

La visualisation interactive de données avec animint2 dans R



1

Ce site est le travail en cours du projet de traduction en français du site en anglais [The animint2 Manual](#), financé par [FabriqueREL](#), livraison prévu pour mars 2026.

Pour créer ce site web à partir des [fichiers source](#), nous avons utilisé les versions suivantes :

```
R.version.string
```

```
[1] "R version 4.5.1 (2025-06-13)"
```

```
packageVersion("animint2")
```

```
[1] '2025.10.22'
```



Part I

Fonctions de base



2

La grammaire des graphiques

Dans ce chapitre nous expliquons la grammaire des graphiques, un modèle puissant pour décrire une vaste gamme de visualisations de données. Après avoir lu ce chapitre, vous serez en mesure

- De citer les avantages de la grammaire des graphiques par rapport aux systèmes précédents de visualisation de données
 - D'installer le package R `animint2`
 - De traduire les esquisses de graphique en code `ggplot` dans R
 - D'afficher des `ggplots` sur des pages web en utilisant `animint2`
 - De créer des `ggplots` multicouches (avec plusieurs `geoms`)
 - De créer des `ggplots` avec facettes
-

2.1 Historique et objectifs de la grammaire des graphiques

La plupart des systèmes informatiques d'analyse de données offrent des fonctions de création de graphiques pour visualiser les tendances des données. Les systèmes les plus anciens proposent des fonctions très générales permettant de tracer les éléments de base d'un graphique, tels que des lignes et des points (par exemple, les packages `graphics` et `grid` dans R). Si vous utilisez l'un de ces systèmes généraux, vous devez assembler vous-même les composants pour obtenir un graphique significatif et interprétable. L'avantage des systèmes généraux réside dans le peu de limites imposées aux types de graphiques qui peuvent être créés. Leur principal inconvénient est qu'ils ne fournissent généralement pas de fonctions permettant d'automatiser les tâches courantes (axes, panneaux, légendes).

Pour remédier aux inconvénients de ces systèmes de tracé généraux, des packages de visualisation tels que `lattice` ont été développés ([Sarkar, 2008](#)). Ils disposent de plusieurs types de graphiques prédéfinis et fournissent une fonction dédiée à la création de chaque type de graphique. Par exemple, `lattice` fournit la fonction `bwplot` pour créer des diagrammes en boîte et des boîtes à moustaches. L'avantage de ces systèmes est qu'ils facilitent grandement la création de graphiques complets avec légendes et panneaux. Par contre, ils ont l'inconvénient d'offrir un ensemble de types de graphiques prédéfinis, ce qui fait qu'il n'est pas facile de créer des graphiques plus complexes.

Les nouveaux systèmes basés sur la grammaire des graphiques se situent entre ces deux extrêmes. Ensuite, Leland Wilkinson propose la grammaire des graphiques pour décrire et créer un grand éventail de graphiques ([Wilkinson, 2005](#)). Hadley Wickham implémente ultérieurement plusieurs idées de la grammaire des graphiques dans le package R `ggplot2` ([Wickham, 2009](#)). Le package `ggplot2` présente plusieurs avantages par rapport aux systèmes précédents.

- Comme les systèmes généraux, et contrairement à **lattice**, **ggplot2** impose peu de limites aux types de graphiques qui peuvent être créés (il n'y a pas de types de graphiques prédéfinis).
- Contrairement aux systèmes généraux, et à l'instar de **lattice**, **ggplot2** permet d'inclure facilement des éléments de graphique courants tels que des axes, des panneaux et des légendes.
- Puisque **ggplot2** est basé sur la grammaire des graphiques, il est nécessaire de mapper explicitement les variables de données aux propriétés visuelles. Plus loin dans ce chapitre, nous expliquerons comment ce mappage permet la création d'esquisses qui peuvent être directement traduites en code R.

Enfin, tous les systèmes évoqués précédemment sont destinés à la création de graphiques statiques, qui peuvent être visualisés aussi bien sur un écran d'ordinateur que sur du papier. Cependant, le sujet principal de ce manuel est **animint2**, un package R pour les graphiques interactifs. Contrairement aux graphiques statiques, les graphiques interactifs sont conçus pour être visualisés sur un ordinateur équipé d'une souris et d'un clavier pour interagir avec le graphique. Étant donné que de nombreux concepts des graphiques statiques sont également utiles dans les graphiques interactifs, le package **animint2** est implémenté en tant que fork du package **ggplot2**. Dans ce chapitre, nous présenterons les principales caractéristiques de **ggplot2** qui seront également utiles pour la conception de graphiques interactifs dans les chapitres suivants.

En 2013, Toby Dylan Hocking et ses étudiants ont créé le package **animint**, qui dépend du package **ggplot2**. Cependant, de 2014 à 2017, **ggplot2** a introduit de nombreux changements incompatibles avec la grammaire interactive du package **animint**, ce qui a conduit en 2018, à la création du package **animint2** qui copie les parties pertinentes du package **ggplot2**. Désormais, **animint2** peut être utilisé sans que **ggplot2** soit installé. Il est en fait conseillé d'utiliser `library(animint2)` sans utiliser `library(ggplot2)`. On peut cependant utiliser **animint2** avec des packages qui importent **ggplot2** (sans l'attacher). À titre d'exemple, voir le [Chapitre 16](#) qui utilise le package **penaltyLearning** (lequel importe **ggplot2**).

2.2 Installer et attacher animint2

Pour installer la dernière version d' **animint2** à partir du dépôt CRAN,

```
if(!requireNamespace("animint2"))install.packages("animint2")
```

Loading required namespace: animint2

Pour installer une version de développement encore plus récente d' **animint2** depuis GitHub,

```
if(!requireNamespace("animint2")){  
  if(!requireNamespace("remotes"))install.packages("remotes")  
  remotes::install_github("tdhock/animint2")  
}
```

Une fois que vous avez installé **animint2**, vous pouvez charger et attacher toutes ses fonctions exportées via :

```
library(animint2)
```

2.3 Traduire des esquisses en ggplots

Dans cette section, nous expliquons comment traduire une esquisse de graphique en code R. Dans ce livre, nous utilisons le mot « esquisse » dans le sens « Première forme, traitée à grands traits et généralement en dimensions réduites, de l'œuvre projetée » ([Larousse](#)). Nous allons souvent présenter une esquisse, dessinée à la main, pour donner l'idée du graphique qu'on veut créer, avant d'écrire le code nécessaire. Chaque esquisse va comporter certains éléments qui nous permettra de facilement écrire le code correspondant. Avant d'écrire le code R pour un graphique que vous voulez créer, nous vous encourageons de faire vos propres esquisses à la main, soit dans un cahier, soit sur un tableau. Nous utilisons comme exemple un ensemble de données de la Banque mondiale. Commençons par charger et examiner un sous-ensemble de ces données.

```
data(BanqueMondiale, package="animint2fr")
BanqueMondiale$Region <- sub(
  " (all income levels)", "", BanqueMondiale$région, fixed=TRUE)
subset(BanqueMondiale, pays=="Canada" & année==2010)
```

```
iso2c country year fertility.rate life.expectancy population
1959    CA  Canada 2010          1.6269          80.7978  34126547
GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c      region
1959                                46211.53                                NA  CAN North America
capital longitude latitude          income          lending
1959  Ottawa  -75.6919  45.4215 High income: OECD Not classified
              Region          région espérance.de.vie taux.de.fertilité année
1959 Amérique du Nord Amérique du Nord          80.7978          1.6269  2010
              pays PIB.par.habitant.USD alphabétisation          revenu
1959 Canada          46211.53          NA Revenu élevé : OCDE
```

```
dim(BanqueMondiale)
```

```
[1] 11342    24
```

Le jeu de données **BanqueMondiale** comprend des mesures telles que le taux de fertilité et l'espérance de vie pour chaque pays sur la période 1960-2010. Le code ci-dessus imprime la ligne pour Canada en 2010, ainsi que la dimension (lignes 11342 et colonnes 24) du tableau de données. Supposons que nous voulions voir s'il existe une relation entre l'espérance de vie et le taux de fertilité. Nous pourrions fixer une année, puis utiliser ces deux variables de données dans un nuage de points. L'esquisse ci-dessous contient les principaux éléments de cette visualisation de données.

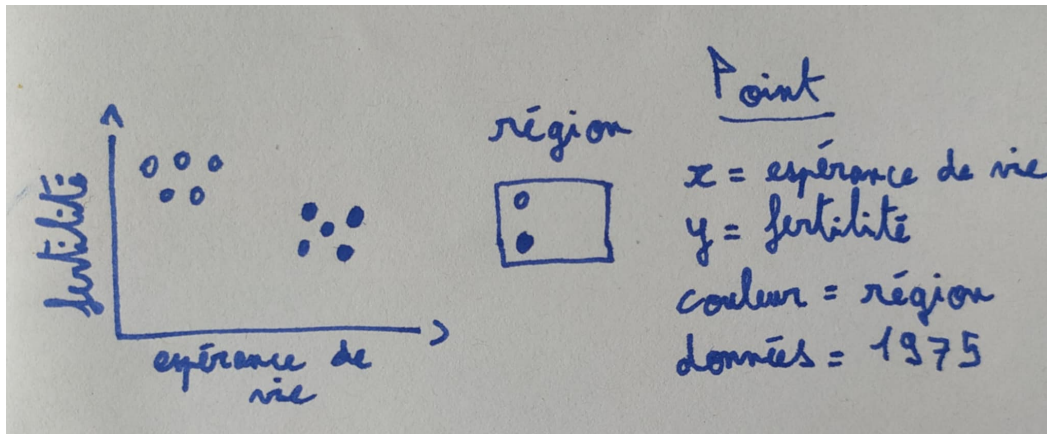


Figure 2.1: World Bank scatterplot

L'esquisse ci-dessus montre l'espérance de vie sur l'axe horizontal (x), le taux de fertilité sur l'axe vertical (y) et une légende pour la région. Ces éléments de l'esquisse peuvent être directement traduits en code R à l'aide de la méthode suivante. Tout d'abord, nous devons construire un tableau de données comportant une ligne par pays pour l'année 1975, et des colonnes nommées `espérance.de.vie`, `taux.de.fertilité`, et `région`. Le jeu de données `BanqueMondiale` possède déjà ces colonnes, il suffit donc de considérer le sous-ensemble pour l'année 1975 :

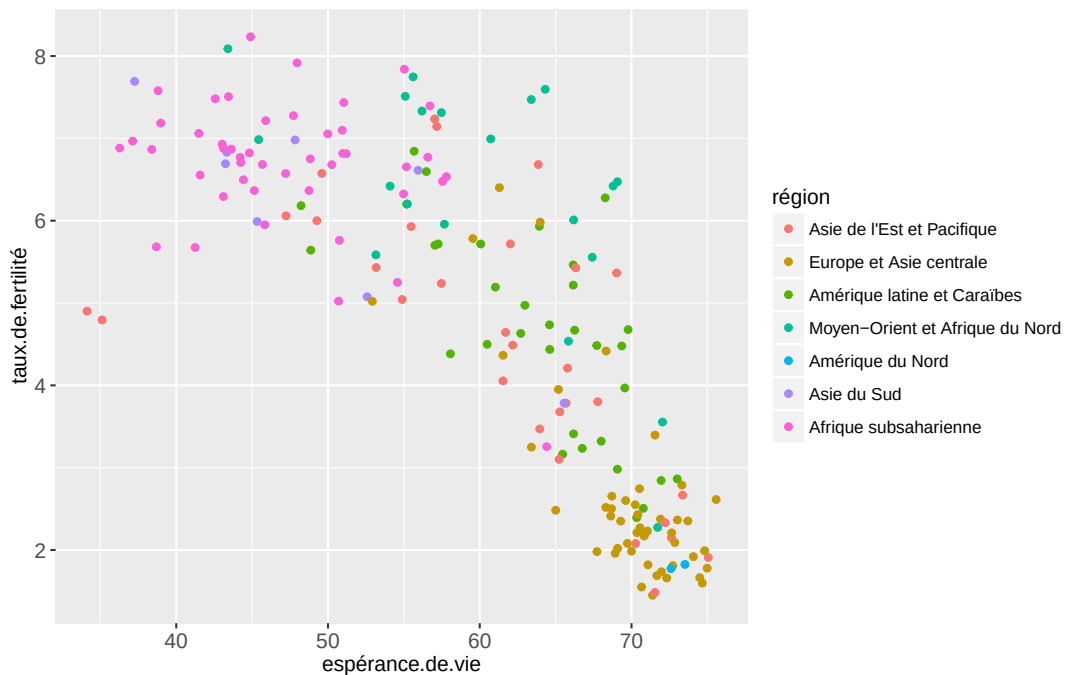
```
BanqueMondiale1975 <- subset(BanqueMondiale, année==1975)
subset(BanqueMondiale1975, pays=="Canada")
```

```
iso2c country year fertility.rate life.expectancy population
1924 CA Canada 1975 1.824 73.52171 23209000
GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c region
1924 7354.268 NA CAN North America
capital longitude latitude income lending
1924 Ottawa -75.6919 45.4215 High income: OECD Not classified
Region région espérance.de.vie taux.de.fertilité année
1924 Amérique du Nord Amérique du Nord 73.52171 1.824 1975
pays PIB.par.habitant.USD alphabétisation revenu
1924 Canada 7354.268 NA Revenu élevé : OCDE
```

Le code ci-dessus imprime la ligne pour Canada en 1975. L'étape suivante consiste à utiliser les notes de l'esquisse pour coder un ggplot avec `aes()` qui fait correspondance entre variables et propriétés visuelles :

```
nuage_de_points <- ggplot()+
  geom_point(
    mapping=aes(x=espérance.de.vie, y=taux.de.fertilité, color=région),
    data=BanqueMondiale1975)
nuage_de_points
```

Warning: Removed 27 rows containing missing values (geom_point).

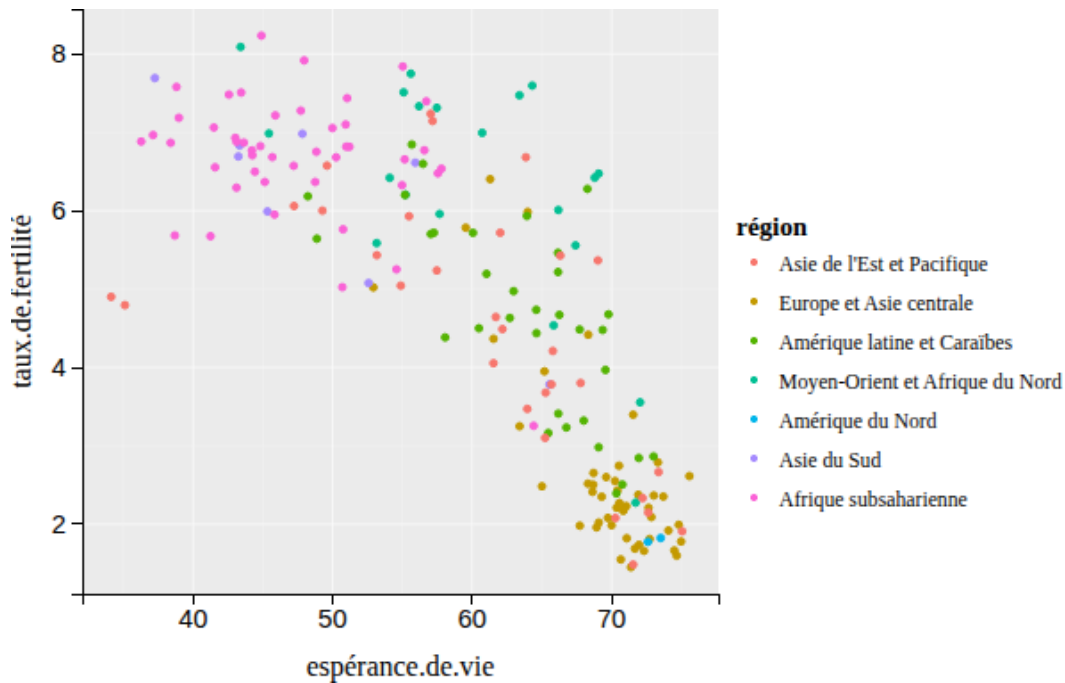


La fonction `aes()` est appelée avec les noms des propriétés visuelles (`x`, `y`, `color`) et les valeurs des variables de données correspondantes (`espérance.de.vie`, `taux.de.fertilité`, `région`). Cette correspondance est appliquée aux variables du tableau de données `BanqueMondiale1975`, afin de créer les propriétés du `geom_point`. Le ggplot a été sauvegardé dans une variable nommée `nuage_de_points` qui, lorsqu'imprimé sur la ligne de commande R, affiche le graphique sur un périphérique graphique. Notez que nous avons automatiquement une légende `région` en couleur.

2.4 Afficher des ggplots sur des pages web à l'aide d'animint

Cette section explique comment `animint2` peut être utilisé pour afficher des ggplots sur des pages web. Le ggplot de la section précédente peut être affiché avec `animint2`, en utilisant la fonction `animint`

```
animint(nuage_de_points)
```



Lors de l'exécution du code ci-dessus, si l'animint ne s'affiche pas dans votre navigateur web pour une raison quelconque (par exemple si vous voyez une page web vierge), veuillez consulter notre [FAQ](#) qui vous aidera à trouver une solution. La fonction `animint()` crée une liste de classe "`animint`", puis R exécute la fonction `print.animint()` grâce au [système d'objets S3](#). La fonction `print.animint()` fait appel ensuite à notre compilateur, qui prend la liste en entrée et qui produit une page web avec une visualisation de données. Le compilateur est la fonction `animint2dir()` qui compile `nuage_de_points` en un dossier de fichiers de données (CSV) et de codes (HTML, JavaScript) qui peuvent être affichés dans un navigateur web.

Lorsqu'il est visualisé dans un navigateur web, le graphique animint devrait être pratiquement identique aux versions statiques produites par les périphériques graphiques R standard, à la différence que la légende de la région est interactive : en cliquant sur une entrée de la légende, les points de cette couleur sont cachés ou affichés.

Exercice: essayez de modifier le contenu du `aes()` du `ggplot`, puis de créer une nouvelle visualisation. Les variables quantitatives comme `population` sont mieux représentées par les axes (`x` ou `y`) ou par la taille du cercle (`size`). Les variables qualitatives comme `lending` sont mieux représentées en utilisant les propriétés visuelles `color` ou `fill`.

2.5 Graphiques multicouches (plusieurs geoms)

Dans les ggplots, nous utilisons le mot « couche » comme synonym de « geom » comme `geom_point` que nous avons utilisé ci-dessus. Il existe d'autres geoms comme `geom_line` et `geom_path` qu'on peut utiliser pour dessiner les courbes. On appelle un graphique « multicouche » quand on utilise plusieurs geoms dans le même graphique, dessinés les uns sur les autres. Un graphique multicouche est utile lorsque vous souhaitez afficher

plusieurs geoms ou ensembles de données différents dans le même graphique. Par exemple, considérons l'esquisse suivante qui ajoute une couche de `geom_path` à la visualisation de données précédente.

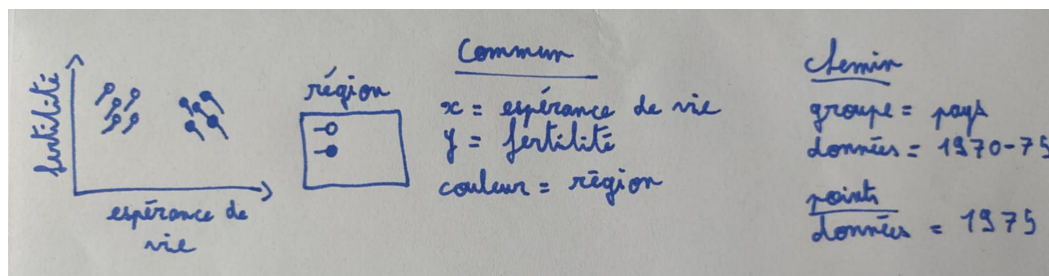
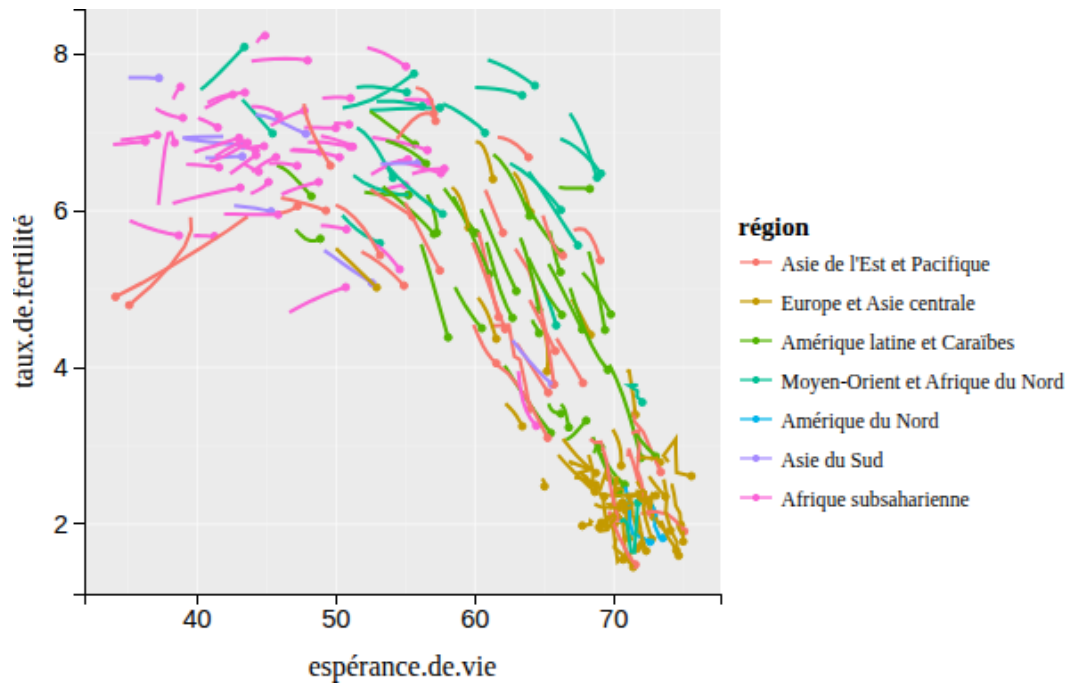


Figure 2.2: multi-layer WorldBank data viz

Notez que l'esquisse ci-dessus comprend deux geoms différents (point et path). Les deux geoms partagent une définition commune des éléments `x`, `y`, et `color`, mais avec des ensembles de données différents.

Nous traduisons ci-dessous cette esquisse en code R.

```
BanqueMondialeAvant1975 <- subset(
  BanqueMondiale, 1970 <= année & année <= 1975)
gg_deux_couches <- nuage_de_points+
  geom_path(aes(
    x=espérance.de.vie,
    y=taux.de.fertilité,
    color=région,
    group=pays),
  data=BanqueMondialeAvant1975)
vis.deux.couches <- animint(gg_deux_couches)
vis.deux.couches
```



Dans le code ci-dessus, nous faisons un nouveau jeu de données `BanqueMondialeAvant1975`, et ensuite nous rajoutons un `geom_path` au `nuage_de_points` ce qui donne `gg_deux_couches`, un ggplot avec deux couches. Ensuite nous sauvegardons la valeur rendue de l'appel à `animint()` dans l'objet `vis.deux.couches`, qu'on imprime pour l'afficher. Dans ce manuel, nous utiliserons souvent des noms de variables commençant par `vis` pour désigner les objets de classe "`animint`", qui sont en fait des listes de ggplots et d'options.

Le graphique ci-dessus est une visualisation de données avec deux couches :

- le `geom_point` montre l'espérance de vie, le taux de fertilité et la région de tous les pays en 1975.
- le `geom_path` montre les mêmes variables pour les cinq années précédentes.

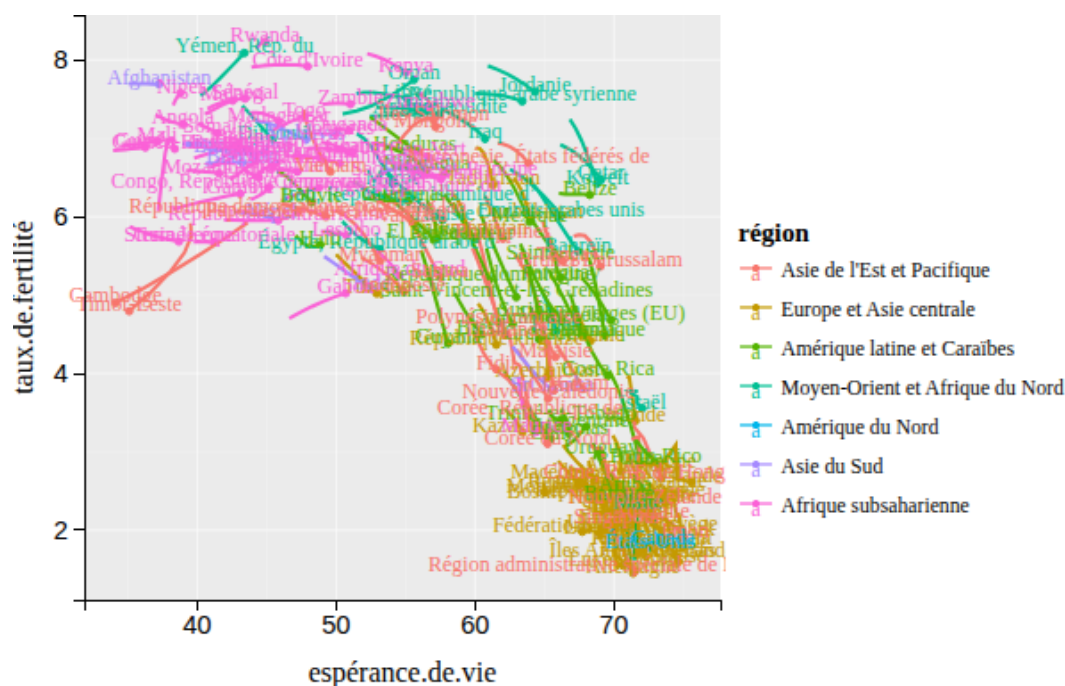
L'ajout de `geom_path` montre comment les pays ont changé au fil du temps. Les données de la plupart des pays se sont déplacées vers la droite et vers le bas, ce qui signifie une espérance de vie plus élevée et un taux de fertilité plus faible. Il y a cependant quelques exceptions. Par exemple, les deux pays d'Asie de l'Est situés en bas à gauche ont vu leur espérance de vie diminuer au cours de cette période. Par ailleurs, certains pays ont vu leur taux de fertilité augmenter.

Exercice: essayez de remplacer la légende `région` par une légende `revenu`. Indice : vous devez utiliser la même spécification `aes(color=revenu)` pour tous les geoms. Vous pouvez utiliser `scale_color_manual` avec une palette de couleurs séquentielle, voir `RColorBrewer::display.brewer.all(type="seq")` et lire l'annexe pour plus de détails.

Pouvons-nous ajouter le nom des pays à la visualisation de données ? Ci-dessous, nous ajoutons une autre couche avec une étiquette de texte pour le nom de chaque pays.

```
gg_trois_couches <- gg_deux_couches +
  geom_text(aes(
```

```
x=espérance.de.vie,
y=taux.de.fertilité,
color=région,
label=pays),
data=BanqueMondiale1975)
animint(gg_trois_couches)
```



Cette visualisation n'est pas très facile à lire, car il y a beaucoup d'étiquettes de texte qui se chevauchent. La légende interactive des régions aide un peu, puisqu'elle permet à l'utilisateur de masquer les données des régions sélectionnées. Cependant, ce serait encore mieux si l'utilisateur pouvait afficher et masquer le texte pour les pays individuellement. Ce type d'interaction peut être réalisé en utilisant les mots-clés `showSelected` et `clickSelects` que nous expliquons dans les chapitres 3 et 4.

Pour l'instant, passons à une des grandes forces d'`animint` : la visualisation de données à l'aide de plusieurs graphiques interconnectés.

2.6 Visualisations avec plusieurs ggplots interconnectés

L'utilisation de plusieurs ggplots est utile lorsque vous souhaitez présenter des plusieurs jeux de données, ou bien plusieurs façons de regarder un seul jeu de données. Dans ce genre de visualisation, il est courant d'utiliser au moins deux ggplots :

- un pour afficher un résumé ou survol de données, qu'on peut cliquer pour sélectionner un sous-ensemble d'intérêt,
- un pour afficher plus de détails pour les données sélectionnées.

Prenons l'exemple d'une visualisation comportant deux graphiques : une série temporelle avec les données de la Banque mondiale de 1960 à 2010 (sommaire) et un nuage de points avec les données de 1975 (détails). Nous esquissons ci-dessous le graphique des séries temporelles.

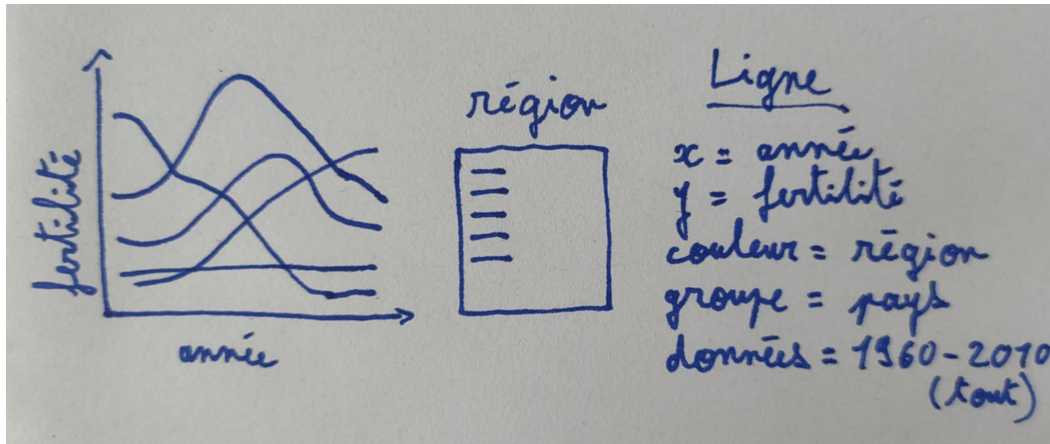


Figure 2.3: WorldBank data viz with two plots

Notez que l'esquisse ci-dessus peut être directement traduite dans le code R ci-dessous. Nous copions ensuite la visualisation existante (`vis.deux.couches`), puis nous rajoutons un `ggplot` à un nouvel élément nommé `timeSeries`.

```
vis.deux.graphes <- vis.deux.couches
vis.deux.graphes$timeSeries <- ggplot()+
  geom_line(aes(
    x=année,
    y=taux.de.fertilité,
    color=région,
    group=pays),
    data=BanqueMondiale)
```

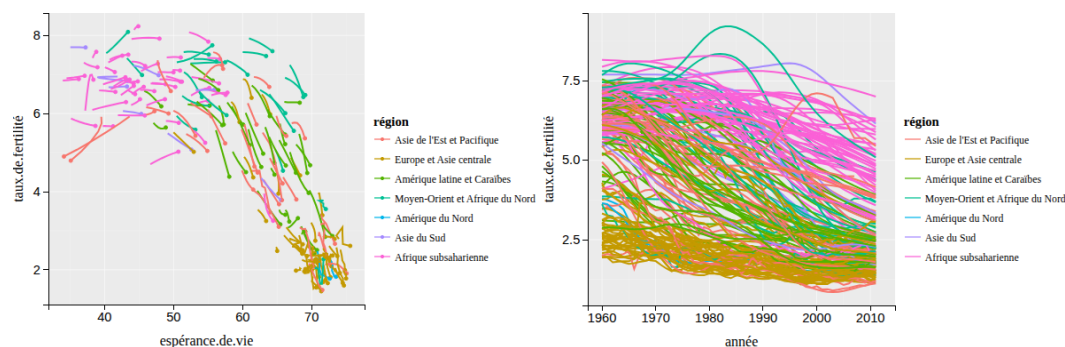
Il en résulte une liste nommée de deux éléments (les deux éléments sont des `ggplots` de classe `gganimint`).

```
summary(vis.deux.graphes)
```

	Length	Class	Mode
plot1	9	gganimint	list
timeSeries	9	gganimint	list

On peut imprimer ou afficher cette liste de visualisation de données en tapant son nom. Puisque la liste contient deux `ggplots`, `animint2` affiche la visualisation de données sous la forme de deux graphiques liés.

```
vis.deux.graphes
```



La visualisation de données ci-dessus contient deux ggplots, qui mappent chacun différentes variables de données sur le plan horizontal x. La série temporelle utilise `aes(x=année)` et montre un résumé des valeurs du taux de fertilité pour toutes les années. Le nuage de points utilise `aes(x=espérance.de.vie)` et montre les détails de la relation entre le taux de fertilité et l'espérance de vie en 1975.

Essayez de cliquer sur une entrée de légende dans le nuage de points ou dans la série temporelle ci-dessus. Vous devriez voir les données et les légendes des deux graphiques se mettre à jour simultanément. Étant donné que `aes(color=région)` a été spécifié pour les deux graphiques, `animint` crée une seule variable de sélection partagée appelée `région`. Cliquer sur l'une des légendes a pour effet de mettre à jour l'ensemble des régions sélectionnées, `animint` met donc à jour les légendes et les données dans les deux graphiques en conséquence. Il s'agit du mécanisme principal utilisé par `animint` pour créer des visualisations de données interactives avec des graphiques liés, nous en discuterons plus en détail dans les deux prochains chapitres.

Exercice: utilisez `animint` pour créer une visualisation de données avec trois graphiques, en créant une liste avec trois ggplots. Par exemple, vous pouvez ajouter une série temporelle d'une autre variable de données telle que `espérance.de.vie` ou `population`.

Notez que les deux ggplots mappent la variable `taux.de.fertilité` sur l'axe y. Cependant, comme il s'agit de deux graphiques distincts, les plages de leurs axes y sont calculées séparément. Cela signifie que même lorsque les deux graphiques sont affichés côte à côte, les deux axes y ne sont pas exactement alignés. C'est problématique, la visualisation des données serait plus facile à décoder si chaque unité d'espace vertical représentait la même quantité de taux de fertilité. Pour obtenir cet effet, nous utilisons les facettes dans la section suivante.

2.7 Visualisation de données avec plusieurs panneaux (facettes)

Un ggplot peut avoir des facettes (panneaux), qui sont des sous-graphiques, chacun d'eux présentant un sous-ensemble de données différent. On utilise le mot « facette » pour les fonctions `facet_*` correspondantes. L'un des principaux atouts de ggplots est qu'il est relativement facile de créer différents types de graphiques avec plusieurs facettes. Les facettes peuvent servir deux objectifs différentes :

- L'alignement des axes de plusieurs graphiques connexes contenant différents geoms. Cette technique facilite la comparaison entre plusieurs geoms différents et est également utile pour la visualisation interactive des données.

- La division des données d'un geom en plusieurs panneaux. Cette technique facilite la comparaison entre des sous-ensembles de données mais elle est moins utile pour la visualisation interactive de données (l'interactivité peut souvent la remplacer et produire le même effet comparatif pour les sous-ensembles de données).

2.7.1 Des geoms distincts dans chaque panneau (axes alignés)

Nous commençons par expliquer comment les facettes sont utiles pour aligner les axes quand on veut afficher des différents geoms dans chaque facette. Considérons l'esquisse ci-dessous qui contient un graphique avec deux panneaux.

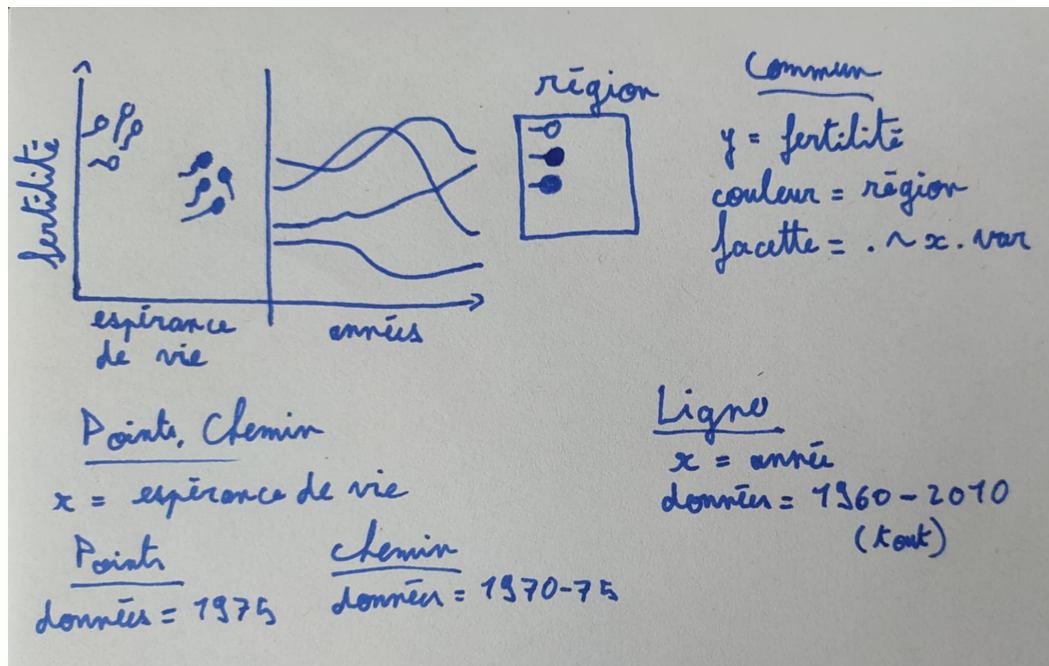


Figure 2.4: World Bank aligned plot

Notez que les deux panneaux tracent des geoms différents, chacun avec son propre `aes()`. Le `geom_point` et le `geom_path` dans le panneau de gauche ont `x=espérance.de.vie`, et le `geom_line` du panneau de droite a `x=année`. Notez également que nous avons écrit `facet=x.var`, nous devons donc ajouter une variable `x.var` à chacun des trois ensembles de données. Nous traduisons cette esquisse en code R ci-dessous.

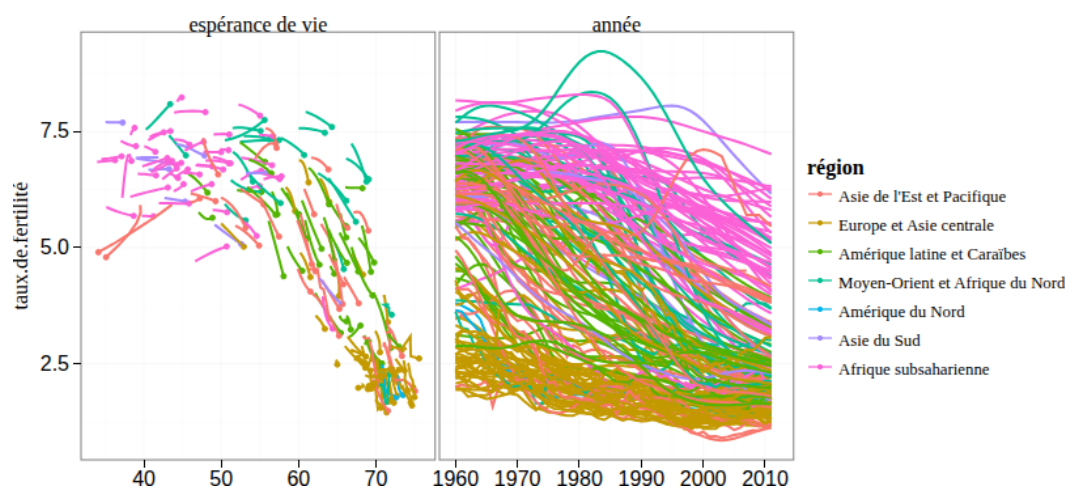
```
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("espérance de vie", "année")))
}
(vis.aligné <- animint(
  scatter=ggplot()+
    theme_bw()+
    theme_animint(width=600)+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
```



```

data=add.x.var(BanqueMondiale1975, "espérance de vie")+
geom_path(aes(
  x=espérance.de.vie, y=taux.de.fertilité, color=région,
  group=pays),
data=add.x.var(BanqueMondialeAvant1975, "espérance de vie")+
geom_line(aes(
  x=year, y=taux.de.fertilité, color=région, group=pays),
data=add.x.var(BanqueMondiale, "année")+
xlab("")+
facet_grid(. ~ x.var, scales="free"))

```



La visualisation de données ci-dessus contient un seul ggplot avec deux panneaux et trois couches. Le panneau de gauche montre les `geom_point` et `geom_path` et le panneau de droite montre les `geom_line`. Les panneaux ont un axe commun pour le taux de fertilité, ce qui permet de comparer directement le `geom_line` du panneau des séries temporelles avec le `geom_point` et le `geom_path` du panneau du nuage de points.

Il est à noter que nous avons utilisé la méthode `add.x.var` pour ajouter une variable `x.var` à chaque ensemble de données, puis nous avons utilisé cette variable dans la fonction `facet_grid(scales="free")`. Nous appelons cette méthode Ajouter une variable ensuite des facettes. Elle est généralement utile pour créer une visualisation de données à plusieurs panneaux avec des axes alignés. Ainsi, pour changer l'ordre des panneaux dans la visualisation, il nous suffirait de modifier l'ordre des niveaux de facteurs dans la définition de `add.x.var`.

Notez également que `theme_bw()` fait les bordures de panneaux noires et des fonds de panneaux blancs, et que `theme(panel.margin=0)` pourrait être utilisé pour supprimer l'espacement entre les panneaux. Cette élimination libère de l'espace pour les panneaux, ce qui permet de mettre l'accent sur les données. Nous appelons cette méthode Les facettes économes en espace, elle est généralement utile dans tout ggplot avec des facettes.

Dans la visualisation ci-dessus, les étiquettes de texte se chevauchent un peu, ce qui peut être corrigé par l'une des méthodes suivantes (exercice pour le lecteur).

- en utilisant l'argument `breaks` de `scale_x_continuous()`
- en réduisant la taille du texte avec `theme()` et `element_text()`
- en augmentant la largeur du graphique à l'aide de `theme_animated()` voir le Chapitre 6 pour plus d'informations.

2.7.2 Facettes pour comparaison des sous-ensembles de données

La deuxième raison d'utiliser des graphes avec plusieurs panneaux dans une visualisation de données est de comparer des sous-ensembles d'observations. Cette approche facilite la comparaison entre les sous-ensembles de données et peut être utilisée dans au moins deux situations différentes :

- L'ensemble de données d'un geom comporte trop d'observations pour être visualisé efficacement dans un seul panneau.
- Vous souhaitez comparer différents sous-ensembles de données tracées pour un geom

Prenons l'exemple de l'esquisse ci-dessous.

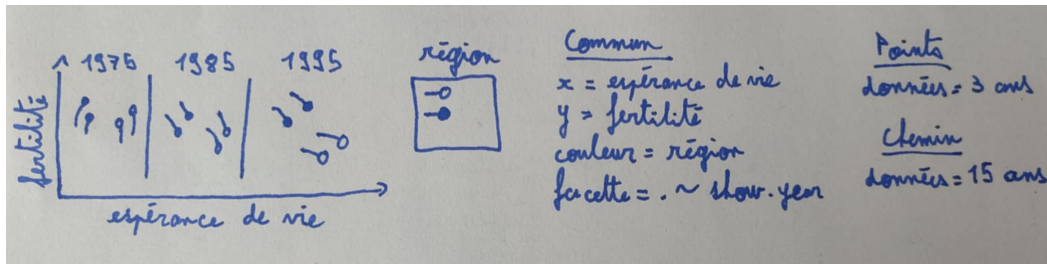


Figure 2.5: World Bank panels plot

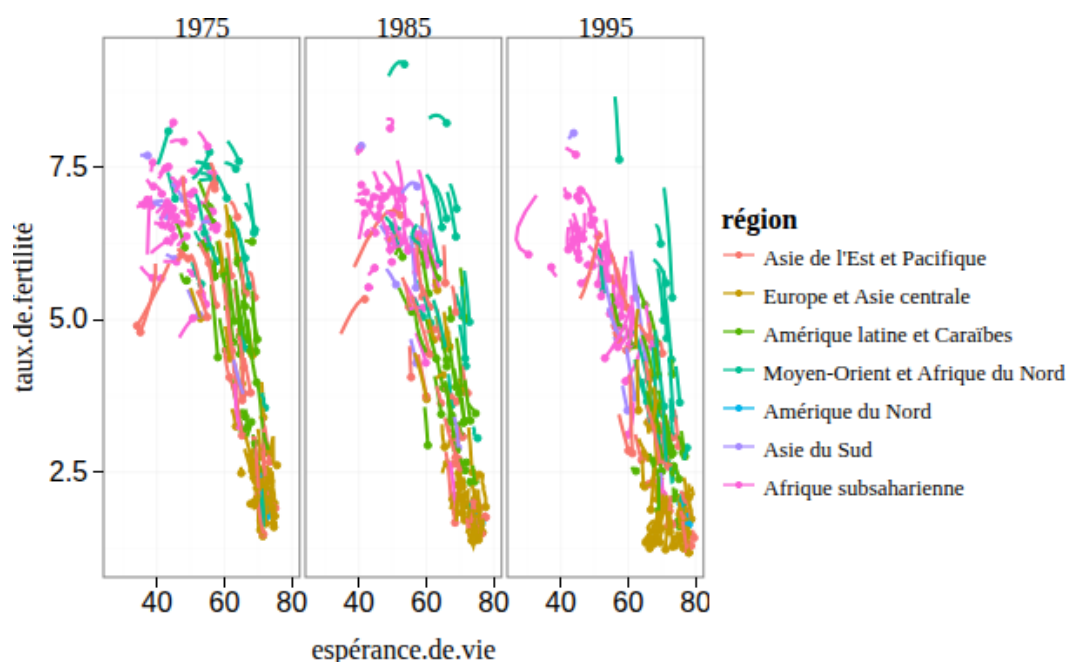
Notez que les trois panneaux tracent les deux mêmes geoms, `geom_point` et `geom_path`. Puisque `facet=afficher.année` et qu'il y a trois panneaux, nous devons créer des tableaux de données qui auront trois valeurs pour la variable `afficher.année`. Le `geom_point` ne contient des données que pour 3 ans, et le tableau `geom_path` a des données pour 15 ans (mais 3 valeurs de `afficher.année`). Le code ci-dessous crée ces deux ensembles de données pour trois années de l'ensemble de données de la Banque mondiale.

```
afficher.point.list <- list()
afficher.path.list <- list()
for(afficher.année in c(1975, 1985, 1995)){
  afficher.point.list[[paste(afficher.année)]] <- data.frame(
    afficher.année, subset(
      BanqueMondiale, année==afficher.année))
  afficher.path.list[[paste(afficher.année)]] <- data.frame(
    afficher.année, subset(
      BanqueMondiale,
      afficher.année - 5 <= année & année <= afficher.année))
}
afficher.point <- do.call(rbind, afficher.point.list)
afficher.path <- do.call(rbind, afficher.path.list)
```

Nous avons utilisé une boucle `for` sur trois valeurs de `afficher.année`, la variable que nous utiliserons plus tard dans `facet_grid`. Pour chaque valeur de `afficher.année` nous stockons un sous-ensemble de données sous la forme d'un élément nommé d'une liste. Après la boucle `for`, nous utilisons `do.call` avec `rbind` pour combiner les sous-ensembles de données. Il s'agit d'un exemple de la méthode Liste de tableau de données qui est généralement utile pour la visualisation interactive des données.

Ci-dessous, nous appliquons les facettes sur la variable `afficher.année` pour créer une visualisation de données avec trois panneaux.

```
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
      data=afficher.point)+
    geom_path(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région,
      group=pays),
      data=afficher.path)+
    facet_grid(. ~ afficher.année)+
    theme_bw())
```



La visualisation de données ci-dessus contient un seul ggplot avec trois facettes. Elle montre une plus grande partie de l'ensemble des données de la Banque mondiale que les visualisations précédentes qui ne montraient que les données de 1975. Cependant, elle ne montre qu'un sous-ensemble de données relativement restreint. Vous pourriez être tenté d'utiliser un panneau pour afficher chaque année (pas seulement 1975, 1985 et 1995). Sachez toutefois que ce type de visualisation de données à plusieurs panneaux fonctionne bien seulement s'il y a un nombre limité de sous-ensembles de données. Au-delà d'une dizaine de panneaux, il devient difficile de voir toutes les données simultanément, et donc de faire des comparaisons significatives.

Au lieu de montrer toutes les données à la fois, nous pouvons créer une visualisation de données animée qui présente à l'observateur différents sous-ensembles de données au fil du temps. Dans le [chapitre suivant](#) nous montrerons comment le nouveau mot-clé `showSelected` peut être utilisé pour réaliser une animation, et nous révélerons plus de détails sur cet ensemble de données.

2.8 Résumé du chapitre et exercices

Ce chapitre a présenté les bases de la visualisation de données statiques avec `ggplot2`. Nous avons montré comment `animint` peut être utilisé pour afficher une liste de `ggplots` dans un navigateur web. Nous avons expliqué comment `ggplot2` facilite la création de graphiques avec plusieurs couches, et plusieurs facettes.

Exercices :

- Quels sont les trois principaux avantages de `ggplot2` par rapport aux systèmes antérieurs tels que `grid` et `lattice` ?
- Pourquoi utiliserait-on plusieurs couches dans un même graphique ?
- Quelles sont les deux raisons différentes de créer des graphiques à plusieurs facettes ? Laquelle est plus utile dans les visualisations interactives ?
- Définissons “ $A < B$ ” comme signifiant qu’“un B peut contenir plusieurs A ”. Laquelle des affirmations suivantes est vraie ?
 - `ggplot < panel`
 - `panel < ggplot`
 - `ggplot < animint`
 - `animint < ggplot`
 - `layer < panel`
 - `panel < layer`
 - `layer < ggplot`
 - `ggplot < layer`
- Dans les facettes `vis.alignée`, pourquoi est-il important d’utiliser l’argument `scales="free"` ?
- Dans `vis.alignée` nous avons montré un `ggplot` avec un panneau de nuage de points à gauche et un panneau de séries temporelles à droite. Réalisez une autre version de la visualisation de données avec le panneau des séries temporelles à gauche et le panneau du nuage de points à droite.
- Dans `vis.alignée` le nuage de points affiche le taux de fertilité et l’espérance de vie, mais la série chronologique n’affiche que le taux de fertilité. Réalisez une autre version de la visualisation de données qui montre les deux séries temporelles. Indice : utilisez les panneaux horizontaux et verticaux dans `facet_grid`.
- Utilisez `aes(size=population)` dans le nuage de points pour indiquer la population de chaque pays. Indice : `scale_size_animint(pixel.range=c(5, 10))` signifie que des cercles d’un rayon de 5/10 pixels doivent être utilisés pour représenter la population minimale/maximale.
- Créez une visualisation de données à plusieurs panneaux qui montre chaque année des données `BanqueMondiale` dans un différent panneau. Quelles sont les limites de l’utilisation de graphiques statiques pour visualiser ces données ?
- Créez un `vis.alignée` en utilisant un système de tracé qui n’est pas basé sur la grammaire des graphiques. Par exemple, vous pouvez utiliser les fonctions du package `graphics` dans R (`plot`, `points`, `lines` etc.), ou `matplotlib` en Python. Quels sont les avantages de `ggplot2` et `animint` ?

Ensuite, dans le [Chapitre 3](#) nous expliquons le mot-clé `showSelected`, qui nous permettra de préciser l’interactivité ainsi que l’animation.

3

Le mot-clé `showSelected`

Dans ce chapitre, nous vous présentons `showSelected`, l'un des deux principaux mots-clés introduits par `animint2` pour la visualisation interactive de données. Après avoir lu ce chapitre, vous saurez :

- Utiliser le mot-clé `showSelected` dans vos esquisses pour spécifier des geoms pour lesquels seul un sous-ensemble de données doit être tracé à la fois.
- Utiliser les menus de sélection pour modifier le sous-ensemble de données tracées.
- Préciser des transitions fluides entre les sous-ensembles de données avec `aes(key)` et l'option `duration`.
- Créer des visualisations de données animées à l'aide de l'option `time`.

3.1 Esquisses avec `showSelected`

Dans le chapitre 2, nous avons expliqué comment planifier les codes nécessaires pour créer une visualisation avec une esquisse qui comprend les éléments principaux (geoms, axes, légendes, données). Dans cette section, nous allons expliquer comment le mot-clé `showSelected` peut être utilisé dans les esquisses. Le mot-clé `showSelected` est utilisé pour préciser les variables à utiliser pour sélectionner un sous-ensemble de données. Chaque geom possède son propre ensemble de données et sa propre définition de variables `showSelected`. Cela signifie que chaque geom peut afficher son propre sous-ensemble de données (qui est différent que les autres geoms).

En fait, nous avons déjà utilisé le mot-clé `showSelected` : il avait été généré automatiquement par les légendes interactives que nous avons créées dans les deux chapitres précédents. Considérons, par exemple, la courbe de Keeling du chapitre 1 présentée ci-dessous.

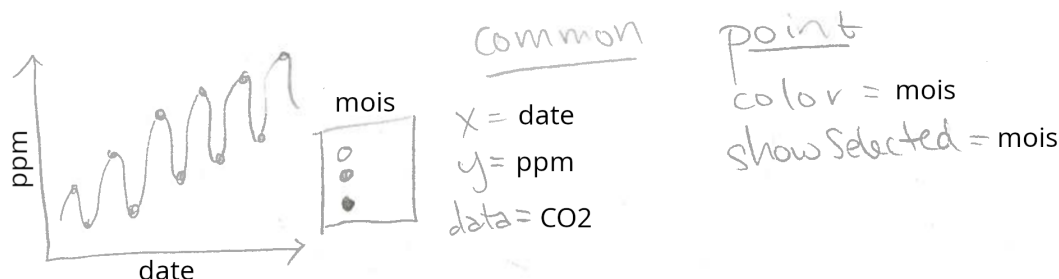


Figure 3.1: Esquisse de la courbe de Keeling

L'esquisse ci-dessus comprend `showSelected=mois` pour le `geom_point` ce qui signifie qu'elle

doit afficher le sous-ensemble de données pour les mois sélectionnés. En revanche, puisque `geom_line` n'inclut pas `showSelected`, il affiche toujours l'ensemble complet des données (peu importe les mois sélectionnés).

Prenons un autre exemple : l'esquisse ci-dessous de la première visualisation de données de la Banque mondiale, du chapitre 2.

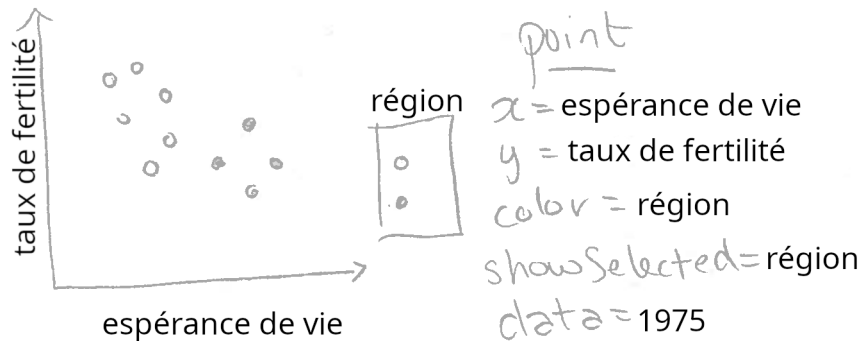


Figure 3.2: Esquisse Banque mondiale avec `showSelected` région

L'esquisse ci-dessus spécifie `showSelected=région` pour le `geom_point`, ce qui signifie qu'elle doit afficher le sous-ensemble de données pour les régions sélectionnées.

Notez que l'esquisse dans chapitre 2 ne précisait pas explicitement `showSelected=région`. Nous avons simplement indiqué `aes(color=région)`, et `animint2` a automatiquement ajouté une variable `showSelected` correspondante. En général, `animint2` rajoute un mot-clé `showSelected` pour chaque variable utilisée dans une légende qualitative.

Cependant, le mot-clé `showSelected` n'est pas limité aux légendes qualitatives. Vous pouvez utiliser `showSelected` pour toutes les variables de votre choix, en précisant explicitement les variables dans l'argument `showSelected` du `geom`.

Chaque variable utilisée avec `showSelected` est traitée par `animint2` comme une variable de sélection. Par exemple, la visualisation de la courbe de Keeling a une variable de sélection (`mois`), tout comme la visualisation de la Banque mondiale (`région`). Pour chaque variable de sélection, `animint2` stocke les valeurs sélectionnées. Lorsque la sélection change, `animint2` met à jour le sous-ensemble de données affiché.

Les deux visualisations présentées ci-dessus ne comportent qu'une seule variable de sélection. Par contre, une visualisation de données peut avoir autant de variables de sélection que vous le souhaitez. Dans la section suivante, nous étudierons une visualisation des données de la Banque mondiale, avec les variables de sélection `région` et `année`.

3.2 Sélection de sous-ensembles avec menus

Considérons l'esquisse suivante avec une variable `showSelected` de plus et un ensemble de données différent.

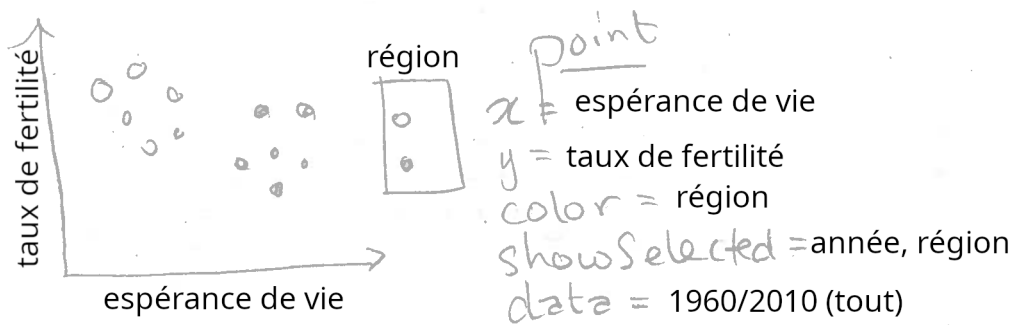
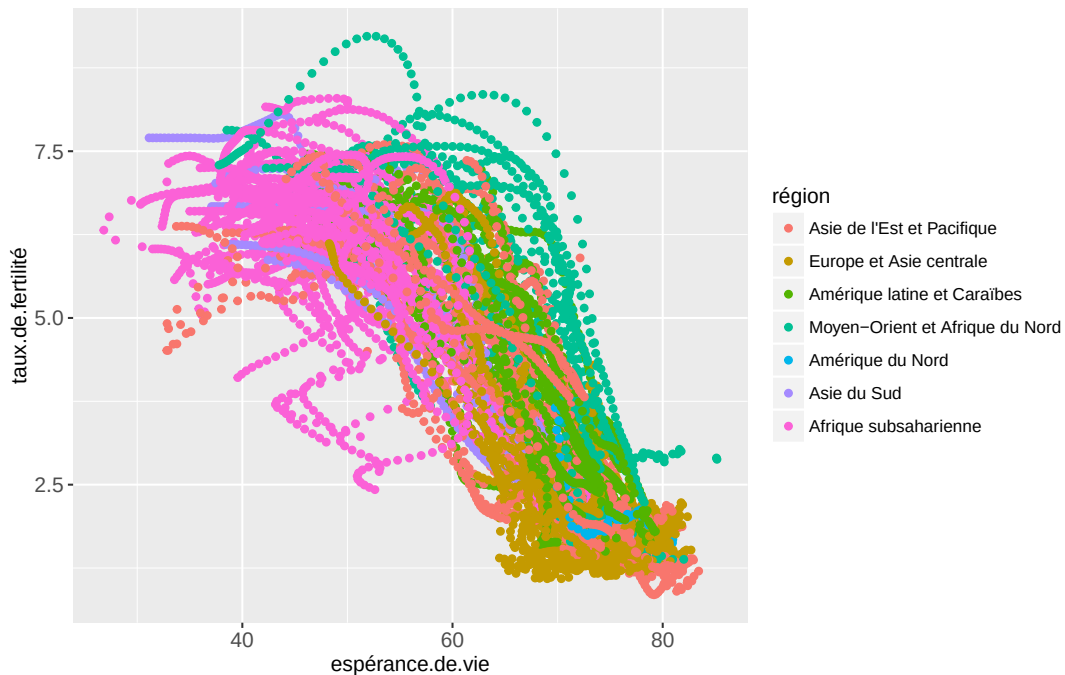


Figure 3.3: Esquisse Banque mondiale avec showSelected région et année

Notez qu'il y a deux variables `showSelected` : `région` et `année`. Notez également que les données sont spécifiées pour toutes les années (mais une seule année sera affichée à la fois, grâce au `showSelected=année`). Ci-dessous, nous traduisons l'esquisse en code R.

```
library(animint2)
data(BanqueMondiale, package="animint2fr")
nuage <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région),
    showSelected="année",
    data=BanqueMondiale)
nuage
```

Warning: Removed 1490 rows containing missing values (geom_point).



Notez que le code ci-dessus contient le mot-clé `showSelected`, une nouveauté dans `animint2` par rapport à `ggplot2`. Le mot-clé `showSelected` est ignoré lorsque le graphique est affiché comme ci-dessus, avec les fonctions d’affichage habituelles de R, qui dessinent un nuage de points incluant toutes les années.

En revanche, donner le code ci-dessus comme argument pour `animint()` donne la visualisation interactive ci-dessous, qui dessine une année à la fois.



Notez que la visualisation ci-dessus comporte deux variables de sélection : `région` et `année` (`color=région` fait automatiquement de `région` une variable `showSelected`). Chaque variable dispose d’un menu sous la visualisation de données pour la modification de la sélection en cours. Dans cette visualisation, ces menus sont affichés par défaut. Pour les masquer, cliquez sur le bouton “Hide selection menus”, et pour les réafficher sur “Show selection menus”.

Les variables discrètes, telles que `région`, ont une sélection multiple par défaut, de sorte que plusieurs valeurs sont sélectionnées et affichées à la fois. Essayez de modifier la région sélectionnée dans la légende interactive et dans le menu de sélection. Lorsque vous modifiez la sélection à l’aide d’une des méthodes, la légende interactive et le menu de sélection sont tous les deux mis à jour, pour refléter la nouvelle sélection.

Nous utilisons les termes “manipulation directe” et “manipulation indirecte” pour décrire ces différentes façons de modifier la sélection. La manipulation directe est généralement plus facile à comprendre, parce qu’il s’agit de cliquer sur les objets que l’on souhaite modifier dans le graphique. De leur côté, les techniques de manipulation indirecte, telles que les menus, sont utiles dans d’autres cas (par exemple, la sélection du nom d’un pays). Dans la visualisation ci-dessus, vous pouvez modifier la valeur de `région` en utilisant soit la légende, soit le menu. L’utilisation de la légende est une technique de manipulation plus directe,

puisque la légende est dessinée plus près du nuage de points qui sera mis à jour.

D'autres variables de sélection, comme `année`, ont une sélection simple par défaut, de sorte qu'une seule valeur est sélectionnée et affichée à la fois. Essayez de modifier la valeur sélectionnée pour la variable `année` à l'aide du menu de sélection. Vous devriez voir le nuage de points se mettre à jour immédiatement pour afficher le taux de fertilité et l'espérance de vie de tous les pays au cours de l'année que vous avez sélectionnée.

Exercice couches multiples : ajoutez un autre geom à ce nuage de points interactif. Comme dans le chapitre 2, vous pouvez utiliser un `geom_text` pour afficher le nom de chaque pays (facile), ou un `geom_text` pour afficher l'année sélectionnée (moyen), ou un `geom_path` pour afficher les données des 5 années précédentes (difficile). Astuce : utilisez `showSelected=année` dans tous les geoms.

Exercice multi-plot : ajoutez une série temporelle à la visualisation de données ci-dessus. Comme dans le chapitre 2, vous pouvez utiliser un `geom_line` pour afficher le taux de fertilité de chaque pays, pour toutes les années. Ajoutez un `geom_vline` avec `showSelected=année` pour mettre en évidence l'année sélectionnée.

3.3 Transitions : l'option *duration* et *aes(key)*

Vous avez peut-être remarqué la présence de boutons sous chaque visualisation de données créée par `animint2`. Cliquez sur le bouton "Show animation controls" de la visualisation présentée ci-dessus. Vous verrez alors que ce tableau contient une ligne pour chaque variable de sélection. Les zones de texte indiquent le nombre de millisecondes utilisées pour les durées de transition après la mise à jour de chaque variable de sélection. Pour chaque variable de sélection, la durée de transition par défaut est 0, ce qui signifie que les données seront immédiatement placées à leur nouvelle position après la mise à jour de chaque variable.

Pour illustrer l'importance des durées de transition, essayez de changer la valeur de la durée de transition de la variable `année` pour 2000. Ensuite, utilisez le menu pour modifier la valeur sélectionnée de la variable `année`. Vous devriez voir les points de données se déplacer lentement vers leurs nouvelles positions en 2 secondes.

Certaines transitions n'entraînent qu'un léger déplacement des points vers des positions proches (par exemple, 1979-1980). D'autres transitions entraînent un déplacement beaucoup plus important des points, vers des localisations plus éloignées (par exemple 1980-1981). Pourquoi?

Les transitions fluides n'ont de sens que pour les points de données qui existent à la fois avant et après la modification de la sélection. Dans le code R ci-dessous, nous calculons un tableau de contingence des points de données qui peuvent être tracés pour chacune de ces trois années.

```
trois.ans <- subset(BanqueMondiale, 1979 <= année & année <= 1981)
can.plot <- with(trois.ans, {
  (!is.na(espérance.de.vie)) & (!is.na(taux.de.fertilité))
})
table(trois.ans$année, can.plot)
```

`can.plot`

	FALSE	TRUE
1979	27	187
1980	27	187
1981	26	188

Le tableau de contingence ci-dessus montre clairement que 187 points peuvent être tracés en 1979 et 1980. Cependant, en 1981, il y a un point de données supplémentaire correspondant à un pays pour lequel nous n'avons pas de données en 1980. Nous présentons ci-dessous les données de ce pays, le Kosovo.

```
subset(trois.ans, pays=="Kosovo")
```

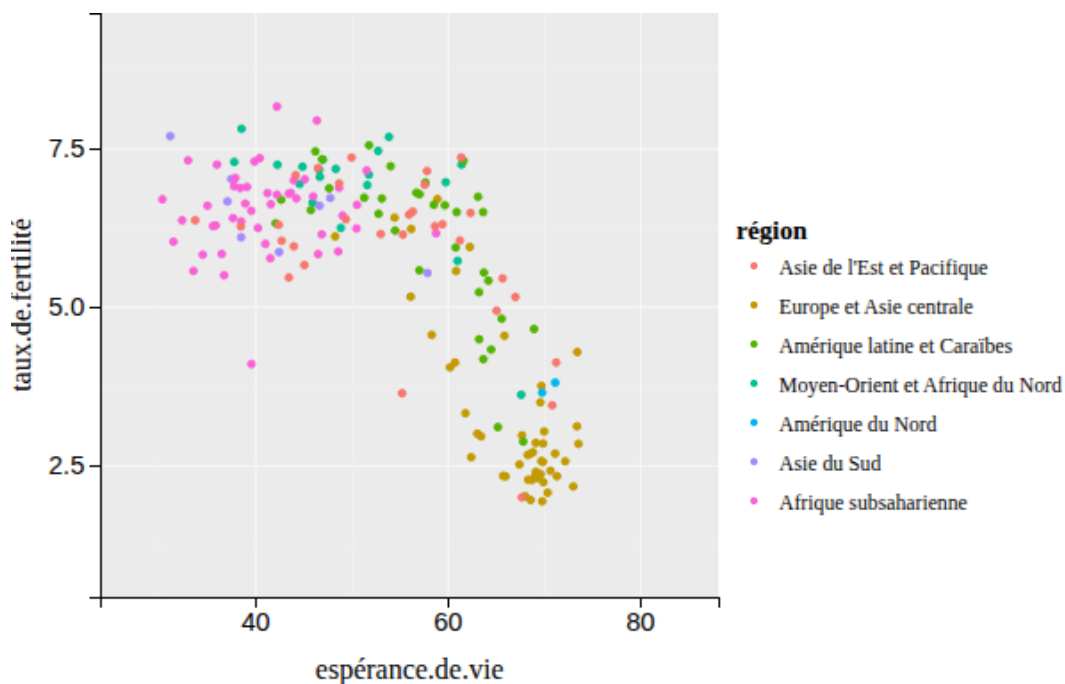
	iso2c	country	year	fertility.rate	life.expectancy	population
5850	KV	Kosovo	1979	NA	NA	1491000
5851	KV	Kosovo	1980	NA	NA	1521000
5852	KV	Kosovo	1981	4.5758	65.93268	1552000
	GDP.per.capita.Current.USD	15.to.25.yr.female.literacy	iso3c			
5850	NA	NA	KSV			
5851	NA	NA	KSV			
5852	NA	NA	KSV			
	region	capital	longitude	latitude		
5850	Europe & Central Asia (all income levels)	Pristina	20.926	42.565		
5851	Europe & Central Asia (all income levels)	Pristina	20.926	42.565		
5852	Europe & Central Asia (all income levels)	Pristina	20.926	42.565		
	income	lending	Region	région		
5850	Lower middle income	IDA	Europe & Central Asia	Europe et Asie centrale		
5851	Lower middle income	IDA	Europe & Central Asia	Europe et Asie centrale		
5852	Lower middle income	IDA	Europe & Central Asia	Europe et Asie centrale		
	espérance.de.vie	taux.de.fertilité	année	pays	PIB.par.habitant.USD	
5850	NA	NA	1979	Kosovo	NA	
5851	NA	NA	1980	Kosovo	NA	
5852	65.93268	4.5758	1981	Kosovo	NA	
	alphabétisation	revenu				
5850	NA	Revenu moyen bas				
5851	NA	Revenu moyen bas				
5852	NA	Revenu moyen bas				

On voit bien que les données sur le taux de fertilité et sur l'espérance de vie sont absentes du tableau pour le Kosovo en 1979 et en 1980. Il ne serait donc pas logique d'effectuer une transition fluide pour ce pays, puisque les données ne sont présentes ni avant ni après la transition. Comment le spécifier dans la visualisation de données? Dans le code ci-dessous, nous utilisons `aes(key=pays)` pour spécifier que la variable `pays` doit être utilisée pour faire correspondre les points de données avant et après la modification de la sélection.

```
nuage.key <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
  showSelected="année",
  data=BanqueMondiale)
```


Le `aes(key)` dans le ggplot ci-dessus n'a de sens que pour la visualisation interactive des données, il est donc ignoré lorsqu'il est affiché avec les périphériques graphiques R habituels. Cependant, si nous affichons ce ggplot en utilisant `animint2`, la variable `pays` permettra des durées de transitions pertinentes. Pour spécifier une durée de transition par défaut pour la variable `année` nous utilisons l'option `duration` dans l'image de données ci-dessous.

```
(viz.duration <- animint(nuage.key, duration=list(année=2000)))
```



L'option `duration` doit être une liste nommée. Chaque nom doit être une variable de sélection et chaque valeur doit spécifier le nombre de millisecondes à utiliser pour la durée de la transition lorsque la valeur sélectionnée de cette variable est modifiée.

Si vous cliquez sur “Show animation controls” (Afficher les contrôles d’animation) dans le graphique ci-dessus, vous verrez que la zone de texte pour la variable `année` est 2000, comme spécifié dans le code R. Si vous changez la sélection de 1980 à 1981, vous devriez voir une transition correcte.

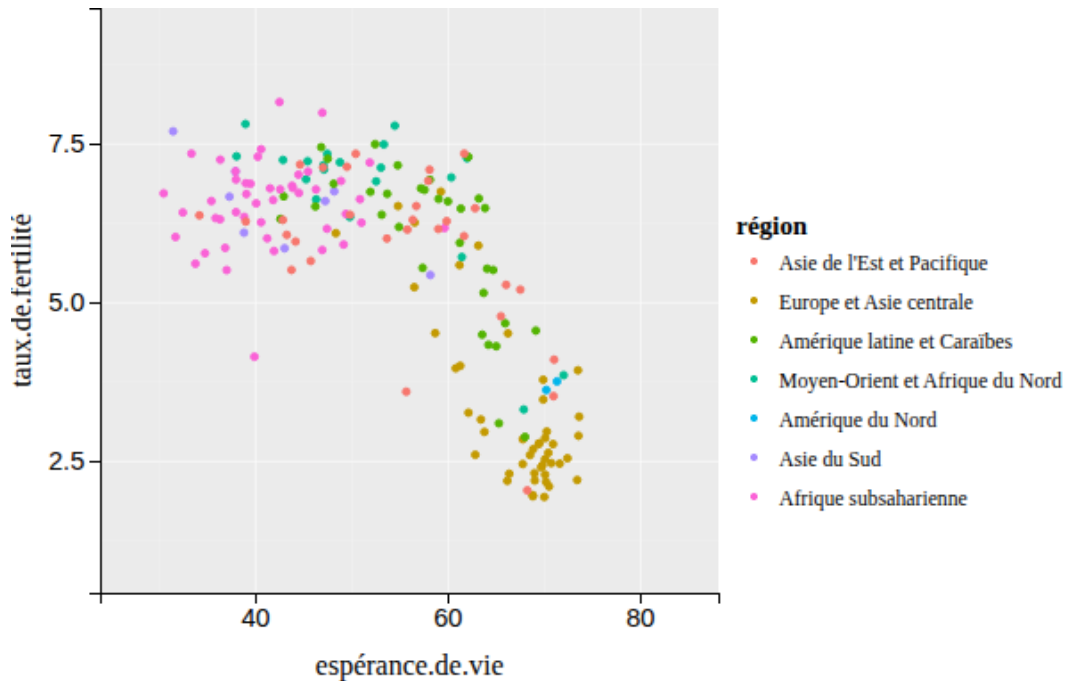
De façon générale, `aes(key)` doit être spécifié pour tous les geoms qui utilisent le mot-clé `showSelected` avec une variable qui apparaît dans l'option `duration`. Dans cet exemple, nous avons utilisé l'option `duration` pour spécifier une transition fluide pour la variable `année`. Puisque nous utilisons `showSelected=année` dans `geom_point` nous avons également spécifié `aes(key)` pour ce geom.

3.4 Animation : l'option `time`

L'option `time` permet de spécifier une variable à utiliser pour l'animation. Dans le code ci-dessous, on utilise l'option `time` pour préciser `année` comme variable d'animation, et

mettre l'animation à jour toutes les 2000 millisecondes.

```
viz.duration.time <- viz.duration
viz.duration.time$time <- list(variable="année", ms=2000)
viz.duration.time
```



On dit que la visualisation ci-dessus est animée, parce que la sélection pour `année` va changer toutes les deux secondes.

Exercice : réalisez une visualisation de données animée qui n'utilise PAS de transitions fluides. Indice : créez une liste de ggplots qui intègrent l'option `time` mais pas l'option `duration`.

3.5 Résumé du chapitre et exercices

Dans ce chapitre, nous avons expliqué l'utilisation du mot-clé `showSelected`, des menus de sélection, des transitions et de l'animation.

Exercices :

- Réalisez une version améliorée de `vis.alignée` du chapitre précédent. Au lieu de fixer l'année à 1975, utilisez `showSelected=année` pour que l'utilisateur puisse sélectionner une année. Ajoutez des geoms qui affichent l'année sélectionnée : un `geom_text` sur le nuage de points et un `geom_vline` sur la série chronologique.
- Traduisez l'un des [exemples de library\(animation\)](#) en `animint2`. Indice : dans le code de `library(animation)`, il y a toujours une boucle `for` dans l'option `time`. Au lieu d'appeler une fonction d'affichage à l'intérieur de la boucle `for`, utilisez l'idiome liste de tableaux de données pour stocker les données qui doivent être tracées. Utilisez ensuite

ces données avec `showSelected` pour créer des ggplots, et affichez-les avec `animint2`.

Dans le [chapitre 4](#), nous vous expliquerons le mot-clé `clickSelects` qui indique un geom sur lequel on peut cliquer pour mettre à jour une variable de sélection.



4

Le mot-clé `clickSelects`

Traduction de l'[anglais Ch04-clickSelects](#)

Dans ce chapitre, nous expliquons `clickSelects`, l'un des deux principaux mots-clés qu'`animint2` propose pour la visualisation interactive de données. Le mot-clé `clickSelects` précise quelle variable de sélection sera mise à jour quand on clique sur le geom correspondant. Chaque geom d'une visualisation de données possède son propre ensemble de données et sa propre définition du mot-clé `clickSelects`. Ainsi, en cliquant sur différents geoms, on peut modifier différentes variables de sélection.

Après avoir lu ce chapitre, vous serez en mesure de

- Comprendre comment les légendes interactives utilisent implicitement les `clickSelects`.
- Utiliser le mot-clé `clickSelects` dans vos esquisses de graphiques.
- Traduire vos esquisses de graphiques avec `clickSelects` en code R.
- Utiliser l'option `selector.types` pour spécifier plusieurs variables de sélection.

4.1 Les légendes interactives utilisent implicitement `clickSelects`

Dans cette section, nous expliquerons comment le mot-clé `clickSelects` est implicitement utilisé dans les légendes interactives. Si vous avez lu les chapitres précédents, vous avez déjà utilisé implicitement `clickSelects`, qui a été créé automatiquement pour les légendes interactives dans les chapitres précédents. Considérons l'esquisse de la visualisation de données de la Banque mondiale présentée au dernier chapitre.

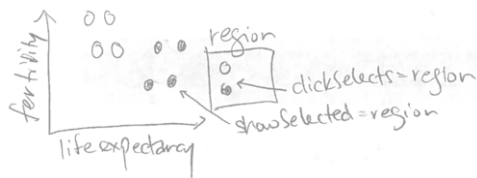


Figure 4.1: légende `clickSelects`

Puisque la légende a `clickSelects=région`, cliquer sur une entrée de cette légende met à jour la variable de sélection `région`. Notez que `animint2` rend automatiquement chaque légende discrète interactive, vous n'avez donc pas besoin de spécifier explicitement `clickSelects=région` pour la légende. En fait, lorsque nous avons spécifié `color=région` pour le geom_point `animint2` fait deux choses automatiquement :

- `showSelected=région` est aussi affecté au même `geom_point`.

- `clickSelects=région` est attribué à la légende des couleurs.

Notez que les mots-clés `clickSelects` ne sont pas limités aux légendes interactives. Chaque geom a sa propre définition de `clickSelects` qui détermine la variable de sélection qui doit être mise à jour lorsqu'on clique sur ce geom. Dans la section suivante, nous donnerons plusieurs exemples sur la manière dont `clickSelects` peut être utilisé en combinaison avec `showSelected` pour créer des visualisations de données interactives.

4.2 Utiliser `clickSelects` pour identifier des points sur un nuage de points

L'objectif de cette section est de créer la visualisation suivante des données de la Banque mondiale.

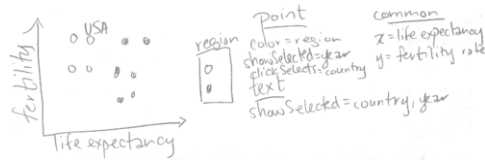
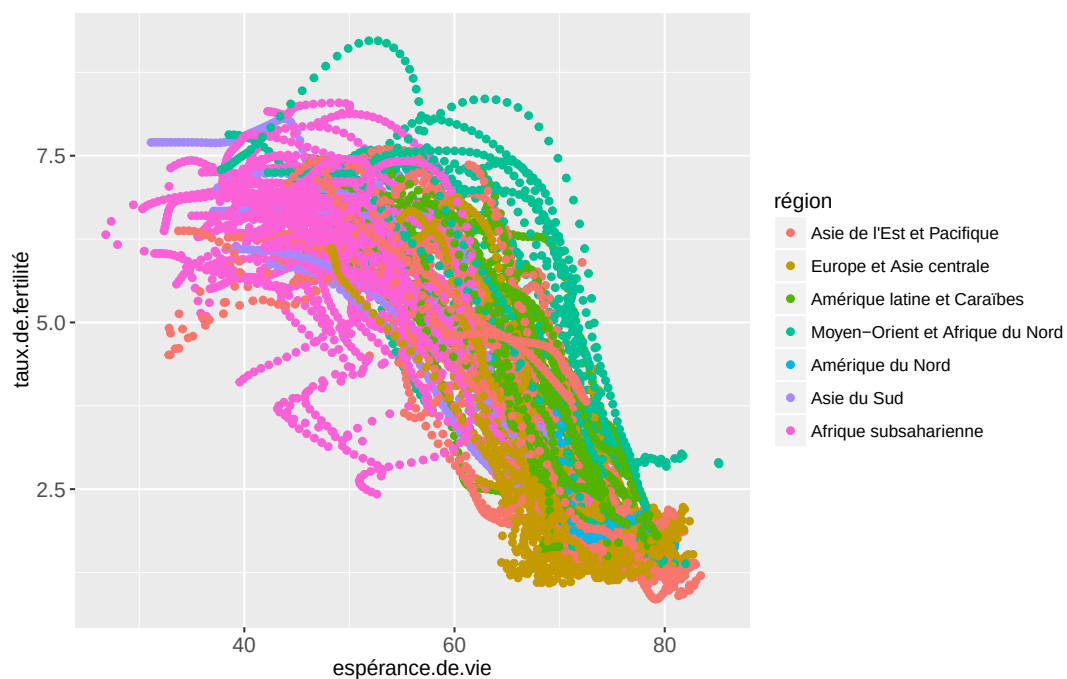


Figure 4.2: Texte de la visualisation de la Banque mondiale

Pour commencer, considérons le code R suivant qui génère un nuage de points des données de la Banque mondiale :

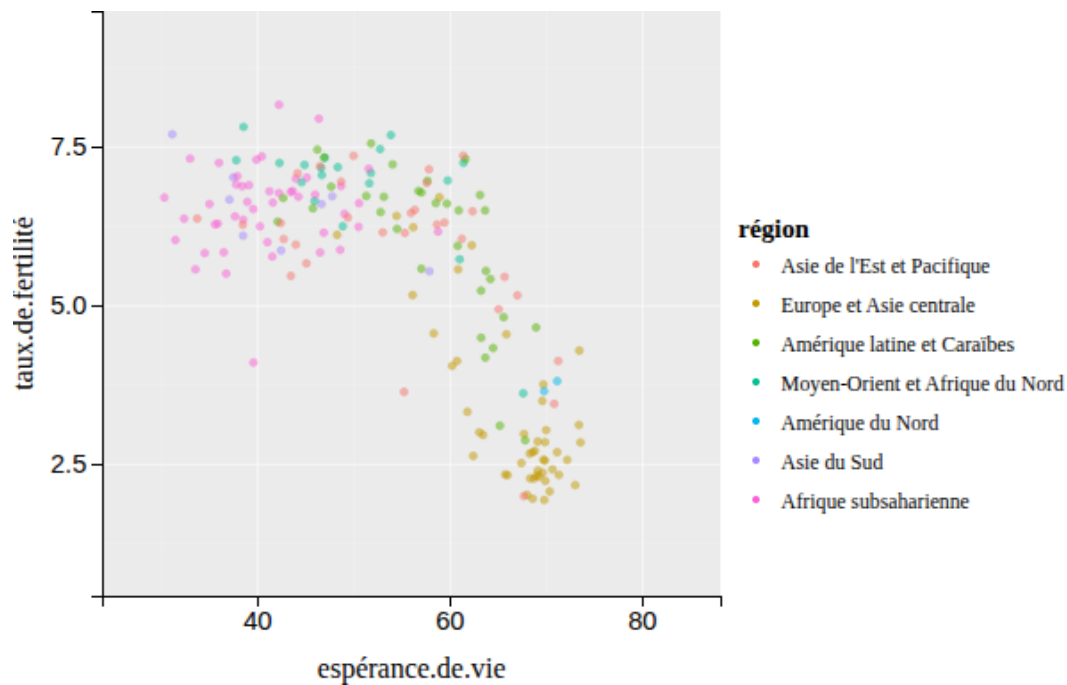
```
library(animint2)
data(BanqueMondiale, package="animint2fr")
gg_nuage <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
  showSelected="année",
  clickSelects="pays",
  data=BanqueMondiale)
gg_nuage
```

Warning: Removed 1490 rows containing missing values (geom_point).



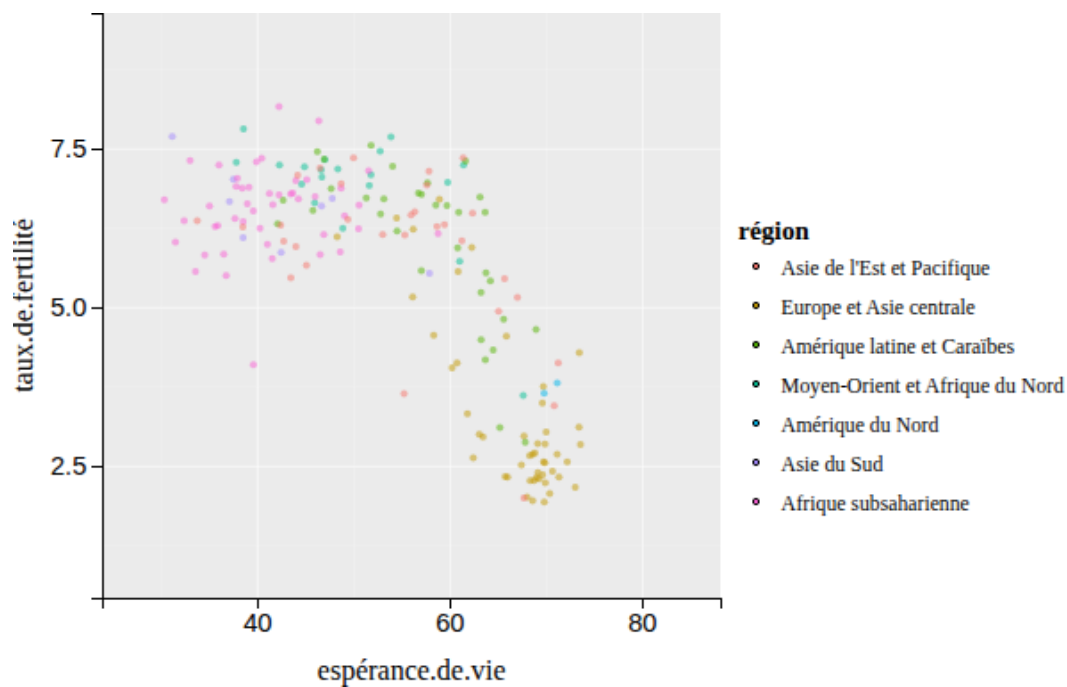
Notez que le graphique ci-dessus n'est pas interactif, car il est affiché à l'aide du périphérique graphique R traditionnel. En revanche, afficher le même ggplot en utilisant `animint2` donne le graphique interactif suivant :

```
(vis.nuage <- animint(  
  nuage=gg_nuage,  
  duration=list(année=2000)))
```



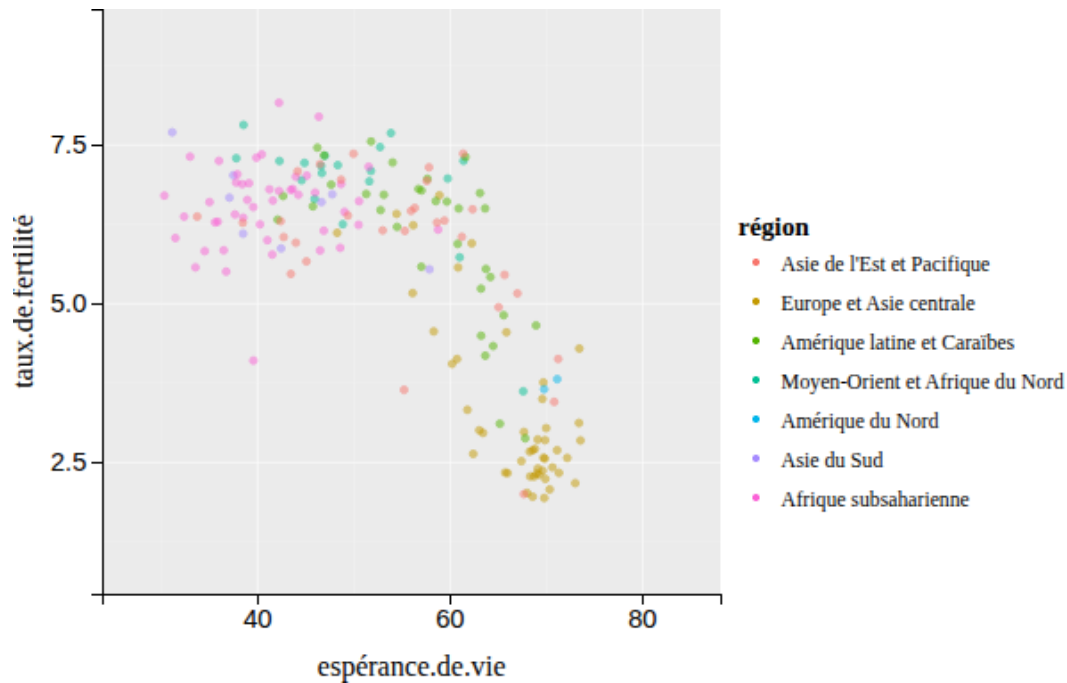
Essayez de cliquer sur des points de données dans le nuage de points ci-dessus. Vous devriez voir la valeur de la variable `pays` changer après avoir cliqué sur un point de données. Vous devriez également voir que le point de données pour le pays sélectionné est plus foncé que les autres. Cela permet de mettre en évidence la sélection en cours, cette surbrillance est générée automatiquement pour chaque geom avec `clickSelects`. Par défaut, le point sélectionné a `alpha=1` (totalement opaque, pas de transparence), et les autres points ont `alpha=0.5` (moitié semi-transparent). Ces valeurs par défaut peuvent être personnalisées ; par exemple, dans le code ci-dessous, un contour noir est utilisé pour mettre en évidence la sélection actuelle.

```
animint(
  ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, fill=région,
      key=pays),
      shape=21,
      color="black",
      color_off=NA,
      showSelected="année",
      clickSelects="pays",
      data=BanqueMondiale))
```

La visualisation de données ci-dessus affiche le nom du pays sélectionné dans le menu de sélection, mais il serait préférable de l'afficher sous forme d'une étiquette de texte sur le nuage de points. Nous pouvons le faire en ajoutant une couche `geom_text` avec deux variables `showSelected` :

```
vis.text <- vis.nuage
vis.text$nuage <- gg_nuage+
  geom_text(aes(
    x=espérance.de.vie, y=taux.de.fertilité, label=pays,
    key=pays),
    showSelected=c("année", "pays"),
    data=BanqueMondiale)
vis.text
```



Après avoir cliqué sur un point de données dans le nuage de points ci-dessus, vous devriez voir apparaître une étiquette de texte avec le nom du pays. Essayez également de changer l'année à l'aide du menu de sélection. Vous devriez voir l'étiquette de texte se déplacer dans une transition fluide en même temps que le point de données correspondant.

La visualisation de données ci-dessus contient plus d'un geom, chacun avec des caractéristiques interactives différentes. Essayez de cliquer sur le bouton "Démarrer la visite" au bas de la visualisation de données, qui affichera les fonctions interactives disponibles pour le premier geom de la visualisation de données. En cliquant sur "Next", vous obtiendrez des informations sur le prochain geom, et en cliquant sur "Done" ou sur l'arrière-plan gris, vous mettrez fin à la visite guidée. La fonction "Start Tour" peut être utile aux nouveaux utilisateurs de votre visualisation de données pour découvrir les fonctions interactives présentes dans chaque geom. Les informations affichées pendant la visite peuvent être personnalisées, en définissant les paramètres `help` et `title` de chaque geom.

Comme nous l'avons expliqué dans le dernier chapitre, toute variable spécifiée à l'aide du mot-clé `showSelected` d'un geom est traitée comme une variable interactive. Dans l'exemple ci-dessus, nous avons spécifié deux variables `showSelected` pour le `geom_text`. Cela signifie que l'on ne dessine qu'une étiquette de texte pour les lignes de la base de données `BanqueMondiale` qui correspondent aux valeurs actuelles des deux variables de sélection. Étant donné que chaque combinaison de `pays` et `année` a une ligne dans ces données, une seule étiquette de texte sera affichée à la fois.

Essayez de cliquer sur l'entrée de la légende qui correspond à la région du pays actuellement sélectionné (par exemple, si le Canada est sélectionné, essayez de cliquer sur l'entrée de la légende Amérique du Nord). Vous devriez voir le point disparaître, mais le texte rester affiché.

Exercice : comment faire disparaître le texte en même temps que le point ? Indice : il faut ajouter un mot-clé au `geom_text`.

Dans le dernier chapitre, nous avons introduit les termes “manipulation directe” et “manipulation indirecte” pour décrire les interactions avec les légendes et les menus. Dans la visualisation de données ci-dessus, nous pouvons modifier la valeur de la variable de sélection **pays** en cliquant sur un point de données (manipulation directe) ou en utilisant le menu de sélection (manipulation indirecte). Les deux techniques sont utiles, mais à des fins différentes :

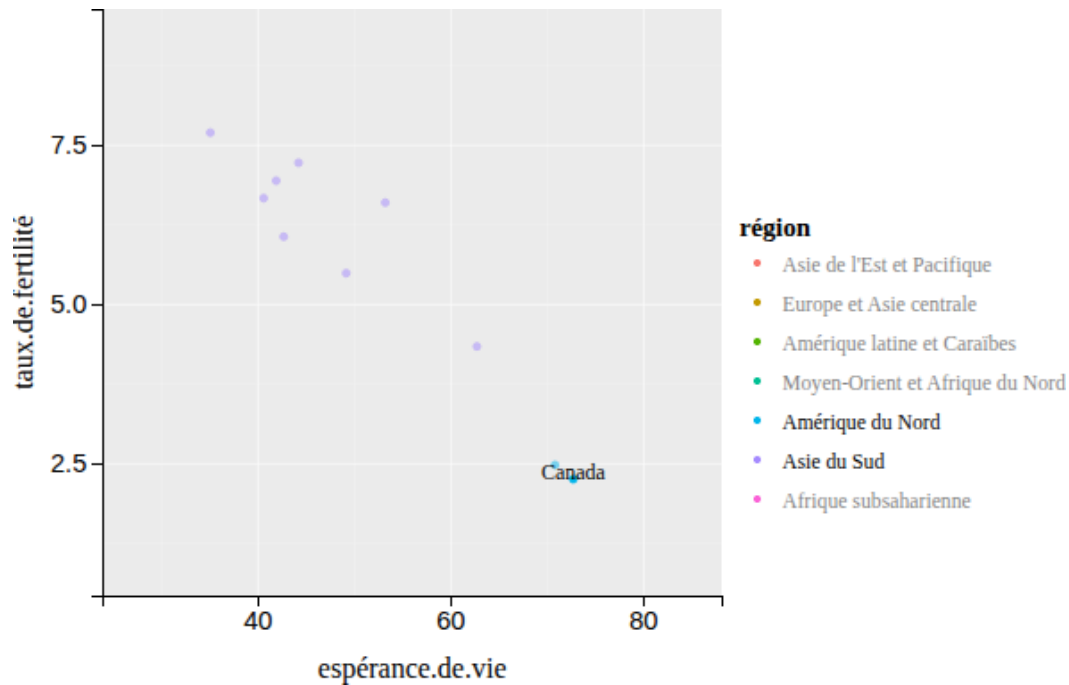
- La manipulation directe, soit de cliquer sur les points de données, est utile pour trouver le nom des pays ayant des valeurs extrêmes de taux de fertilité et d'espérance de vie. Par exemple, pour l'année 1960, en cliquant sur le point en bas à gauche du graphique, on découvre le nom du pays, le Gabon.
- La manipulation indirecte à l'aide des menus est utile pour voir la position tracée d'un pays d'intérêt. Par exemple, il serait difficile de trouver la France en cliquant sur tous les différents points, mais il est simple de trouver la France en tapant son nom dans le menu de sélection.

Notez que lorsque la visualisation ci-dessus est affichée pour la première fois, le pays sélectionné est Andorre et l'année sélectionnée est 1960. Comme les données pour Andorre sont manquantes en 1960, aucune étiquette de texte n'est affichée au départ. Pour modifier la première sélection, vous pouvez spécifier l'option **first** comme expliqué dans la section suivante.

4.3 L'option **first**

Pour spécifier la sélection qui doit être affichée lorsque la visualisation de données est exécutée pour la première fois, utilisez l'option **first**. Il doit s'agir d'une liste nommée avec des entrées pour chaque variable de sélection. Par exemple, le code ci-dessous spécifie 1970 comme première année, le Canada comme premier pays, et l'Amérique du Nord et l'Asie du Sud comme premières régions.

```
vis.first <- vis.text
vis.first$first <- list(
  année=1970,
  pays="Canada",
  région=c("Amérique du Nord", "Asie du Sud"))
vis.first
```



Notez que dans la visualisation des données ci-dessus, il n'y a qu'un seul pays sélectionné à la fois. Dans la section suivante, nous expliquerons comment l'option `selector.types` peut être utilisée pour modifier la variable de sélection `pays` en une variable de sélection multiple.

4.4 L'option `selector.types`

Dans cette section, notre objectif est de produire une version légèrement plus complexe du nuage de points de la section précédente. L'esquisse ci-dessous ne présente qu'une seule différence par rapport à l'esquisse de la dernière section : les étiquettes de texte sont affichées pour plus d'un pays.

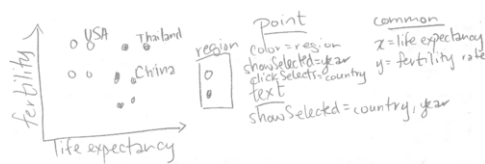


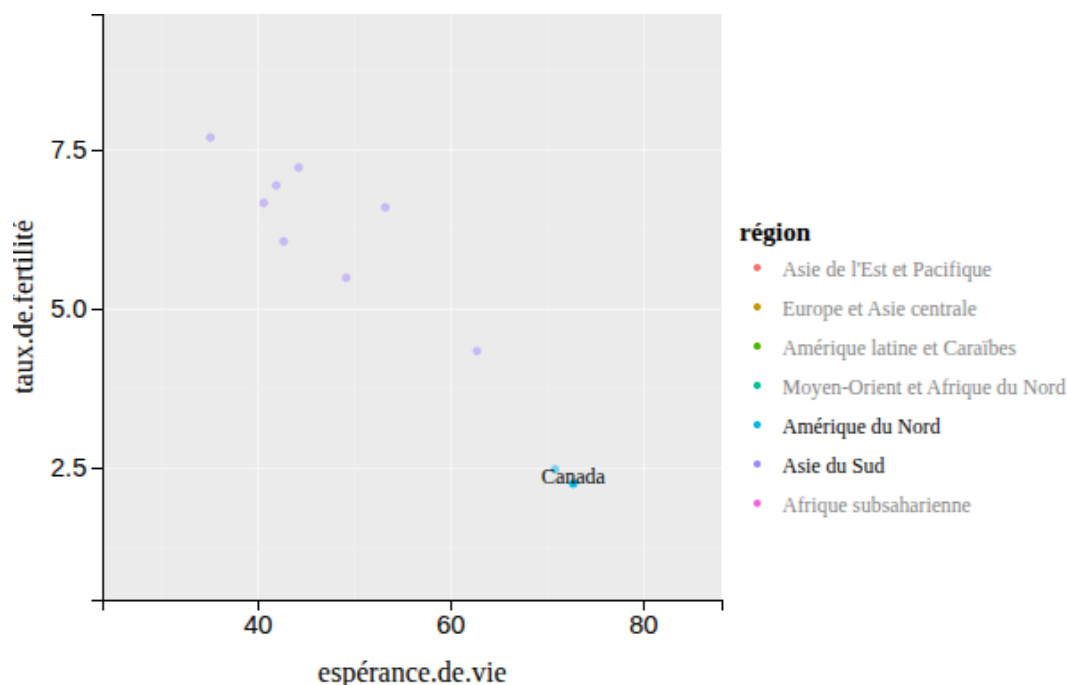
Figure 4.3: Banque mondiale vis texte

Dans `animint2`, chaque variable de sélection a un type, soit simple, soit multiple. La sélection unique signifie qu'une seule valeur peut être sélectionnée à la fois. La sélection multiple signifie qu'un nombre quelconque de valeurs peut être sélectionné à la fois. Dans les graphiques de la section précédente, la sélection multiple a été utilisée pour la variable `région` mais pas pour les variables `année` et `pays`. Comment cela se fait-il ?

Par défaut, `animint2` attribue une sélection multiple à toutes les variables qui apparaissent

dans les légendes discrètes interactives, et une sélection unique aux autres variables. Cependant, une sélection unique ("`single`") ou multiple ("`multiple`") peut être précisée en utilisant l'option `selector.types`. Dans le code R ci-dessous, nous utilisons l'option `selector.types` pour spécifier que la variable `pays` doit être traitée comme une variable de sélection multiple.

```
vis.multiple <- vis.first  
vis.multiple$selector.types <- list(pays="multiple")  
vis.multiple
```



Lorsque la visualisation de données ci-dessus est affichée pour la première fois, elle montre les points de données de l'année 1970 pour chaque pays d'Amérique du Nord et d'Asie du Sud. Elle affiche également une étiquette de texte pour le Canada.

Vous avez peut-être remarqué qu'il est facile d'ajouter des pays à la sélection actuelle en cliquant sur des points de données. Normalement, le fait de cliquer sur un point de données sélectionné supprime ce pays de la sélection actuelle. Toutefois, dans cette visualisation de données en particulier, il n'est pas si facile de les supprimer, car les étiquettes de texte sont affichées au-dessus des points de données.

Exercice : Recréez la visualisation des données ci-dessus de manière à ce qu'un clic sur une étiquette de texte supprime le pays en question de l'ensemble de sélection. Indice : vous devez ajouter un mot-clé `clickSelects` au `geom_text`.

Notez que dans la visualisation de données ci-dessus, la variable `année` ne peut être modifiée que par le menu de sélection.

Dans la section suivante, nous ajouterons une facette avec un `geom` directement cliquable pour changer la variable `année`.

4.5 Sélection d'une année sur un graphique de série temporelle

L'objectif de cette section est d'ajouter un graphique de série temporelle sur lequel on peut cliquer pour changer l'année sélectionnée.

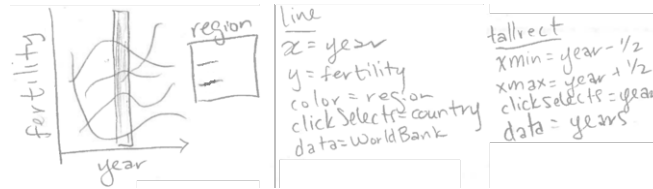


Figure 4.4: Série temporelle de la Banque mondiale

Notez que l'esquisse ci-dessus comprend `geom_tallrect`, un nouveau geom introduit dans `animint2`. Il est qualifié de "grand" car il occupe tout l'espace vertical du graphique, il ne nécessite donc que la définition de ses limites horizontales dans `aes()` avec `xmin` et `xmax`. En spécifiant `clickSelects=année` nous indiquons que nous voulons tracer un `tallrect` pour chaque année, et que cliquer sur un `tallrect` changera l'année sélectionnée. Nous devons donc créer un nouvel ensemble de données appelé `années` avec une ligne pour chaque année des données `BanqueMondiale`.

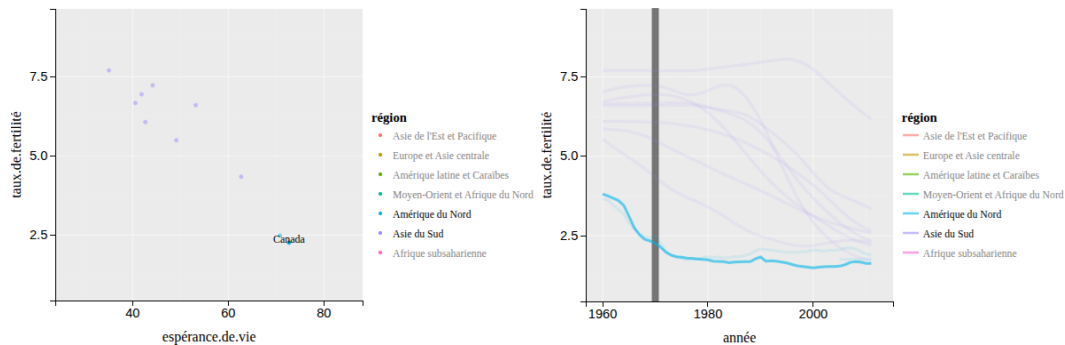
```
années <- data.frame(année=unique(BanqueMondiale$année))
head(années)
```

```
année
1  1960
2  1961
3  1962
4  1963
5  1964
6  1965
```

Ensuite, nous ajoutons la série temporelle à la visualisation existante.

```
vis.timeSeries <- vis.multiple
vis.timeSeries$timeSeries <- ggplot()+
  geom_tallrect(aes(
    xmin=année-0.5, xmax=année+0.5),
    clickSelects="année",
    alpha=0.5,
    data=années)+
  geom_line(aes(
    x=année, y=taux.de.fertilité, group=pays, color=région),
    clickSelects="pays",
    size=3,
    alpha=0.6,
    data=BanqueMondiale)
```

```
vis.timeSeries
```



Essayez de cliquer sur l'arrière-plan de la série temporelle dans la visualisation de données ci-dessus. Vous devriez voir les points de données et les étiquettes de texte se déplacer dans une transition graduelle vers une position correspondant à l'année nouvellement sélectionnée.

Exercice : animez la visualisation de données en spécifiant l'option `time`, comme expliqué au chapitre 3.

Exercices avec multicouches : ajoutez un `geom_text` qui indique l'année en cours sur le nuage de points. Ajoutez un `geom_path` qui présente les données des 5 dernières années. Ajoutez un `geom_label_aligned` pour les pays sélectionnés dans la série temporelle.

4.6 Sélection d'une année sur une facette de série temporelle

L'objectif de cette section est d'ajouter une facette avec un graphique de série temporelle sur lequel on peut cliquer pour changer l'année sélectionnée.

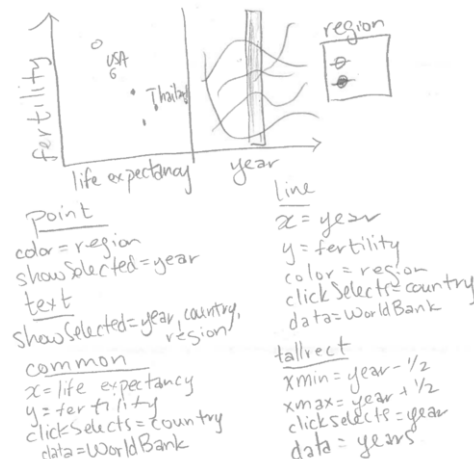


Figure 4.5: Facette de dispersion de la Banque mondiale

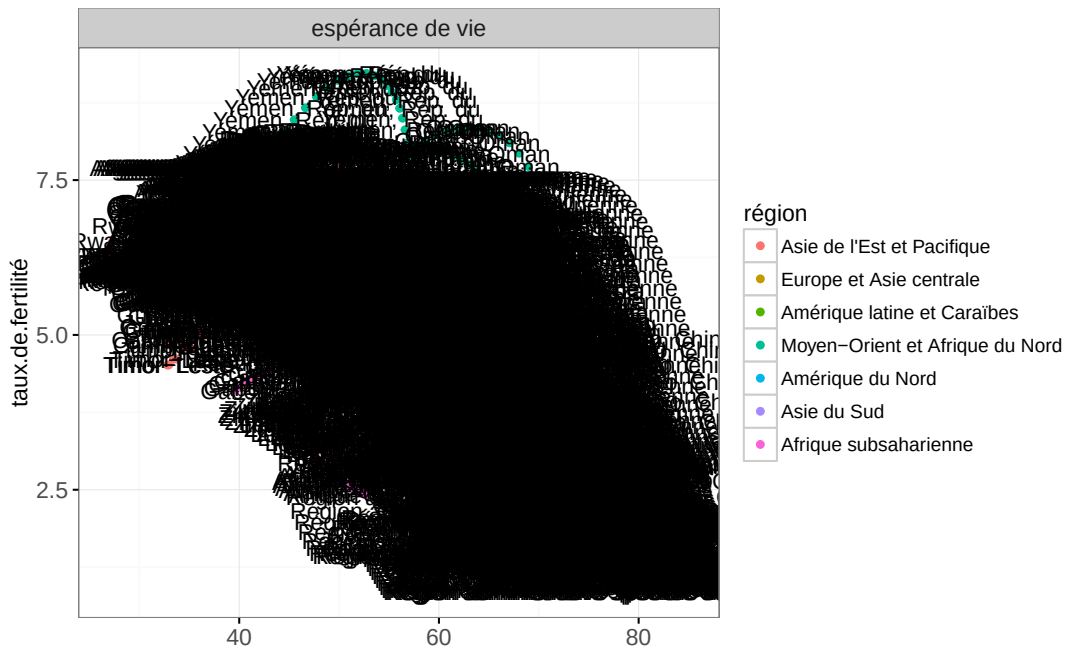
Tout d'abord, nous recréons le nuage de points de la section précédente à l'aide de la méthode

Ajouter une variable ensuite des facettes qui est utile pour créer des ggplots avec des axes alignés.

```
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("espérance de vie", "année")))
}
nuageFacet <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
    showSelected="année",
    clickSelects="pays",
    data=add.x.var(BanqueMondiale, "espérance de vie"))+
  geom_text(aes(
    x=espérance.de.vie, y=taux.de.fertilité, label=pays,
    key=pays),
    clickSelects="pays",
    showSelected=c("année", "pays", "région"),
    data=add.x.var(BanqueMondiale, "espérance de vie"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))
nuageFacet
```

Warning: Removed 1490 rows containing missing values (geom_point).

Warning: Removed 1490 rows containing missing values (geom_text).



Notez que le ggplot ci-dessus utilise les mêmes définitions `aes` que le nuage de points de la section précédente. La seule différence est que nous avons utilisé un jeu de données

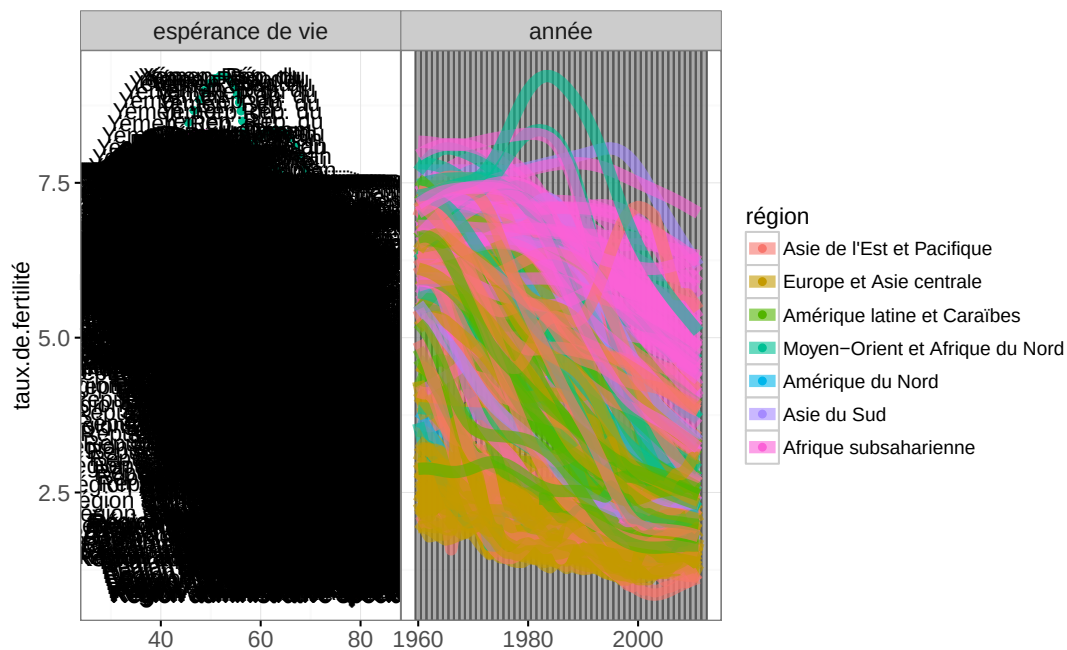
BanqueMondiale enrichi avec une variable `x.var` supplémentaire que nous utilisons avec `facet_grid`. Ci-dessous, nous ajoutons des geoms pour un graphique de série temporelle qui est aligné sur l'axe du taux de fertilité.

```
nuage_serie_temporelle <- nuageFacet+
  geom_tallrect(aes(
    xmin=année-0.5, xmax=année+0.5),
    clickSelects="année",
    alpha=0.5,
    data=add.x.var(années, "année"))+
  geom_line(aes(
    x=année, y=taux.de.fertilité, group=pays, color=région),
    clickSelects="pays",
    size=3,
    alpha=0.6,
    data=add.x.var(BanqueMondiale, "année"))
nuage_serie_temporelle
```

Warning: Removed 1490 rows containing missing values (geom_point).

Warning: Removed 1490 rows containing missing values (geom_text).

Warning: Removed 759 rows containing missing values (geom_path).



Les deux geom définis ci-dessus occupent un nouveau `facet` pour la valeur d'`année` de la variable `x.var` (définie par la fonction `add.x.var`). Puisque ces deux geoms ont des définitions différentes de `clickSelects`, un clic sur chaque geom mettra à jour le graphique d'une manière différente. Notez que pour le `geom_line` nous spécifions `size=3` ce qui signifie une largeur de trait de 3 pixels. En général, il est judicieux d'augmenter la taille des geoms avec `clickSelects`, afin de les rendre plus faciles à cliquer.

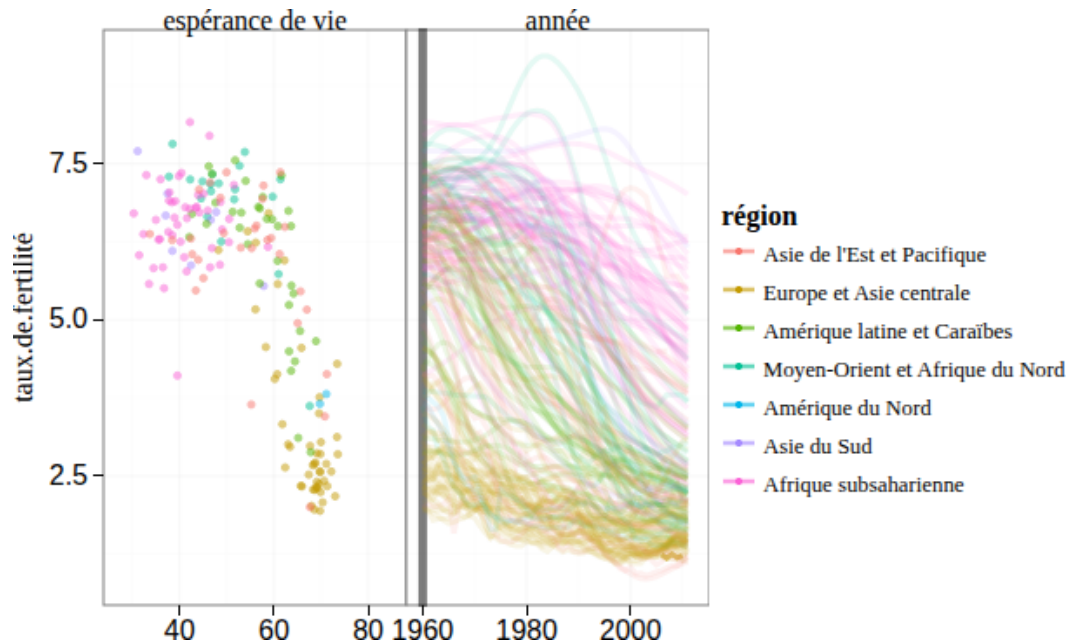
Notez également que nous avons spécifié `alpha=0.5` pour le `geom_tallrect` et `alpha=0.6`

pour le `geom_line`. Puisque ces deux geoms définissent `clickSelects`, certaines lignes et rectangles seront sélectionnées, et d'autres ne le seront pas. Les valeurs `alpha` précisent l'opacité des objets sélectionnés, et les autres objets auront une valeur d'opacité de 0,5 inférieure à cette valeur. Dans l'exemple ci-dessus, les lignes non sélectionnées auront `alpha=0.1` et les `tallrects` non sélectionnés auront `alpha=0` (complètement transparent).

Depuis septembre 2023, il est également possible de spécifier l'opacité, le remplissage et la couleur des objets qui ne sont pas actuellement sélectionnés (`alpha_off`, `fill_off`, `color_off`). Les utilisateurs peuvent spécifier ces paramètres dans le geom (pas dans le `aes`) afin de créer librement une apparence différente pour les éléments sélectionnés et non sélectionnés, au lieu d'être contraints de dépendre du comportement décrit ci-dessus. Pour plus d'informations, voir la discussion et l'exemple dans chapitre 6, section Préciser le mode d'affichage de l'état de la sélection .

Enfin, nous utilisons le code R ci-dessous pour afficher le nouveau nuage de points aligné avec la série temporelle.

```
(vis.facets <- animint(nuage_série_temporelle))
```



La visualisation de données interactives ci-dessus contient un nouveau panneau avec des lignes qui montrent une série temporelle de taux de fertilité sur toutes les années. Puisque nous avons précisé `clickSelects=pays` pour le `geom_line`, un clic sur une ligne met à jour l'ensemble des pays sélectionnés. Puisque nous avons spécifié `clickSelects=année` pour le `geom_tallrect`, un clic sur un `tallrect` met à jour l'année sélectionnée.

Exercice : ajoutez les options `time`, `duration`, `first` et `selector.types` à la visualisation de données ci-dessus.

4.7 Résumé du chapitre et exercices

Dans ce chapitre, nous avons vu l'utilisation de `clickSelects`, l'un des deux principaux mots-clés introduits par `animint2` pour la conception de visualisations de données interactives. Nous avons utilisé l'ensemble de données de la Banque mondiale pour montrer comment `clickSelects` peut être utilisé pour définir différentes interactions pour chacun des geoms tracés. Nous avons expliqué comment l'option `first` peut être utilisée pour préciser les valeurs à utiliser lorsque le `animint2` est affiché pour la première fois. Nous avons également expliqué comment l'option `selector.types` peut être utilisée pour préciser le type de sélection (unique ou multiple).

Exercices :

- Jusqu'à présent, nous avons vu trois façons différentes de modifier les variables de sélection : (1) utiliser des légendes interactives, (2) utiliser des menus de sélection et (3) cliquer sur les données à l'aide de `clickSelects`. Classez ces trois techniques de manipulation, de la plus directe, à la moins directe. Quelle technique privilégier et selon quelles circonstances ?
- Quand `geom_point(clickSelects=something, alpha=0.75)` est affiché avec le périphérique graphique R habituel, quel est le degré d'opacité/transparence pour tous les points de données ? Lorsque `animint2` affiche le même geom, certains points seront sélectionnés et d'autres non. Quelle est l'opacité/transparence des points sélectionnés ? Quelle est l'opacité/transparence des points non sélectionnés ?
- Ajouter `aes(size=population)` aux points du nuage de points de la Banque mondiale. La légende de la taille est-elle interactive ? Pourquoi ?
- Ajoutez un `geom_text` au nuage de points de la Banque mondiale qui montre l'année sélectionnée.
- Ajoutez un `geom_text` à la série chronologique de la Banque mondiale pour afficher le nom des pays sélectionnés.
- Ajoutez un `geom_path` au nuage de points de la Banque mondiale pour afficher les données des cinq dernières années.
- Utilisez l'option `time` pour créer une version animée de `vis.facets`.
- Utilisez `help` et `title` pour chaque geom dans `vis.facets` afin de créer un tour guidé qui fournit plus d'informations sur ce qui est affiché dans chaque geom.

Dans le [Chapitre 5](#) nous expliquons les différentes méthodes pour publier et partager des animints sur le web.



5

Partage

Traduction de l'[anglais Ch05-sharing](#)

Dans ce chapitre, nous expliquons plusieurs méthodes pour partager vos visualisations de données interactives sur le web. Après avoir lu ce chapitre, vous serez en mesure de visualiser des animints :

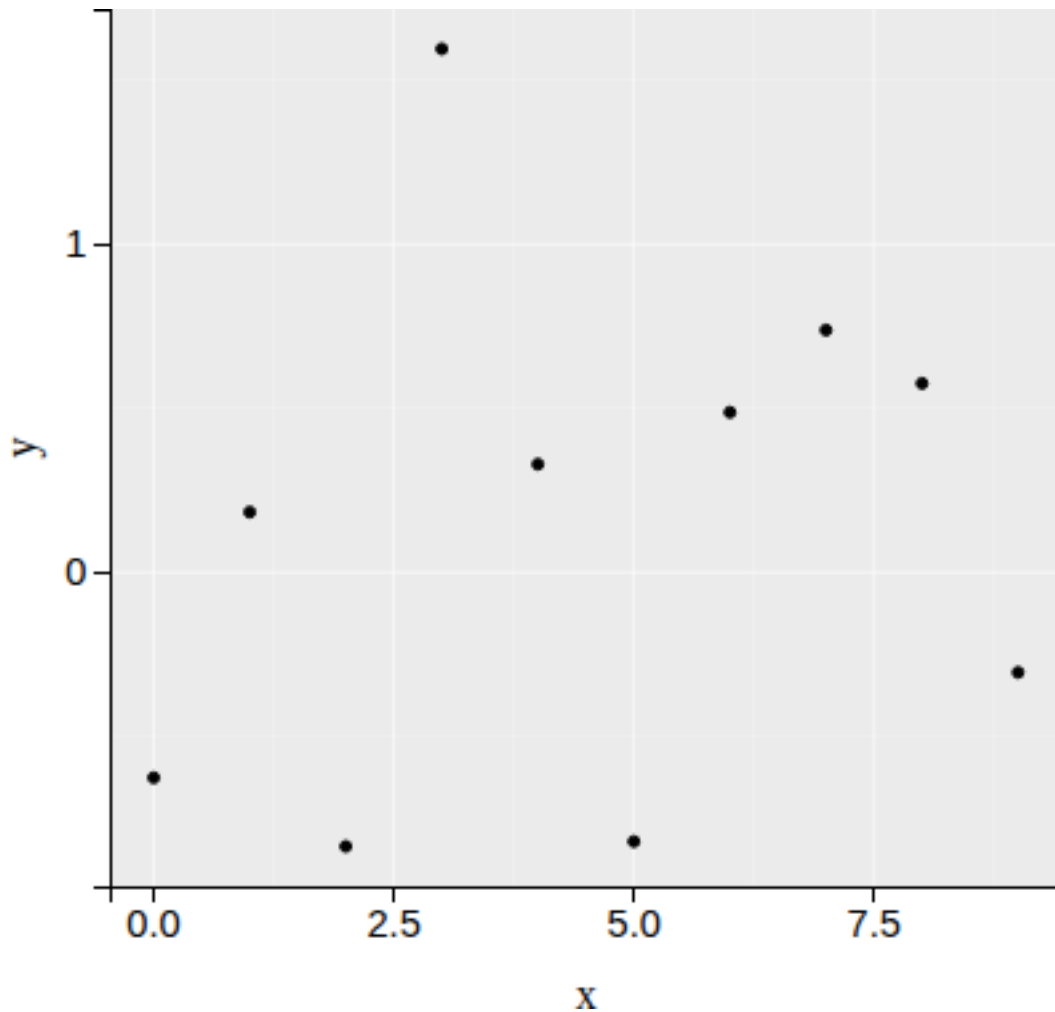
- à partir d'un dossier local sur votre ordinateur personnel;
- dans des documents [R Markdown](#);
- en utilisant n'importe quel serveur web, y compris [NetlifyDrop](#);
- publiés en utilisant [GitHub Pages](#) et organisés dans une galerie.

5.1 Compiler un animint dans un dossier local

Lorsque vous testez différentes visualisations de données interactives, il est utile de les prévisualiser sur votre ordinateur personnel avant de les publier sur le web. Cette section traite de deux méthodes pour compiler les animints dans un répertoire local.

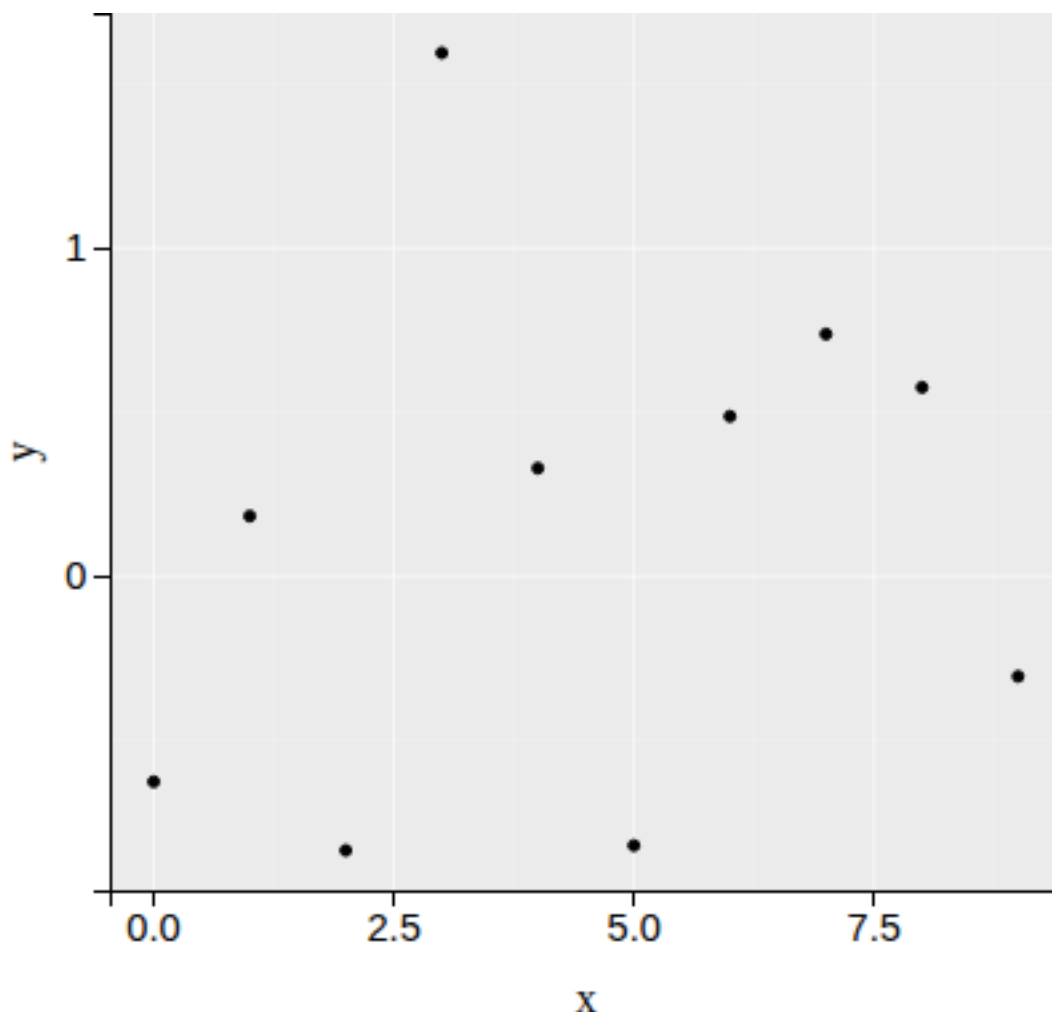
Dans les chapitres précédents, nous n'avons abordé qu'une seule méthode pour créer des visualisations de données interactives. Si `vis` est un animint (liste de ggplots et d'options avec la classe animint), alors `print(vis)` va compiler cet animint dans un dossier temporaire, en utilisant un code comme celui-ci,

```
set.seed(1)
dix.points <- data.frame(x=0:9, y=rnorm(10))
library(animint2)
animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=dix.points))
```



Le code ci-dessus sauvegarde l'animint dans un nouveau dossier temporaire. Plutôt que de sauvegarder chaque animint dans un dossier temporaire séparé, vous pouvez définir un dossier de sortie via l'argument `out.dir` de la commande `animint`. Si vous voulez sauvegarder l'animint dans le fichier "Ch05-partager-dix-points" utilisez :

```
animint(  
  point=ggplot()+  
    geom_point(aes(  
      x, y,  
      data=dix.points),  
    out.dir="Ch05-partager-dix-points")
```



Si le dossier parent d'`out.dir` n'existe pas, vous générerez une erreur (vous pouvez utiliser `dir.create` pour créer le parent si nécessaire). Si `out.dir` n'existe pas, il sera créé. Si `out.dir` existe (et contient un fichier nommé `animint.js`), tous les fichiers de ce répertoire seront écrasés. Pour afficher la visualisation, il suffit d'aller à `Ch05-partager-dix-points/index.html` dans un navigateur web (ce qui devrait être fait automatiquement / par défaut). Si la page web est vide, il se peut que vous deviez configurer votre navigateur pour autoriser l'exécution du code JavaScript local, [comme expliqué dans notre FAQ](#).

En interne, R appelle la méthode S3 `print.animint`, laquelle appelle `animint2dir` afin de compiler la visualisation dans un nouveau dossier temporaire sur votre ordinateur personnel. En général, nous déconseillons d'appeler `animint2dir` directement, mais cela peut être utile si vous voulez éviter d'ouvrir plusieurs fenêtres similaires dans le navigateur lorsque vous révisez un animint. Vous pouvez empêcher l'ouverture par défaut d'une fenêtre du navigateur avec :

```
vis <- animint(  
  point=ggplot()+  
  geom_point(aes(  
    x=1:10,  
    y=rnorm(10)  ))  
)
```

```
      x, y),
      data=dix.points))
animint2dir(
  vis,
  out.dir="Ch05-partager-dix-points",
  open.browser=FALSE)
```

5.2 Publier en R Markdown

Pour inclure un animint dans un document R Markdown, utilisez `animint(...)` à l'intérieur d'un bloc de code R. R exécutera la méthode S3 `knit_print.animint`, qui compile l'animint dans un dossier local, nommé en fonction du nom du bloc de code R. Par exemple, un bloc de code nommé `vis-facettes` sera sauvegardé dans le dossier `visfacettes`. Veillez à placer chaque animint dans son propre morceau de code (ne mettez pas deux animints dans le même morceau de code).

5.3 Publier sur un serveur web

Comme les animints ne sont que des dossiers contenant des fichiers HTML, TSV et JavaScript, vous pouvez les publier sur n'importe quel serveur web en copiant simplement le dossier sur ce serveur.

Par exemple, j'ai exécuté le code dans <https://github.com/animint/animint2/blob/master/inst/examples/WorldBank-facets.R> pour créer le `WorldBank-facets` dossier sur mon ordinateur personnel. J'ai copié ce dossier sur le serveur web de mon laboratoire à l'aide de la commande `rsync -r WorldBank-facets/ monsoon.hpc.nau.edu:genomic-ml/WorldBank-facets/` et je peux ainsi le visualiser sur <https://rcdata.nau.edu/genomic-ml/WorldBank-facets/>

Si vous n'avez pas accès à un serveur web personnel ou par le biais d'un laboratoire, vous pouvez utiliser l'une des méthodes décrites ci-dessous, elles sont gratuites et accessibles à tous.

5.4 Publier sur Netlify Drop

[Netlify Drop](#) est un service d'hébergement de sites web statiques. Pour y publier votre visualisation de données, il suffit de faire glisser un dossier sur cette page web (il peut s'agir d'un dossier issu de `animint2dir` ou de `rmarkdown::render` si votre animint est à l'intérieur de Rmd, comme décrit ci-dessus). Une fois le téléversement terminé, vous obtiendrez un lien qui vous permettra de visualiser les fichiers contenus dans ce dossier. Aucun enregistrement ou connexion n'est nécessaire, mais si vous n'avez pas de compte enregistré, vos données seront supprimées après une heure. Vous pouvez souscrire à un compte gratuit si vous

souhaitez que vos vis données soient disponibles pour une plus longue période.

5.5 Publier sur GitHub Pages

[GitHub Pages](#) est un service d'hébergement de sites web statiques et peut être utilisé pour publier des animints. Pour publier un animint sur GitHub Pages, vous avez besoin d'un compte GitHub et des packages `gert` (pour exécuter git à partir de R), `gh` (pour utiliser l'API GitHub à partir de R), et `gitcreds` (pour interagir avec le stockage des identifiants de git et faciliter l'authentification lors de l'envoi à GitHub). Tout d'abord, installez ces packages. Si vous n'avez pas de compte GitHub, vous pouvez [vous inscrire gratuitement](#). Ensuite, assurez-vous d'indiquer à R le nom et le courriel à utiliser pour la publication des commits git :

```
gert::git_config_global_set("user.name", "<votre_identifiant>")
gert::git_config_global_set("user.email", "<votre_courriel>")
```

Vous pouvez ensuite utiliser `animint2pages(vis, "nouveau_dépôt")` pour publier votre visualisation dans la branche `gh-pages` de `votre_identifiant_github/nouveau_dépôt` (à noter que `votre_identifiant_github` n'est pas écrit dans le code, puisque `gitcreds` obtiendra cette information à partir du stockage des identifiants de git). Un message devrait s'afficher vous indiquant l'URL ou le lien pour accéder à votre visualisation de données. Il faut compter quelques minutes (généralement pas plus de cinq) entre le moment où vous lancez `animint2pages` et celui où la visualisation est publiée sur GitHub Pages.

Si vous voulez mettre à jour un animint qui a déjà été publié sur GitHub Pages, vous pouvez simplement lancer `animint2pages(nouvelle_version, "dépôt_existant")` qui mettra à jour la branche `gh-pages` du dépôt spécifié.

Attention, GitHub Pages impose une limite de 100MB par fichier. Pour la plupart des visualisations, cette limite ne devrait pas poser de problème. Si votre visualisation de données contient un fichier TSV de plus de 100 Mo, vous pouvez utiliser l'option `chunk_vars` pour convertir ce grand fichier en plusieurs fichiers plus petits.

5.6 Organiser des animints dans une galerie

Une galerie est une collection de méta-données pour un ensemble de visualisations publiées sur GitHub Pages. Par exemple, la galerie principale des animints <https://animint.github.io/gallery/> (en anglais) est une page web contenant des liens vers divers animints, organisés en tableau de contingence. Nous avons aussi [une galerie en français](#).

- Il y a une ligne pour chaque animint.
- La première colonne `viz.link` montre une capture d'écran de l'animint, qui renvoie à la page web de l'animint.
- La deuxième colonne `title` indique le nom de la visualisation (extrait de l'option `title` de chaque animint).
- La troisième colonne `links` contient des liens vers son dépôt sur GitHub, ainsi que son

code source et éventuellement une vidéo.

Une galerie est un dépôt sur GitHub qui doit avoir deux fichiers sources dans la branche `gh-pages` :

- `repos.txt` (liste des dépôts sur GitHub qui contiennent des animints, un propriétaire/dépôt par ligne) et
- `index.Rmd` (source de la page web avec des liens vers animints).

Pour ajouter un nouvel animint à la galerie qui est publiée sur la branche `gh-pages` de `votre_identifiant_github/galerie` vous pouvez utiliser la méthode suivante :

- créez un nouveau animint dans le code R, et assurez-vous de définir les options `source` et `title`;
- utilisez `animint2pages(vis, "dépôt_de_vis")` pour publier cet animint sur la branche `gh-pages` de `votre_identifiant_github/dépôt_de_vis`;
- prenez une capture d'écran de cet `animint2`, et validez/poussez cette capture d'écran dans un fichier nommé `Capture.PNG` (sensible à la casse), dans la branche `gh-pages` de `votre_identifiant_github/dépôt_de_vis` ;
- ajoutez `votre_identifiant_github/dépôt_de_vis` au fichier `repos.txt` dans la branche `gh-pages` de `votre_identifiant_github/galerie`;
- exécutez le code R `animint2::update_gallery("path/to/galerie")` (notez qu'un clone du dossier de la galerie doit être présent sur le système où vous exécutez cette fonction, et que le GitHub distant doit être nommé `origin`). Le code R lira `galerie/repos.txt`, puis les méta-données (`title`, `source`, `Capture.PNG`) de chaque dossier qui n'est pas déjà présent dans `galerie/meta.csv`, il écrira ensuite les fichiers de méta-données mis à jour dans la galerie, affichera les fichiers de `galerieo/index.Rmd` dans `galerie/index.html`, validera les fichiers et les poussera vers `origin`.
- la galerie mise à jour devrait pouvoir être consultée sur le web, à l'adresse https://votre_identifiant_github.github.io/galerie après quelques minutes (généralement pas plus de cinq).

5.7 Résumé du chapitre et exercices

Ce chapitre explique comment partager des animints sur le web.

Exercices :

- Créez un animint en utilisant les options mentionnées dans ce chapitre : `out.dir` (nom du dossier où sauvegarder l'animint sur votre ordinateur), `source` (lien vers le code source R utilisé pour créer l'animint), `title` (description de l'animint).
- Utilisez Netlify Drop pour publier cette animation sur le web.
- Utilisez `animint2pages` pour publier cet animint dans un nouveau dépôt GitHub. Faites une capture d'écran et enregistrez la en tant que `Capture.PNG` dans la branche `gh-pages` de ce dépôt. Ajoutez ce dépôt à la galerie principale de l'animint [repos.txt](#) en soumettant une Pull Request sur GitHub.
- Créez votre propre galerie animint, et ajoutez y au moins deux de vos propres animints.

Dans le [Chapitre 6](#) nous vous expliquerons les différentes options qui peuvent être utilisées pour personnaliser un animint.

6

Animint options

This chapter gives a complete list of new features that animint introduces to the grammar of graphics. After reading this chapter, you will understand how to customize your animint graphics via

- the `href`, `tooltip`, `id` aesthetics for observation-specific characteristics;
- named elements of `clickSelects` and `showSelected` for specifying several selection variables at once;
- the `chunk_vars` geom-specific option;
- the `color_off`, `fill_off`, and `alpha_off` geom parameters for specifying how selection state is displayed;
- the `help` and `title` geom-specific parameters, which can be set to text strings that will be shown in the guided tour.
- plot-specific legends and height/width options; and
- global data viz options.

6.1 Observation-specific options (new aesthetics)

This section explains the new aesthetics that are recognized by animint2.

6.1.1 Review of previously introduced aesthetics

First we discuss the new aesthetics that we have already introduced in previous chapters.

Chapter 3 also introduced `aes(key)` to designate a variable to use for smooth transitions that are interpretable.

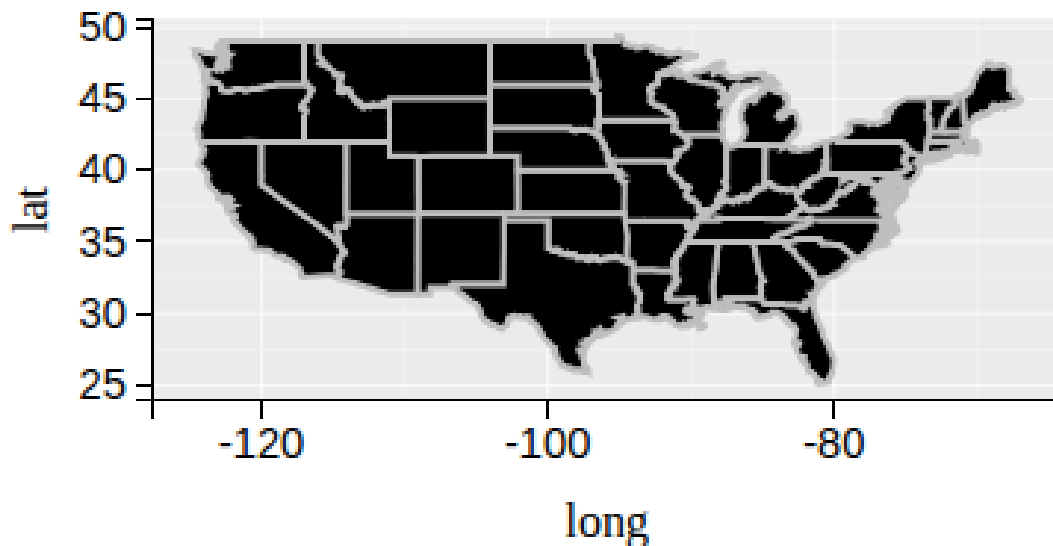
6.1.2 Hyperlinks using `aes(href)`

The code below uses animint to draw a map of the United States.

```
library(animint2)
USpolygons <- map_data("state")
animint(
  map=ggplot()+
    ggtitle("click a state to read its Wikipedia page")+
    coord_equal()+
    geom_polygon(aes(
      x=long, y=lat, group=group,
```

```
href=paste0("http://en.wikipedia.org/wiki/", region)),
data=USpolygons, fill="black", colour="grey"))
```

click a state to read its Wikipedia page

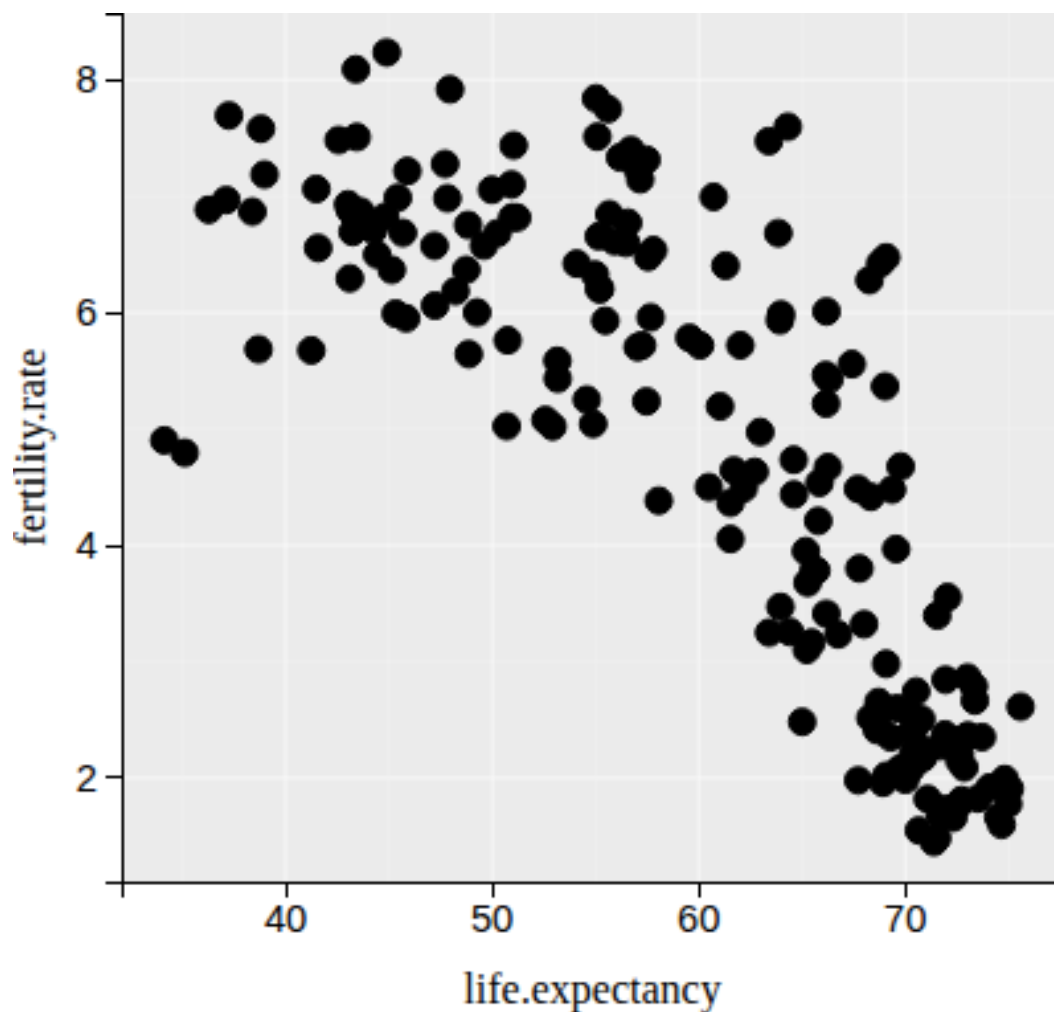


Try clicking a state in the data viz above. You should see the corresponding wikipedia page open in a new tab.

6.1.3 Tooltips using `aes(tooltip)`

Tooltips are little windows of text information that appear when you hover the cursor over something on the screen. In animint you can use `aes(tooltip)` to designate the observation-specific message that appears. For example we use it to display the population and country name in the scatterplot of the World Bank data below.

```
data(WorldBank)
WorldBank1975 <- subset(WorldBank, year == 1975)
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate,
      tooltip=paste(country, "population =", population)),
      size=5,
      data=WorldBank1975))
```



Try hovering the cursor over one of the data points. You should see a small box appear with the country name and population for that data point.

Note that a tooltip of the form “variable value” is specified by default for each geom with `aes(clickSelects)`. For example a geom with `aes(clickSelects=year)` shows the default tooltip “year 1984” for an observation with year 1984. You can change this default by explicitly specifying `aes(tooltip)`.

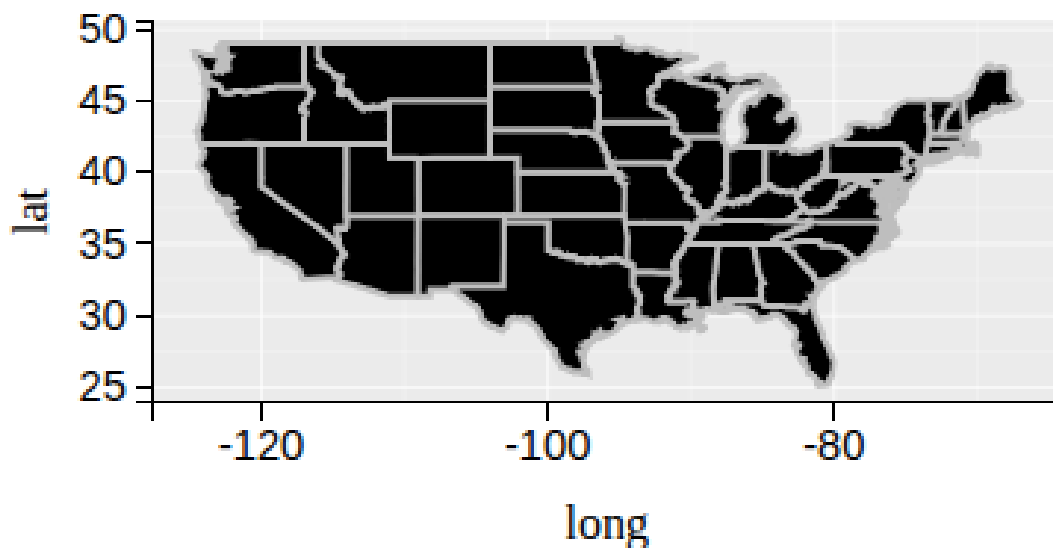
6.1.4 HTML id attribute using `aes(id)`

Since everything plotted by animint is rendered as an [SVG](#) element in a web page, you may want to specify a [HTML id attribute](#) using `aes(id)` as below.

```
animint(  
  map=ggplot()+  
    ggtitle("each state/region/group has a unique id")+  
    coord_equal()+  
    geom_polygon(aes(  
      id=state, # or region, or group, or any other variable  
      x=long,  # or lon, or x, or any other spatial variable  
      y=lat,   # or lat, or y, or any other spatial variable  
    ))  
)
```

```
x=long, y=lat, group=group,
id=gsub(" ", "_", paste(region, group))),
data=USpolygons, fill="black", colour="grey"))
```

each state/region/group has a unique id



Note how `gsub` is used to convert spaces to underscores, since a well-defined id must not include spaces. Note also that `paste` is used to add a group number, since there may be more than one polygon per state/region, and each id must be unique on a web page. The animint2 developers use this feature [for testing the animint JavaScript renderer code](#).

6.1.5 Data-driven selector names using named `clickSelects` and `showSelected`

Chapter 3 introduced `showSelected` for designating a geom which shows only the selected subset of its data.

Chapter 4 introduced `clickSelects` to designate a geom which can be clicked to change a selection variable.

Usually selector names are defined in `showSelected` or `clickSelects`. For example, `showSelected=c("year", "country")` means to create two selection variables (named `year` and `country`). However, that method becomes inconvenient if you have many selectors in your data viz. To illustrate we consider the following theoretical example (the code in this section is not directly executable). Say you want to use 20 different selector variable names, `selector1value... selector20value`. The usual way to define your data viz would be

```
viz <- list(
  points=ggplot()+
    geom_point(clickSelects="selector1value", data=data1)+
    ...
  <!-- comment -->
  geom_point(clickSelects="selector20value", data=data20)
)
```

However that method is bad since it violates the DRY principle (Don't Repeat Yourself). Another way to do that would be to use a for loop:

```
viz <- list(points=ggplot())
for(selector.name in paste0("selector", 1:20, "value")){
  data.for.selector <- all.data.list[[selector.name]]
  <!-- comment -->
  viz$points <- viz$points +
    geom_point(clickSelects=selector.name, data=data.for.selector)
}
```

That method is bad since it is slow to construct `viz`, and the compiled `viz` potentially takes up a lot of disk space since there is at least one TSV file created for each `geom_point`. The preferable method is to use a named character vector for `clickSelects`. The names should be used to indicate the column that contains the selector variable name. For example:

```
viz <- list(
  points=ggplot()+
    geom_point(
      clickSelects=c(selector.name="selector.value"),
      data=all.data)
)
```

The animint compiler looks through the data.frame `all.data` and create selectors for each of the distinct values of `all.data$selector.name`. Clicking one of the data points updates the corresponding selector with the value indicated in `all.data$selector.value`.

You can similarly use one geom with a named `showSelected` instead of a bunch of different geoms with `showSelected`.

This feature is useful not only to avoid repetition in the definition of the data viz, but also because they are more computationally efficient. For a detailed example with timings and disk space measurements, see [Chapter 14](#).

6.2 Geom options

In animint, there are several options for customization at the geom level: `chunk_vars` is used to specify how to split data sets for storage on disk, and `\*_off` parameters are used to specify how a `clickSelects` geom should be displayed when it is not selected. Additionally, `help` and `title` may be specified, to add information to the guided tour.

6.2.1 The `chunk_vars` geom-specific compilation option

The `chunk_vars` option defines the selection variables that are used to split the data set into separate chunks (TSV files) to download. There is one TSV file created for each combination of values of the `chunk_vars` variables. More selection variables specified in `chunk_vars` means to split the data set into more TSV files, each of a smaller size.

The `chunk_vars` option should be specified as an argument to a `geom_*` function, and its value should be a character vector of selection variable names. When `chunk_vars=character(0)`, a character vector of length zero, all of the data is stored in a single TSV file. When `chunk_vars` is set to all of the `showSelected` variable names, then a TSV file is created for each combination of values of those variables.

In general the animint compiler chooses a sensible default for `chunk_vars`, but you may want to specify `chunk_vars` if the data viz is loading slowly, or taking up too much space on disk. If the data viz is loading slowly, you should add selection variables to `chunk_vars` to reduce the size of the first TSV file to download. If the data viz takes up too much space on disk, you should remove selection variables from `chunk_vars` to decrease the number of TSV files. Lots of small TSV files can take more disk space than a single TSV file because some filesystems store a constant amount of metadata for every file.

To illustrate the usage of `chunk_vars`, consider the following visualization of the `breakpoints` data set.

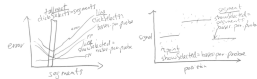


Figure 6.1: Breakpoints data viz

The sketch above consists of two plots. We begin by creating the plot of error curves on the left.

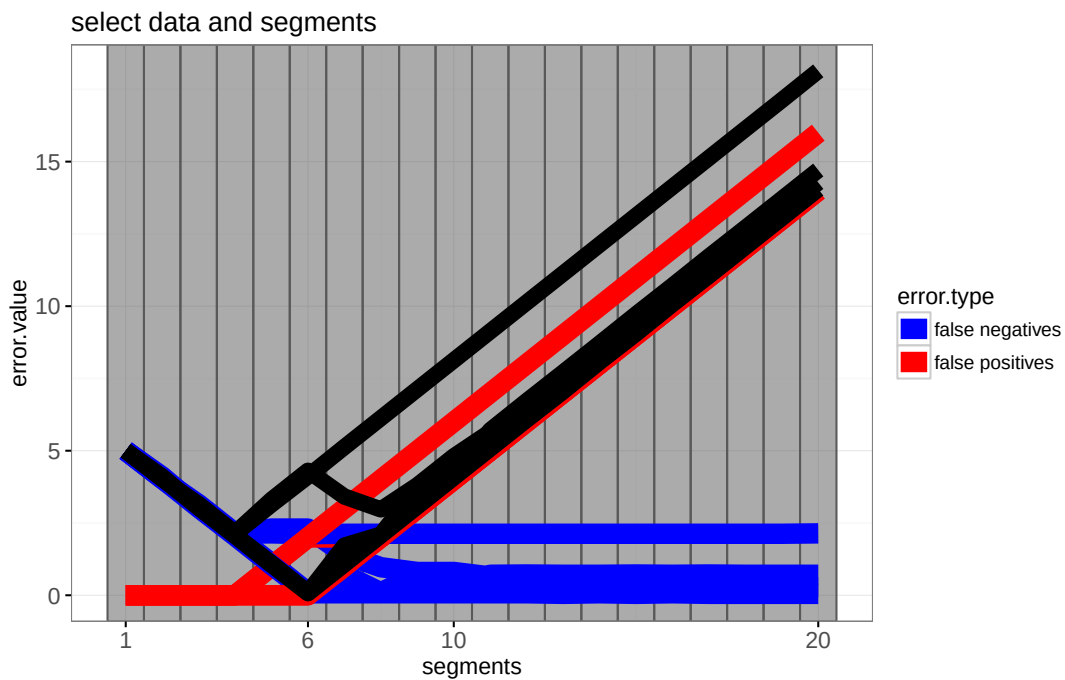
```
data(breakpoints)
only.error <- subset(breakpoints$error, type=="E")
only.segments <- subset(only.error, bases.per.probe==bases.per.probe[1])
library(data.table)
fp.fn.names <- rbind(
  data.table(error.type="false positives", type="FP"),
  data.table(error.type="false negatives", type=c("I", "FN")))
error.dt <- data.table(breakpoints$error)
error.type.dt <- error.dt[fp.fn.names, on=list(type)]
fp.fn.dt <- error.type.dt[, list(
  error.value=sum(error)
), by=(error.type, segments, bases.per.probe)]
errorPlot <- ggplot()+
  ggtitle("select data and segments")+
  theme_bw()+
  geom_tallrect(aes(
    xmin=segments-0.5, xmax=segments+0.5),
    clickSelects="segments",
    data=only.segments,
```



```

    alpha=1/2)+
  geom_line(aes(
    segments, error.value, color=error.type,
    group=paste(bases.per.probe, error.type)),
    showSelected="bases.per.probe",
    data=fp.fn.dt,
    size=5)+
  scale_color_manual(values=c(
    "false positives"="red", "false negatives"="blue"))+
  geom_line(aes(
    segments, error, group=bases.per.probe),
    clickSelects="bases.per.probe",
    data=only.error,
    size=4)+
  scale_x_continuous(breaks=c(1, 6, 10, 20))
errorPlot

```



The plot above includes a `geom_tallrect` with `clickSelects=segments` and a `geom_line` with `clickSelects=bases.per.probe`. It will be used to select the data and model in the plot below.

```

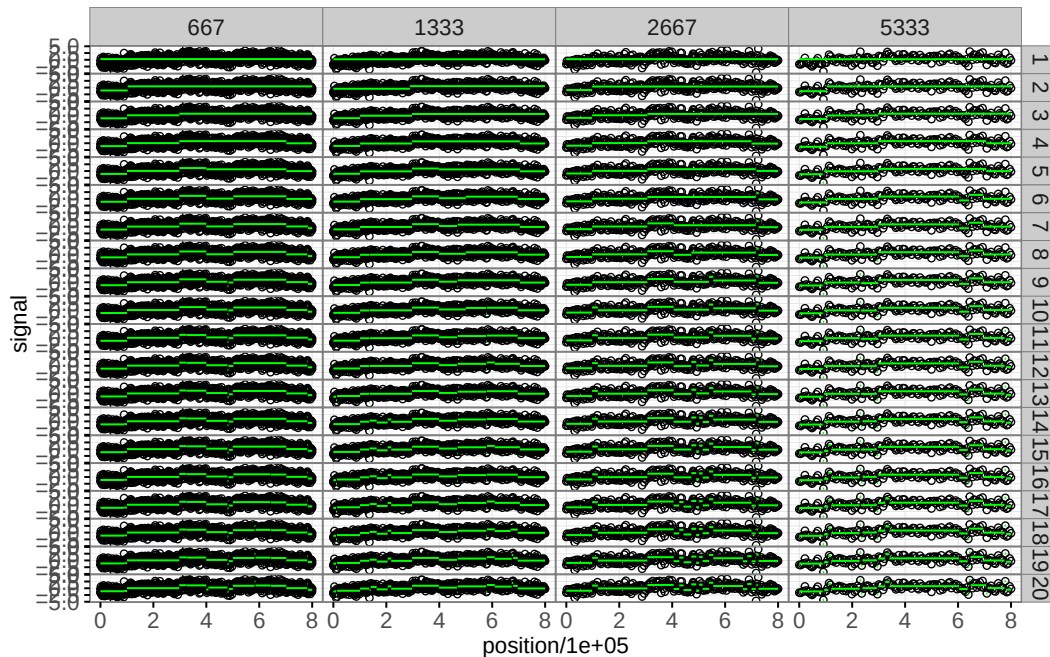
signalPlot <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  theme_animint(height=800)+
  geom_point(aes(
    position/1e5, signal),

```

```

showSelected="bases.per.probe",
shape=1,
data=breakpoints$signals)+
geom_segment(aes(
  first.base/1e5, mean, xend=last.base/1e5, yend=mean),
showSelected=c("segments", "bases.per.probe"),
color="green",
data=breakpoints$segments)
signalPlot+facet_grid(segments ~ bases.per.probe)

```

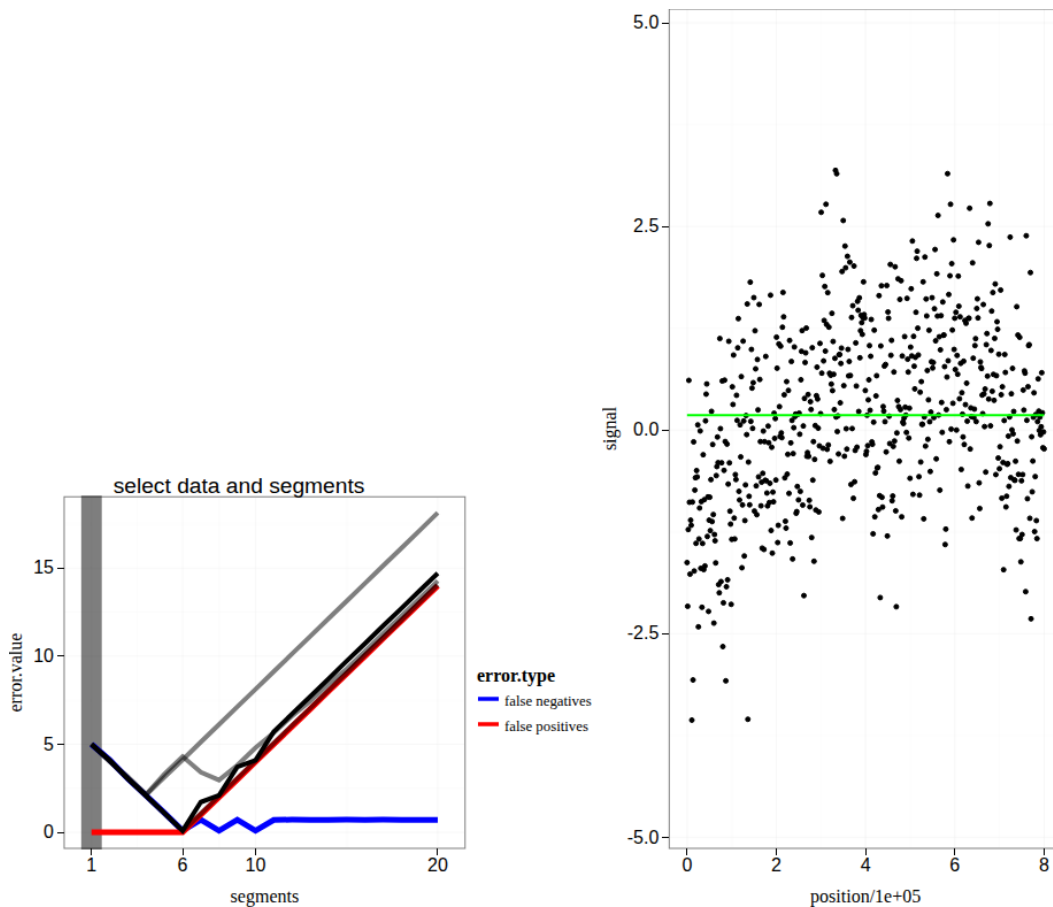


The non-interactive plot above has 80 facets, one for each combination of the two `showSelected` variables, `bases.per.probe` and `segments`. Below we make an interactive version in which only one of these facets will be shown.

```

(viz.chunk.vars <- animint(
  errorPlot,
  signal=signalPlot+
    geom_vline(aes(
      xintercept=base/1e5),
      showSelected=c("segments", "bases.per.probe"),
      color="green",
      chunk_vars=character(),
      linetype="dashed",
      data=breakpoints$breaks)))

```



Click the “Show download status table” button, and you should see counts of chunks (TSV files). Note that `geom6_vline_signal` has only 1 chunk, since `chunk_vars=character()` is specified for the `geom_vline` in the R code above. If another value of `chunk_vars` was specified, it would create a different number of TSV files, but the appearance of the data viz should be the same.

Below we use the `du` command line program to determine the disk usage of the data viz for different choices of `chunk_vars`.

```
tsvSizes <- function(segment.chunk.vars){
  viz <- list(
    error=errorPlot,
    signal=signalPlot+
      geom_vline(aes(
        xintercept=base/1e5),
        showSelected=c("segments", "bases.per.probe"),
        color="green",
        chunk_vars=segment.chunk.vars,
        linetype="dashed",
        data=breakpoints$breaks)
  )
}
```

```

info <- animint2dir(viz, open.browser=FALSE)
cmd <- paste("du -ks", info$out.dir)
kb.dt <- fread(cmd=cmd)
setnames(kb.dt, c("kb", "dir"))
tsv.vec <- Sys.glob(paste0(info$out.dir, "/*.tsv"))
is.geom6 <- grepl("geom6", tsv.vec)
data.frame(
  kb=kb.dt$kb, geom6.tsv=sum(is.geom6), other.tsv=sum(!is.geom6))
}
chunk_vars_list <- list(
  neither=c(),
  bases.per.probe=c("bases.per.probe"),
  segments=c("segments"),
  both=c("segments", "bases.per.probe"))
sizes.list <- lapply(chunk_vars_list, tsvSizes)
(sizes <- do.call(rbind, sizes.list))

```

	kb	geom6.tsv	other.tsv
neither	844	1	12
bases.per.probe	848	5	12
segments	904	19	12
both	1132	76	12

The table above includes counts of kilobytes for the data viz, along with counts of TSV files for `geom6_vline_signal` and the other geoms. Note how the choice of `chunk_vars` affects the number of TSV files and the disk space usage. Since `chunk_vars` was only specified for `geom6_vline_signal`, the number of TSV files for the other geoms does not change. When both `segments` and `bases.per.probe` are specified for `chunk_vars`, there are 76 TSV files for `geom6_vline_signal`, and the data viz takes 1132 kilobytes. In contrast, `chunk_vars=character()` produces only one TSV file for `geom6_vline_signal`, and the data viz uses 844 kilobytes.

In conclusion, the geom-specific `chunk_vars` option defines the number of TSV files created for each geom. When deciding the value of `chunk_vars`, you should consider both disk usage and loading time. A few large files take up less disk space but are slower to download than many small files.

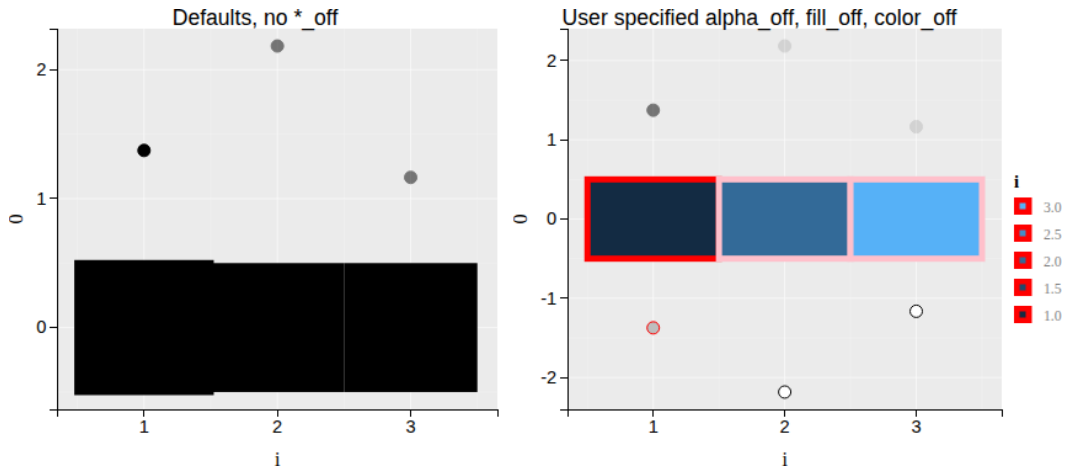
6.2.2 Specifying how selection state is displayed

Animint has sensible defaults for displaying selection state. In particular,

- when there is a rect or tile with `clickSelects`, we use black color/border to show items which are selected, and transparent for items which are not selected.
- for any other geom with `clickSelects`, we use full opacity `alpha` to show items which are selected, and `alpha=0.5` opacity to show items which are not selected.

The defaults explained above are illustrated in the first plot below. Those defaults may be customized by using the `alpha_off`, `fill_off`, and `color_off` geom parameters as in the code below,

```
N <- 3
set.seed(1)
demo_df <- data.frame(i=1:N, num=rnorm(N,2))
animint(
  defaults=ggplot()+
    ggtitle("Defaults, no *_off")+
    geom_tile(aes(
      i, 0),
      size=5,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      clickSelects="i",
      data=demo_df),
  off=ggplot()+
    ggtitle("User specified alpha_off, fill_off, color_off")+
    geom_tile(aes(
      i, 0, fill=i),
      clickSelects="i",
      color="red",
      color_off="pink",
      size=5,
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      alpha=0.5,
      alpha_off=0.1,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, -num),
      size=5,
      alpha=1,
      alpha_off=1,
      color="red",
      color_off="black",
      fill="grey",
      fill_off="white",
      clickSelects="i",
      data=demo_df))
```

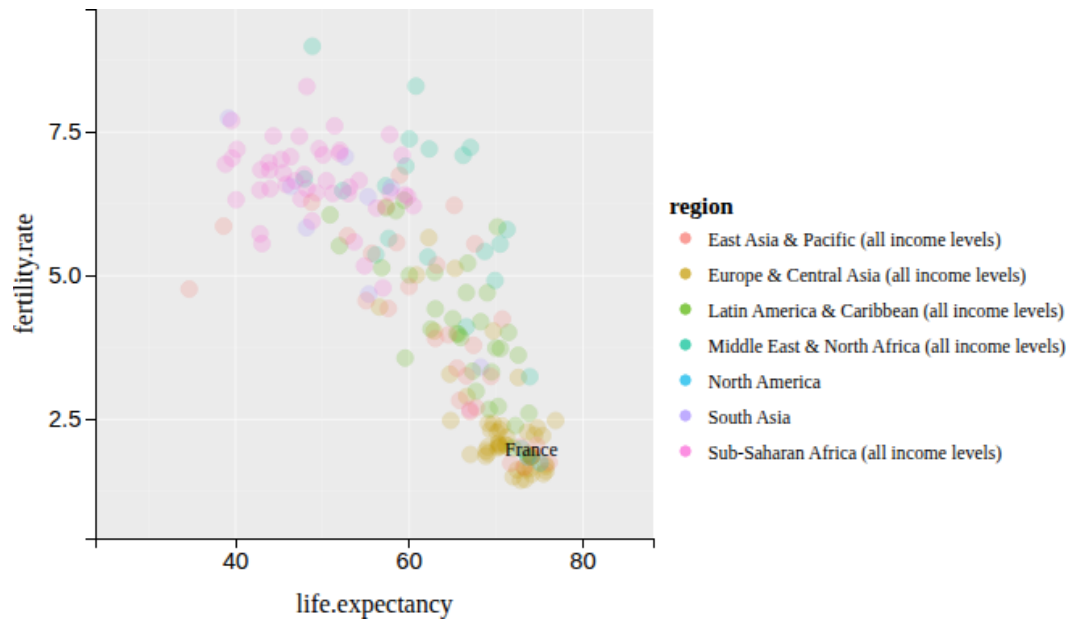


Note that when using any one of these visual properties in the `aes` mapping, it should not be specified as a geom parameter. For example in the tile above, we used `aes(fill)`, so `fill` and `fill_off` should not be specified as parameters for that geom (in order to make it clear that fill is used for displaying data values, not selection state).

6.2.3 Specifying guided tour text

Since Jan 2025, animint supports a guided tour, which displays information about possible interactions with each geom. To customize what is displayed for each geom, you can specify the `help` and `title` parameters, as in the code below.

```
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=region),
      size=5,
      showSelected="year",
      clickSelects="country",
      help="One point drawn for each country in the selected year",
      alpha=0.7,
      data=WorldBank)+
    geom_text(aes(
      x=life.expectancy, y=fertility.rate, label=country),
      data=WorldBank,
      title="Selected country",
      showSelected=c("year", "country")),
  first=list(
    country="France",
    year=1980))
```



In the code above, we specify `help` for `geom_point`, which controls the sub-text which is displayed for that geom, after clicking the “Start Tour” button at the bottom of the data visualization. After clicking the “Next” button, we can see the `title` that was specified in the code, shown at the top of the tour window, for the `geom_text`. This mechanism can be used to provide extra helpful information for the users of your data visualization, so they can more easily understand what is displayed, and what interactions are possible.

6.3 Plot-specific options

This section discusses options which are specific to one ggplot of a data viz. The `theme_animint` function is used to attach animint options to ggplot objects.

6.3.1 Plot height and width

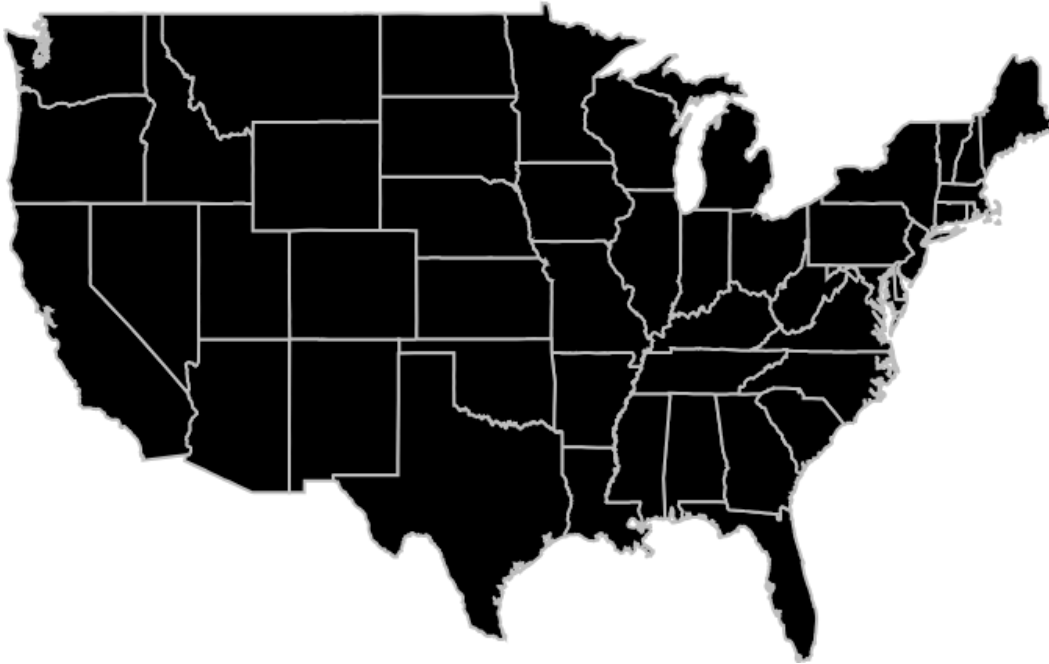
The `width` and `height` options are for specifying the dimensions (in pixels) of a ggplot rendered by animint. For example, consider the following re-design of the plot of the United States:

```
animint(
  map=ggplot()+
  theme_animint(width=750, height=500)+
  theme(
    axis.line=element_blank(),
    axis.text=element_blank(),
    axis.ticks=element_blank(),
    axis.title=element_blank(),
    panel.border=element_blank(),
```

```

panel.background=element_blank(),
panel.grid.major=element_blank(),
panel.grid.minor=element_blank()+
geom_polygon(aes(
  x=long, y=lat, group=group),
  data=USpolygons, fill="black", colour="grey"))

```



Note that the plot above was rendered with a width of 750 pixels and a height of 500 pixels, due to the `theme_animint` options. If either of these options is not specified for any ggplot, then animint uses a default of 400 pixels.

Also note that `theme` was used to specify several blank elements. This has the effect of removing the axes and background, and is generally useful for rendering maps.

6.3.2 Size scale in pixels

The `scale_size_animint` scale should be used in all ggplots where you specify `aes(size)`. To see why, consider the following examples.

```

scatter1975 <- ggplot()+
  geom_point(
    aes(x=life.expectancy, y=fertility.rate, size=population),
    WorldBank1975,
    shape=21,
    color="red",
    fill="black")
(viz.scale.size <- animint(
  ggplotDefault=scatter1975+

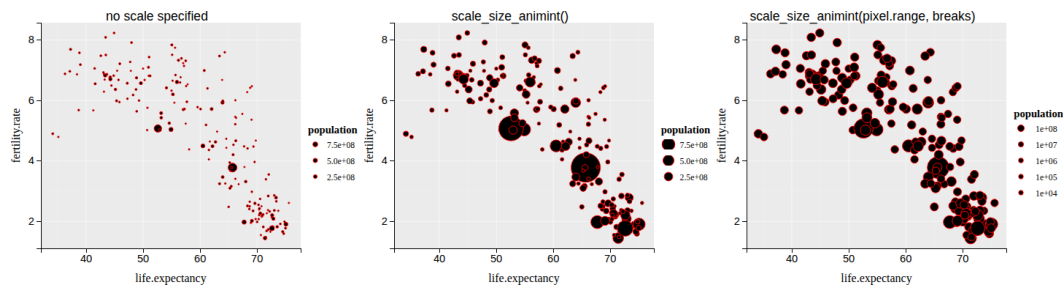
```



```

ggtitle("no scale specified"),
animintDefault=scatter1975+
ggtitle("scale_size_animint()")+
scale_size_animint(),
animintOptions=scatter1975+
ggtitle("scale_size_animint(pixel.range, breaks)")+
scale_size_animint(pixel.range=c(5, 15), breaks=10^(10:1)))

```



The first ggplot above has no scale specified, so it uses the default ggplot2 scale, which has two problems. The first problem is that it seems that all countries have about the same size except the two really big countries. That problem can be fixed by simply adding `scale_size_animint()` to the ggplot, which results in the second plot above. However, a second problem is that the legend entries do not show the full range of the data. That problem is fixed in the third plot above, by manually specifying the `breaks` to use for legend entries. Note that the `pixel.range` argument can also be used to specify the radius of the largest and smallest circles.

6.3.3 Axes and legend text size

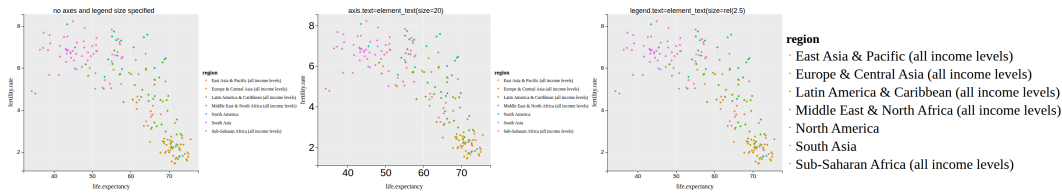
The syntax of defining axes and legend text size (in pixels) is almost the same as ggplot2. Inside `theme`, you can use numbers directly to change the font size, or you can use `rel()` to define the relative size.

```

scatter1975 <- ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=WorldBank1975)
(viz.text.size <- animint(
  animintDefault=scatter1975+
  theme_animint(width=500, height=500)+
  ggtitle("no axes and legend size specified"),
  animintAxesOptions=scatter1975+
  theme_animint(width=500, height=500)+
  theme(axis.text=element_text(size=20))+
  ggtitle("axis.text=element_text(size=20)"),
  animintLegendOptions=scatter1975+
  theme_animint(width=500, height=500)+
  theme(
    legend.title=element_text(size=24),
    legend.text=element_text(size=rel(2.5)))+

```

```
ggtitle("legend.text=element_text(size=rel(2.5)))")
```



This allows you to change the font size while changing the size of the plot to make it look more coherent.

Note that the default font size in animint is 11px for the axes and 16px for the legend.

6.4 Global data viz options

Global data viz options are any named elements of the `viz` list that are not ggplots.

6.4.1 Review of previously introduced global options

Chapter 3 introduced the `duration` option for specifying the duration of smooth transitions.

Chapter 3 introduced the `time` option for specifying a selection variable which is automatically updated (animation).

Chapter 4 introduced the `first` option for specifying the selection when the data viz is first rendered.

Chapter 4 introduced the `selector.types` option for specifying multiple selection variables.

6.4.2 Web page title with the `title` option

The `title` option should be a character string, and will be used to set the `\<title>` element of the web page. It does not make sense to use the `title` option in an Rmd document such as this page. A title can and should be used with `animint2dir`, as in the code below.

```
viz.title <- viz.scale.size
viz.title$title <- "Several size scales"
animint2dir(viz.title, "Ch06-viz-title")
```

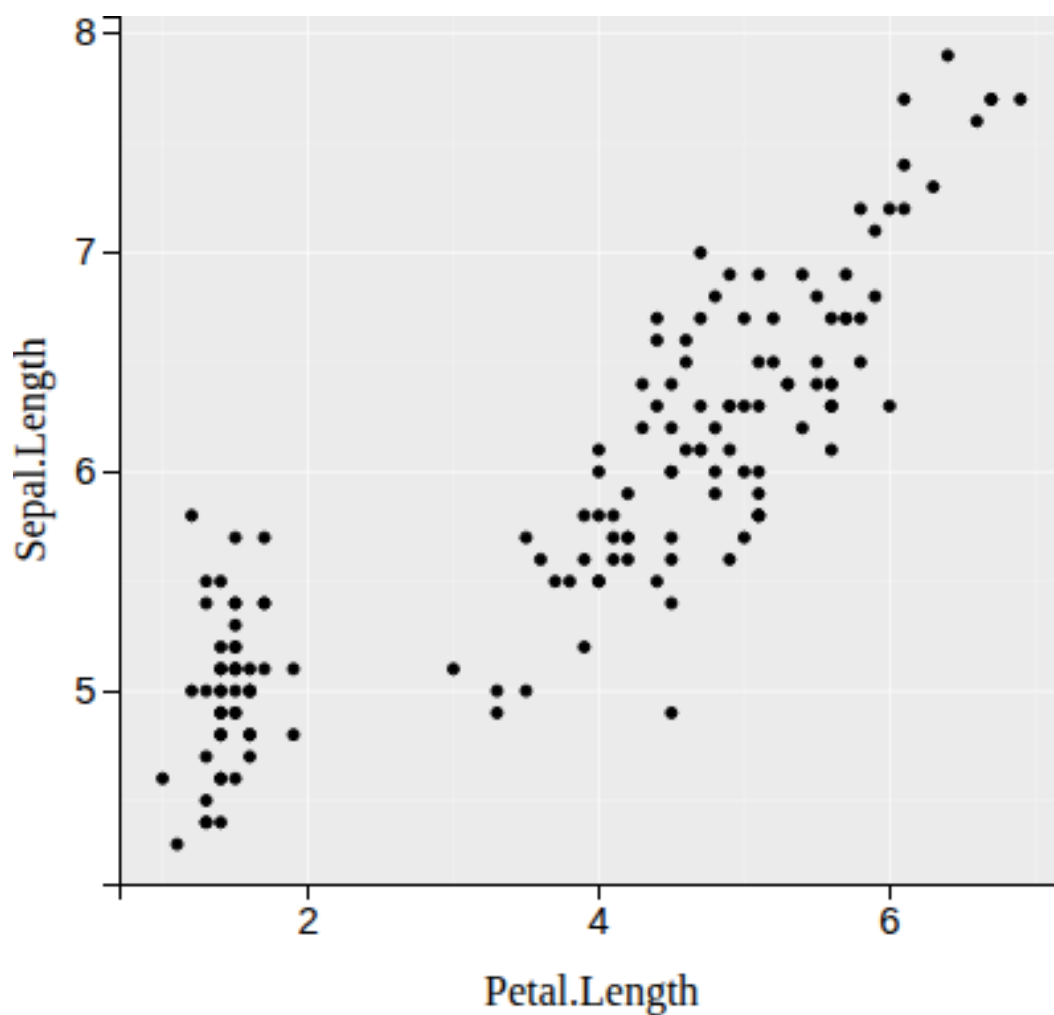
Note that `viz.scale.size` already has three ggplots, each with a `ggtitle`. Adding the global `title` option has the effect of defining a title for [the web page](#).

Chapter 5 introduced the `animint2pages` function, which is used to publish an animint to GitHub Pages. It requires that the animint defines the `title` option, because that meta-data is required for organizing the animint in a gallery.

6.4.3 Link R code with source option

The `source` option should be a character string: a link to the R source code which was used to create the animint.

```
animint(  
  demo=ggplot()+  
    geom_point(aes(  
      Petal.Length, Sepal.Length),  
      data=iris),  
  source=paste0(  
    "https://github.com/animint/animint-manual-fr",  
    "/blob/main/Chapitres/Ch06/Ch06_source.Rmd"))
```



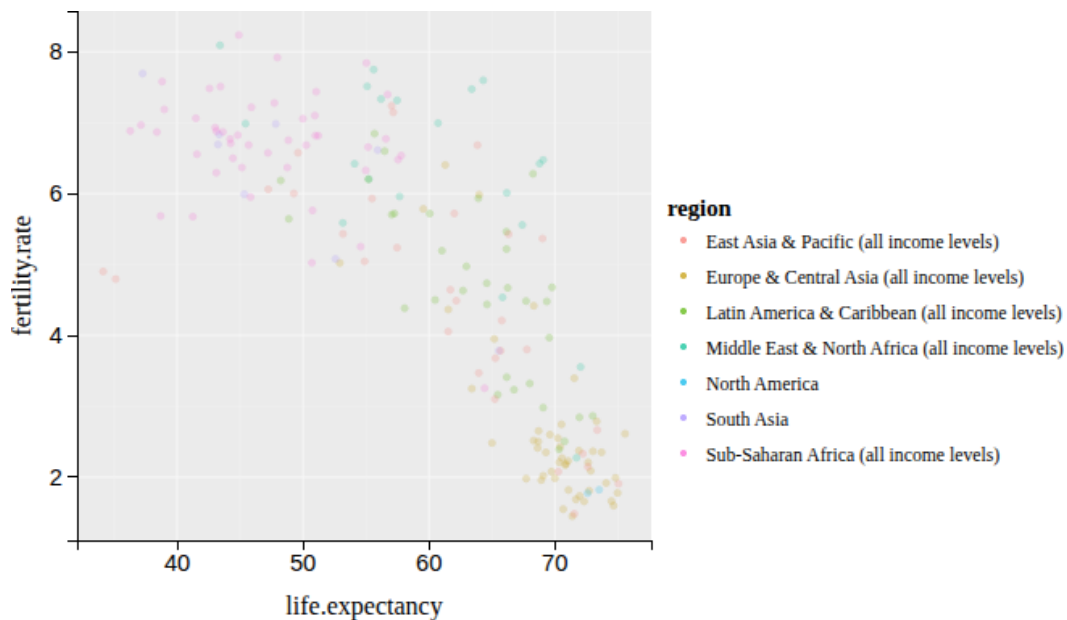
Note above how there is a source link at the bottom of the data viz.

Chapter 5 introduced the `animint2pages` function, which is used to publish an animint to GitHub Pages. It requires that the animint defines the `source` option, because that meta-data is required for organizing the animint in a gallery .

6.4.4 Link a video

The `video` option should be a character string: a link to a video which shows typical interactions with the animint. This mechanism can be used to help the users of your data visualization understand what is displayed, and what interactions they can use.

```
animint(
  video="https://vimeo.com/1050117030",
  scatter=ggplot()+
    geom_point(aes(
      x=life.expectancy, y=fertility.rate, color=region),
      clickSelects="country",
      alpha=0.7,
      data=WorldBank1975))
```



In the data visualization above, notice the “video” link which appears in the bottom right. Clicking that link leads to a video that was recorded to explain a more complex data visualization based on the World Bank data. The idea is that you can record a video for each of your animints, and then include a link to the video using this mechanism, so your users can more easily understand what is displayed, and what interactions are possible.

6.4.5 Show or hide selection menus with the selectize option

The `selectize` option should be a named list of logical values. Names should be selector variables, and values should indicate whether or not you would like to render a selection menu via `selectize.js`. By default, animint will render a selection menu for every selection variable, with two exceptions:

- data-driven selection variables that are defined using `namedclickSelects/showSelectedvariables`.
- selection variables that have a lot of values (they are slow to render).

These defaults should work well for the vast majority of animints. For those who are interested to see an example of how the `selectize` option works, please see the [PredictedPeaks test in the animint2 source code](#) .

6.5 Chapter summary and exercises

This chapter explained several options for customizing animints at the observation, geom, plot, and global level.

Exercises:

- Create other versions of `viz.chunk.vars` with different values of `chunk_vars` for the `geom_point` and `geom_segment`. How does the choice of `chunk_vars` affect the appearance of the visualization? The disk space? The loading time?

Next, [Chapter 7](#) explains the limitations of the current implementation of animint2.



Limitations of animint2, and workarounds

This chapter explains several known limitations of animint2 for some interactive data visualization tasks. It also explains some workarounds that you can use in these situations. After reading this chapter, you will understand how to

- Use a normalized variable when different `showSelected` subsets have very different values of a variable you want to display.
- Compute statistics for each `showSelected` subset, rather than relying on the `stat_*` functions in `ggplot2`.
- Add data one at a time to a multiple selection variable set, rather than using a rectangular selection brush.
- Avoid using `vjust` and labels with multiple lines in `geom_text`.
- Order the plots on the page.
- Avoid using some unsupported `theme` options.
- Use facets with multiple variables per axis.
- Use the `shiny` web server package to interactively change the aesthetic mapping of an animint, or to perform computations based on selected values.

If you have an idea for improving animint2 so that it overcomes one of these limitations, the animint2 developers would be more than happy to accept your [Pull Request](#) .

7.1 Use normalized variables to work with fixed scales

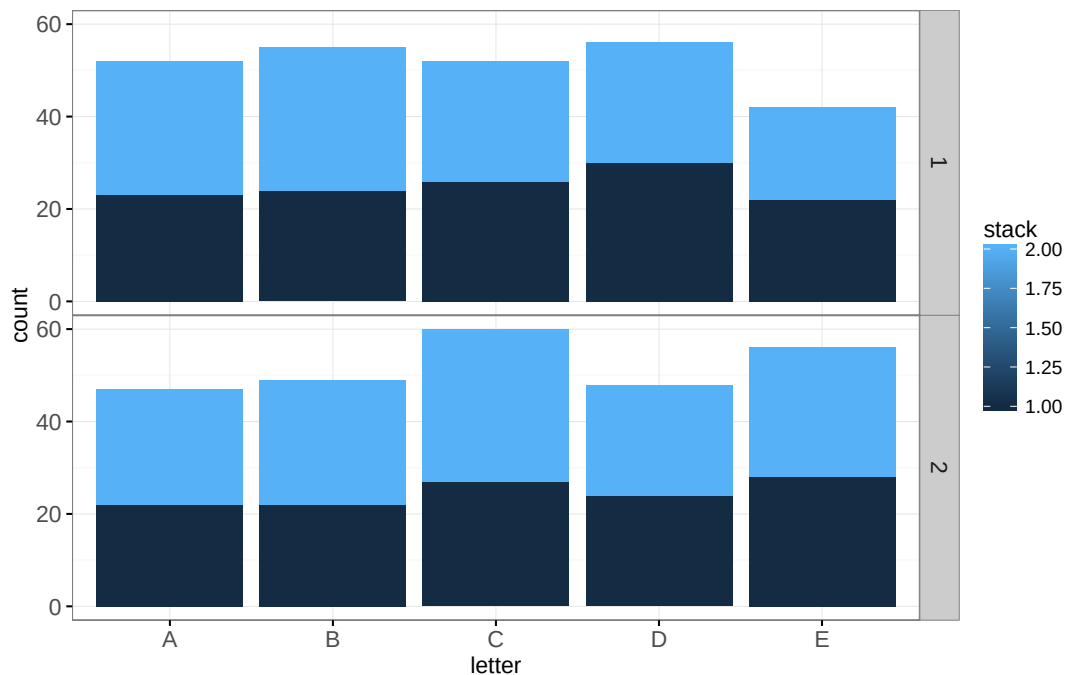
We implement axes and legends in the same way `ggplot2` does, by computing them once when the plot is first rendered. As a consequence, the axes and legends in each animint plot are not interactive. For most animated data visualizations , fixed axes make it easy to understand how the data changes along with the `time` variable.

There are some situations where it would be useful to have axes that interactively update. One example is when different `showSelected` subsets have very different values for variables that are shown with an axis or legend. In this case it would be useful to have interactive axes that update and change along with the data. We have experimental support for this, see [the update axetest](#) for details.

7.2 Compute statistics for each showSelected subset

Animint does not support ggplot2 statistics with `showSelected`. For example, consider the faceted ggplot below.

```
set.seed(1)
library(data.table)
random.counts <- data.table(
  letter = c(replicate(4, LETTERS[1:5])),
  count = c(replicate(4, rbinom(5, 50, 0.5))),
  stack = rep(rep(1:2, each = 5), 2),
  facet = rep(1:2, each = 10))
library(animint2)
ggstat <- ggplot() +
  theme_bw() +
  theme(panel.margin=grid::unit(0, "lines")) +
  geom_bar(
    aes(letter, count, fill = stack),
    showSelected="facet",
    data = random.counts,
    stat = "identity",
    position="stack"
  )
ggstat+facet_grid(facet ~ .)
```



Using `showSelected` instead of facets does not result in what you may expect.


```

animint(
  plot = ggstat,
  time = list(variable = "facet", ms = 1000),
  duration = list(facet = 1000))

```

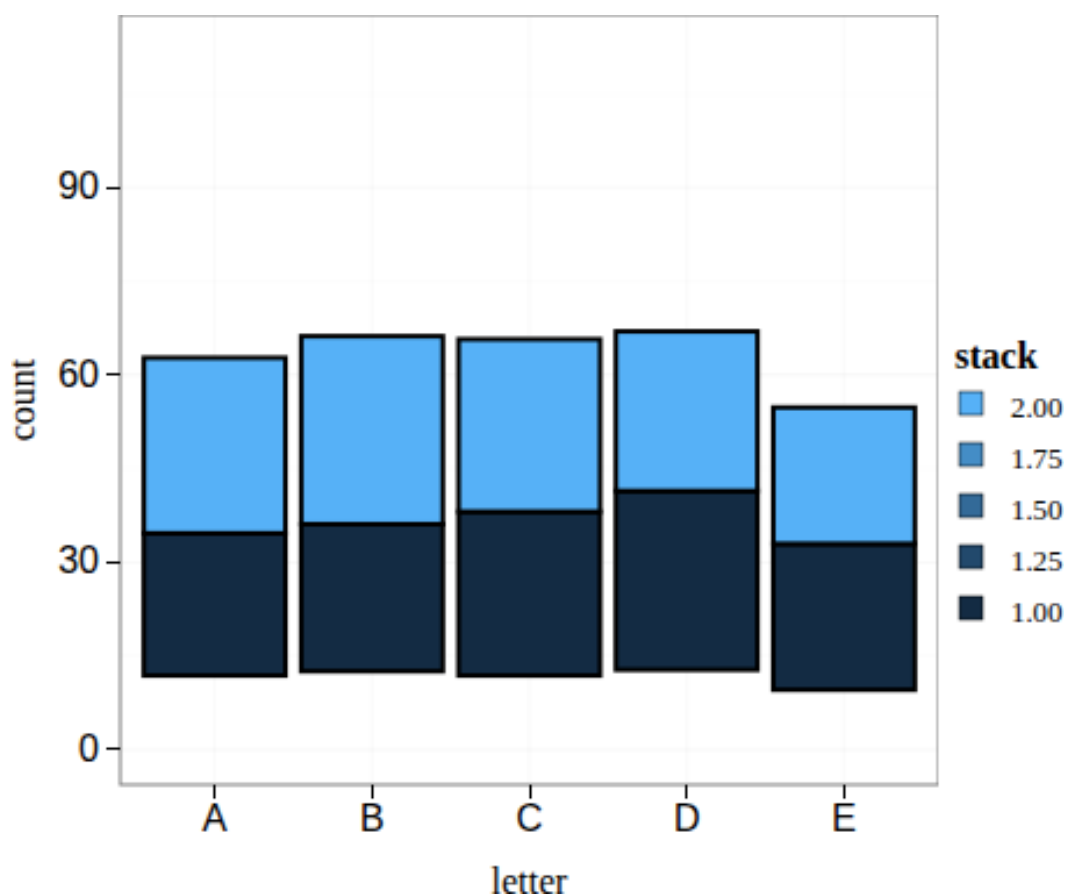
```

mapping: x = letter
y = count
fill = stack
showSelected1 = facet
geom_bar: width = NULL
na.rm = FALSE
stat_identity: na.rm = FALSE
position_stack

```

Warning in f(...): showSelected only works with position=identity, problem: geom1_bar_plot

Warning in issueSelectorWarnings(meta\$geoms, meta\$selector.aes, meta\$duration): to ensure that smooth transitions are interpretable, aes(key) should be specified for geoms with showSelected=facet, problem: geom1_bar_plot

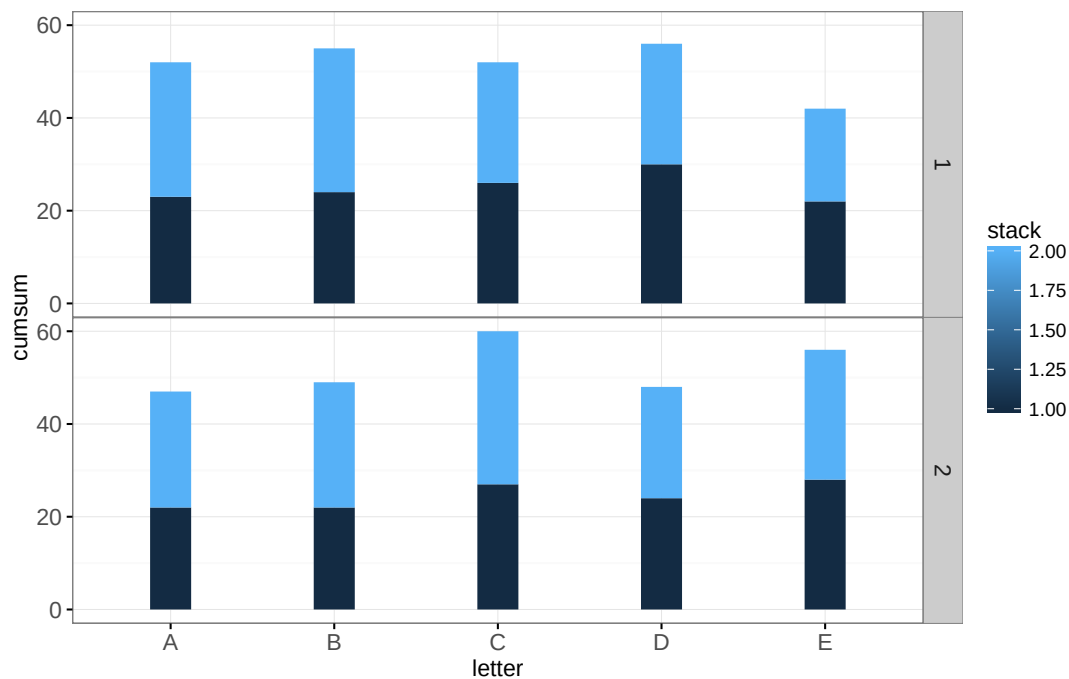


A workaround is to compute what you want to display, and use position=identity.

```

random.cumsums <- random.counts[, list(
  cumsum=cumsum(count),
  count=count,
  stack=stack),
  by=.(letter, facet)]
ggidentity <- ggplot() +
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_segment(aes(
    x=letter, xend=letter,
    y=cumsum, yend=cumsum-count,
    color=stack),
  showSelected="facet",
  data = random.cumsums,
  size=10,
  stat = "identity",
  position="identity")
ggidentity+facet_grid(facet ~ .)

```



Note how we used `geom_segment` instead of `geom_bar`, but their appearance is similar.

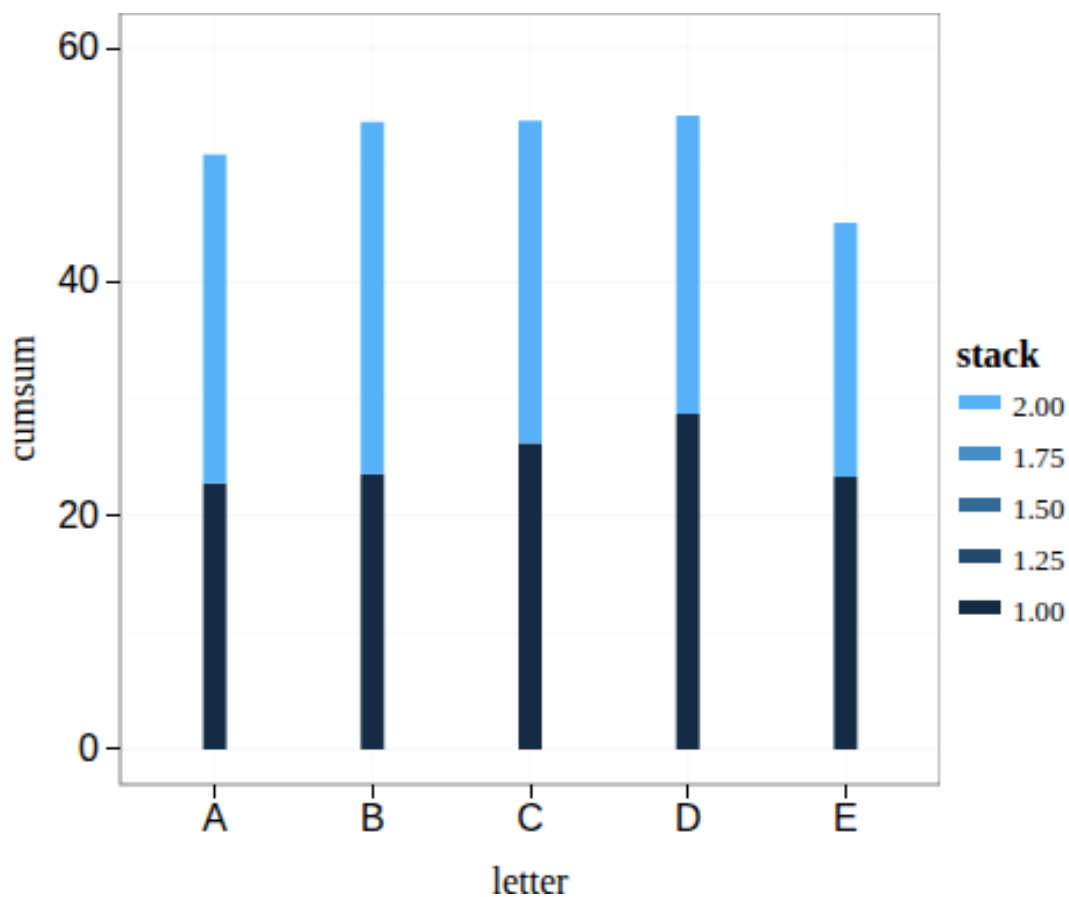
```

animint(
  plot = ggidentity,
  time = list(variable = "facet", ms = 1000),
  duration = list(facet = 1000))

```

Warning in `issueSelectorWarnings(meta$geoms, meta$selector.aes, meta$duration)`:
to ensure that smooth transitions are interpretable, `aes(key)` should be

specified for geoms with `showSelected=facet`, problem: `geom1_segment_plot`



7.3 Add values to a multiple selection set one at a time

Animint does not support a rectangular brush or lasso for interactively defining a set of selected values. Instead, animint supports multiple selection by adding values one by one to the multiple selection set. Use the `selector.types` option to declare a multiple selection variable.

7.4 Adjust y instead of using `vjust`

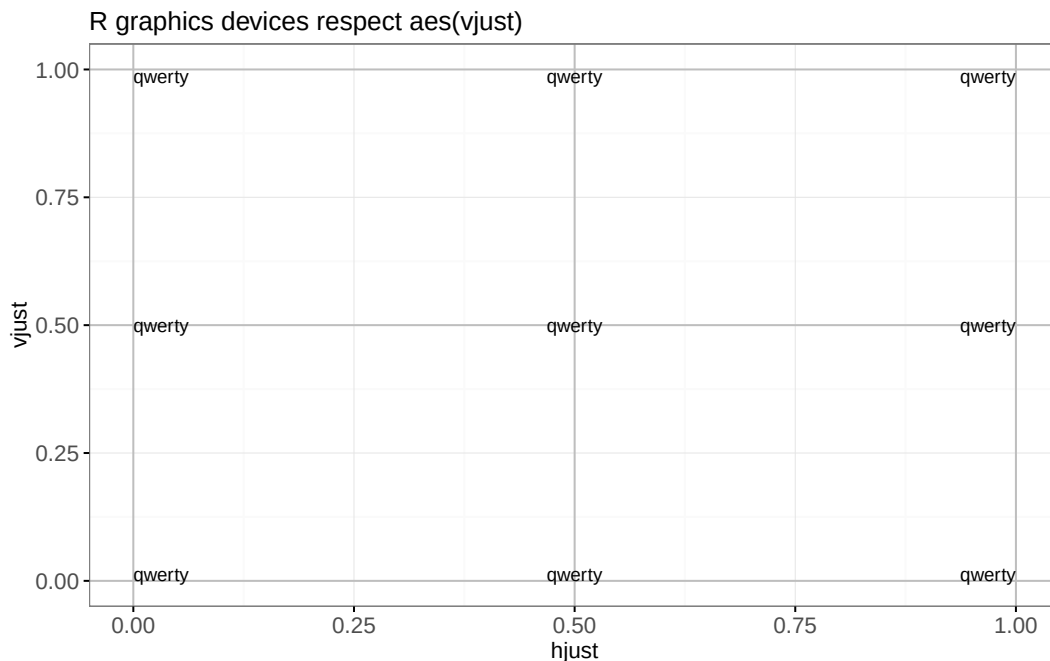
For horizontal text alignment, animint supports using `hjust` in R with the most common values: 0 for left alignment, 0.5 for middle alignment, and 1 for right alignment. Animint translates these three `hjust` values to the `text-anchor` property in the rendered data viz.

However, animint does not support vertical text alignment using `vjust` in R, because there is no property that can be used for vertical text alignment in SVG.

```

line.df <- data.frame(just=c(0, 0.5, 1))
text.df <- expand.grid(
  vjust=line.df$just,
  hjust=line.df$just)
gg.lines <- ggplot()+
  theme_bw()+
  geom_vline(aes(xintercept=just), data=line.df, color="grey")+
  geom_hline(aes(yintercept=just), data=line.df, color="grey")
gg.vjust <- gg.lines+
  geom_text(aes(
    hjust, vjust, label="qwerty", hjust=hjust, vjust=vjust),
    size=10,
    data=text.df)
gg.vjust+ggtitle("R graphics devices respect aes(vjust)")

```



Note how both `hjust` and `vjust` are respected in the static `ggplot` above. In contrast, consider the `animint` below. The left plot should be the same as the `ggplot` above, but there are clear differences in terms of vertical placement of the text elements.

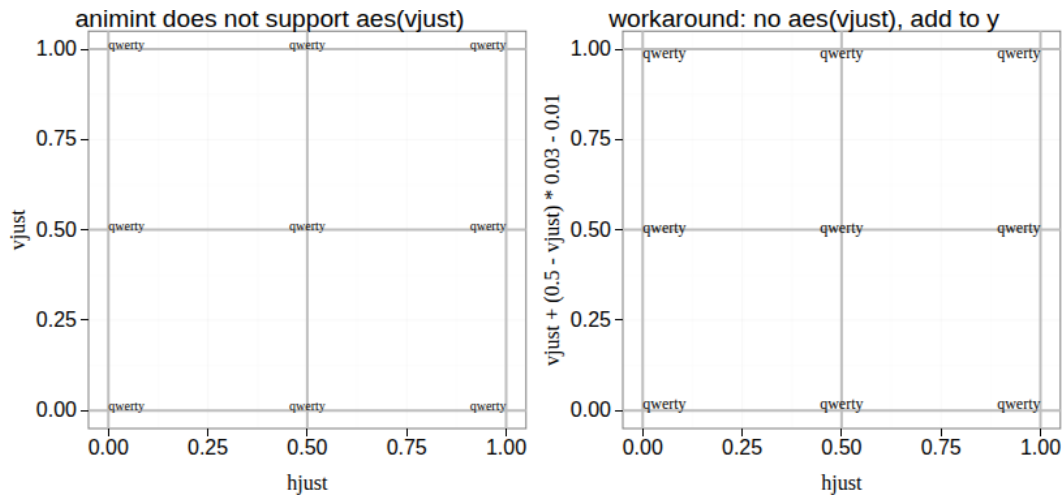
```

(viz.just <- animint(
  vjust=gg.vjust+
  ggtitle("animint does not support aes(vjust)"),
  workaround=gg.lines+
  ggtitle("workaround: no aes(vjust), add to y")+
  geom_text(aes(
    hjust, vjust + (0.5-vjust)*0.03 - 0.01,
    label="qwerty", hjust=hjust),
    data=text.df)))

```

```
[1] 0.5 1.0
```

Warning in `vjustWarning(g.data$vjust)`: `geom_text` currently only supports `vjust=0`, but you may want to try `geom_label_aligned`, which supports `vjust` values 0, 0.5, and 1



The workaround in animint is shown in the right panel above. You can adjust the vertical position `y` values of the text elements for which you would have used `vjust`.

It is possible to implement `vjust` support in animint, but we haven't yet had time to work on it. If you would like to implement it, we would be more than happy to accept a Pull Request. We already have an [issue explaining how to implement it](#).

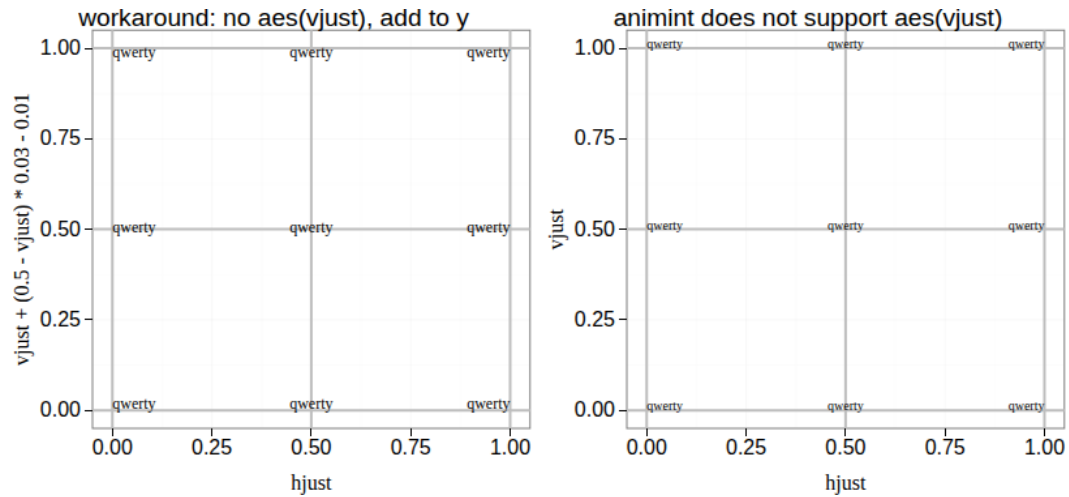
7.5 Order the plots on the page

Currently the only way to organize a multi-plot data viz is using the order of the ggplots in the animint `viz` list. The plots will appear on the web page in the same order as they occur in the animint `viz` list. For example, compare the data viz below with the data viz from the previous section.

```
animint(
  first=viz.just$workaround,
  second=viz.just$vjust)
```

```
[1] 0.5 1.0
```

Warning in `vjustWarning(g.data$vjust)`: `geom_text` currently only supports `vjust=0`, but you may want to try `geom_label_aligned`, which supports `vjust` values 0, 0.5, and 1



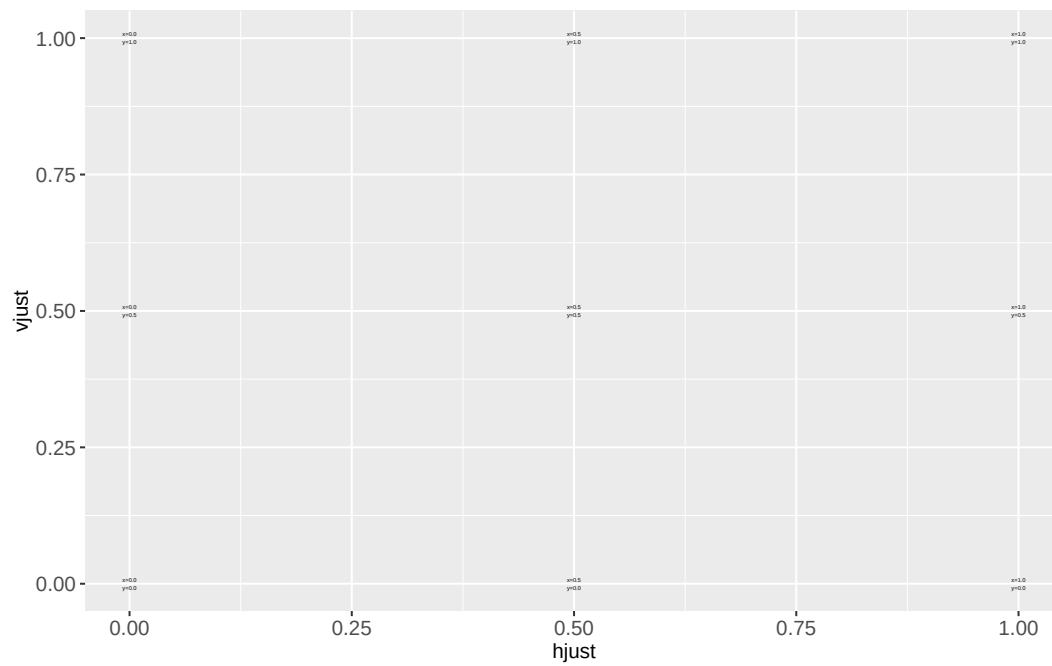
Note how the order of plots is reversed with respect to the data viz from the previous section. The main limitation of this method for plot layout is that only the order can be controlled. For example, depending on the width of the web page element in which the data viz above is rendered, the second plot will appear either below the first plot, or to the right of it.

If you have an idea for a better way to define the layout of plots in an *animint*, [please tell us](#)!

7.6 Avoid line breaks in text labels

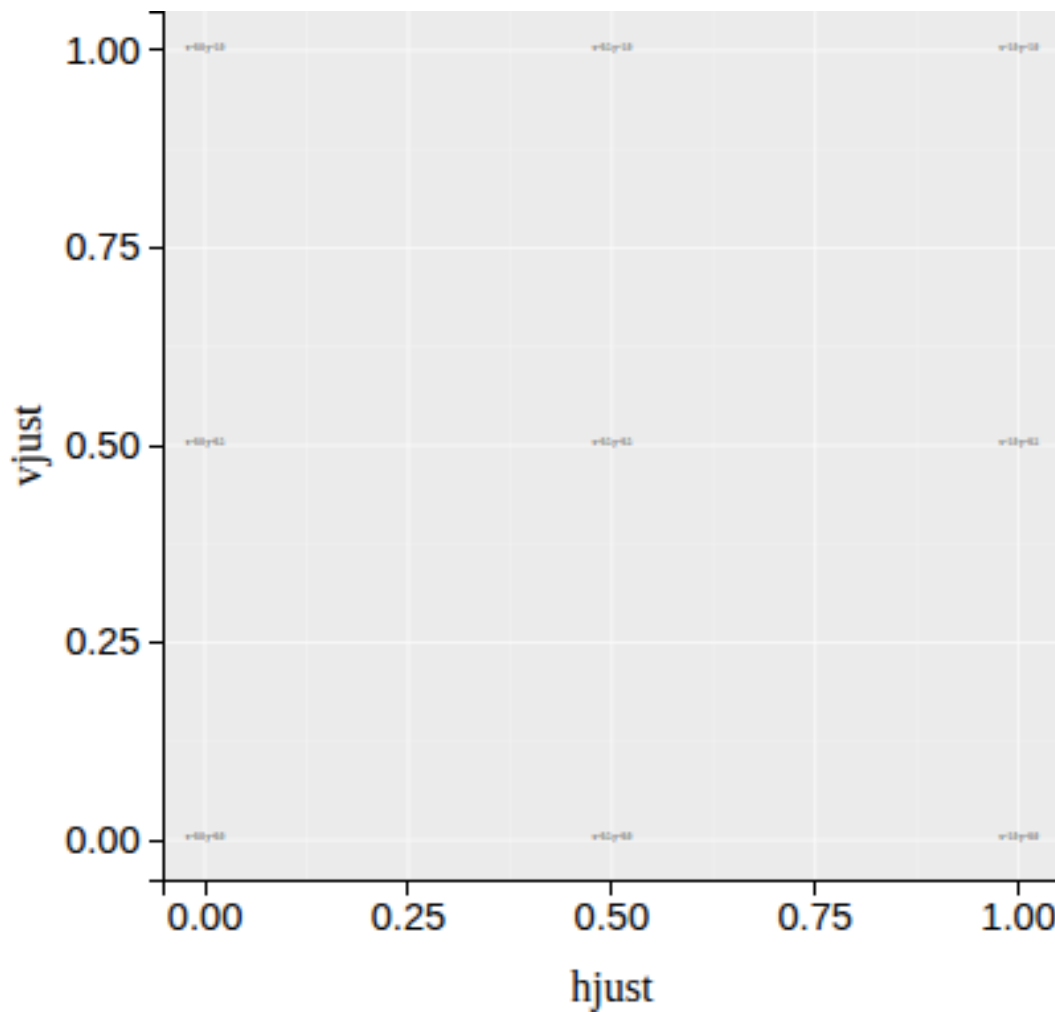
When rendering a ggplot using regular R graphics devices, using a line break or newline `\n` in a `geom_text` label results in multiple lines of text on the plot.

```
gg.return <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f\ny=%.1f", hjust, vjust)),
    size=3,
    data=text.df)
gg.return
```



However, animint only supports drawing the first line of text.

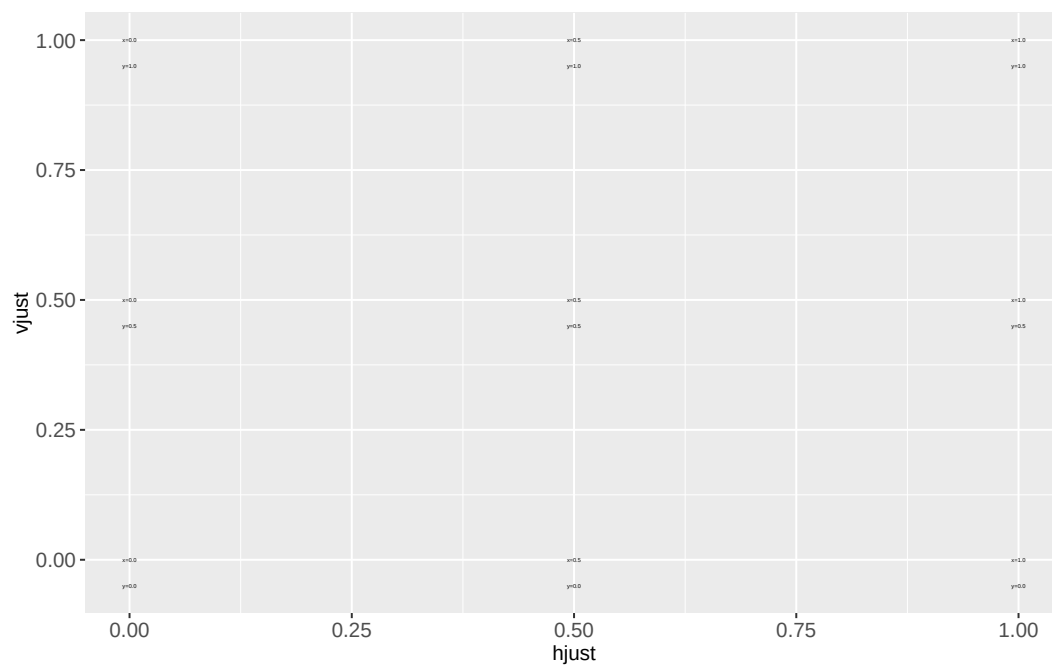
```
animint(gg.return)
```



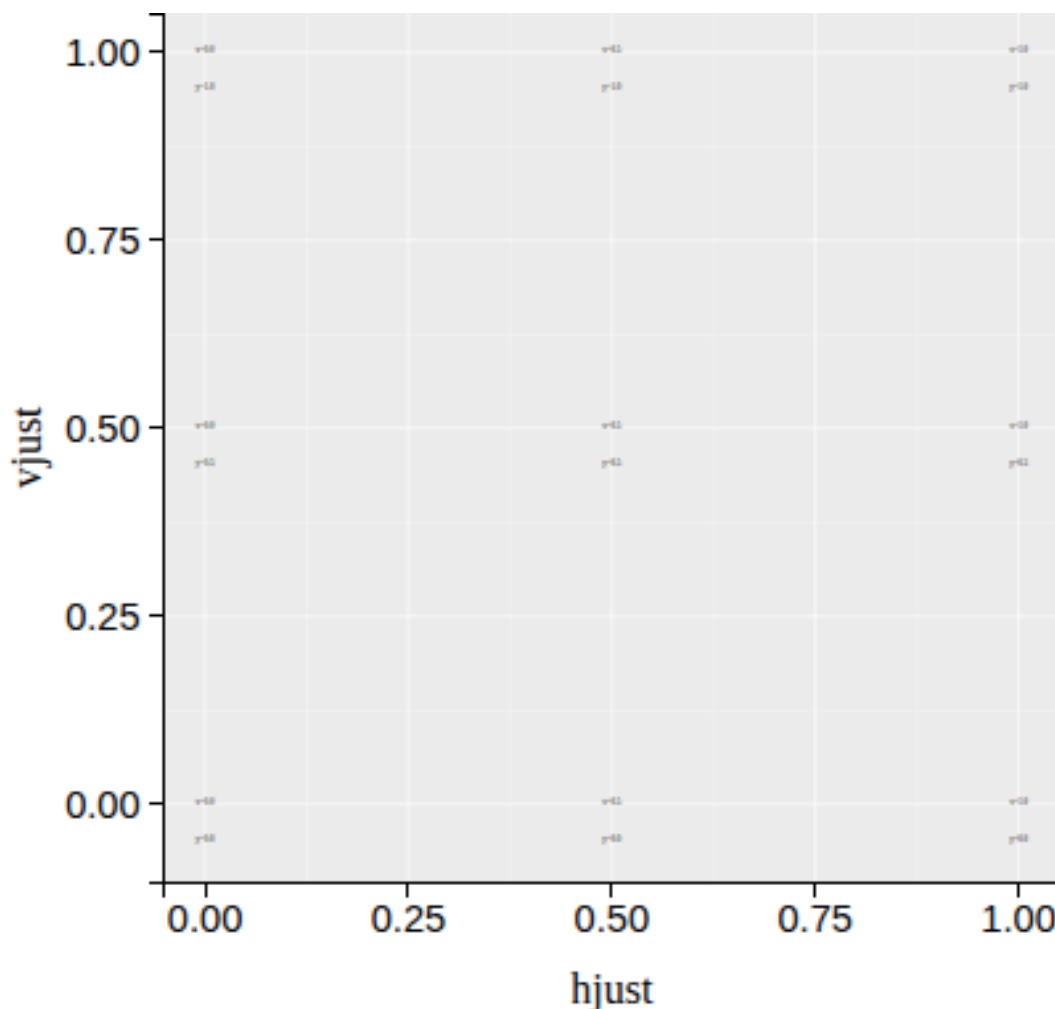
It would be nice to support multiple lines in `geom_text` labels, but we have not yet had time to implement that. However, we have [an issue](#), and would be willing to accept a Pull Request which implements that.

Until then, the workaround is to use one `geom_text` layer for each line of text that you want to display:

```
gg.two.lines <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f", hjust)),
    size=3,
    data=text.df)+
  geom_text(aes(
    hjust, vjust-0.05, label=sprintf("y=%.1f", vjust)),
    size=3,
    data=text.df)
gg.two.lines
```

```
animint(gg.two.lines)
```



7.7 Avoid some ggplot theme options

One goal of *animint* is to support [all of the theme options](#), but we have not yet had time to implement them all. If there is a theme option that you use and *animint* does not yet support, then please send us a Pull Request. The following list documents all the **theme** options that *animint* currently supports.

- **panel.margin** designates the distance between panels, and is used in the space saving facets idiom to eliminate the distance between panels.
- **panel.grid.major** is used to draw the the grid lines which are designated by the **breaks** argument to the scale.
- **panel.grid.minor** is used to draw the grid lines between the major grid lines.
- **panel.background** is used for the **<rect>** in background of each panel.
- **panel.border** is used for the border **<rect>** of each panel (on top of the background **<rect>**).
- **legend.position="none"** works for hiding all of the legends, but none of the other

legend positions are supported (the legends always appear on the right of the plot).

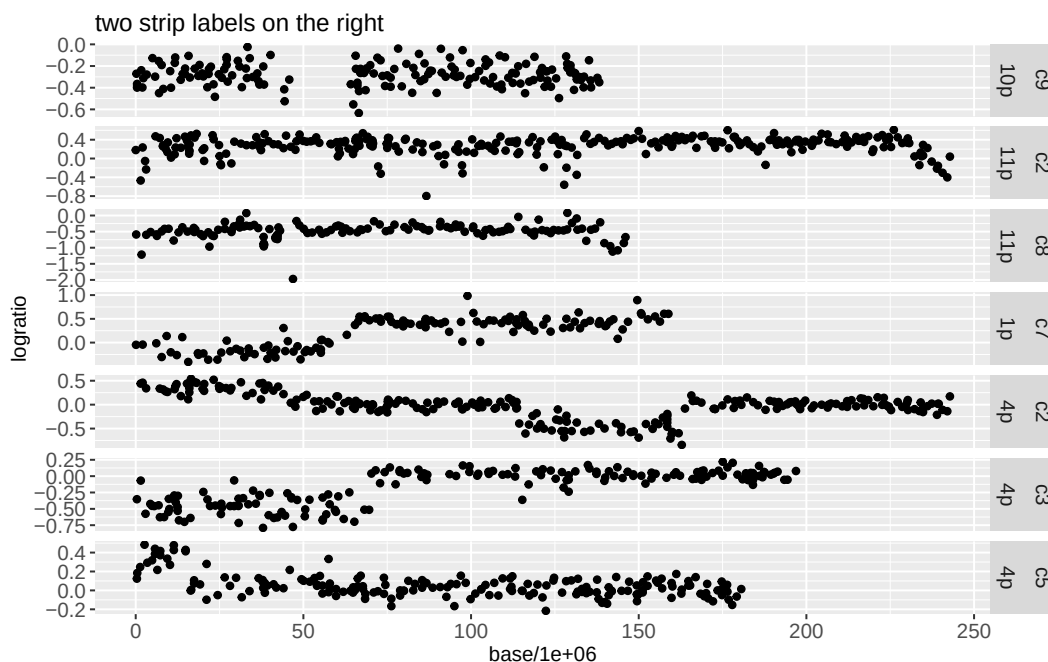
The following theme options can be set to `element_blank` to hide the axes.

- `axis.title`, `axis.title.x`, `axis.title.y` designate the axis title.
- `axis.ticks`, `axis.ticks.x`, `axis.ticks.y` designate the axis ticks.
- `axis.line`, `axis.line.x`, `axis.line.y` designate the axis line.
- `axis.text`, `axis.text.x`, `axis.text.y` designate the axis tick label text, and support the `angle` and `hjust` arguments of `element_text`.

7.8 Facets with multiple variables per axis

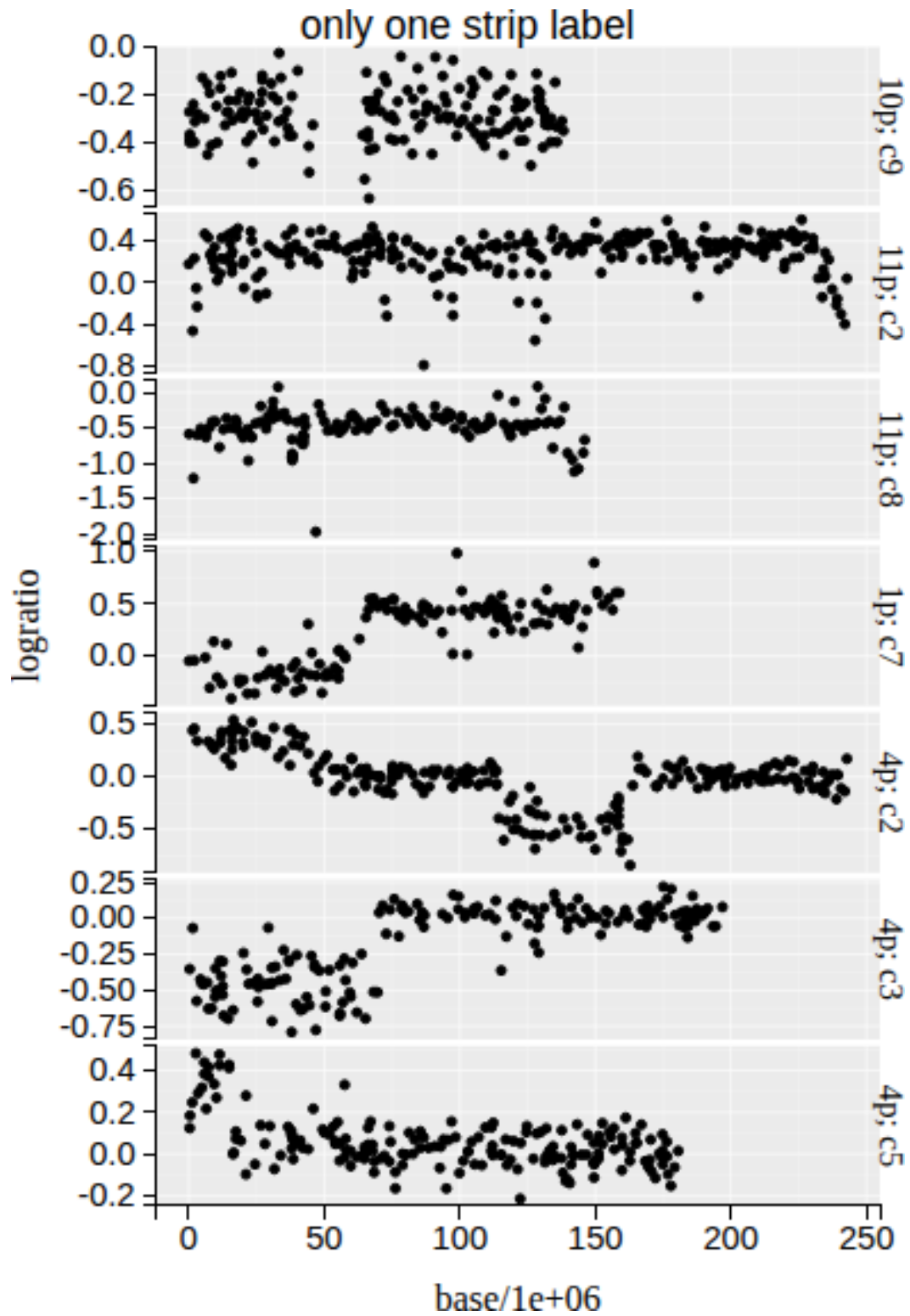
Animint supports `facet_grid` for creating multi-panel data visualizations, but only has limited support for multiple variables per axis. For example the ggplot below uses two variables to create vertical facets, which results in two strip labels when rendered with `ggplot2`.

```
data(intreg)
signals.df <- transform(
  intreg$signals,
  person=sub("[.]*", "p", signal),
  chromosome=sub(".*[.]", "c", signal))
two.strips <- ggplot()+
  theme_animint(height=600)+
  facet_grid(person + chromosome ~ ., scales="free")+
  geom_point(aes(base/1e6, logratio), data=signals.df)
two.strips+ggtitle("two strip labels on the right")
```



In contrast, animint renders the same ggplot below using only one strip label.

```
animint(two.strips+ggtitle("only one strip label"))
```



It would be nice to support multiple strip labels per axis, but we have not yet had time to implement it. If you would like to implement this feature, we would be happy to accept your

Pull Request.

7.9 Interactive definition of aesthetic mappings using shiny

Normally, aesthetic mappings are defined once in R code, and can not be changed after rendering an animint. One way to overcome this limitation is by defining `shiny` inputs that are used as animint aesthetics, as in the following example.

```
shiny::runApp(system.file(
  "examples", "shiny-WorldBank", package="animint2"))
```

7.10 Interactive computation

Another limitation is that animint can only display data that can be computed and stored in a data table before creating the visualization. This means that animint is not appropriate when there are more subsets of data to plot than you could ever compute. In that case, it would be better to use shiny.

7.11 Chapter summary and exercises

We discussed limitations of the current implementation of animint2, and explained several workarounds.

Exercises:

- Make a faceted ggplot with `stat_bin` that will not work with animint when the facet variable is instead used as a `showSelected` variable. Compute the stat yourself for each facet, and use `stat_identity` to make it work with animint.
- Make a ggplot which displays fine using `facet_grid(. <!-- comment --> ~ var, scales="free")` but does not display well in animint with `showSelected=var`. To fix the problem, compute a normalized version of `var` and use that for the `showSelected` variable.

Next, [Chapter 8](#) explains how to create a multi-panel visualization of the World Bank data.

Part II

Exemples avancés



8

World Bank data viz

In this chapter we will explore several data visualizations of the World Bank data set.

Chapter outline:

- We begin by loading the World Bank data set and defining some helper functions for creating a multi-panel ggplot with several geoms.
- We then create a time series plot for life expectancy.
- We then add a scatterplot of life expectancy versus fertility rate as a second panel.
- We then add a third panel with a time series for fertility rate.

8.1 Load data and define helper functions

First we load the WorldBank data set, and consider only the subset which has both non-missing values for both life.expectancy and fertility.rate.

```
library(animint2)
data(WorldBank)
WorldBank$Region <- sub(" (all income levels)", "", WorldBank$region, fixed=TRUE)
library(data.table)
not.na <- data.table(
  WorldBank[!(is.na(life.expectancy) | is.na(fertility.rate))])
```

We will also be plotting the population variable using a size legend. Before plotting, we will make sure that none of the values are missing.

```
not.na[is.na(not.na$population)]
```

```
   iso2c country year fertility.rate life.expectancy population
1:    KW  Kuwait 1992          2.338         72.95266         NA
2:    KW  Kuwait 1993          2.341         73.07373         NA
3:    KW  Kuwait 1994          2.413         73.18724         NA
10 variables not shown: [GDP.per.capita.Current.USD, 15.to.25.yr.female.literacy, iso3c, region, c
```

The table above shows that there are three rows with missing values for the population variable. They are for the country Kuwait during 1992-1994. The table below shows the data from the neighboring years, 1991-1995.

```
not.na[country == "Kuwait" & 1991 <= year & year <= 1995]
```

```

iso2c country year fertility.rate life.expectancy population
1:   KW  Kuwait 1991          2.418          72.82254    1999651
2:   KW  Kuwait 1992          2.338          72.95266         NA
3:   KW  Kuwait 1993          2.341          73.07373         NA
4:   KW  Kuwait 1994          2.413          73.18724         NA
5:   KW  Kuwait 1995          2.525          73.29422    1586123
10 variables not shown: [GDP.per.capita.Current.USD, 15.to.25.yr.female.literacy, iso3c, region, c

```

The table above shows that the population of Kuwait decreased over the period 1991-1995, consistent with the Gulf War of that time period. We fill in those missing values below.

```

not.na[is.na(population), population := 1700000]
not.na[country == "Kuwait" & 1991 <= year & year <= 1995]

```

```

iso2c country year fertility.rate life.expectancy population
1:   KW  Kuwait 1991          2.418          72.82254    1999651
2:   KW  Kuwait 1992          2.338          72.95266    1700000
3:   KW  Kuwait 1993          2.341          73.07373    1700000
4:   KW  Kuwait 1994          2.413          73.18724    1700000
5:   KW  Kuwait 1995          2.525          73.29422    1586123
10 variables not shown: [GDP.per.capita.Current.USD, 15.to.25.yr.female.literacy, iso3c, region, c

```

Next, we define the following helper function, which will be used to add columns to data sets in order to assign geoms to facets.

```

FACETS <- function(df, top, side){
  data.frame(df,
             top=factor(top, c("Fertility rate", "Years")),
             side=factor(side, c("Years", "Life expectancy")))
}

```

Note that the factor levels will specify the order of the facets in the ggplot. This is an example of the addColumn then facet idiom . Below, we define three helper functions, one for each facet.

```

TS.RIGHT <- function(df)FACETS(df, "Years", "Life expectancy")
SCATTER <- function(df)FACETS(df, "Fertility rate", "Life expectancy")
TS.ABOVE <- function(df)FACETS(df, "Fertility rate", "Years")

```

8.2 First time series plot

First we define a data set with one row for each year, which we will use for selecting years using a `geom_tallrect` in the background.

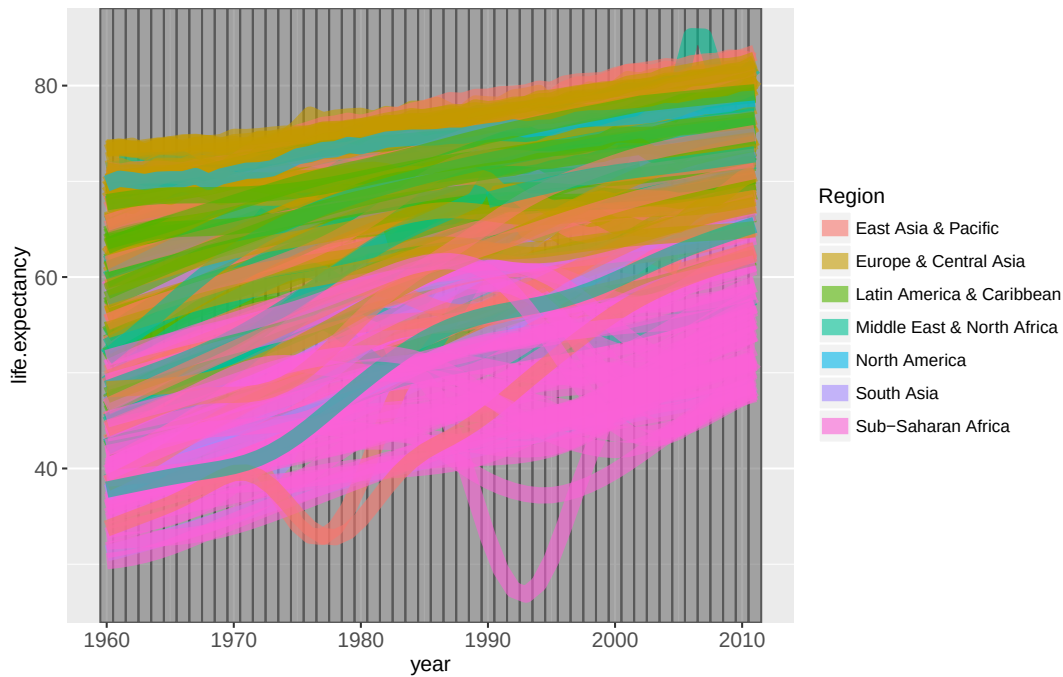
```

years <- unique(not.na[, .(year)])

```

We define the ggplot with a `geom_tallrect` in the background, and a `geom_line` for the time series.

```
ts.right <- ggplot()+
  geom_tallrect(aes(
    xmin=year-1/2, xmax=year+1/2),
    clickSelects="year",
    data=TS.RIGHT(years), alpha=1/2)+
  geom_line(aes(
    year, life.expectancy, group=country, colour=Region),
    clickSelects="country",
    data=TS.RIGHT(not.na), size=4, alpha=3/5)
ts.right
```



Note that we specified `clickSelects=year` so that clicking a tallrect will change the selected year, and `clickSelects=country` so that clicking a line will select or de-select a country. Also note that we used `TS.RIGHT` to specify columns that we will use in the facet specification (next section).

8.3 Add a scatterplot facet

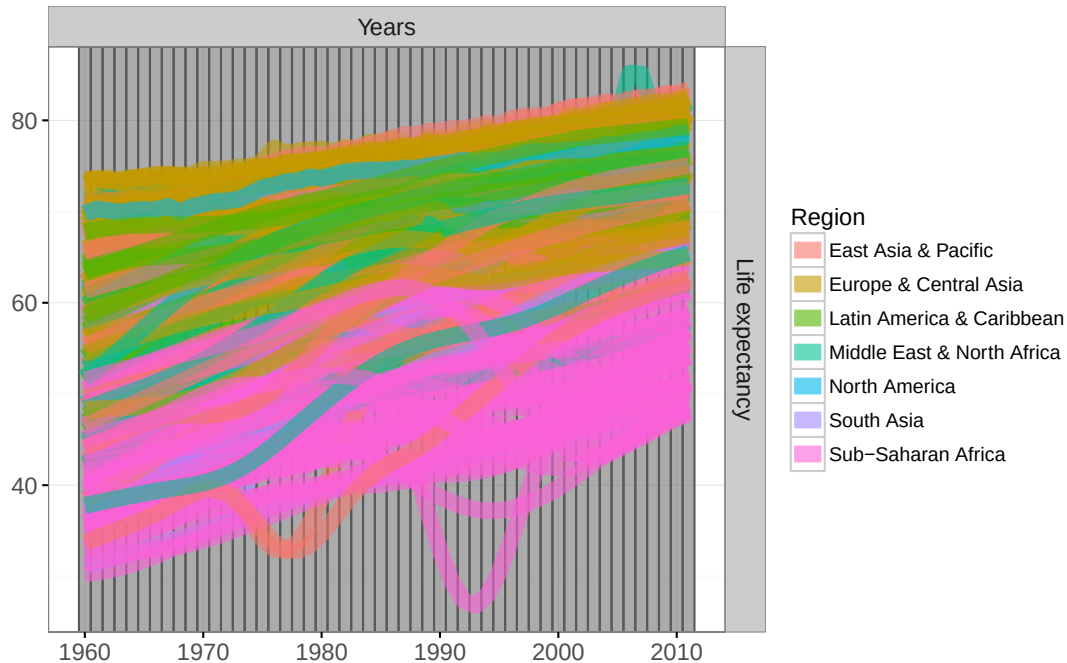
We begin by simply adding facets to the previous time series plot.

```
ts.facet <- ts.right+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(side ~ top, scales="free")+
```

```

xlab("")+
ylab("")
ts.facet

```

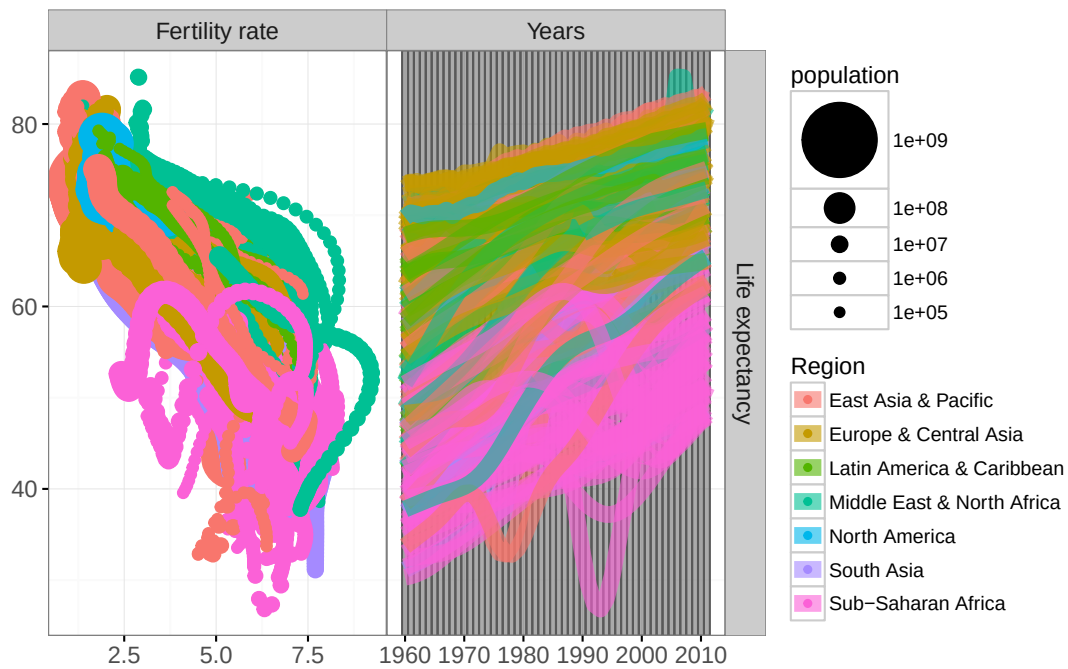


We set the `panel.margin` to 0, which is always a good idea to save space in a ggplot with facets. We use `scales="free"` and hide the axis labels, in an example of the `addColumn` then `facet` idiom. Instead, we use the facet label to show the variable encoded on each axis. Below, we add a scatterplot facet with a point for each year and country.

```

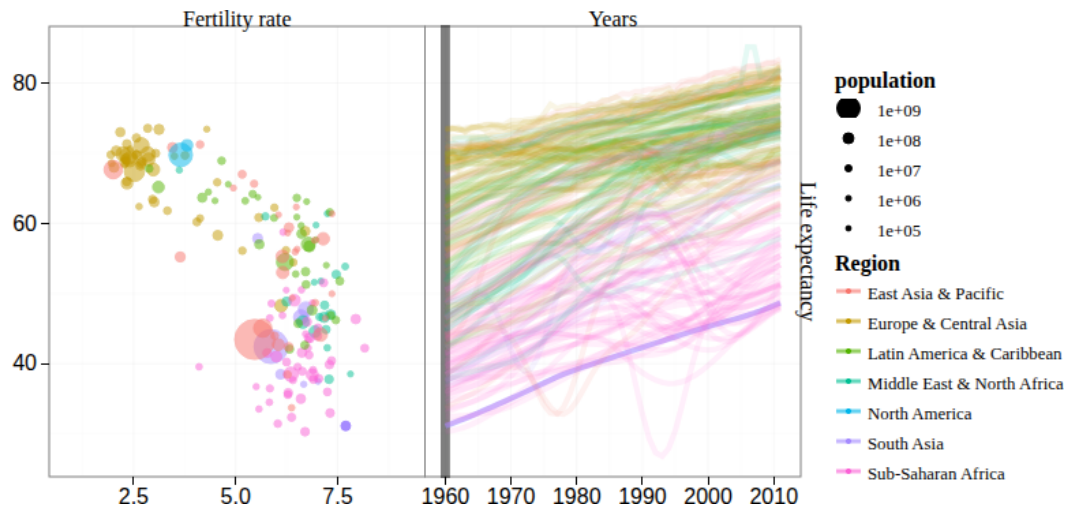
ts.scatter <- ts.facet+
  theme_animint(width=600)+
  geom_point(aes(
    fertility.rate, life.expectancy,
    colour=Region, size=population,
    key=country), # key aesthetic for animated transitions!
  clickSelects="country",
  showSelected="year",
  data=SCATTER(not.na))+
  scale_size_animint(pixel.range=c(2, 20), breaks=10^(9:5))
ts.scatter

```



Note how we use `scale_size_animint` to specify the range of sizes in pixels, and the breaks in the legend. Also note that we use `SCATTER` to specify `top` and `side` columns which are used in the facet specification. We also render this ggplot interactively below.

```
animint(ts.scatter)
```

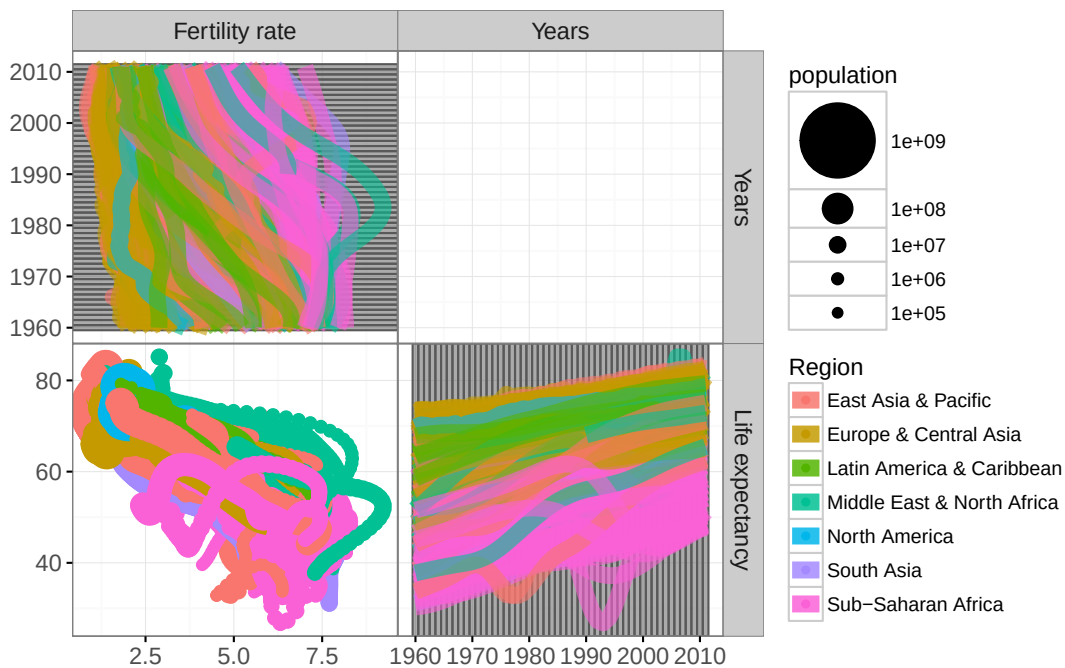


Note that single selection is used by default for both year and country.

8.4 Adding another time series facet

Below we add `widirects` for selecting years, and paths for showing fertility rate.

```
scatter.both <- ts.scatter+
  geom_widirect(aes(
    ymin=year-1/2, ymax=year+1/2),
    clickSelects="year",
    data=TS.ABOVE(years), alpha=1/2)+
  geom_path(aes(
    fertility.rate, year, group=country, colour=Region),
    clickSelects="country",
    data=TS.ABOVE(not.na), size=4, alpha=3/5)
scatter.both
```



Note that `TS.ABOVE` was used to specify facet columns `top` and `side`. We render an interactive version below.

```
viz.scatter.both <- animint(
  title="World Bank data (multiple selection, facets)",
  scatterBoth=scatter.both+
    theme_animint(width=1000, height=800),
  duration=list(year=1000),
  time=list(variable="year", ms=3000),
  first=list(year=1975, country=c("United States", "Vietnam")),
  selector.types=list(country="multiple"))
```

8.5 Chapter summary and exercises

We showed how to create a multi-layer, multi-panel (but single-plot) visualization of the World Bank data.

Exercises:

- Add a points on each time series plot, with size proportional to population as in the scatterplot. The points should appear only when the country is selected, and clicking the points should de-select that country.
- Add text labels to the time series plot on the right, with names for each country. Each label should appear only when the country is selected, and should disappear after clicking on the label.
- Add a text label to the scatterplot to indicate the selected year.
- Add text labels to the scatterplot, with names for each country. Each label should appear only when the country is selected, and should disappear after clicking on the label.

Next, [Chapter 9](#) explains how to visualize the Montreal bike data set.



9

Montreal bikes data viz

In this chapter we will explore several data visualizations of the Montreal bike data set.

Chapter outline:

- We begin with some static data visualizations.
- We create an interactive visualization of accident frequency over time.
- We create a interactive data viz with four plots, showing monthly accident trends, daily details, and a map of counter locations.

9.1 Static figures

We begin by loading the `montreal.bikes` data set, which is not available in the CRAN release of `animint2`, in order to save space on CRAN. Therefore to access this data set, you will need to install `animint2` from GitHub:

```
tryCatch({
  data(montreal.bikes, package="animint2")
}, warning=function(w){
  devtools::install_github("tdhock/animint2")
})
```

We begin by examining the accidents data table.

```
library(animint2)
library(dplyr)
```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

`filter`, `lag`

The following objects are masked from 'package:base':

`intersect`, `setdiff`, `setequal`, `union`

```
data(montreal.bikes) #only present if installed from github
Sys.setlocale("LC_ALL", "")
```

```
[1] "LC_CTYPE=C.UTF-8;LC_NUMERIC=C;LC_TIME=C.UTF-8;LC_COLLATE=C.UTF-
8;LC_MONETARY=C.UTF-8;LC_MESSAGES=C.UTF-8;LC_PAPER=C.UTF-
8;LC_NAME=C;LC_ADDRESS=C;LC_TELEPHONE=C;LC_MEASUREMENT=C.UTF-
8;LC_IDENTIFICATION=C"
```

```
montreal.bikes$counter.locations <- montreal.bikes$counter.locations %>%
  mutate(across(where(is.character), ~ iconv(.x, from = "latin1", to = "UTF-8")))

print(table(montreal.bikes$counter.counts$location))
```

Berri	Brébeuf	Côte-Sainte-Catherine
1666	730	1666
Maisonneuve 1	Maisonneuve 2	du Parc
1666	1666	1666
Pierre-Dupuy	Rachel	Saint-Urbain
1666	1666	730
Totem_Laurier		
261		

```
library(data.table)
```

Attaching package: 'data.table'

The following objects are masked from 'package:dplyr':

between, first, last

```
accidents.dt <- data.table(montreal.bikes$accidents)
str(accidents.dt)
```

Classes 'data.table' and 'data.frame': 5595 obs. of 12 variables:

```
$ date.str      : chr  "2012-01-02" "2012-01-05" "2012-01-
09" "2012-01-10" ...
```

```
$ time.str      : chr  "18:35" "21:50" "21:15" "15:40" ...
```

```
$ deaths        : int   0 0 0 0 0 0 0 1 0 0 ...
```

```
$ people.severely.injured: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
$ people.slightly.injured: int  1 1 1 1 1 1 1 0 1 1 ...
```

```
$ street.number  : int   NA NA NA NA NA 2330 NA NA 4160 NA ...
```

```
$ street         : chr   "ST JEAN BAPTISTE 0" "FOSTER" "ROSEMONT" "ST ANTOINE" ...
```

```
$ cross.street   : chr   "AV ROULEAU" "JANELLE" "DES ERABLES" "MANSFIELD" ...
```

```
$ location.int   : int   32 34 NA 32 32 34 32 32 33 31 ...
```

```
$ position.int   : int   6 6 6 6 6 6 6 6 6 5 ...
```

```
$ position       : chr   "Voie de circulation" "Voie de circulation" "Voie de circulation"
```

```
$ location       : chr   "En intersection (moins de 5 mètres)" "Entre intersections (100 m
```

```
- attr(*, ".internal.selfref")=<externalptr>
```

Each accident has data about its date, time, location, and counts of death and slight/severe injury. Some of the values are in French (e.g. position Voie de circulation, location En intersection, etc).

We calculate the time period of the accidents below.

```
(accidents.dt[
, date.POSIXct := as.POSIXct(strptime(date.str, "%Y-%m-%d"))
][
, month.str := strptime(date.POSIXct, "%Y-%m")
][[])
```

	date.str	time.str	deaths	people.severely.injured
1:	2012-01-02	18:35	0	0
2:	2012-01-05	21:50	0	0
3:	2012-01-09	21:15	0	0
4:	2012-01-10	15:40	0	0
5:	2012-01-10	0:15	0	0

5591:	2014-12-19	12:22	0	0
5592:	2014-12-19	19:50	0	0
5593:	2014-12-26	19:56	0	0
5594:	2014-12-27	12:35	0	0
5595:	2014-12-30	11:55	0	0

10 variables not shown: [people.slightly.injured, street.number, street, cross.street, location.in]

```
range(accidents.dt$month.str)
```

```
[1] "2012-01" "2014-12"
```

Below we also compute the range of months for the bike counter data table.

```
(counts.dt <- data.table(montreal.bikes$counter.counts))
```

	location	date	count
1:	Berri	2009-01-01 05:00:00	29
2:	Berri	2009-01-02 05:00:00	19
3:	Berri	2009-01-03 05:00:00	24
4:	Berri	2009-01-04 05:00:00	24
5:	Berri	2009-01-05 05:00:00	120

13379:	Totem_Laurier	2013-09-14 04:00:00	2456
13380:	Totem_Laurier	2013-09-15 04:00:00	2527
13381:	Totem_Laurier	2013-09-16 04:00:00	3012
13382:	Totem_Laurier	2013-09-17 04:00:00	3745
13383:	Totem_Laurier	2013-09-18 04:00:00	3921

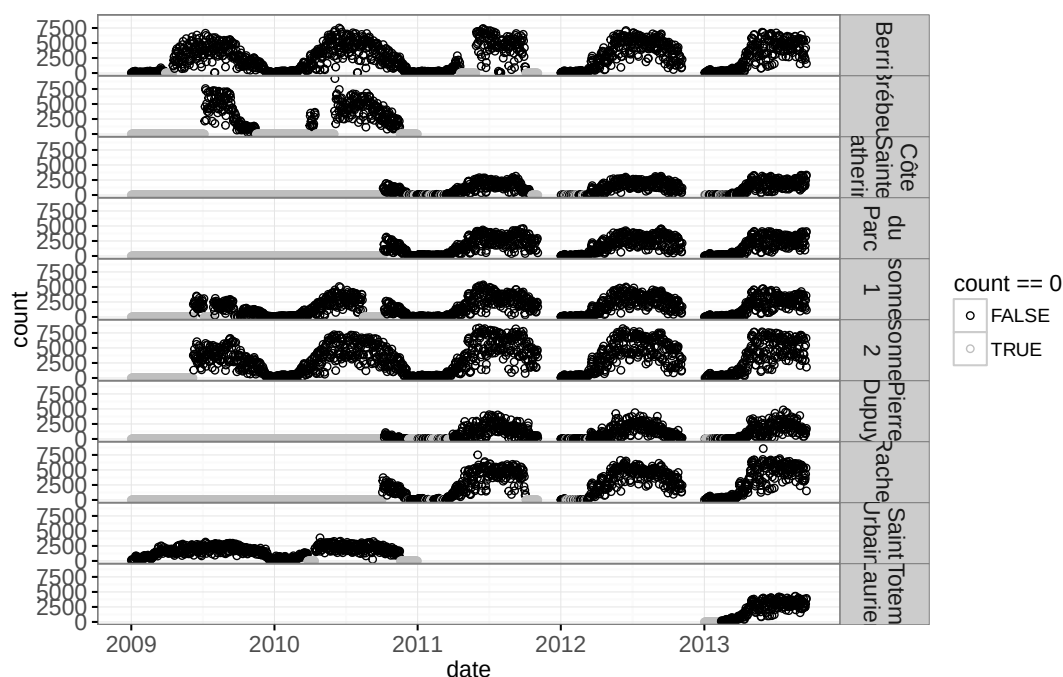
```
counts.dt[, month.str := strptime(date, "%Y-%m")]
range(counts.dt$month.str)
```

```
[1] "2009-01" "2013-09"
```

The bike counts are time series data which we visualize below.

```
counts.dt[, loc.lines := gsub("[-_]", "\n", location)]
ggplot()+
```

Warning: Removed 407 rows containing missing values (geom_point).



We will compute a summary of all accidents per month in this time period, so we first create a data table for each month below. (and make sure to set the locale to C for English month names)

```
uniq.month.vec <- unique(c(
  accidents.dt$month.str,
  counts.dt$month.str))
one.day <- 60 * 60 * 24
months <- data.table(month.str=uniq.month.vec)[
, month01.str := paste0(month.str, "-01")
][
, month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
][, let(
  next.POSIXct = month01.POSIXct + one.day * 31,
  month.str = strftime(month01.POSIXct, "%B %Y")

```

```

)] [
, next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
] [
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
]
month.levs <- months[order(month01.POSIXct), month.str]
(months[, month := factor(month.str, month.levs)][])

```

```

      month.str month01.str month01.POSIXct next.POSIXct next01.str
1:   January 2012  2012-01-01    2012-01-01  2012-02-01 2012-02-01
2:  February 2012  2012-02-01    2012-02-01  2012-03-03 2012-03-01
3:    March 2012  2012-03-01    2012-03-01  2012-04-01 2012-04-01
4:    April 2012  2012-04-01    2012-04-01  2012-05-02 2012-05-01
5:     May 2012  2012-05-01    2012-05-01  2012-06-01 2012-06-01
---
68:   August 2011  2011-08-01    2011-08-01  2011-09-01 2011-09-01
69: September 2011  2011-09-01    2011-09-01  2011-10-02 2011-10-01
70:  October 2011  2011-10-01    2011-10-01  2011-11-01 2011-11-01
71: November 2011  2011-11-01    2011-11-01  2011-12-02 2011-12-01
72: December 2011  2011-12-01    2011-12-01  2012-01-01 2012-01-01
2 variables not shown: [next01.POSIXct, month]

```

Note that we created a `month` column which is a factor ordered by `month.levs`.

```

(accidents.dt[
, month.text := strftime(date.POSIXct, "%B %Y")
] [
, month := factor(month.text, month.levs)
] [
, month.POSIXct := as.POSIXct(strptime(paste0(month.str, "-15"), "%Y-%m-%d"))
] [])

```

```

      date.str time.str deaths people.severely.injured
1: 2012-01-02   18:35      0                      0
2: 2012-01-05   21:50      0                      0
3: 2012-01-09   21:15      0                      0
4: 2012-01-10   15:40      0                      0
5: 2012-01-10    0:15      0                      0
---
5591: 2014-12-19   12:22      0                      0
5592: 2014-12-19   19:50      0                      0
5593: 2014-12-26   19:56      0                      0
5594: 2014-12-27   12:35      0                      0
5595: 2014-12-30   11:55      0                      0
13 variables not shown: [people.slightly.injured, street.number, street, cross.street, location.in]

```

```

stopifnot(!is.na(accidents.dt$month.POSIXct))
(accidents.per.month <- accidents.dt[, list(
  total.accidents=.N,
  total.people=sum(deaths+people.severely.injured+people.slightly.injured),

```

```

deaths=sum(deaths),
people.severely.injured=sum(people.severely.injured),
people.slightly.injured=sum(people.slightly.injured),
next.POSIXct = month.POSIXct + one.day * 30,
month01.str = paste0(strftime(month.POSIXct, "%Y-%m"), "-01")
), by=(month, month.str, month.text, month.POSIXct)][, let(
  month01.POSIXct = as.POSIXct(strptime(month01.str, "%Y-%m-%d")),
  next01.str = paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
)]
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
][[])

```

	month	month.str	month.text	month.POSIXct	total.accidents	
1:	January	2012	2012-01	January 2012	2012-01-15	11
2:	February	2012	2012-02	February 2012	2012-02-15	19
3:	March	2012	2012-03	March 2012	2012-03-15	76
4:	April	2012	2012-04	April 2012	2012-04-15	113
5:	May	2012	2012-05	May 2012	2012-05-15	224

32:	August	2014	2014-08	August 2014	2014-08-15	308
33:	September	2014	2014-09	September 2014	2014-09-15	279
34:	October	2014	2014-10	October 2014	2014-10-15	166
35:	November	2014	2014-11	November 2014	2014-11-15	71
36:	December	2014	2014-12	December 2014	2014-12-15	10

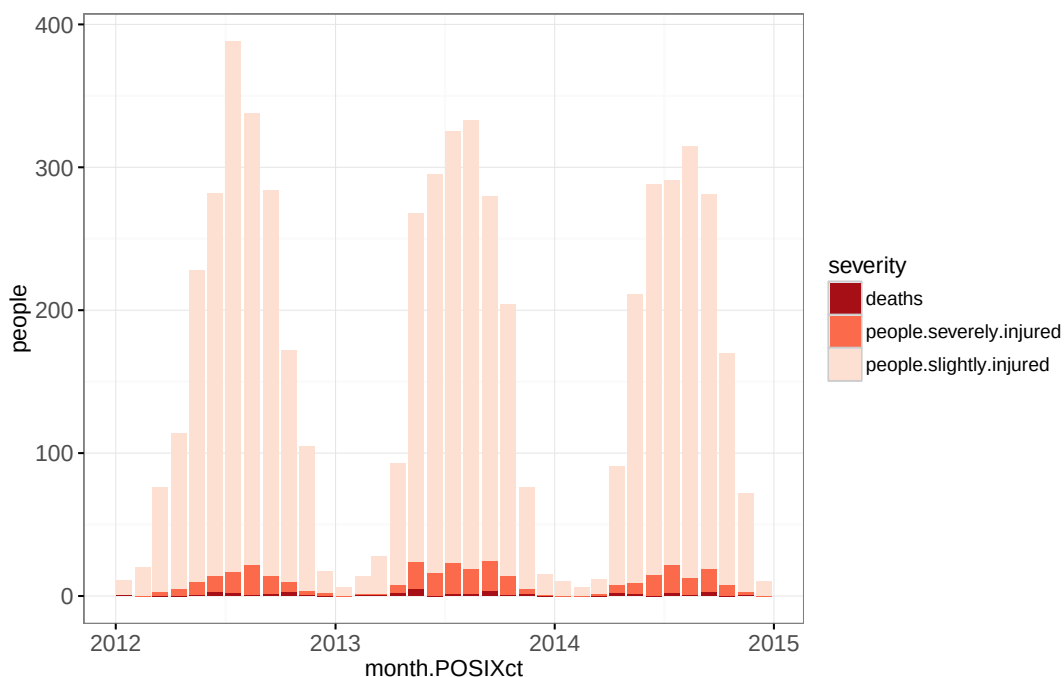
9 variables not shown: [total.people, deaths, people.severely.injured, people.slightly.injured, ne

We plot the accidents per month below.

```

accidents.tall <- melt(
  accidents.per.month,
  measure.vars=c(
    "deaths", "people.severely.injured", "people.slightly.injured"),
  variable.name="severity",
  value.name="people")
severity.colors <- c(
  "people.slightly.injured"="#FEE0D2",#lite red
  "people.severely.injured"="#FB6A4A",
  deaths="#A50F15")#dark red
ggplot()+
  theme_bw()+
  geom_bar(aes(
    month.POSIXct, people, fill=severity),
  stat="identity",
  data=accidents.tall)+
  scale_fill_manual(values=severity.colors)

```



In each accident, there are counts of people who died, along with people who suffered severe and slight injuries. Below we classify the severity of each accident according to the worst outcome among the people affected.

```
accidents.dt[
, severity.str := fcase(
  0 < deaths, "deaths",
  0 < people.severely.injured, "people.severely.injured",
  default="people.slightly.injured")
][
, severity := factor(severity.str, names(severity.colors))
][
, table(severity)
]
```

```
severity
people.slightly.injured people.severely.injured deaths
                    5262                    289         44
```

The output above shows that accidents with only slight injuries are most frequent, and accidents with at least one death are least frequent. Below we compute counts per month.

```
(counts.per.month <- counts.dt[, let(
  month.POSIXct = as.POSIXct(strptime(paste0(month.str, "-15"), "%Y-%m-%d")),
  month.text = strftime(date, "%B %Y"),
  day.of.the.month = as.integer(strftime(date, "%d"))
)])[
, month := factor(month.text, month.levs)
][, list(
```

```

    days=.N,
    mean.per.day=mean(count),
    count=sum(count),
    month01.str = paste0(month.str, "-01")
  ), by=.(location, month, month.str, month.POSIXct)][
    0 < count
  ][
    , month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
  ][
    , next.POSIXct := month01.POSIXct + one.day * 31
  ][
    , next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
  ][
    , next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
  ][
    , days.in.month := as.integer(round(difftime(next01.POSIXct, month01.POSIXct, units="days")))
  ][])

```

	location	month	month.str	month.POSIXct	days	mean.per.day
1:	Berri	January 2009	2009-01	2009-01-15	31	100.3226
2:	Berri	February 2009	2009-02	2009-02-15	28	159.6786
3:	Berri	March 2009	2009-03	2009-03-15	31	271.3226
4:	Berri	May 2009	2009-05	2009-05-15	31	2972.8710
5:	Berri	June 2009	2009-06	2009-06-15	30	3909.9333

```

---
321: Totem_Laurier      May 2013    2013-05    2013-05-15    31    2746.4194
322: Totem_Laurier      June 2013   2013-06    2013-06-15    30    2828.6000
323: Totem_Laurier      July 2013   2013-07    2013-07-15    31    3238.3871
324: Totem_Laurier      August 2013 2013-08    2013-08-15    31    3162.7097
325: Totem_Laurier      September 2013 2013-09    2013-09-15    18    2888.7778

```

7 variables not shown: [count, month01.str, month01.POSIXct, next.POSIXct, next01.str, next01.POSIXct]

```

counts.per.month[days < days.in.month, {
  list(location, month, days, days.in.month)
}]

```

	location	month	days	days.in.month
1:	Berri	November 2012	5	30
2:	Côte-Sainte-Catherine	November 2012	5	30
3:	Maisonnette 1	November 2012	5	30
4:	Maisonnette 2	November 2012	5	30
5:	du Parc	November 2012	5	30

11:	Maisonnette 2	September 2013	18	30
12:	du Parc	September 2013	18	30
13:	Pierre-Dupuy	September 2013	18	30
14:	Rachel	September 2013	18	30
15:	Totem_Laurier	September 2013	18	30

As shown above, some months do not have observations for all days.

9.2 Interactive viz of accident frequency

```
complete.months <- counts.per.month[days == days.in.month]
month.labels <- counts.per.month[, {
  .SD[which.max(count), ]
}, by=location]
day.labels <- counts.dt[, {
  .SD[which.max(count), ]
}, by=(location, month)]
(city.wide.cyclists <- counts.per.month[0 < count, list(
  locations=.N,
  count=sum(count),
  month01.str = paste0(month.str, "-01")
), by=(month, month.str, month.POSIXct)])[
, month01.POSIXct := as.POSIXct(strptime(month01.str, "%Y-%m-%d"))
][
, next.POSIXct := month01.POSIXct + one.day * 31
][
, next01.str := paste0(strftime(next.POSIXct, "%Y-%m"), "-01")
][
, next01.POSIXct := as.POSIXct(strptime(next01.str, "%Y-%m-%d"))
][[])
```

	month	month.str	month.POSIXct	locations	count	month01.str
1:	January	2009	2009-01	2009-01-15	2 14245	2009-01-01
2:	February	2009	2009-02	2009-02-15	2 24002	2009-02-01
3:	March	2009	2009-03	2009-03-15	2 57980	2009-03-01
4:	May	2009	2009-05	2009-05-15	2 149327	2009-05-01
5:	June	2009	2009-06	2009-06-15	4 305555	2009-06-01

50:	May	2013	2013-05	2013-05-15	8 811126	2013-05-01
51:	June	2013	2013-06	2013-06-15	8 767951	2013-06-01
52:	July	2013	2013-07	2013-07-15	8 916662	2013-07-01
53:	August	2013	2013-08	2013-08-15	8 856066	2013-08-01
54:	September	2013	2013-09	2013-09-15	8 450513	2013-09-01

4 variables not shown: [month01.POSIXct, next.POSIXct, next01.str, next01.POSIXct]

```
month.str.vec <- strptime(seq(
  strptime("2012-01-15", "%Y-%m-%d"),
  strptime("2013-01-15", "%Y-%m-%d"),
  by="month"), "%Y-%m")
city.wide.complete <- complete.months[0 < count, list(
  locations=.N,
  count=sum(count),
  month01.str = paste0(month.str, "-01")
), by=(month, month.str, month.POSIXct)]
setkey(city.wide.complete, month.str)
```

```

scatter.cyclists <- city.wide.complete[month.str.vec]
scatter.accidents <- accidents.per.month[scatter.cyclists, on=list(month.str)]
scatter.not.na <- scatter.accidents[!is.na(locations),]
scatter.max <- scatter.not.na[locations==max(locations)]
fit <- lm(total.accidents ~ count - 1, scatter.max)
scatter.max[
  , pred.accidents := predict(fit)
][
  , mean(total.accidents/count)
]

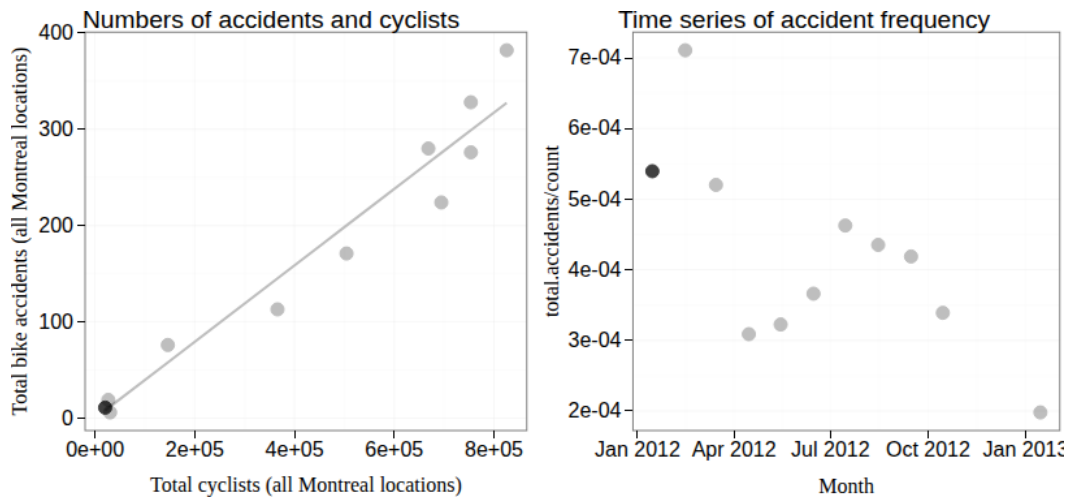
```

```
[1] 0.0004201563
```

```

animint(
  regression=ggplot()+
    theme_bw()+
    ggtitle("Numbers of accidents and cyclists")+
    geom_line(aes(
      count, pred.accidents,
      color="grey",
      data=scatter.max)+
    geom_point(aes(
      count, total.accidents,
      shape=1,
      clickSelects="month",
      size=5,
      alpha=0.75,
      data=scatter.max)+
    ylab("Total bike accidents (all Montreal locations)")+
    xlab("Total cyclists (all Montreal locations)"),
  timeSeries=ggplot()+
    theme_bw()+
    ggtitle("Time series of accident frequency")+
    xlab("Month")+
    geom_point(aes(
      month.POSIXct, total.accidents/count),
      clickSelects="month",
      size=5,
      alpha=0.75,
      data=scatter.max))

```



The data viz above shows two data visualizations of city-wide accident frequency over time. The plot on the left shows that the number of accidents grows with the number of cyclists. The plot on the right shows the frequency of accidents over time.

9.3 Interactive viz with map and details

The plot below is a dotplot of accidents for each month. Each dot represents one person who got in an accident.

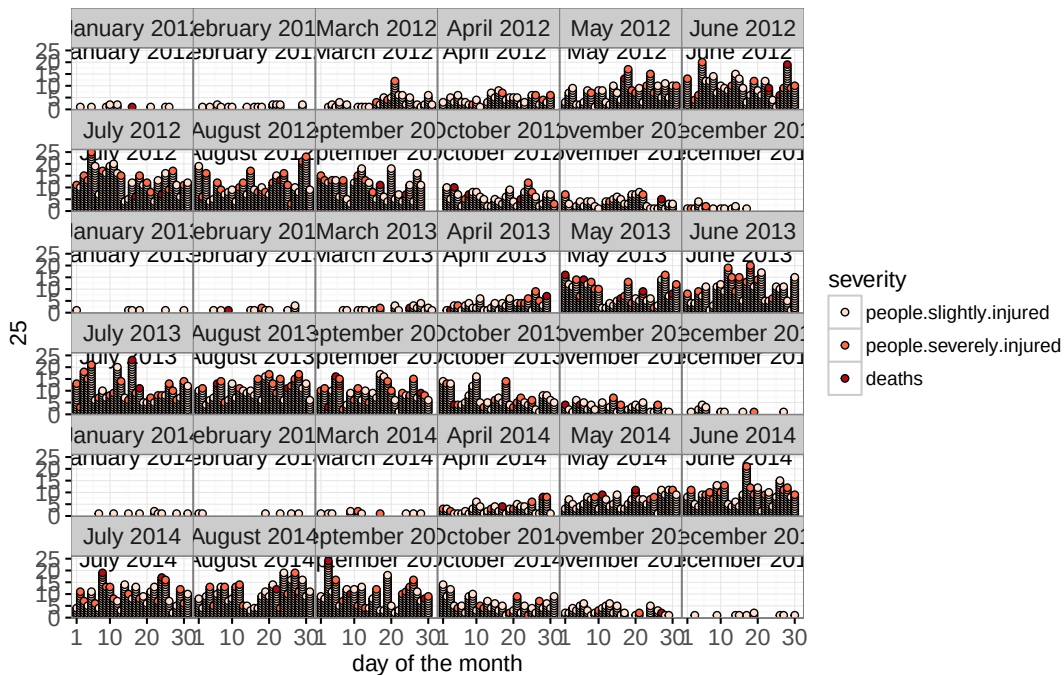
```
(accidents.cumsum <- accidents.dt[
  order(date.POSIXct, month, severity)
][
  , accident.i := seq_along(severity)
  , by=.(date.POSIXct, month)
][
  , day.of.the.month := as.integer(strftime(date.POSIXct, "%d"))
][
  ])
```

	date.str	time.str	deaths	people.severely.injured
1:	2012-01-02	18:35	0	0
2:	2012-01-05	21:50	0	0
3:	2012-01-09	21:15	0	0
4:	2012-01-10	15:40	0	0
5:	2012-01-10	0:15	0	0

5591:	2014-12-19	12:22	0	0
5592:	2014-12-19	19:50	0	0
5593:	2014-12-26	19:56	0	0
5594:	2014-12-27	12:35	0	0
5595:	2014-12-30	11:55	0	0

17 variables not shown: [people.slightly.injured, street.number, street, cross.street, location.in]

```
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_wrap("month")+
  geom_text(aes(15, 25, label=month), data=accidents.per.month)+
  scale_fill_manual(values=severity.colors)+
  scale_x_continuous("day of the month", breaks=c(1, 10, 20, 30))+
  geom_point(aes(
    day.of.the.month, accident.i, fill=severity),
    shape=21,
    data=accidents.cumsum)
```



```
(counter.locations <- data.table(montreal.bikes$counter.locations)[, let(
  lon = coord_X,
  lat = coord_Y
)])[[])
```

	id	nom	nom_comptage	Etat	Type
1:	1	St-Urbain_1	Saint-Urbain	Existant	compteur
2:	2	Brebeuf_1	Brebeuf	Existant	compteur
3:	4	Maisonneuve_1	Maisonneuve_1	À réinstaller	compteur
4:	5	Maisonneuve_2	Maisonneuve_2	Existant	compteur
5:	6	Rachel/Papineau	Rachel/Papineau	Existant	compteur

17:	36	René-Lévesque_2	René-Lévesque	Existant	compteur
18:	37	Totem_Laurier	Totem_Laurier	Existant	totem
19:	3	Berri_1	Berri1	Existant	compteur
20:	38	Parc_2	Parc U-Zelt Test	Existant	Projet-pilot

21: 39 St-Laurent Saint-Laurent U-Zelt Test Existant Projet-pilot
 5 variables not shown: [Annee_implante, coord_X, coord_Y, lon, lat]

```
loc.name.code <- c(
  "Berri1"="Berri",
  "Brebeuf"="Brébeuf",
  "CSC"="Côte-Sainte-Catherine",
  "Maisonneuve_1"="Maisonneuve 1",
  "Maisonneuve_2"="Maisonneuve 2",
  "Parc"="du Parc",
  "PierDup"="Pierre-Dupuy",
  "Rachel/Papineau"="Rachel",
  "Saint-Urbain"="Saint-Urbain",
  "Totem_Laurier"="Totem_Laurier")
counter.locations[, location := loc.name.code[nom_comptage] ]
velo.counts <- table(counts.dt$location)
(show.locations <- counter.locations[names(velo.counts), on=list(location)])
```

	id	nom	nom_comptage	Etat	Type
1:	3	Berri_1	Berri1	Existant	compteur
2:	2	Brebeuf_1	Brebeuf	Existant	compteur
3:	8	Cote-Ste-Catherine_1	CSC	Existant	compteur
4:	4	Maisonneuve_1	Maisonneuve_1	À réinstaller	compteur
5:	5	Maisonneuve_2	Maisonneuve_2	Existant	compteur
6:	22	Parc_1	Parc	Existant	compteur
7:	12	Pierre-Dupuy_1	PierDup	Existant	compteur
8:	6	Rachel/Papineau	Rachel/Papineau	Existant	compteur
9:	1	St-Urbain_1	Saint-Urbain	Existant	compteur
10:	37	Totem_Laurier	Totem_Laurier	Existant	totem

6 variables not shown: [Annee_implante, coord_X, coord_Y, lon, lat, location]

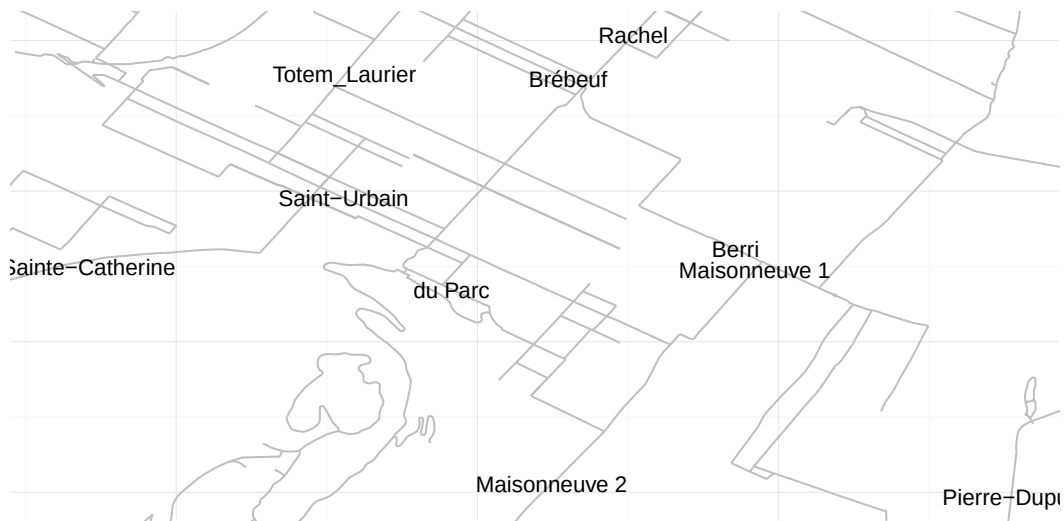
The counter locations above will be plotted below. Note that we use `showSelected=month` and `clickSelects=location`.

```
map.lim <- show.locations[, list(
  range.lat=range(lat),
  range.lon=range(lon)
)]
diff.vec <- sapply(map.lim, diff)
diff.mat <- c(-1, 1) * matrix(diff.vec, 2, 2, byrow=TRUE)
scale.mat <- as.matrix(map.lim) + diff.mat
location.colors <-
  c("#8DD3C7", "#FFFFB3", "#BEBADA", "#FB8072", "#80B1D3", "#FDB462",
    "#B3DE69", "#FCCDE5", "#D9D9D9", "#BC80BD", "#CCEBC5", "#FFED6F")
names(location.colors) <- show.locations$location
counts.per.month.loc <- counts.per.month[show.locations, on=list(location)]
bike.paths <- data.table(montreal.bikes$path.locations)
some.paths <- bike.paths[
  scale.mat[1, "range.lat"] < lat &
  scale.mat[1, "range.lon"] < lon &
```

```

lat <- scale.mat[2, "range.lat"] &
lon <- scale.mat[2, "range.lon"]
mtl.map <- ggplot()+
  theme_bw()+
  theme(
    panel.margin=grid::unit(0, "lines"),
    axis.line=element_blank(), axis.text=element_blank(),
    axis.ticks=element_blank(), axis.title=element_blank(),
    panel.background = element_blank(),
    panel.border = element_blank())+
  coord_equal(xlim=map.lim$range.lon, ylim=map.lim$range.lat)+
  scale_color_manual(values=location.colors)+
  scale_x_continuous(limits=scale.mat[, "range.lon"])+
  scale_y_continuous(limits=scale.mat[, "range.lat"])+
  geom_path(aes(
    lon, lat,
    tooltip=TYPE_VOIE,
    group=paste(feature.i, path.i)),
    color="grey",
    data=some.paths)+
  guides(color="none")+
  geom_text(aes(
    lon, lat,
    label=location),
    clickSelects="location",
    data=show.locations)
mtl.map

```

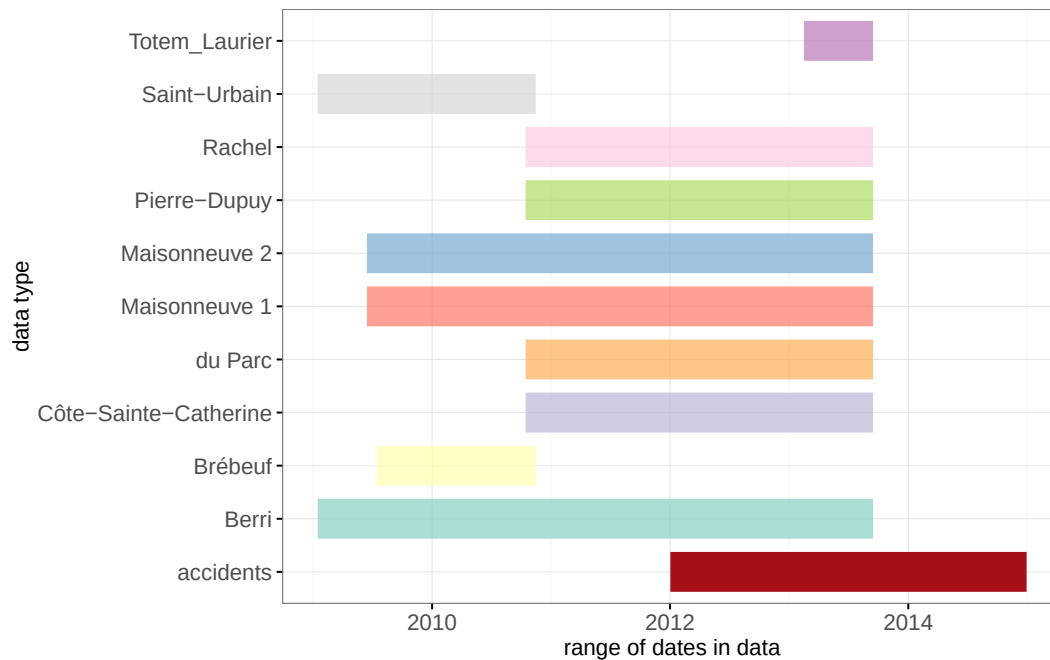


The plot below shows the time period that each counter was in operation. Note that we use `geom_tallrect` with `clickSelects` to select the month.

```

location.ranges <- counts.per.month[0 < count, list(
  min=min(month.POSIXct),
  max=max(month.POSIXct)
), by=location]
accidents.range <- accidents.dt[, data.table(
  location="accidents",
  min=min(date.POSIXct),
  max=max(date.POSIXct))]
MonthSummary <- ggplot()+
  theme_bw()+
  theme_animint(width=450, height=250)+
  xlab("range of dates in data")+
  ylab("data type")+
  scale_color_manual(values=location.colors)+
  guides(color="none")+
  geom_segment(aes(
    min, location,
    xend=max, yend=location,
    color=location),
    clickSelects="location",
    data=location.ranges, alpha=3/4, size=10)+
  geom_segment(aes(
    min, location,
    xend=max, yend=location),
    color=severity.colors[["deaths"]],
    data=accidents.range,
    size=10)
print(MonthSummary)

```



The plot below shows the bike counts at each location and day.

```
(dates <- counts.dt[, list(
  min.date = date-one.day/2,
  max.date = date+one.day/2,
  locations=sum(!is.na(count))
), by=list(date)])[0 < locations])
```

	date	min.date	max.date	locations
1:	2009-01-01 05:00:00	2008-12-31 17:00:00	2009-01-01 17:00:00	9
2:	2009-01-02 05:00:00	2009-01-01 17:00:00	2009-01-02 17:00:00	9
3:	2009-01-03 05:00:00	2009-01-02 17:00:00	2009-01-03 17:00:00	9
4:	2009-01-04 05:00:00	2009-01-03 17:00:00	2009-01-04 17:00:00	9
5:	2009-01-05 05:00:00	2009-01-04 17:00:00	2009-01-05 17:00:00	9

1604:	2013-09-14 04:00:00	2013-09-13 16:00:00	2013-09-14 16:00:00	8
1605:	2013-09-15 04:00:00	2013-09-14 16:00:00	2013-09-15 16:00:00	8
1606:	2013-09-16 04:00:00	2013-09-15 16:00:00	2013-09-16 16:00:00	8
1607:	2013-09-17 04:00:00	2013-09-16 16:00:00	2013-09-17 16:00:00	8
1608:	2013-09-18 04:00:00	2013-09-17 16:00:00	2013-09-18 16:00:00	8

```
(location.labels <- counts.dt[
  , .SD[which.max(count)]
  , by=list(location)])
```

	location	date	count	month.str
1:	Berri	2010-06-15 04:00:00	7495	2010-06
2:	Brébeuf	2010-06-04 04:00:00	9235	2010-06
3:	Côte-Sainte-Catherine	2013-09-18 04:00:00	3330	2013-09
4:	Maisonnette 1	2011-06-17 04:00:00	5355	2011-06
5:	Maisonnette 2	2011-06-07 04:00:00	8332	2011-06
6:	du Parc	2011-09-27 04:00:00	4577	2011-09
7:	Pierre-Dupuy	2013-07-21 04:00:00	4841	2013-07
8:	Rachel	2013-05-31 04:00:00	8555	2013-05
9:	Saint-Urbain	2010-04-27 04:00:00	3856	2010-04
10:	Totem_Laurier	2013-08-21 04:00:00	4293	2013-08

5 variables not shown: [loc.lines, month.POSIXct, month.text, day.of.the.month, month]

```
TimeSeries <- ggplot()+
  theme_bw()+
  geom_tallrect(aes(
    xmin=date-one.day/2, xmax=date+one.day/2,
    clickSelects=date),
  data=dates, alpha=1/2)+
  geom_line(aes(
    date, count, group=location,
    showSelected=location,
    clickSelects=location),
  data=counts.dt)+
```

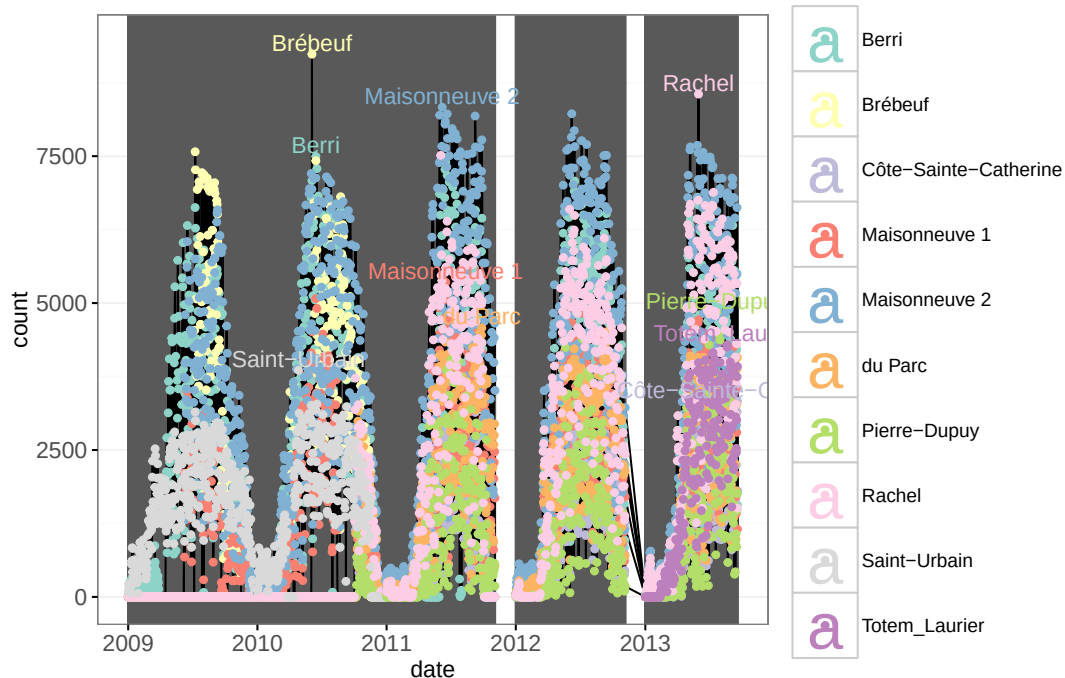


```

scale_color_manual(values=location.colors)+
geom_point(aes(
  date, count, color=location,
  showSelected=location,
  clickSelects=location),
  data=counts.dt)+
geom_text(aes(
  date, count+200, color=location, label=location,
  showSelected=location,
  clickSelects=location),
  data=location.labels)
print(TimeSeries)

```

Warning: Removed 407 rows containing missing values (geom_point).



The plot below shows the same data but for each month.

```

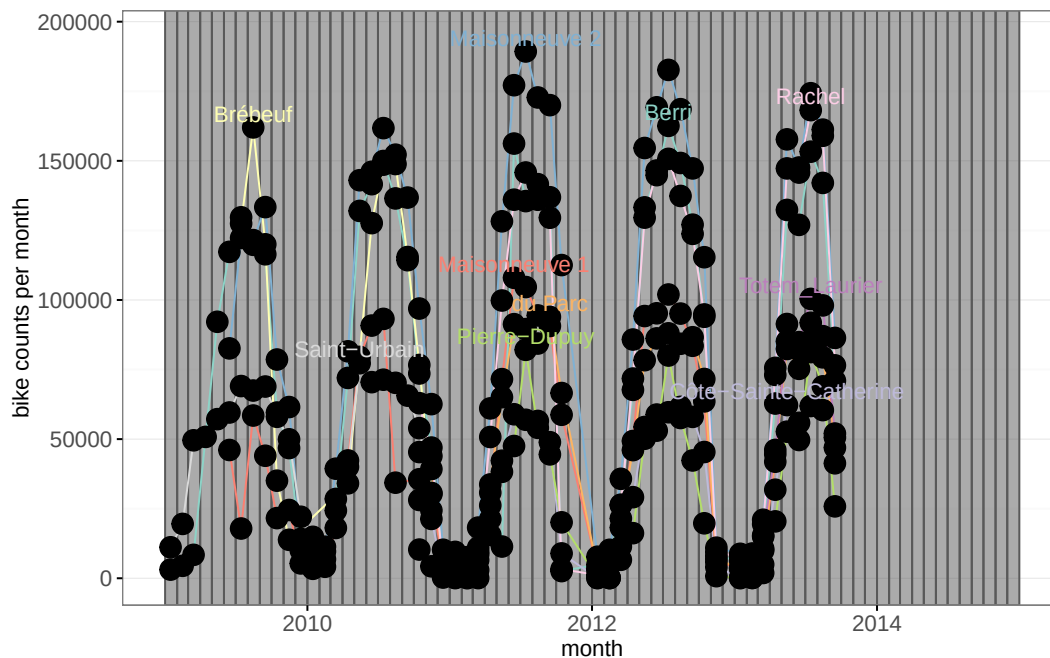
MonthSeries <- ggplot()+
  guides(color="none", fill="none")+
  theme_bw()+
  geom_tallrect(aes(
    xmin=month01.POSIXct, xmax=next01.POSIXct),
    clickSelects="month",
    data=months,
    alpha=1/2)+
  geom_line(aes(
    month.POSIXct, count, group=location,
    color=location),

```

```

    showSelected="location",
    clickSelects="location",
    data=counts.per.month)+
scale_color_manual(values=location.colors)+
scale_fill_manual(values=location.colors)+
xlab("month")+
ylab("bike counts per month")+
geom_point(aes(
  month.POSIXct, count, fill=location,
  tooltip=paste(
    count, "bikers counted at",
    location, "in", month)),
  showSelected="location",
  clickSelects="location",
  size=5,
  color="black",
  data=counts.per.month)+
geom_text(aes(
  month.POSIXct, count+5000, color=location, label=location),
  showSelected="location",
  clickSelects="location",
  data=month.labels)
print(MonthSeries)

```



```

counter.title <- "mean cyclists per day"
accidents.title <- "city-wide accidents"
MonthFacet <- ggplot()+
  ggtitle("All data, select month")+

```

```

guides(color="none", fill="none")+
theme_bw()+
facet_grid(facet ~ ., scales="free")+
theme(panel.margin=grid::unit(0, "lines"))+
geom_tallrect(aes(
  xmin=month01.POSIXct, xmax=next01.POSIXct),
  clickSelects="month",
  data=data.table(
    city.wide.cyclists,
    facet=counter.title),
  alpha=1/2)+
geom_line(aes(
  month.POSIXct, mean.per.day, group=location,
  color=location),
  showSelected="location",
  clickSelects="location",
  data=data.table(counts.per.month, facet=counter.title))+
scale_color_manual(values=location.colors)+
xlab("month")+
ylab("")+
geom_point(aes(
  month.POSIXct, mean.per.day, color=location,
  tooltip=paste(
    count, "cyclists counted at",
    location, "in",
    days, "days of", month,
    sprintf("(mean %d cyclists/day)", as.integer(mean.per.day)))),
  showSelected="location",
  clickSelects="location",
  size=5,
  fill="grey",
  data=data.table(counts.per.month, facet=counter.title))+
geom_text(aes(
  month.POSIXct, mean.per.day+300, color=location, label=location),
  showSelected="location",
  clickSelects="location",
  data=data.table(month.labels, facet=counter.title))+
scale_fill_manual(values=severity.colors, breaks=names(severity.colors))+
geom_bar(aes(
  month.POSIXct, people,
  fill=severity),
  showSelected="severity",
  stat="identity",
  position="identity",
  color=NA,
  data=data.table(accidents.tall, facet=accidents.title))+
geom_tallrect(aes(
  xmin=month01.POSIXct, xmax=next01.POSIXct,
  tooltip=paste(

```

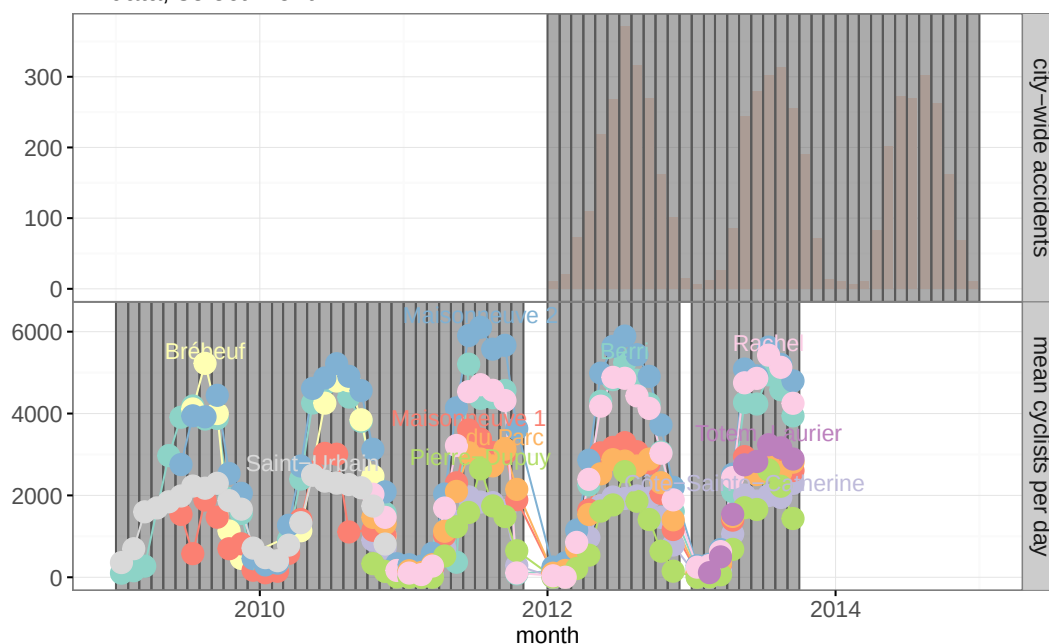
```

ifelse(deaths==0, "",
ifelse(deaths==1,
      "1 death,",
      paste(deaths, "deaths,"))),
ifelse(people.severely.injured==0, "",
ifelse(people.severely.injured==1,
      "1 person severely injured,",
      paste(people.severely.injured,
            "people severely injured,"))),
people.slightly.injured,
"people slightly injured in",
month)),
clickSelects="month",
alpha=0.5,
data=data.table(accidents.per.month,
                  facet=accidents.title))

```

MonthFacet

All data, select month



```

(days.dt <- data.table(
  day.POSIXct=with(months, seq(
    min(month01.POSIXct),
    max(next01.POSIXct),
    by="day"))
)[
  , day.of.the.week := strftime(day.POSIXct, "%a")
][

```

day.POSIXct day.of.the.week

```

1: 2009-01-01 Thu
2: 2009-01-02 Fri
3: 2009-01-03 Sat
4: 2009-01-04 Sun
5: 2009-01-05 Mon
---
2188: 2014-12-28 Sun
2189: 2014-12-29 Mon
2190: 2014-12-30 Tue
2191: 2014-12-31 Wed
2192: 2015-01-01 Thu

```

```

## The following only works in locales with English days of the week.
(weekend.dt <- days.dt[
  day.of.the.week %in% c("Sat", "Sun")
][, let(
  month.text = strftime(day.POSIXct, "%B %Y"),
  day.of.the.month = as.integer(strftime(day.POSIXct, "%d"))
)]
, month := factor(month.text, month.levs)
][])

```

```

      day.POSIXct day.of.the.week   month.text day.of.the.month      month
1: 2009-01-03      Sat January 2009           3 January 2009
2: 2009-01-04      Sun January 2009           4 January 2009
3: 2009-01-10      Sat January 2009          10 January 2009
4: 2009-01-11      Sun January 2009          11 January 2009
5: 2009-01-17      Sat January 2009          17 January 2009
---
622: 2014-12-14      Sun December 2014          14 December 2014
623: 2014-12-20      Sat December 2014          20 December 2014
624: 2014-12-21      Sun December 2014          21 December 2014
625: 2014-12-27      Sat December 2014          27 December 2014
626: 2014-12-28      Sun December 2014          28 December 2014

```

```

counter.title <- "cyclists per day"
DaysFacet <- ggplot()+
  ggtitle("Selected month (weekends in grey)")+
  geom_tallrect(aes(
    xmin=day.of.the.month-0.5, xmax=day.of.the.month+0.5,
    key=paste(day.POSIXct)),
    showSelected="month",
    fill="grey",
    color="white",
    data=weekend.dt)+
  guides(color="none")+
  theme_bw()+
  facet_grid(facet ~ ., scales="free")+
  geom_line(aes(
    day.of.the.month, count, group=location,

```

```

    key=location,
    color=location),
    showSelected=c("location", "month"),
    clickSelects="location",
    chunk_vars=c("month"),
    data=data.table(counts.dt, facet=counter.title))+
scale_color_manual(values=location.colors)+
ylab("")+
geom_point(aes(
  day.of.the.month, count, color=location,
  key=paste(day.of.the.month, location),
  tooltip=paste(
    count, "cyclists counted at",
    location, "on",
    date)),
  showSelected=c("location", "month"),
  clickSelects="location",
  size=5,
  chunk_vars=c("month"),
  fill="white",
  data=data.table(counts.dt, facet=counter.title))+
scale_fill_manual(values=severity.colors, breaks=names(severity.colors))+
geom_text(aes(
  15, 23, label=month, key=1),
  showSelected="month",
  data=data.table(months, facet=accidents.title))+
scale_x_continuous("day of the month", breaks=c(1, 10, 20, 30))+
geom_text(aes(
  day.of.the.month, count+500, color=location, label=location,
  key=location),
  showSelected=c("location", "month"),
  clickSelects="location",
  data=data.table(day.labels, facet=counter.title))+
geom_point(aes(
  day.of.the.month, accident.i,
  key=paste(date.str, accident.i),
  tooltip=paste(
    ifelse(deaths==0, "",
    ifelse(deaths==1,
      "1 death,",
      paste(deaths, "deaths,"))),
    ifelse(people.severely.injured==0, "",
    ifelse(people.severely.injured==1,
      "1 person severely injured,",
      paste(people.severely.injured,
        "people severely injured,"))),
    people.slightly.injured,
    "people slightly injured at",
    ifelse(is.na(street.number), "", street.number),

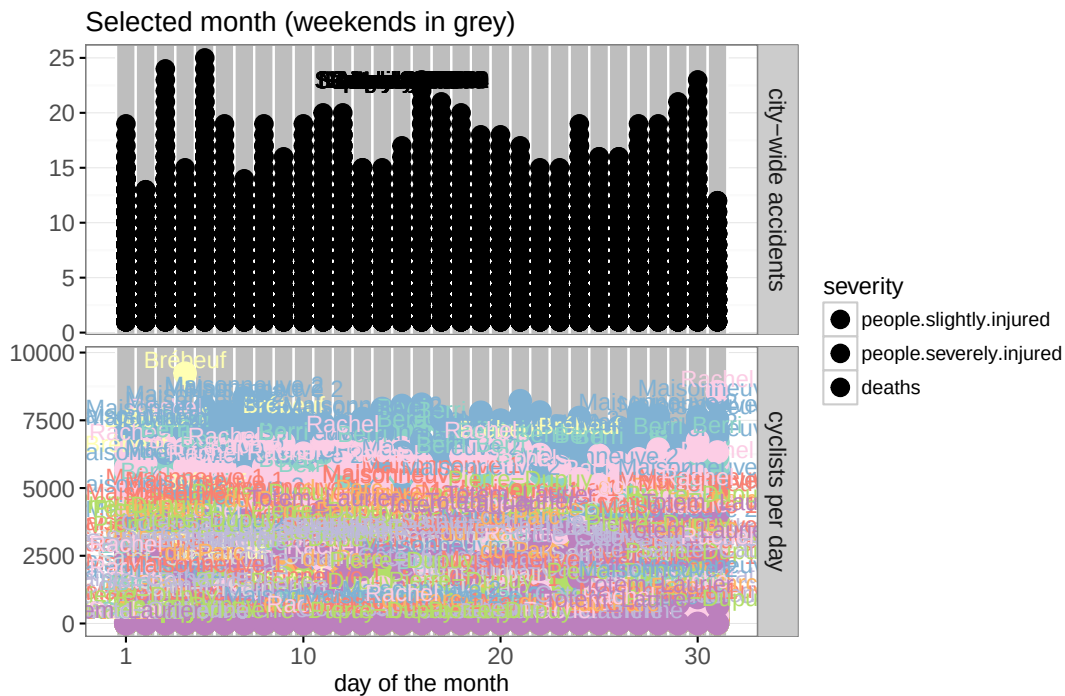
```

```

    street, "/", cross.street,
    date.str, time.str),
  fill=severity),
  showSelected="month",
  size=4,
  chunk_vars=c("month"),
  data=data.table(accidents.cumsum, facet=accidents.title))
DaysFacet

```

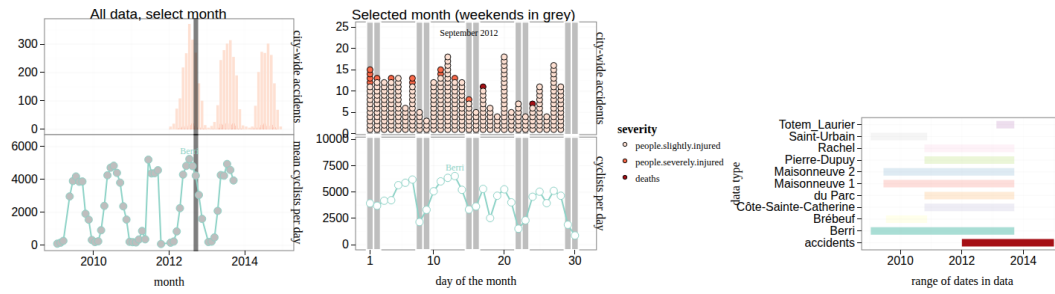
Warning: Removed 407 rows containing missing values (geom_point).



```

animint(
  MonthFacet,
  DaysFacet,
  MonthSummary,
  selector.types=list(severity="multiple"),
  duration=list(month=2000),
  first=list(
    location="Berri",
    month="September 2012"),
  time=list(variable="month", ms=5000))#buggy.

```



9.4 Chapter summary and exercises

Exercises:

- Change location to a multiple selection variable.
- Add a plot for the map to the data viz.
- On the map, draw a circle for each location, with size that changes based on the `count` of the accidents in the currently selected `month`.
- On the `MonthSummary` plot, add a background rectangle that can be used to select the `month`.
- Remove the `MonthSummary` plot and add a similar visualization as a third panel in the `MonthFacet` plot.

Next, [Chapter 10](#) explains how to visualize the K-Nearest-Neighbors machine learning model.

10

K-Nearest-Neighbors example

In this chapter we will explore several data visualizations of the K-Nearest-Neighbors (KNN) classifier.

Chapter outline:

- We will start with the original static data visualization, re-designed as two ggplots rendered by animint. There is a plot of 10-fold cross-validation error, and a plot of the predictions of the 7-Nearest-Neighbors classifier.
- We propose a re-design that allows selecting the number of neighbors used for the model predictions.
- We propose a second re-design that allows selecting the number of folds used to compute the cross-validation error.

10.1 Original static figure

We start by reproducing a static version of Figure 13.4 from [Elements of Statistical Learning by Hastie et al](#) . That Figure consists of two plots:

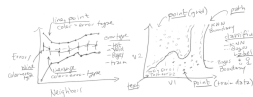


Figure 10.1: Static KNN viz

Left: mis-classification error curves, as a function of the number of neighbors.

- `geom_line` and `geom_point` for the error curves.
- `geom_linerange` for error bars of the validation error curve.
- `geom_hline` for the Bayes error.
- `x = neighbors`.
- `y = percent error`.
- `color = error type`.

Right: data and decision boundaries in the two-dimensional input feature space.

- `geom_point` for the data points.
- `geom_point` for the classification predictions on the grid in the background.
- `geom_path` for the decision boundaries.
- `geom_text` for the train/test/Bayes error rates.

10.1.1 Plot of mis-classification error curves

We begin by loading the data set.

```
if(!file.exists("ESL.mixture.rda")){
  curl::curl_download(
    "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda",
    "ESL.mixture.rda")
}
load("ESL.mixture.rda")
str(ESL.mixture)
```

List of 8

```
$ x      : num [1:200, 1:2] 2.5261 0.367 0.7682 0.6934 -0.0198 ...
$ y      : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
$ xnew    : 'matrix' num [1:6831, 1:2] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -
1.9 -1.8 -1.7 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : chr [1:6831] "1" "2" "3" "4" ...
.. ..$ : chr [1:2] "x1" "x2"
$ prob    : num [1:6831] 3.55e-05 3.05e-05 2.63e-05 2.27e-05 1.96e-05 ...
..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
$ marginal: num [1:6831] 6.65e-15 2.31e-14 7.62e-14 2.39e-13 7.15e-13 ...
..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
$ px1     : num [1:69] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
$ px2     : num [1:99] -2 -1.95 -1.9 -1.85 -1.8 -1.75 -1.7 -1.65 -1.6 -
1.55 ...
$ means   : num [1:20, 1:2] -0.2534 0.2667 2.0965 -0.0613 2.7035 ...
```

We will use the following components of this data set:

- **x**, the input matrix of the training data set (200 observations x 2 numeric features).
- **y**, the output vector of the training data set (200 class labels, either 0 or 1).
- **xnew**, the grid of points in the input space where we will show the classifier predictions (6831 grid points x 2 numeric features).
- **prob**, the true probability of class 1 at each of the grid points (6831 numeric values between 0 and 1).
- **px1**, the grid of points for the first input feature (69 numeric values between -2.6 and 4.2). These will be used to compute the Bayes decision boundary using the `contourLines` function.
- **px2**, the grid of points for the second input feature (99 numeric values between -2 and 2.9).
- **means**, the 20 centers of the normal distributions in the simulation model (20 centers x 2 input features).

First, we create a test set, following the example code from `help(ESL.mixture)`. Note that we use a `data.table` rather than a `data.frame` to store these big data, since `data.table` is often faster and more memory efficient for big data sets.

```
library(MASS)
library(data.table)
set.seed(123)
```

```
centers <- c(
  sample(1:10, 5000, replace=TRUE),
  sample(11:20, 5000, replace=TRUE))
mix.test <- mvrnorm(10000, c(0,0), 0.2*diag(2))
test.points <- data.table(
  mix.test + ESL.mixture$means[centers,],
  label=factor(c(rep(0, 5000), rep(1, 5000))))
test.points
```

	V1	V2	label
1:	2.0210959	1.390512445	0
2:	2.7488414	1.032724096	0
3:	2.2631823	0.003595182	0
4:	0.9215398	0.880968101	0
5:	1.8492359	-0.807985255	0

9996:	-2.0939980	1.602554853	1
9997:	1.6409142	0.889177271	1
9998:	0.9861499	0.936203846	1
9999:	-1.9089417	1.613524647	1
10000:	0.7678115	0.315426519	1

We then create a data table which includes all test points and grid points, which we will use in the test argument to the knn function.

```
pred.grid <- data.table(ESL.mixture$xnew, label=NA)
input.cols <- c("V1", "V2")
names(pred.grid)[1:2] <- input.cols
test.and.grid <- rbind(
  data.table(test.points, set="test"),
  data.table(pred.grid, set="grid"))
test.and.grid$fold <- NA
test.and.grid
```

	V1	V2	label	set	fold
1:	2.0210959	1.390512445	0	test	NA
2:	2.7488414	1.032724096	0	test	NA
3:	2.2631823	0.003595182	0	test	NA
4:	0.9215398	0.880968101	0	test	NA
5:	1.8492359	-0.807985255	0	test	NA

16827:	3.8000000	2.900000000	<NA>	grid	NA
16828:	3.9000000	2.900000000	<NA>	grid	NA
16829:	4.0000000	2.900000000	<NA>	grid	NA
16830:	4.1000000	2.900000000	<NA>	grid	NA
16831:	4.2000000	2.900000000	<NA>	grid	NA

We randomly assign each observation of the training data set to one of ten folds.

```

n.folds <- 10
set.seed(2)
mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
mixture

```

```

      V1      V2 label fold
1:  2.526092968 0.3210504      0      5
2:  0.366954472 0.0314621      0      8
3:  0.768219076 0.7174862      0      6
4:  0.693435680 0.7771940      0     10
5: -0.019836616 0.8672537      0      6
---
196:  0.256750222 2.2936046      1      1
197:  1.925173384 0.1650526      1      3
198:  1.301941035 0.9921996      1      6
199:  0.008130556 2.2422639      1      4
200: -0.196246334 0.5514036      1      8

```

We define the following `OneFold` function, which divides the 200 observations into one train and one validation set. It then computes the predicted probability of the K-Nearest-Neighbors classifier for each of the data points in all sets (train, validation, test, and grid).

```

OneFold <- function(validation.fold){
  set <- ifelse(mixture$fold == validation.fold, "validation", "train")
  fold.data <- rbind(test.and.grid, data.table(mixture, set))
  fold.data$data.i <- 1:nrow(fold.data)
  only.train <- subset(fold.data, set == "train")
  data.by.neighbors <- list()
  for(neighbors in seq(1, 30, by=2)){
    if(interactive())cat(sprintf(
      "n.folds=%4d validation.fold=%d neighbors=%d\n",
      n.folds, validation.fold, neighbors))
    set.seed(1)
    pred.label <- class::knn( # random tie-breaking.
      only.train[, input.cols, with=FALSE],
      fold.data[, input.cols, with=FALSE],
      only.train$label,
      k=neighbors,
      prob=TRUE)
    prob.winning.class <- attr(pred.label, "prob")
    fold.data$probability <- ifelse(
      pred.label=="1", prob.winning.class, 1-prob.winning.class)
    fold.data[, pred.label := ifelse(0.5 < probability, "1", "0")]
    fold.data[, is.error := label != pred.label]
    fold.data[, prediction := ifelse(is.error, "error", "correct")]
    data.by.neighbors[[paste(neighbors)]] <-
      data.table(neighbors, fold.data)
  }#for(neighbors
  do.call(rbind, data.by.neighbors)
}

```

```
}#for(validation.fold
```

Below, we run the `OneFold` function in parallel using the `future` package. Note that validation folds 1:10 will be used to compute the validation set error. The validation fold 0 treats all 200 observations as a train set, and will be used for visualizing the learned decision boundaries of the K-Nearest-Neighbors classifier.

```
future::plan("multisession")
data.all.folds.list <- future.apply::future_lapply(
  0:n.folds, function(validation.fold){
    one.fold <- OneFold(validation.fold)
    data.table(validation.fold, one.fold)
  },
  future.seed = NULL)
(data.all.folds <- do.call(rbind, data.all.folds.list))
```

	validation.fold	neighbors		V1	V2	label	set	fold
1:	0	1	2.021095933	1.390512445	0	test	NA	
2:	0	1	2.748841354	1.032724096	0	test	NA	
3:	0	1	2.263182259	0.003595182	0	test	NA	
4:	0	1	0.921539757	0.880968101	0	test	NA	
5:	0	1	1.849235932	-0.807985255	0	test	NA	

2810111:	10	29	0.256750222	2.293604575	1	train	1	
2810112:	10	29	1.925173384	0.165052639	1	train	3	
2810113:	10	29	1.301941035	0.992199616	1	train	6	
2810114:	10	29	0.008130556	2.242263910	1	train	4	
2810115:	10	29	-0.196246334	0.551403559	1	train	8	

5 variables not shown: [data.i, probability, pred.label, is.error, prediction]

The data table of predictions contains almost 3 million observations! When there are so many data, visualizing all of them at once is not practical or informative. Instead of visualizing them all at once, we will compute and plot summary statistics. In the code below we compute the mean and standard error of the mis-classification error for each model (over the 10 validation folds). This is an example of the summarize data table idiom which is generally useful for computing summary statistics for a single data table.

```
labeled.data <- data.all.folds[!is.na(label),]
error.stats <- labeled.data[, list(
  error.prop=mean(is.error)
), by=.(set, validation.fold, neighbors)]
validation.error <- error.stats[set=="validation", list(
  mean=mean(error.prop),
  sd=sd(error.prop)/sqrt(.N)
), by=.(set, neighbors)]
validation.error
```

	set	neighbors	mean	sd
1:	validation	1	0.240	0.01943651
2:	validation	3	0.165	0.02362908
3:	validation	5	0.165	0.02477678

```

4: validation      7 0.175 0.02713137
5: validation      9 0.195 0.03023060
---
11: validation     21 0.180 0.01699673
12: validation     23 0.195 0.02166667
13: validation     25 0.190 0.01795055
14: validation     27 0.195 0.02034426
15: validation     29 0.205 0.02291288

```

Below we construct data tables for the Bayes error (which we know is 0.21 for the mixture example data), and the train/test error.

```

Bayes.error <- data.table(
  set="Bayes",
  validation.fold=NA,
  neighbors=NA,
  error.prop=0.21)
Bayes.error

```

```

      set validation.fold neighbors error.prop
1: Bayes              NA         NA      0.21

```

```

other.error <- error.stats[validation.fold==0,]
head(other.error)

```

```

      set validation.fold neighbors error.prop
1: test              0          1    0.2938
2: train             0          1    0.0000
3: test              0          3    0.2400
4: train             0          3    0.1300
5: test              0          5    0.2273
6: train             0          5    0.1300

```

Below we construct a color palette from `dput(RColorBrewer::brewer.pal(Inf, "Set1"))`, and linetype palettes.

```

set.colors <- c(
  test="#377EB8", #blue
  validation="#4DAF4A", #green
  Bayes="#984EA3", #purple
  train="#FF7F00") #orange
classifier.linetypes <- c(
  Bayes="dashed",
  KNN="solid")
set.linetypes <- set.colors
set.linetypes[] <- classifier.linetypes[["KNN"]]
set.linetypes["Bayes"] <- classifier.linetypes[["Bayes"]]
cbind(set.linetypes, set.colors)

```

```

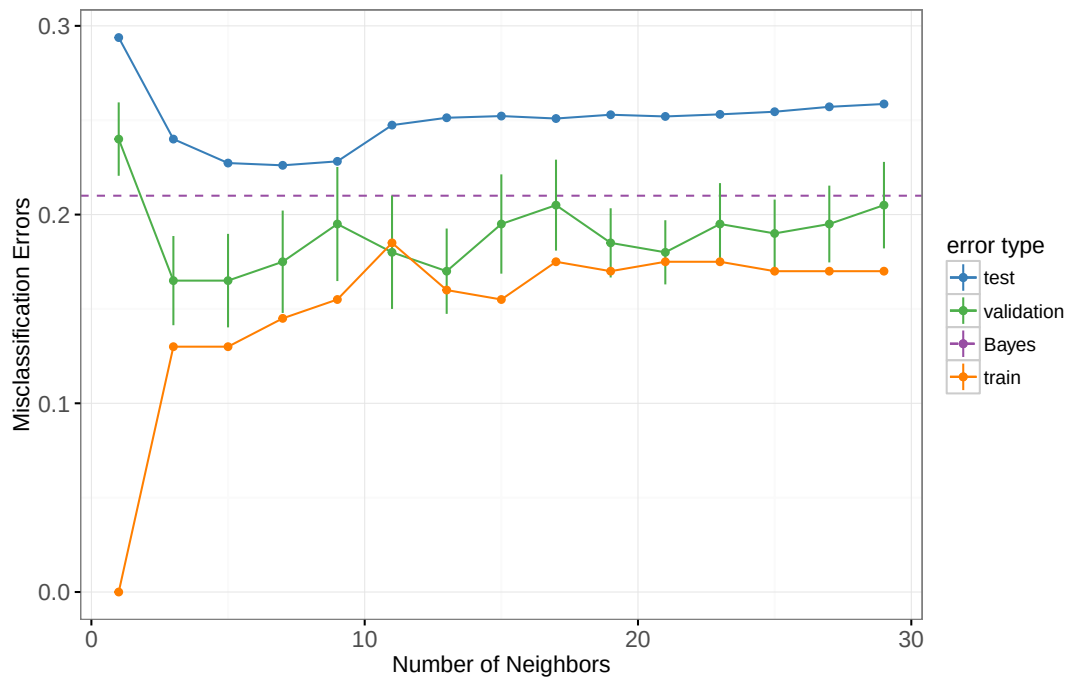
      set.linetypes set.colors
test      "solid"    "#377EB8"

```

validation	"solid"	"#4DAF4A"
Bayes	"dashed"	"#984EA3"
train	"solid"	"#FF7F00"

The code below reproduces the plot of the error curves from the original Figure.

```
library(animint2)
errorPlotStatic <- ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=error.prop, color=set, linetype=set),
    data=Bayes.error)+
  scale_color_manual(
    "error type", values=set.colors, breaks=names(set.colors))+
  scale_linetype_manual(
    "error type", values=set.linetypes, breaks=names(set.linetypes))+
  ylab("Misclassification Errors")+
  xlab("Number of Neighbors")+
  geom_linerange(aes(
    neighbors, ymin=mean-sd, ymax=mean+sd,
    color=set),
    data=validation.error)+
  geom_line(aes(
    neighbors, mean, linetype=set, color=set),
    data=validation.error)+
  geom_line(aes(
    neighbors, error.prop, group=set, linetype=set, color=set),
    data=other.error)+
  geom_point(aes(
    neighbors, mean, color=set),
    data=validation.error)+
  geom_point(aes(
    neighbors, error.prop, color=set),
    data=other.error)
errorPlotStatic
```



10.1.2 Plot of decision boundaries in the input feature space

For the static data visualization of the feature space, we show only the model with 7 neighbors.

```
show.neighbors <- 7
show.data <- data.all.folds[validation.fold==0 & neighbors==show.neighbors,]
show.points <- show.data[set=="train",]
show.points
```

	validation.fold	neighbors	V1	V2	label	set	fold	data.i
1:	0	7	2.526092968	0.3210504	0	train	5	16832
2:	0	7	0.366954472	0.0314621	0	train	8	16833
3:	0	7	0.768219076	0.7174862	0	train	6	16834
4:	0	7	0.693435680	0.7771940	0	train	10	16835
5:	0	7	-0.019836616	0.8672537	0	train	6	16836

196:	0	7	0.256750222	2.2936046	1	train	1	17027
197:	0	7	1.925173384	0.1650526	1	train	3	17028
198:	0	7	1.301941035	0.9921996	1	train	6	17029
199:	0	7	0.008130556	2.2422639	1	train	4	17030
200:	0	7	-0.196246334	0.5514036	1	train	8	17031

4 variables not shown: [probability, pred.label, is.error, prediction]

Next, we compute the Train, Test, and Bayes mis-classification error rates which we will show in the bottom left of the feature space plot.


```

text.height <- 0.25
text.V1.prop <- 0
text.V2.bottom <- -2
text.V1.error <- -2.6
error.text <- rbind(
  Bayes.error,
  other.error[neighbors==show.neighbors,])
error.text[, V2.top := text.V2.bottom + text.height * (1:.N)]
error.text[, V2.bottom := V2.top - text.height]
error.text

```

```

      set validation.fold neighbors error.prop V2.top V2.bottom
1: Bayes          NA          NA      0.2100 -1.75      -2.00
2: test           0           7      0.2261 -1.50      -1.75
3: train          0           7      0.1450 -1.25      -1.50

```

We define the following function which we will use to compute the decision boundaries.

```

getBoundaryDF <- function(prob.vec){
  stopifnot(length(prob.vec) == 6831)
  several.paths <- with(ESL.mixture, contourLines(
    px1, px2,
    matrix(prob.vec, length(px1), length(px2)),
    levels=0.5))
  contour.list <- list()
  for(path.i in seq_along(several.paths)){
    contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
      path.i, V1=x, V2=y))
  }
  do.call(rbind, contour.list)
}

```

We use this function to compute the decision boundaries for the learned 7-Nearest-Neighbors classifier, and for the optimal Bayes classifier.

```

boundary.grid <- show.data[set=="grid",]
boundary.grid[, label := pred.label]
pred.boundary <- getBoundaryDF(boundary.grid$probability)
pred.boundary$classifier <- "KNN"
Bayes.boundary <- getBoundaryDF(ESL.mixture$prob)
Bayes.boundary$classifier <- "Bayes"
Bayes.boundary

```

```

      path.i      V1      V2 classifier
1:      1 -2.600000 -0.528615      Bayes
2:      1 -2.557084 -0.500000      Bayes
3:      1 -2.500000 -0.459723      Bayes
4:      1 -2.484141 -0.450000      Bayes
5:      1 -2.407805 -0.400000      Bayes
---
246:     2  3.004058  2.700000      Bayes

```

```

247:      2  3.010219  2.750000      Bayes
248:      2  3.016359  2.800000      Bayes
249:      2  3.022480  2.850000      Bayes
250:      2  3.028586  2.900000      Bayes

```

Below, we consider only the grid points that do not overlap the text labels.

```

on.text <- function(V1, V2){
  V2 <= max(error.text$V2.top) & V1 <= text.V1.prop
}
show.grid <- boundary.grid[!on.text(V1, V2),]
show.grid

```

```

validation.fold neighbors V1 V2 label set fold data.i probability
1:      0      7 0.1 -2.0    0 grid  NA  10028  0.0000000
2:      0      7 0.2 -2.0    0 grid  NA  10029  0.0000000
3:      0      7 0.3 -2.0    0 grid  NA  10030  0.0000000
4:      0      7 0.4 -2.0    0 grid  NA  10031  0.0000000
5:      0      7 0.5 -2.0    0 grid  NA  10032  0.0000000
---
6395:    0      7 3.8  2.9     1 grid  NA  16827  0.5714286
6396:    0      7 3.9  2.9     1 grid  NA  16828  0.5714286
6397:    0      7 4.0  2.9     1 grid  NA  16829  0.5714286
6398:    0      7 4.1  2.9     1 grid  NA  16830  0.5714286
6399:    0      7 4.2  2.9     1 grid  NA  16831  0.5714286
3 variables not shown: [pred.label, is.error, prediction]

```

The scatterplot below reproduces the 7-Nearest-Neighbors classifier of the original Figure.

```

label.colors <- c(
  "0"="#377EB8",
  "1"="#FF7F00")
scatterPlotStatic <- ggplot()+
  theme_bw()+
  theme(axis.text=element_blank(),
        axis.ticks=element_blank(),
        axis.title=element_blank())+
  ggtitle("7-Nearest Neighbors")+
  scale_color_manual(values=label.colors)+
  scale_linetype_manual(values=classifier.linetypes)+
  geom_point(aes(
    V1, V2, color=label),
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),

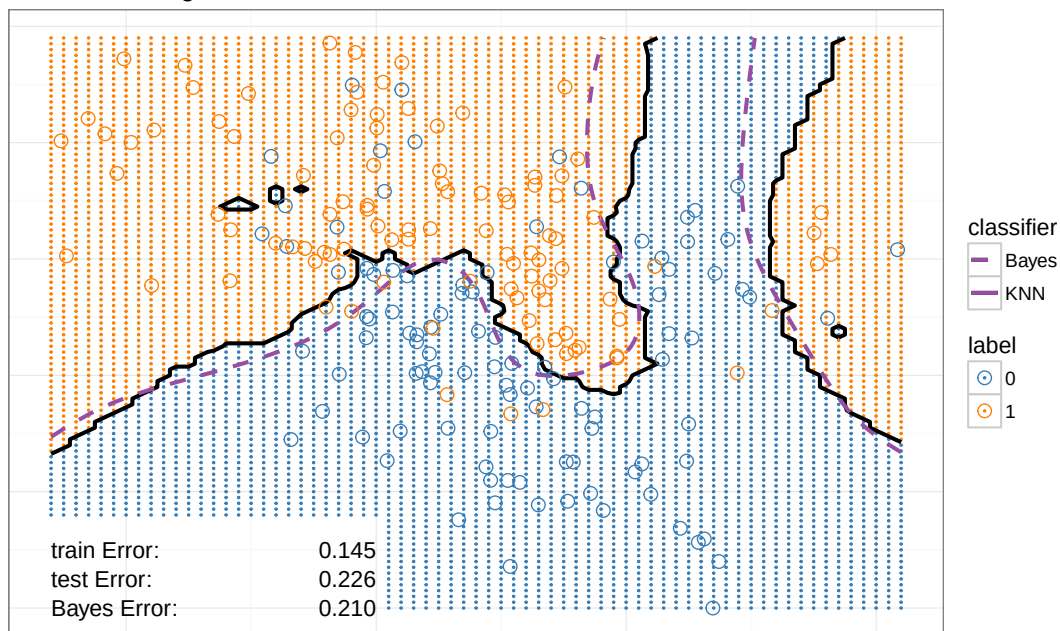
```

```

    color=set.colors[["Bayes"]],
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
    V1, V2, color=label),
    fill=NA,
    size=3,
    shape=21,
    data=show.points)+
  geom_text(aes(
    text.V1.error, V2.bottom, label=paste(set, "Error:")),
    data=error.text,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
    data=error.text,
    hjust=1)
  scatterPlotStatic

```

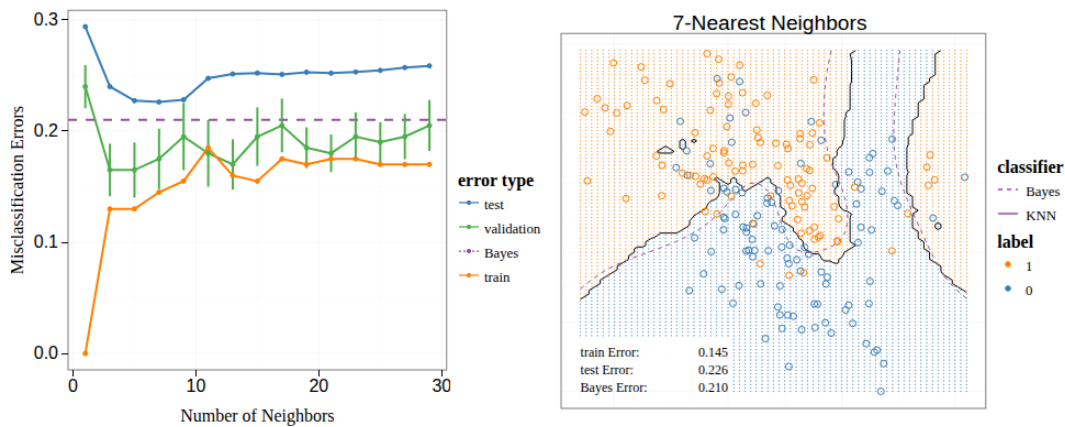
7-Nearest Neighbors



10.1.3 Combined plots

Finally, we combine the two ggplots and render them as an animint.

```
animint(errorPlotStatic, scatterPlotStatic)
```



This data viz does have three interactive legends, but it is static in the sense that it displays only the model predictions for 7-Nearest Neighbors.

10.2 Select the number of neighbors using interactivity

In this section we propose an interactive re-design which allows the user to select K, the number of neighbors in the K-Nearest-Neighbors classifier.

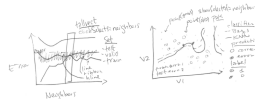


Figure 10.2: Interactive KNN viz

10.2.1 Clickable error curves plot

We begin with a re-design of the error curves plot.

Note the following changes:

- add a selector for the number of neighbors (`geom_tallrect`).
- change the Bayes decision boundary from `geom_hline` with a legend entry, to a `geom_segment` with a text label.
- add a linetype legend to distinguish error rates from the Bayes and KNN models.
- change the error bars (`geom_linerange`) to error bands (`geom_ribbon`).

The only new data that we need to define are the endpoints of the segment that we will use to plot the Bayes decision boundary. Note that we also re-define the set “test” to emphasize the fact that the Bayes error is the best achievable error rate for test data.

```
Bayes.segment <- data.table(
  Bayes.error,
  classifier="Bayes",
  min.neighbors=1,
  max.neighbors=29)
```

```
Bayes.segment$set <- "test"
```

We also add an error variable to the data tables that contain the prediction error of the K-Nearest-Neighbors models. This error variable will be used for the linetype legend.

```
validation.error$classifier <- "KNN"
other.error$classifier <- "KNN"
```

We re-define the plot of the error curves below. Note that

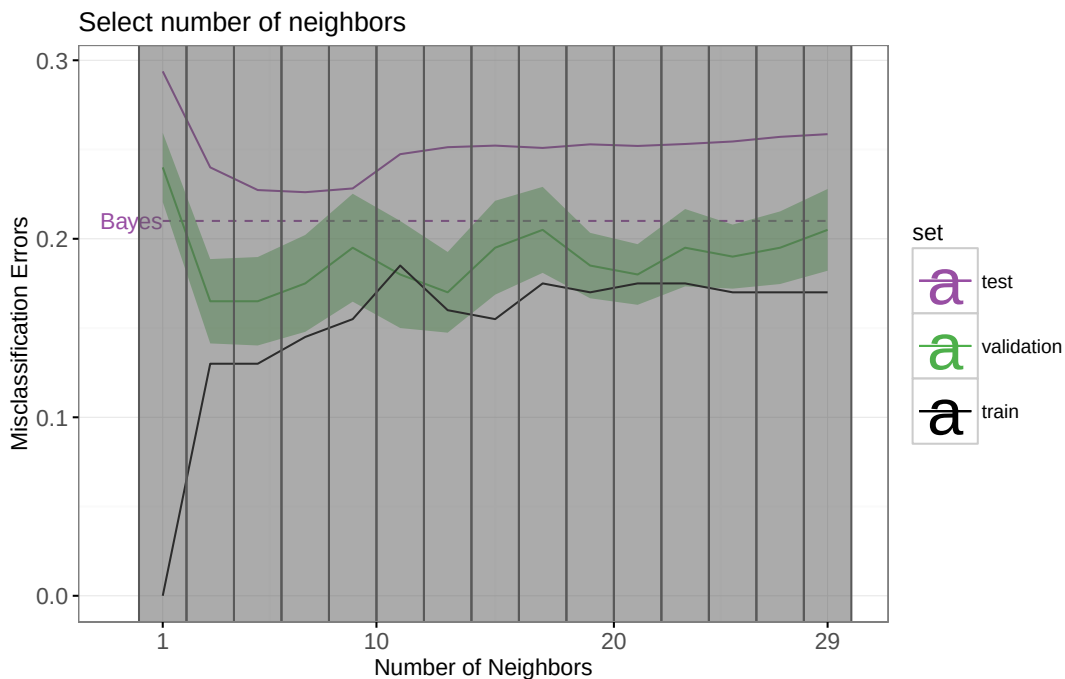
- We use `showSelected` in `geom_text` and `geom_ribbon`, so that they will be hidden when the interactive legends are clicked.
- We use `clickSelects` in `geom_tallrect`, to select the number of neighbors. Clickable geoms should be last (top layer) so that they are not obscured by non-clickable geoms (bottom layers).

```
set.colors <- c(
  test="#984EA3", #purple
  validation="#4DAF4A", #green
  Bayes="#984EA3", #purple
  train="black")
errorPlot <- ggplot()+
  ggtitle("Select number of neighbors")+
  theme_bw()+
  theme_animint(height=500)+
  geom_text(aes(
    min.neighbors, error.prop,
    color=set, label="Bayes"),
    showSelected="classifier",
    hjust=1,
    data=Bayes.segment)+
  geom_segment(aes(
    min.neighbors, error.prop,
    xend=max.neighbors, yend=error.prop,
    color=set,
    linetype=classifier),
    showSelected="classifier",
    data=Bayes.segment)+
  scale_color_manual(values=set.colors, breaks=names(set.colors))+
  scale_fill_manual(values=set.colors)+
  guides(fill="none", linetype="none")+
  scale_linetype_manual(values=classifier.linetypes)+
  ylab("Misclassification Errors")+
  scale_x_continuous(
    "Number of Neighbors",
    limits=c(-1, 30),
    breaks=c(1, 10, 20, 29))+
  geom_ribbon(aes(
    neighbors, ymin=mean-sd, ymax=mean+sd,
    fill=set),
```

```

showSelected=c("classifier", "set"),
alpha=0.5,
color="transparent",
data=validation.error)+
geom_line(aes(
  neighbors, mean, color=set,
  linetype=classifier),
showSelected="classifier",
data=validation.error)+
geom_line(aes(
  neighbors, error.prop, group=set, color=set,
  linetype=classifier),
showSelected="classifier",
data=other.error)+
geom_tallrect(aes(
  xmin=neighbors-1, xmax=neighbors+1),
clickSelects="neighbors",
alpha=0.5,
data=validation.error)
errorPlot

```



10.2.2 Feature space plot that shows the selected number of neighbors

Next, we focus on a re-design of the feature space plot. In the previous section we considered only the subset of data from the model with 7 neighbors. Our re-design includes the following changes:

- We use neighbors as a showSelected variable.

- We add a legend to show which training data points are mis-classified.
- We use equal spaced coordinates so that visual distance (pixels) is the same as the Euclidean distance in the feature space.

```
show.data <- data.all.folds[validation.fold==0,]
show.points <- show.data[set=="train",]
show.points
```

	validation.fold	neighbors	V1	V2	label	set	fold	data.i
1:	0	1	2.526092968	0.3210504	0	train	5	16832
2:	0	1	0.366954472	0.0314621	0	train	8	16833
3:	0	1	0.768219076	0.7174862	0	train	6	16834
4:	0	1	0.693435680	0.7771940	0	train	10	16835
5:	0	1	-0.019836616	0.8672537	0	train	6	16836

2996:	0	29	0.256750222	2.2936046	1	train	1	17027
2997:	0	29	1.925173384	0.1650526	1	train	3	17028
2998:	0	29	1.301941035	0.9921996	1	train	6	17029
2999:	0	29	0.008130556	2.2422639	1	train	4	17030
3000:	0	29	-0.196246334	0.5514036	1	train	8	17031

4 variables not shown: [probability, pred.label, is.error, prediction]

Below, we compute the predicted decision boundaries separately for each K-Nearest-Neighbors model.

```
boundary.grid <- show.data[set=="grid",]
boundary.grid[, label := pred.label]
show.grid <- boundary.grid[!on.text(V1, V2),]
pred.boundary <- boundary.grid[, getBoundaryDF(probability), by=neighbors]
pred.boundary$classifier <- "KNN"
pred.boundary
```

	neighbors	path.i	V1	V2	classifier
1:	1	1	-2.600000	-1.025000	KNN
2:	1	1	-2.550000	-1.000000	KNN
3:	1	1	-2.500000	-0.975000	KNN
4:	1	1	-2.450000	-0.950000	KNN
5:	1	1	-2.450000	-0.900000	KNN

4488:	29	2	2.800000	1.897619	KNN
4489:	29	2	2.795238	1.900000	KNN
4490:	29	2	2.800000	1.902381	KNN
4491:	29	2	2.800990	1.900000	KNN
4492:	29	2	2.800000	1.897619	KNN

Instead of showing the number of neighbors in the plot title, below we create a `geom_text` element that will be updated based on the number of selected neighbors.

```
show.text <- show.grid[, list(
  V1=mean(range(V1)), V2=3.05), by=neighbors]
```

Below we compute the position of the text in the bottom left, which we will use to display the error rate of the selected model.

```
other.error[, V2.bottom := rep(
  text.V2.bottom + text.height * 1:2, l=.N)]
```

Below we re-define the Bayes error data without a neighbors column, so that it appears in each showSelected subset.

```
Bayes.error <- data.table(
  set="Bayes",
  error.prop=0.21)
```

Finally, we re-define the ggplot, using neighbors as a showSelected variable in the point, path, and text geoms.

```
scatterPlot <- ggplot()+
  ggtitle("Mis-classification errors in train set")+
  theme_bw()+
  theme_animint(width=500, height=500)+
  xlab("Input feature 1")+
  ylab("Input feature 2")+
  coord_equal()+
  scale_color_manual(values=label.colors)+
  scale_linetype_manual(values=classifier.linetypes)+
  geom_point(aes(
    V1, V2, color=label),
    showSelected="neighbors",
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    showSelected="neighbors",
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    color=set.colors[["test"]],
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
    V1, V2, color=label,
    fill=prediction),
    showSelected="neighbors",
    size=3,
    shape=21,
    data=show.points)+
  scale_fill_manual(values=c(error="black", correct="transparent"))+
  geom_text(aes(
```



```

    text.V1.error, text.V2.bottom, label=paste(set, "Error:")),
    data=Bayes.error,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, text.V2.bottom, label=sprintf("%.3f", error.prop)),
    data=Bayes.error,
    hjust=1)+
  geom_text(aes(
    text.V1.error, V2.bottom, label=paste(set, "Error:")),
    showSelected="neighbors",
    data=other.error,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
    showSelected="neighbors",
    data=other.error,
    hjust=1)+
  geom_text(aes(
    V1, V2,
    label=paste0(
      neighbors,
      " nearest neighbor",
      ifelse(neighbors==1, "", "s"),
      " classifier")),
    showSelected="neighbors",
    data=show.text)

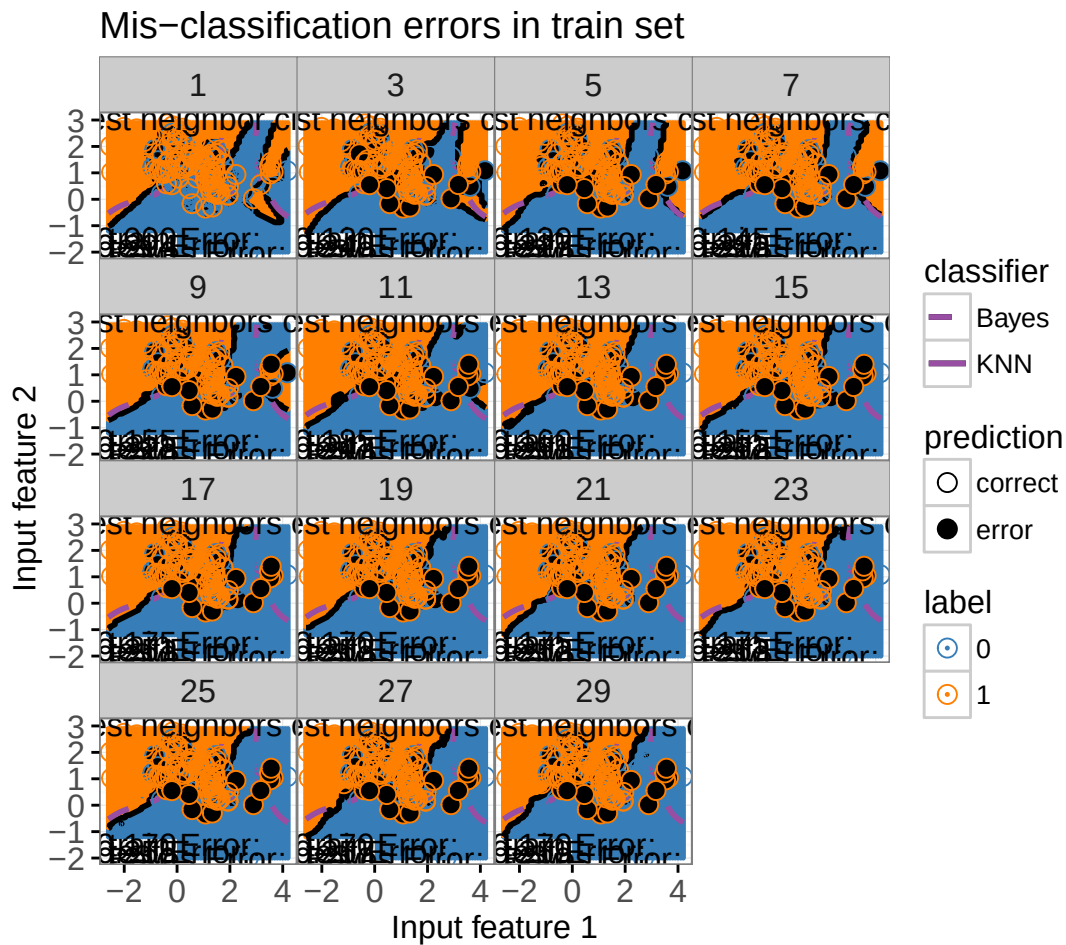
```

Before compiling the interactive data viz, we print a static ggplot with a facet for each value of neighbors.

```

scatterPlot+
  facet_wrap("neighbors")+
  theme(panel.margin=grid::unit(0, "lines"))

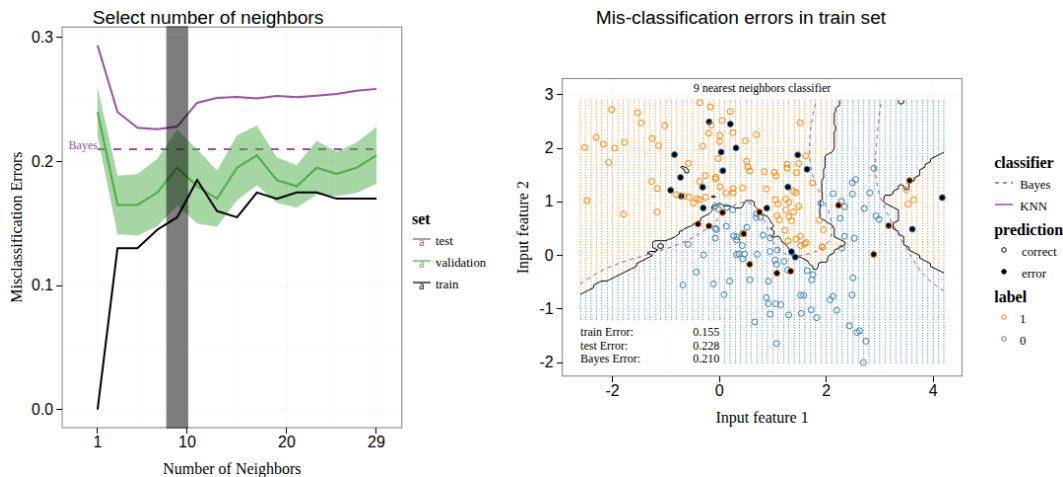
```



10.2.3 Combined interactive data viz

Finally, we combine the two plots in a single data viz with neighbors as a selector variable.

```
animint(
  errorPlot,
  scatterPlot,
  first=list(neighbors=7),
  time=list(variable="neighbors", ms=3000))
```



Note that neighbors is used as a time variable, so animation shows the predictions of the different models.

10.3 Select the number of cross-validation folds using interactivity

In this section we discuss a second re-design which allows the user to select the number of folds used to compute the validation error curve.

The for loop below computes the validation error curve for several different values of `n.folds`.

```
error.by.folds <- list()
error.by.folds[["10"]] <- data.table(n.folds=10, validation.error)
for(n.folds in c(3, 5, 15)){
  set.seed(2)
  mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
  mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
  only.validation.list <- future.apply::future_lapply(
    1:n.folds, function(validation.fold){
      one.fold <- OneFold(validation.fold)
      data.table(validation.fold, one.fold[set=="validation"])
    })
  only.validation <- do.call(rbind, only.validation.list)
  only.validation.error <- only.validation[, list(
    error.prop=mean(is.error)
  ), by=.(set, validation.fold, neighbors)]
  only.validation.stats <- only.validation.error[, list(
    mean=mean(error.prop),
    sd=sd(error.prop)/sqrt(.N)
  ), by=.(set, neighbors)]
  error.by.folds[[paste(n.folds)]] <-
    data.table(n.folds, only.validation.stats, classifier="KNN")
}
```

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option

'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations ('future_lapply-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper, parallel-safe random numbers are produced via a parallel RNG method. To disable this check, use 'future.seed = NULL', or set option 'future.rng.onMisuse' to "ignore".

```
validation.error.several <- do.call(rbind, error.by.folds)
```

The code below computes the minimum of the error curve for each value of n.folds.

```
min.validation <- validation.error.several[, .SD[which.min(mean),], by=n.folds]
```

The code below creates a new error curve plot with two facets.

```
facets <- function(df, facet){
  data.frame(df, facet=factor(facet, c("n.folds", "Misclassification Errors")))
}
errorPlotNew <- ggplot()+
  ggtitle("Select number of folds and neighbors")+
  theme_bw()+
  theme_animint(height=500)+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_grid(facet ~ ., scales="free")+
```

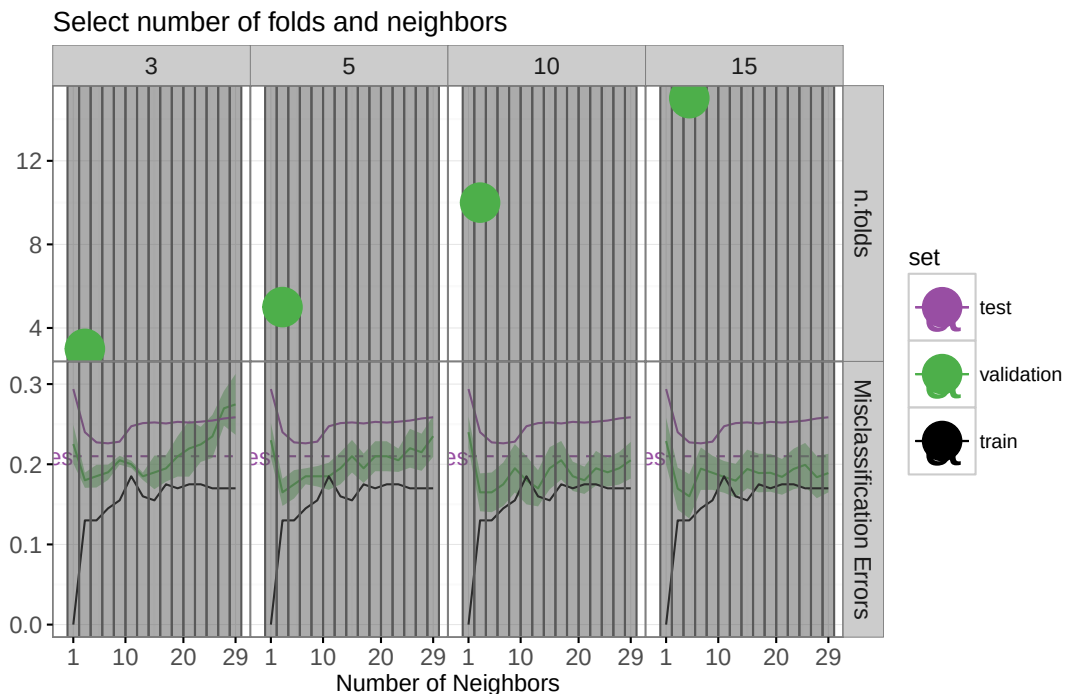
```

geom_text(aes(
  min.neighbors, error.prop,
  color=set, label="Bayes"),
  showSelected="classifier",
  hjust=1,
  data=facets(Bayes.segment, "Misclassification Errors"))+
geom_segment(aes(
  min.neighbors, error.prop,
  xend=max.neighbors, yend=error.prop,
  color=set,
  linetype=classifier),
  showSelected="classifier",
  data=facets(Bayes.segment, "Misclassification Errors"))+
scale_color_manual(values=set.colors, breaks=names(set.colors))+
scale_fill_manual(values=set.colors, breaks=names(set.colors))+
guides(fill="none", linetype="none")+
scale_linetype_manual(values=classifier.linetypes)+
ylab("")+
scale_x_continuous(
  "Number of Neighbors",
  limits=c(-1, 30),
  breaks=c(1, 10, 20, 29))+
geom_ribbon(aes(
  neighbors, ymin=mean-sd, ymax=mean+sd,
  fill=set),
  showSelected=c("classifier", "set", "n.folds"),
  alpha=0.5,
  color="transparent",
  data=facets(validation.error.several, "Misclassification Errors"))+
geom_line(aes(
  neighbors, mean, color=set,
  linetype=classifier),
  showSelected=c("classifier", "n.folds"),
  data=facets(validation.error.several, "Misclassification Errors"))+
geom_line(aes(
  neighbors, error.prop, group=set, color=set,
  linetype=classifier),
  showSelected="classifier",
  data=facets(other.error, "Misclassification Errors"))+
geom_tallrect(aes(
  xmin=neighbors-1, xmax=neighbors+1),
  clickSelects="neighbors",
  alpha=0.5,
  data=validation.error)+
geom_point(aes(
  neighbors, n.folds, color=set),
  clickSelects="n.folds",
  size=9,
  data=facets(min.validation, "n.folds"))

```

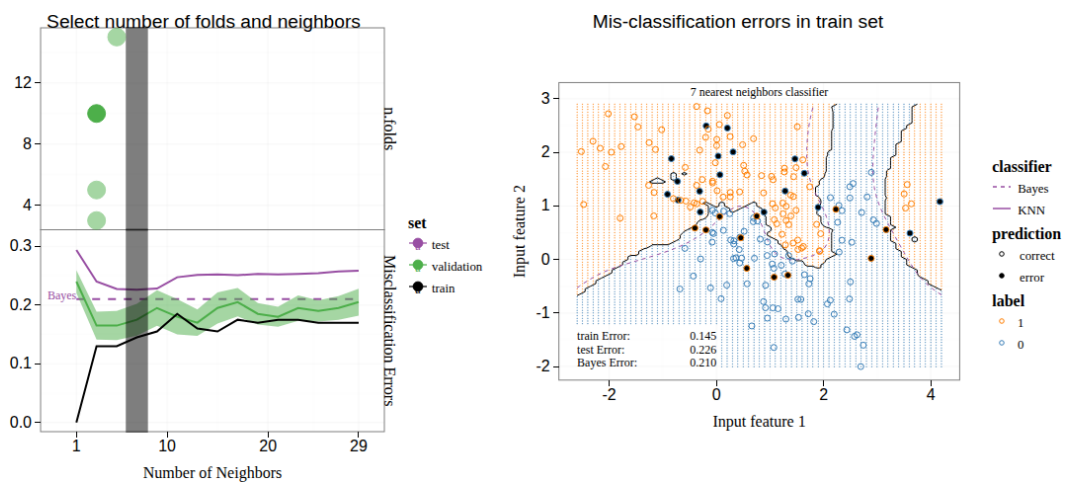
The code below previews the new error curve plot, adding an additional facet for the showSelected variable.

```
errorPlotNew+facet_grid(facet ~ n.folds, scales="free")
```



The code below creates an interactive data viz using the new error curve plot.

```
animint(
  errorPlotNew,
  scatterPlot,
  first=list(neighbors=7, n.folds=10))
```



10.4 Chapter summary and exercises

We showed how to add two interactive features to a data visualization of predictions of the K-Nearest-Neighbors model. We started with a static data visualization which only showed predictions of the 7-Nearest-Neighbors model. Then, we created an interactive re-design which allowed selecting K, the number of neighbors. We did another re-design which added a facet for selecting the number of cross-validation folds.

Exercises:

- Make it so that text error rates in the bottom left of the second plot are hidden after clicking the legend entries for Bayes, train, test. Hint: you can either use one `geom_text` with `showSelected=c(selectorNameColumn="selectorValueColumn")` (as explained in [Chapter 14](#)) or two `geom_text`, each with a different `showSelected` parameter.
- The probability column of the `show.grid` data table is the predicted probability of class 1. How would you re-design the visualization to show the predicted probability rather than the predicted class at each grid point? The main challenge is that probability is a numeric variable, but `ggplot2` enforces that each scale must be either continuous or discrete (not both). You could use a continuous fill scale, but then you would have to use a different scale to show the prediction variable.
- Add a new plot that shows the relative sizes of the train, validation, and test sets. Make sure that the plotted size of the validation and train sets change based on the selected value of `n.folds`.
- So far, the feature space plots only showed model predictions and errors for the entire train data set (`validation.fold==0`). Create a re-design which includes a new plot or facet for selecting `validation.fold`, and a faceted feature space plot (one facet for train set, one facet for validation set).

Next, [Chapter 11](#) explains how to visualize the Lasso, a machine learning model.

11

Lasso

This goal of this chapter is to create an interactive data visualization that explains the [Lasso](#), a machine learning model for regularized linear regression.

Chapter outline:

- We begin with several static data visualizations of the lasso path.
- We then create an interactive version with a facet and plot showing train/validation error and residuals.
- Finally we re-design the interactive data visualization with simplified legends and moving tallrects.

11.1 Static plots of the coefficient regularization path

We begin by loading the prostate cancer data set.

```
if(!file.exists("prostate.data")){
  download.file(
    "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data",
    "prostate.data")
}
prostate <- data.table::fread("prostate.data")
head(prostate)
```

	V1	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa
1:	1	-0.5798185	2.769459	50	-1.386294	0	-1.386294	6	0	-
										0.4307829
2:	2	-0.9942523	3.319626	58	-1.386294	0	-1.386294	6	0	-
										0.1625189
3:	3	-0.5108256	2.691243	74	-1.386294	0	-1.386294	7	20	-
										0.1625189
4:	4	-1.2039728	3.282789	58	-1.386294	0	-1.386294	6	0	-
										0.1625189
5:	5	0.7514161	3.432373	62	-1.386294	0	-1.386294	6	0	0.3715636
6:	6	-1.0498221	3.228826	50	-1.386294	0	-1.386294	6	0	0.7654678

1 variable not shown: [train]

We construct a train inputs x and outputs y using the code below.

```
input.cols <- c(
  "lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason",
  "pgg45")
prostate.inputs <- prostate[, ..input.cols]
is.train <- prostate$train == "T"
x <- as.matrix(prostate.inputs[is.train])
head(x)
```

```
      lcavol lweight age      lbph svi      lcp gleason pgg45
[1,] -0.5798185 2.769459 50 -1.386294 0 -1.386294      6      0
[2,] -0.9942523 3.319626 58 -1.386294 0 -1.386294      6      0
[3,] -0.5108256 2.691243 74 -1.386294 0 -1.386294      7     20
[4,] -1.2039728 3.282789 58 -1.386294 0 -1.386294      6      0
[5,]  0.7514161 3.432373 62 -1.386294 0 -1.386294      6      0
[6,] -1.0498221 3.228826 50 -1.386294 0 -1.386294      6      0
```

```
y <- prostate[is.train, lpsa]
head(y)
```

```
[1] -0.4307829 -0.1625189 -0.1625189 -0.1625189  0.3715636  0.7654678
```

Below we fit the full path of lasso solutions using the `lars` package.

```
if(!requireNamespace("lars"))install.packages("lars")
```

Loading required namespace: lars

```
library(lars)
```

Loaded lars 1.3

```
fit <- lars(x,y,type="lasso")
fit$lambda
```

```
[1] 7.1939462 3.7172742 2.9403866 1.7305064 1.7002813 0.4933166 0.3711651
[8] 0.0403451
```

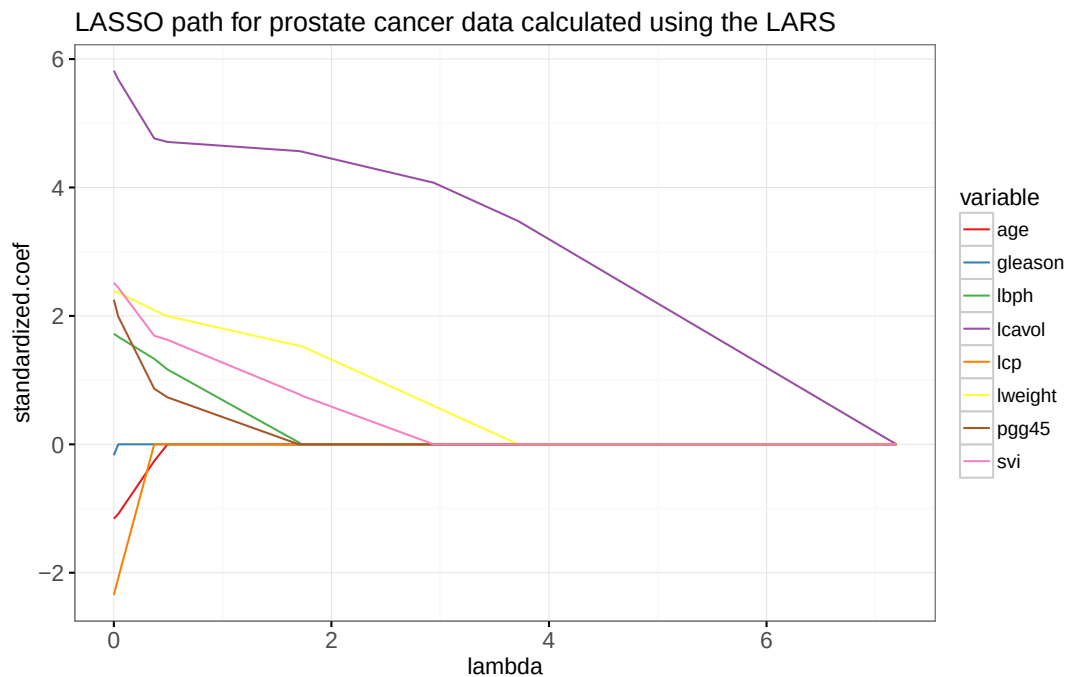
The path of `lambda` values are not evenly spaced.

```
pred.nox <- predict(fit, type="coef")
beta <- scale(pred.nox$coefficients, FALSE, 1/fit$normx)
arclength <- rowSums(abs(beta))
path.list <- list()
for(variable in colnames(beta)){
  standardized.coef <- beta[, variable]
  path.list[[variable]] <- data.table::data.table(
    step=seq_along(standardized.coef),
    lambda=c(fit$lambda, 0),
    variable,
    standardized.coef,
```

```

    fraction=pred.noxx$fraction,
    arclength)
  }
  path <- do.call(rbind, path.list)
  variable.colors <- c(
    "#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
    "#A65628", "#F781BF", "#999999")
  library(animint2)
  gg.lambda <- ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    scale_color_manual(values=variable.colors)+
    geom_line(aes(
      lambda, standardized.coef, color=variable, group=variable),
      data=path)+
    ggtitle("LASSO path for prostate cancer data calculated using the LARS")
  gg.lambda

```



The plot above shows the entire lasso path, the optimal weights in the L1-regularized least squares regression problem, for every regularization parameter λ . The path begins at the least squares solution, $\lambda=0$ on the left. It ends at the completely regularized intercept-only model on the right. To see the equivalence with the ordinary least squares solution, we add dots in the plot below.

```

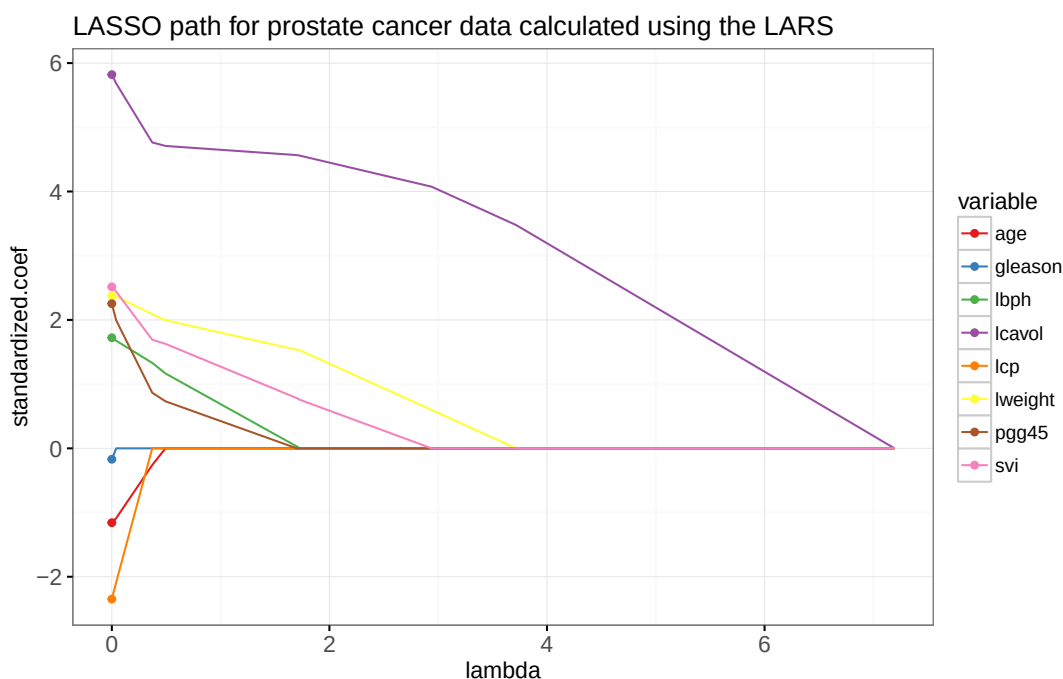
x.scaled <- with(fit, scale(x, meanx, normx))
lfit <- lm.fit(x.scaled, y)
lpoints <- data.table::data.table(

```

```

variable=colnames(x),
standardized.coef=lfit$coefficients,
arclength=sum(abs(lfit$coefficients)))
gg.lambda+
  geom_point(aes(
    0, standardized.coef, color=variable),
    data=lpoints)

```



In the next plot below, we show the path as a function of L1 norm (arclength), with some more points on an evenly spaced grid that we will use later for animation.

```

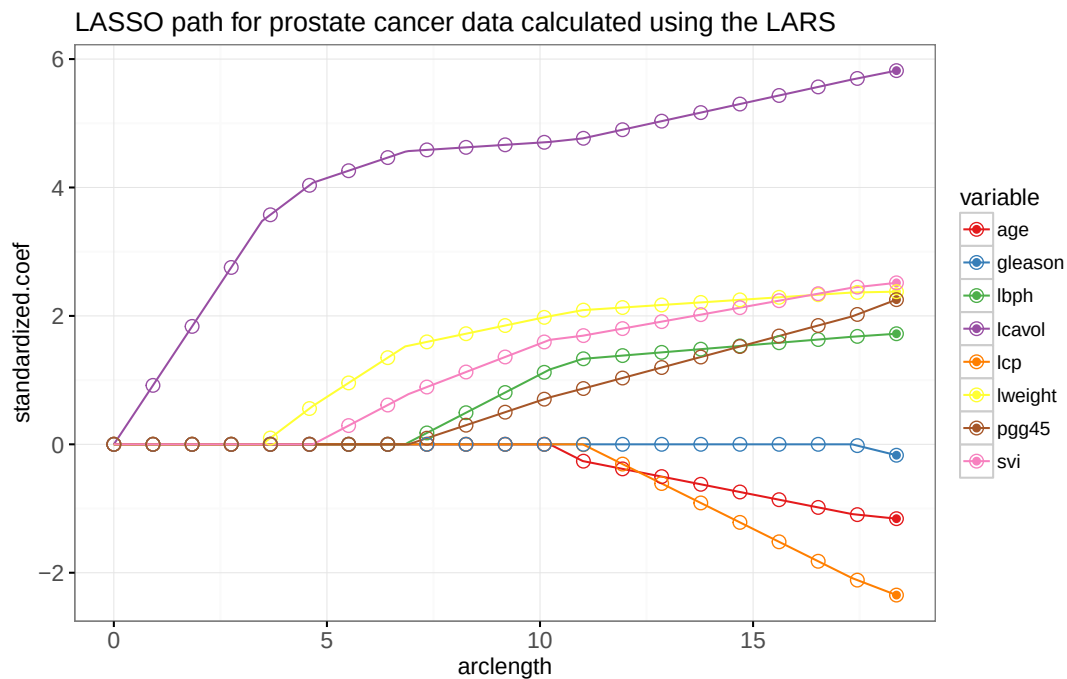
fraction <- sort(unique(c(
  seq(0, 1, l=21))))
pred.fraction <- predict(
  fit, prostate.inputs,
  type="coef", mode="fraction", s=fraction)
coef.grid.list <- list()
coef.grid.mat <- scale(pred.fraction$coefficients, FALSE, 1/fit$normx)
for(fraction.i in seq_along(fraction)){
  standardized.coef <- coef.grid.mat[fraction.i,]
  coef.grid.list[[fraction.i]] <- data.table::data.table(
    fraction=fraction[[fraction.i]],
    variable=colnames(x),
    standardized.coef,
    arclength=sum(abs(standardized.coef)))
}
coef.grid <- do.call(rbind, coef.grid.list)

```

```

ggplot()+
  ggtitle("LASSO path for prostate cancer data calculated using the LARS")+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=path)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    data=lpoints)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    shape=21,
    fill=NA,
    size=3,
    data=coef.grid)

```



The plot above shows that the weights at the grid points are consistent with the lines that represent the entire path of solutions. The LARS algorithm quickly provides Lasso solutions for as many grid points as you like. More precisely, since the LARS only computes the change-points in the piecewise linear path, its time complexity only depends on the number of change-points (not the number of grid points).

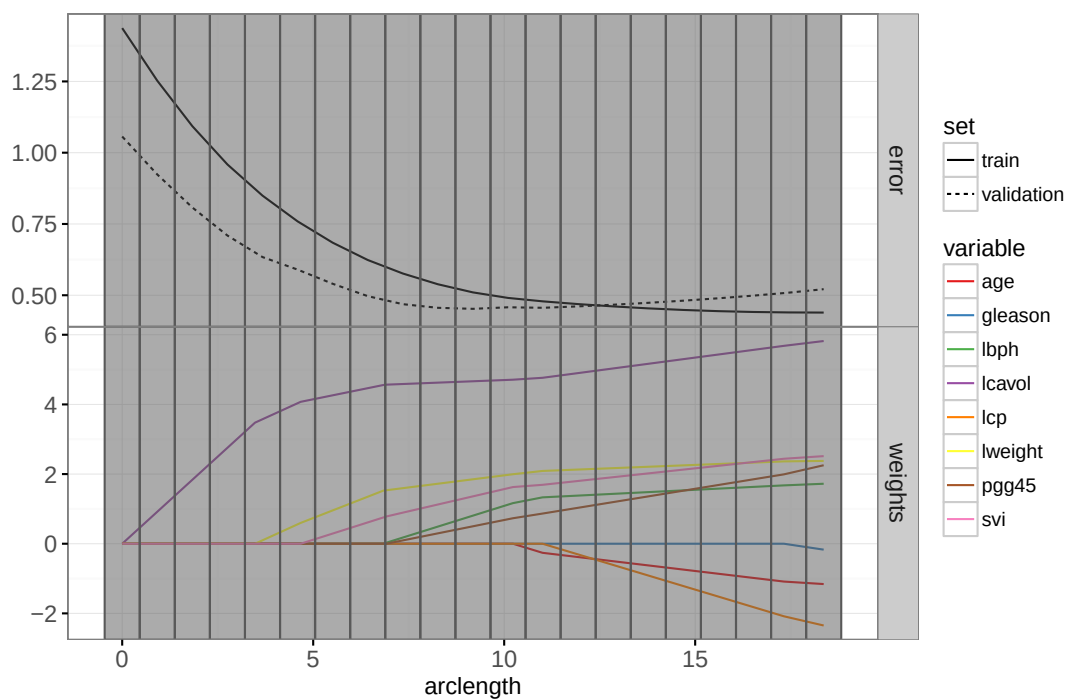
11.2 Interactive visualization of the regularization path

The plot below combines the lasso weight path with the train/test error plot.

```

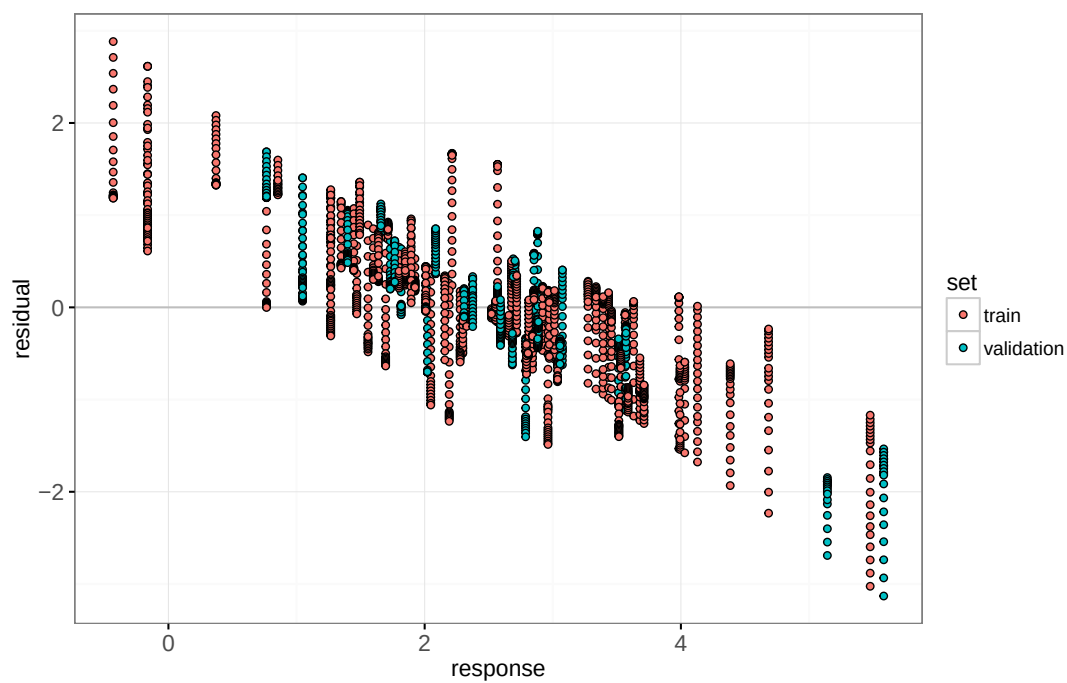
pred.list <- predict(
  fit, prostate.inputs,
  mode="fraction", s=fraction)
residual.mat <- pred.list$fit - prostate$lpsa
squares.mat <- residual.mat * residual.mat
mean.error.list <- list()
for(set in c("train", "validation")){
  val <- if(set=="train")TRUE else FALSE
  is.set <- is.train == val
  mse <- colMeans(squares.mat[is.set, ])
  mean.error.list[[paste(set)]] <- data.table::data.table(
    set, mse, fraction,
    arclength=rowSums(abs(coef.grid.mat)))
}
mean.error <- do.call(rbind, mean.error.list)
rect.width <- diff(mean.error$arclength[1:2])/2
addY <- function(dt, y){
  data.table::data.table(dt, y.var=factor(y, c("error", "weights")))
}
tallrect.dt <- coef.grid[variable==variable[1],]
gg.path <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y.var ~ ., scales="free")+
  ylab("")+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=addY(path, "weights"))+
  geom_line(aes(
    arclength, mse, linetype=set, group=set),
    data=addY(mean.error, "error"))+
  geom_tallrect(aes(
    xmin=arclength-rect.width,
    xmax=arclength+rect.width),
    clickSelects="arclength",
    alpha=0.5,
    data=tallrect.dt)
print(gg.path)

```



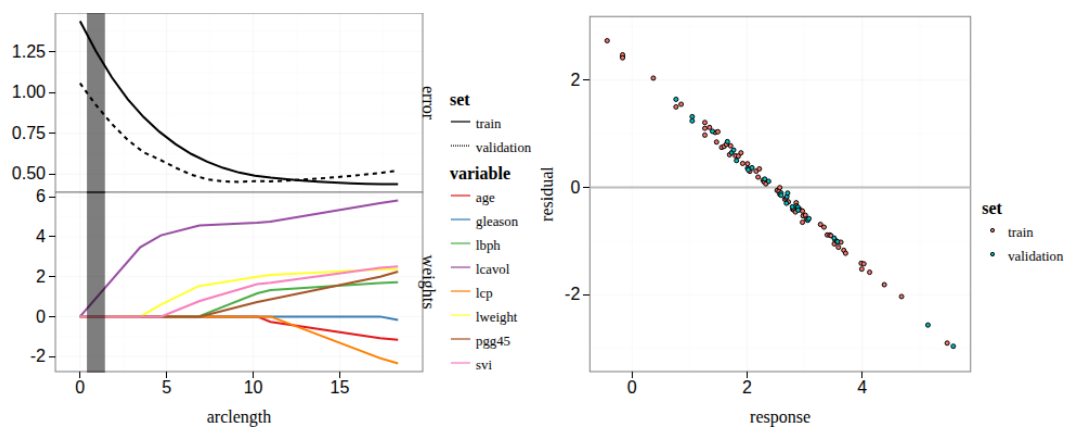
Finally, we add a plot of residuals versus actual values.

```
lasso.res.list <- list()
for(fraction.i in seq_along(fraction)){
  lasso.res.list[[fraction.i]] <- data.table::data.table(
    observation.i=1:nrow(prostate),
    fraction=fraction[[fraction.i]],
    residual=residual.mat[, fraction.i],
    response=prostate$lpse,
    arclength=sum(abs(coef.grid.mat[fraction.i,])),
    set=ifelse(prostate$train, "train", "validation"))
}
lasso.res <- do.call(rbind, lasso.res.list)
hline.dt <- data.table::data.table(residual=0)
gg.res <- ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=residual),
    data=hline.dt,
    color="grey")+
  geom_point(aes(
    response, residual, fill=set,
    key=observation.i),
    showSelected="arclength",
    shape=21,
    data=lasso.res)
print(gg.res)
```



Below, we combine the ggplots above in a single animint below. Clicking the first plot changes the regularization parameter, and the residuals that are shown in the second plot.

```
animint(
  gg.path,
  gg.res,
  duration=list(arclength=2000),
  time=list(variable="arclength", ms=2000))
```



11.3 Re-design with moving tallrects

The re-design below has two changes. First, you may have noticed that there are two different set legends in the previous animint (linetype=set in the first path plot, and color=set in the second residual plot). It would be easier for the reader to decode if the set variable had just one mapping. So in the re-design below we replace the `geom_point` in the second plot with a `geom_segment` with `linetype=set`.

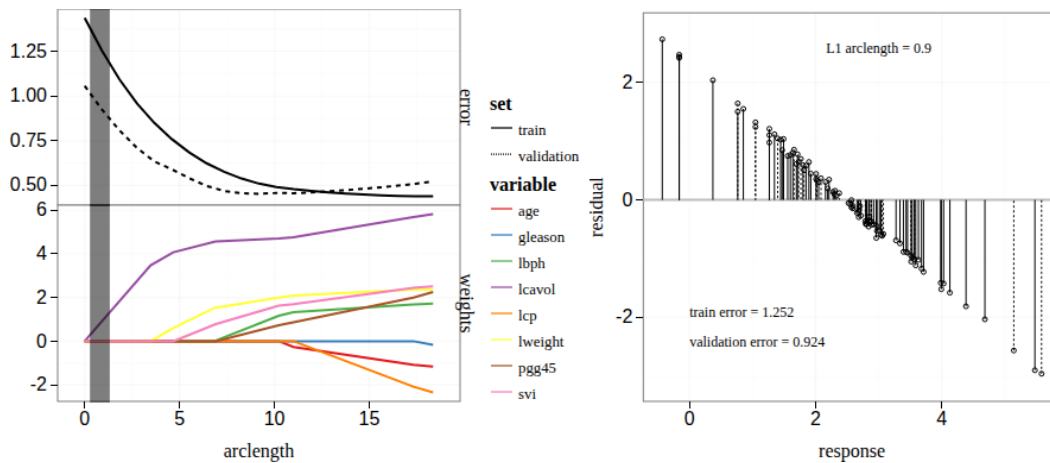
Second, we have replaced the single tallrect in the first plot with two tallrects. The first tallrect has `showSelected=arclength` and is used to show the selected arclength using a grey rectangle. Since we specify a `duration` for the `arclength` variable, and the same `key=1` value, we will observe a smooth transition of the selected grey tallrect. The second tallrect has `clickSelects=arclength` and so clicking it has the effect of changing the selected value of `arclength`. We specify a another data set with more rows, and use the named `clickSelects/showSelected` variables to indicate that `arclength` should also be used as a `showSelected` variable.

```
tallrect.show.list <- list()
for(a in tallrect.dt$arclength){
  is.selected <- tallrect.dt$arclength == a
  not.selected <- tallrect.dt[!is.selected]
  tallrect.show.list[[paste(a)]] <- data.table::data.table(
    not.selected, show.val=a, show.var="arclength")
}
tallrect.show <- do.call(rbind, tallrect.show.list)
animint(
  path=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    ylab("")+
    scale_color_manual(values=variable.colors)+
    geom_line(aes(
      arclength, standardized.coef, color=variable, group=variable),
      data=addY(path, "weights"))+
    geom_line(aes(
      arclength, mse, linetype=set, group=set),
      data=addY(mean.error, "error"))+
    geom_tallrect(aes(
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
      key=1),
      showSelected="arclength",
      alpha=0.5,
      data=tallrect.dt)+
    geom_tallrect(aes(
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
```

```

    key=paste(arclength, show.val)),
    clickSelects="arclength",
    showSelected=c("show.var"="show.val"),
    alpha=0.5,
    data=tallrect.show),
res=ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=residual),
    data=hline.dt,
    color="grey")+
  guides(linetype="none")+
  geom_point(aes(
    response, residual,
    key=observation.i),
    showSelected=c("set", "arclength"),
    shape=21,
    fill=NA,
    color="black",
    data=lasso.res)+
  geom_text(aes(
    3, 2.5, label=sprintf("L1 arclength = %.1f", arclength),
    key=1),
    showSelected="arclength",
    data=tallrect.dt)+
  geom_text(aes(
    0, -2, label=sprintf("train error = %.3f", mse),
    key=1),
    showSelected=c("set", "arclength"),
    hjust=0,
    data=mean.error[set=="train"])+
  geom_text(aes(
    0, -2.5, label=sprintf("validation error = %.3f", mse),
    key=1),
    showSelected=c("set", "arclength"),
    hjust=0,
    data=mean.error[set=="validation"])+
  geom_segment(aes(
    response, residual,
    xend=response, yend=0,
    linetype=set,
    key=observation.i),
    showSelected=c("set", "arclength"),
    size=1,
    data=lasso.res),
duration=list(arclength=2000),
time=list(variable="arclength", ms=2000))

```



11.4 Chapter summary and exercises

We created a visualization of the Lasso machine learning model, which simultaneously shows the regularization path and error curves. Interactivity was used to show details for different values of the regularization parameter.

Exercises:

- Re-make this data viz, including the same visual effect for the tallrects, using only one `geom_tallrect`. Hint: create another data set with `expand.grid(arclength.click=arclength, arclength.show=arclength)`, as in the definition of the `make_tallrect_or_widerect` function.
- Add another scatterplot that shows predicted values versus response, with a `geom_abline` in the background to indicate perfect prediction.
- How would the error curves look if other train/validation splits were chosen? Perform 4-fold cross-validation and add a plot that can be used to select test fold.

Next, [Chapter 12](#) explains how to visualize the Support Vector Machine.



12

Support Vector Machines

This goal of this chapter is to create an interactive data visualization that explains the [Support Vector Machine](#), a machine learning model for binary classification.

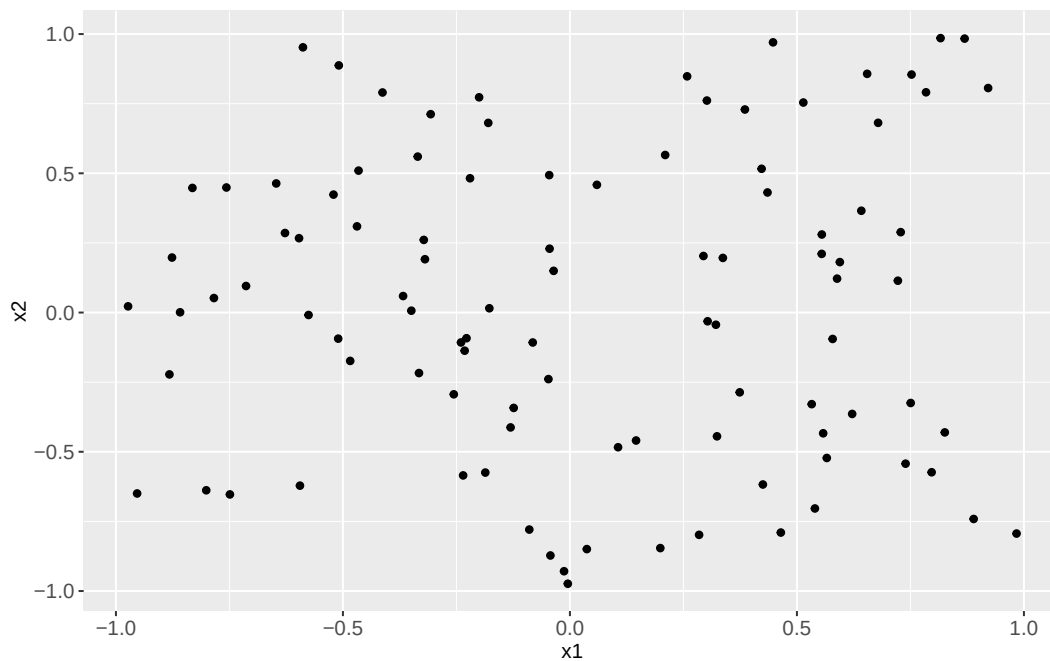
Chapter summary:

- We begin by simulating some data for binary classification in two dimensions, and making some static plots.
 - In the second section, we make an interactive data visualization to show how the linear Support Vector Machine decision boundary changes as a function of the cost hyper-parameter.
 - In the last section, we make an interactive data visualization to show how the decision boundary of the polynomial kernel Support Vector Machine changes as a function of the two hyper-parameters (cost and degree).
-

12.1 Generate and plot some data

We begin by generating two input features, `x1` and `x2`.

```
library(data.table)
N <- 50
set.seed(1)
getInput <- function(){
  c(#rnorm(N, sd=0.3),
    runif(N, -1, 1),
    runif(N, -1, 1)
  )
}
data.dt <- data.table(
  x1=getInput(),
  x2=getInput()
)
library(animint2)
ggplot()+
  geom_point(aes(
    x1, x2,
    data=data.dt
  ))
```



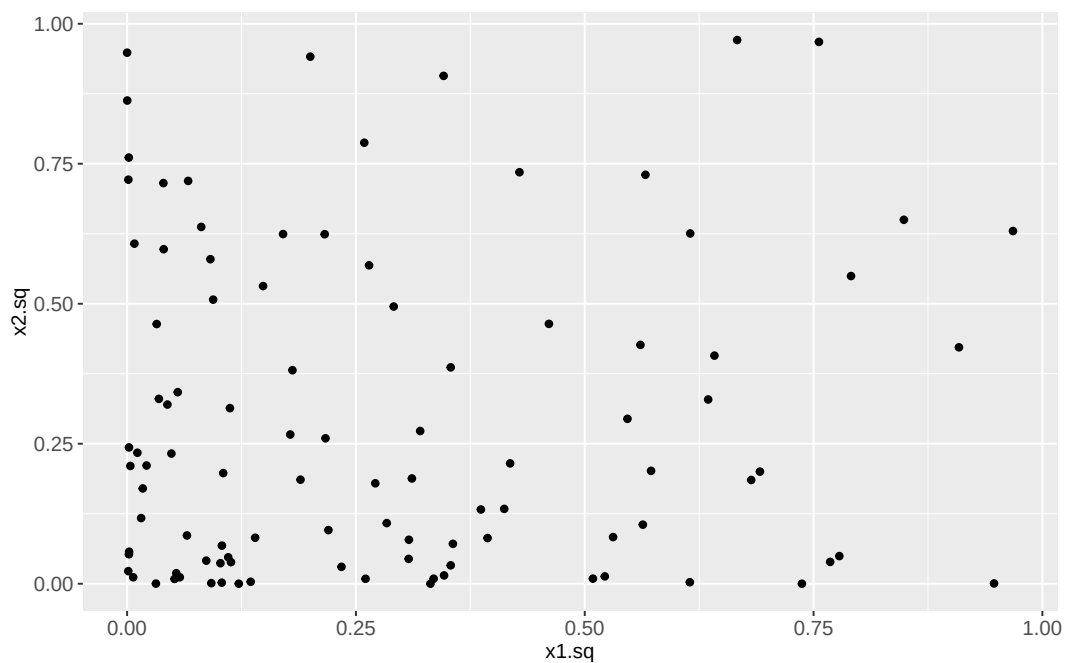
The plot below shows the same data, after computing two additional input features (the squares of the original two inputs).

```
data.dt[, let(
  x1.sq = x1^2,
  x2.sq = x2^2)] []
```

	x1	x2	x1.sq	x2.sq
1:	-0.46898267	0.3094479	0.219944748	0.09575798
2:	-0.25575220	-0.2936055	0.065409188	0.08620416
3:	0.14570673	-0.4594797	0.021230450	0.21112160
4:	0.81641558	0.9853681	0.666534399	0.97095034
5:	-0.59663614	0.2669865	0.355974681	0.07128181

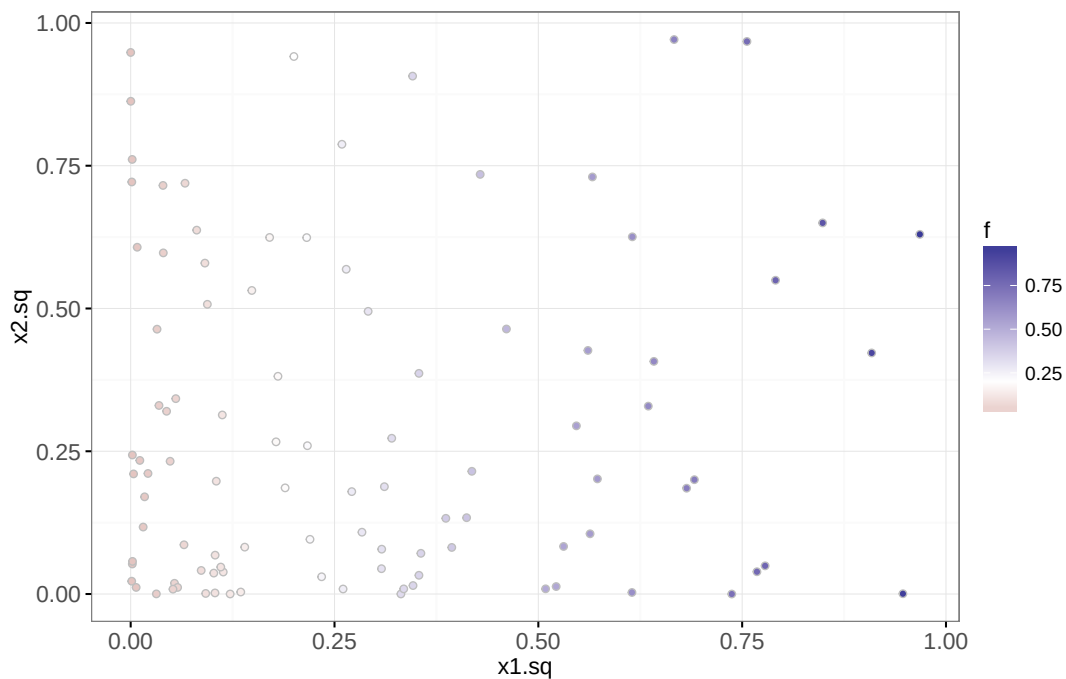
96:	0.59461765	0.1811463	0.353570152	0.03281399
97:	-0.08945109	-0.7792788	0.008001498	0.60727543
98:	-0.17983184	0.6810141	0.032339489	0.46378016
99:	0.62174049	-0.3640726	0.386561231	0.13254888
100:	0.20986658	0.5657027	0.044043982	0.32001952

```
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq),
    data=data.dt)
```



In our simulation, we assume that the output score f is a linear function of $x1.sq$, and ignores $x2.sq$. The plot below visualizes the output scores using the point fill aesthetic.

```
data.dt[, f := x1.sq]
true.decision.boundary <- 0.2
ggplot()+
  theme_bw()+
  scale_fill_gradient2(midpoint=true.decision.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, fill=f),
    shape=21,
    color="grey",
    data=data.dt)
```



In particular, we assume that the label y is negative (-1) if $x1.sq + \text{noise} < \text{threshold}$, and positive (1) otherwise. The plot below visualizes the scores and labels, as a function of the input feature $x1$. It also shows the true score function in black. Of course, we would not be able to make this visualization with real data (only the labels are known in real data, not the scores).

```
data.dt[
  , f.noise := f+rnorm(N, 0, 0.2)
][
  , y.num := ifelse(f.noise<true.decision.boundary, -1, 1)
][
  , y := factor(y.num)
]
table(data.dt$y)
```

```
-1  1
56 44
```

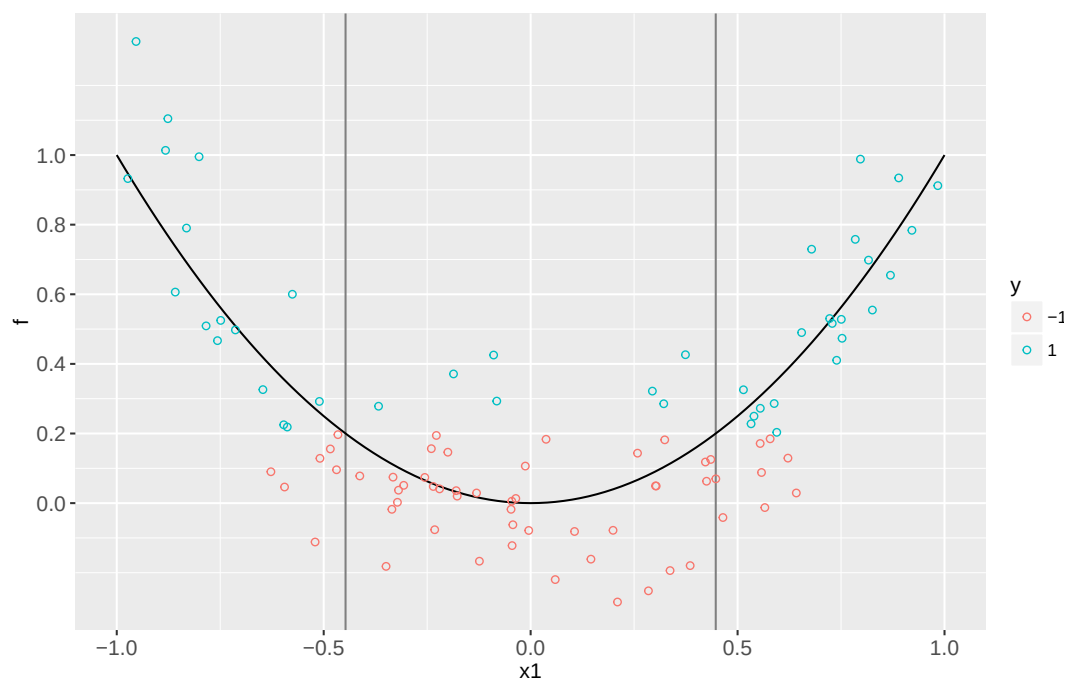
```
scores <- data.table(x1=seq(-1, 1, l=101))[
  , x1.sq := x1^2
][
  , f := x1.sq ]
x1.boundaries <- data.table(
  boundary=c(1, -1)*sqrt(true.decision.boundary))
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
```



```

    color="grey50",
    data=x1.boundaries)+
  geom_line(aes(
    x1, f),
    data=scores)+
  geom_point(aes(
    x1, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)

```

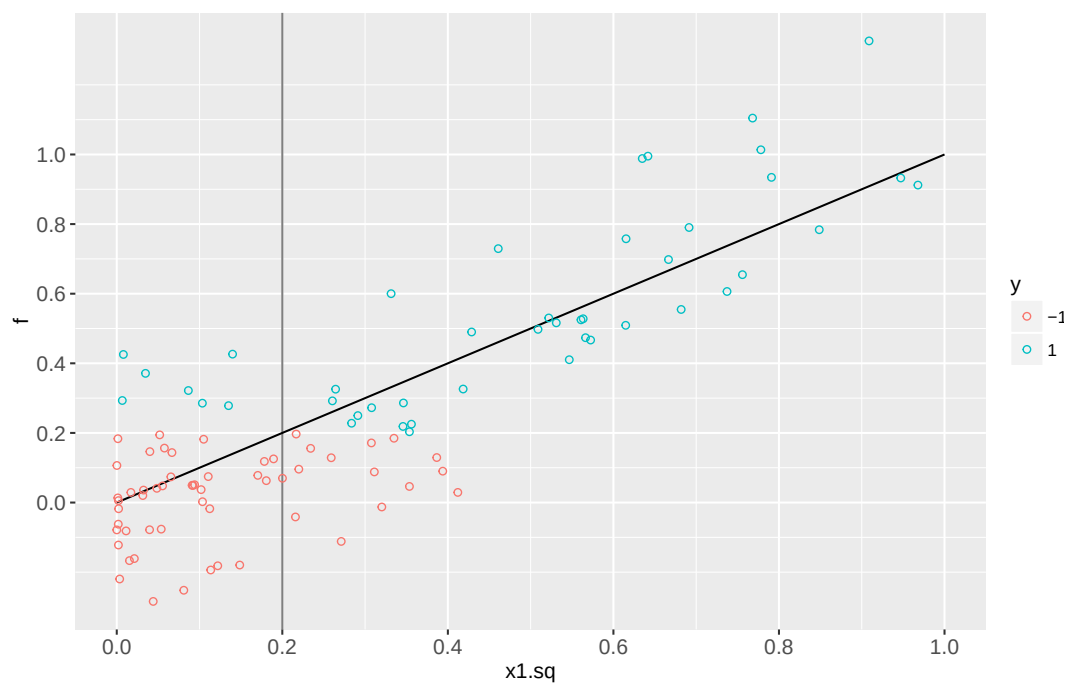


The plot below shows the scores and labels, as a function of the squared feature $x1.sq$. It is clear that the score function that we want to learn is linear in $x1.sq$.

```

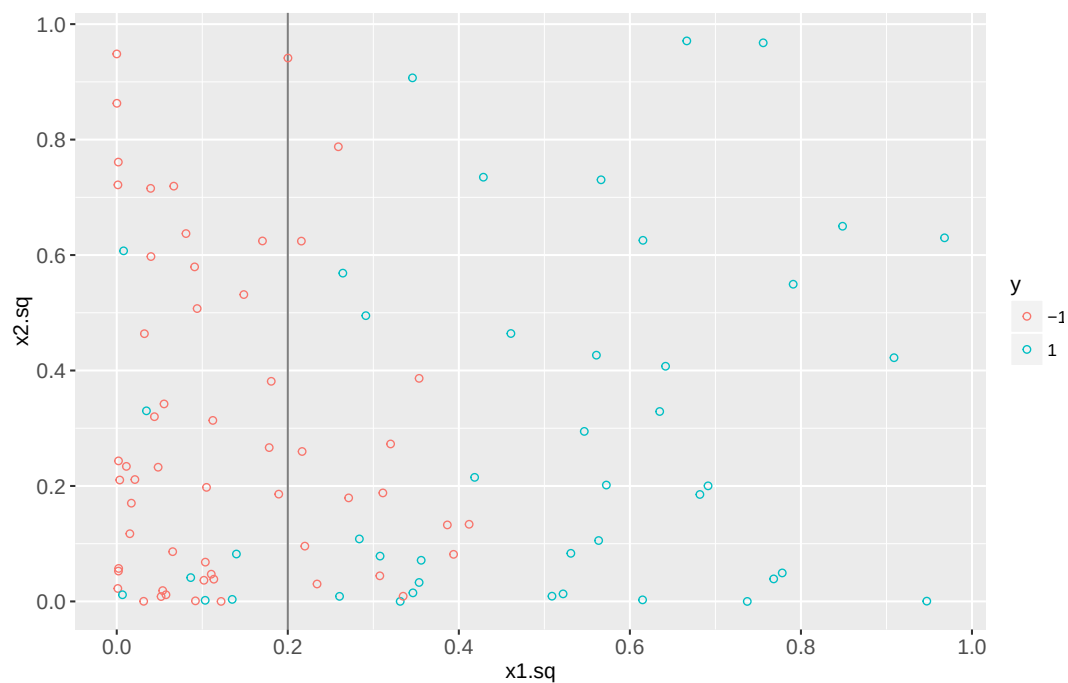
x1sq.boundary <- data.table(boundary=true.decision.boundary)
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_line(aes(x1.sq, f), data=scores)+
  geom_point(aes(
    x1.sq, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)

```



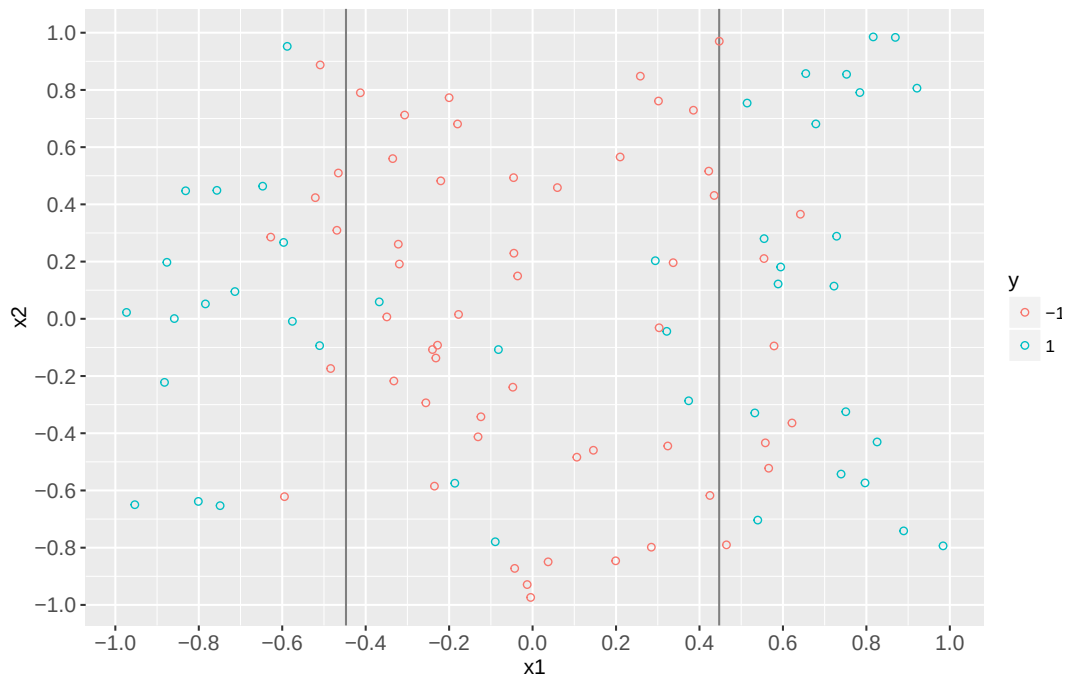
Next, we visualize the labels in the two-dimensional squared feature space. It is clear that the decision boundary is linear in this space.

```
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



The plot below shows the input feature space (x1 and x2). It is clear that the decision boundary is non-linear in x1.

```
ggplot()+  
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+  
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+  
  geom_vline(aes(  
    xintercept=boundary),  
    color="grey50",  
    data=x1.boundaries)+  
  geom_point(aes(  
    x1, x2, color=y),  
    shape=21,  
    fill=NA,  
    data=data.dt)
```

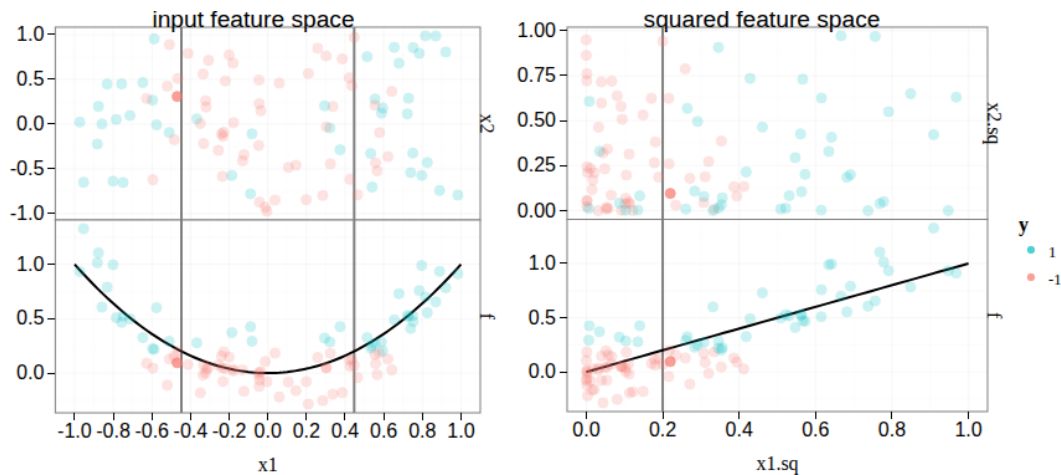


The animint below uses `clickSelects` to show which points in the input and squared space correspond. We just need to create an `data.i` variable that has a unique ID for each data point.

```
data.dt[, data.i := 1:N]
YVAR <- function(dt, y.var){
  dt$y.var <- factor(y.var, c("x2", "x2.sq", "f"))
  dt
}
animint(
  input=ggplot()+
    ggtitle("input feature space")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
    ylab("")+
    guides(color="none")+
    geom_vline(aes(
      xintercept=boundary),
      color="grey50",
      data=x1.boundaries)+
    geom_point(aes(
      x1, x2, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "x2"))+

```

```
geom_line(aes(
  x1, f),
  data=YVAR(scores, "f"))+
geom_point(aes(
  x1, f.noise, color=y),
  clickSelects="data.i",
  size=4,
  alpha=0.7,
  data=YVAR(data.dt, "f")),
square=ggplot()+
  ggtitle("squared feature space")+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y.var ~ ., scales="free")+
  ylab("")+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    clickSelects="data.i",
    size=4,
    alpha=0.7,
    data=YVAR(data.dt, "x2.sq"))+
  geom_line(aes(
    x1.sq, f),
    data=YVAR(scores, "f"))+
  geom_point(aes(
    x1.sq, f.noise, color=y),
    clickSelects="data.i",
    size=4,
    alpha=0.7,
    data=YVAR(data.dt, "f")))
```



Note how we used two multi-panel plots with the `addColumn` then `facet` idiom, rather than creating four separate plots. This emphasizes the fact that some plots/facets have a common `x1` or `x1.sq` axis. Note that we also hid the color legend in the first plot, since it is sufficient to just have one color legend.

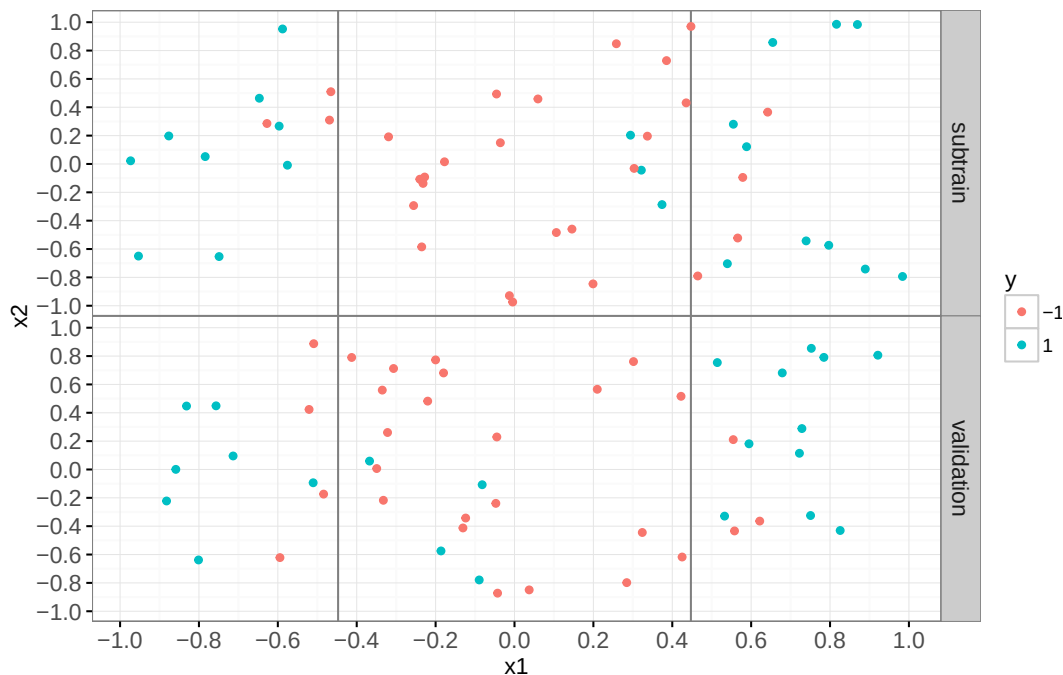
12.2 Linear SVM

```
train.i <- 1:N
data.dt[
  , set := "validation"
][
  train.i, set := "subtrain"
]
table(data.dt$set)
```

```
subtrain validation
      50         50
```

```
subtrain.dt <- data.dt[set=="subtrain",]
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(set ~ .)+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
  geom_point(aes(
    x1, x2, color=y),
```

```
data=data.dt)
```



We begin by fitting a linear SVM to the train data in the squared feature space, and visualizing the true labels `y` along with the predicted labels `pred.y`.

```
library(kernlab)
```

Attaching package: 'kernlab'

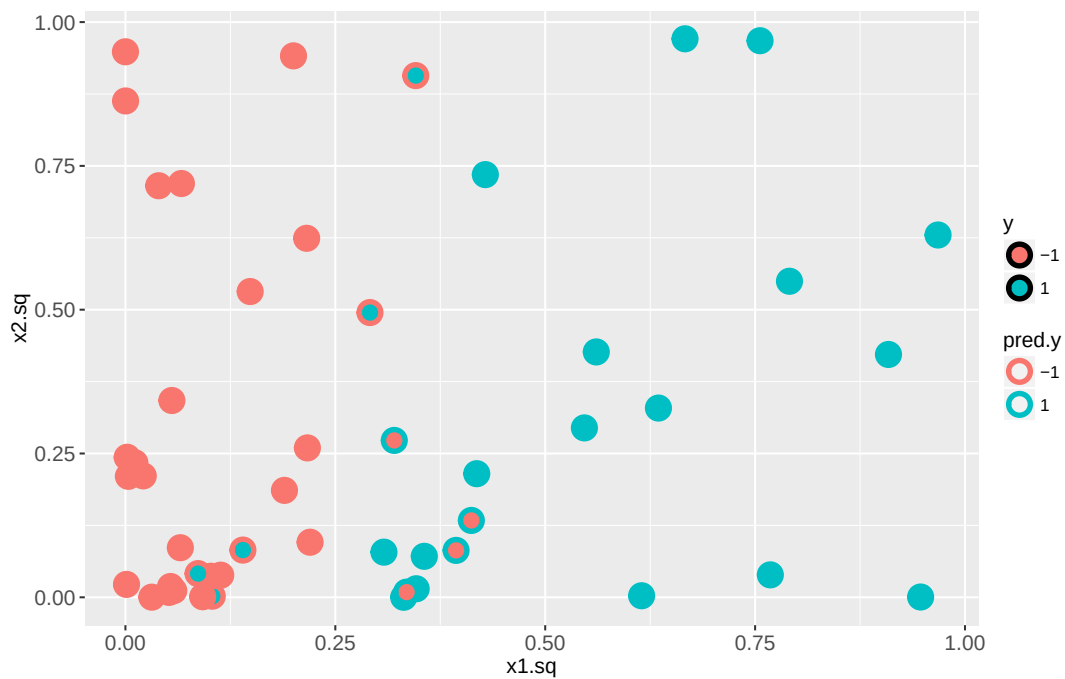
The following object is masked from 'package:animint2':

alpha

```
squared.mat <- subtrain.dt[, cbind(x1.sq, x2.sq)]
y.vec <- subtrain.dt$y
fit <- ksvm(squared.mat, y.vec, kernel="vanilladot")
```

Setting default kernel parameters

```
subtrain.dt$pred.y <- predict(fit)
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq, color=pred.y, fill=y),
    shape=21,
    size=4,
    stroke=2,
    data=subtrain.dt)
```



It is clear from the plot above that there are several mis-classified train data points. In the plot below we visualize the decision boundary and margin.

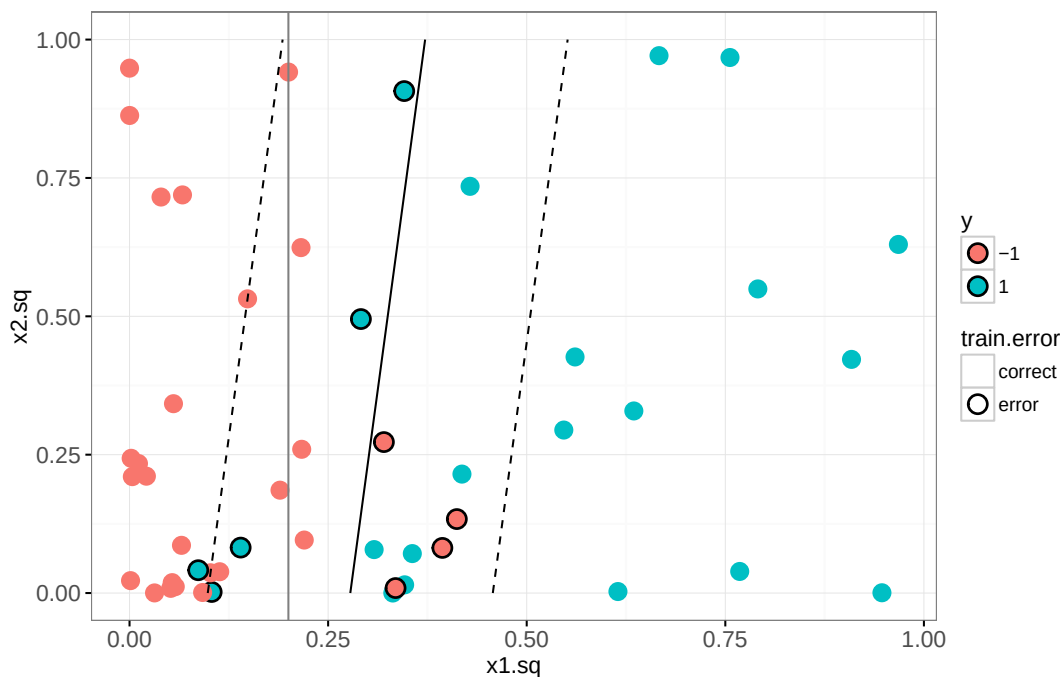
```
predF <- function(fit, X){
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  X.sc <- scale(X, mu, sigma)
  kernelMult(
    kernelf(fit),
    X.sc,
    xmatrix(fit)[[1]],
    coef(fit)[[1]])-b(fit)
}
xsq.vec <- seq(0, 1, l=41)
grid.sq.dt <- data.table(expand.grid(
  x1.sq=xsq.vec,
  x2.sq=xsq.vec
))[,
  pred.f := predF(fit, cbind(x1.sq, x2.sq))]
subtrain.dt[, train.error := ifelse(y==pred.y, "correct", "error")]
ggplot()+
  theme_bw()+
```



```

scale_color_manual(values=c(error="black", correct=NA))+
geom_point(aes(
  x1.sq, x2.sq, fill=y, color=train.error),
  shape=21,
  stroke=1,
  size=4,
  data=subtrain.dt)+
geom_vline(aes(
  xintercept=boundary), color="grey50",
  data=x1sq.boundary)+
geom_contour(aes(
  x1.sq, x2.sq, z=pred.f),
  breaks=0,
  color="black",
  data=grid.sq.dt)+
geom_contour(aes(
  x1.sq, x2.sq, z=pred.f),
  breaks=c(-1, 1),
  color="black",
  linetype="dashed",
  data=grid.sq.dt)

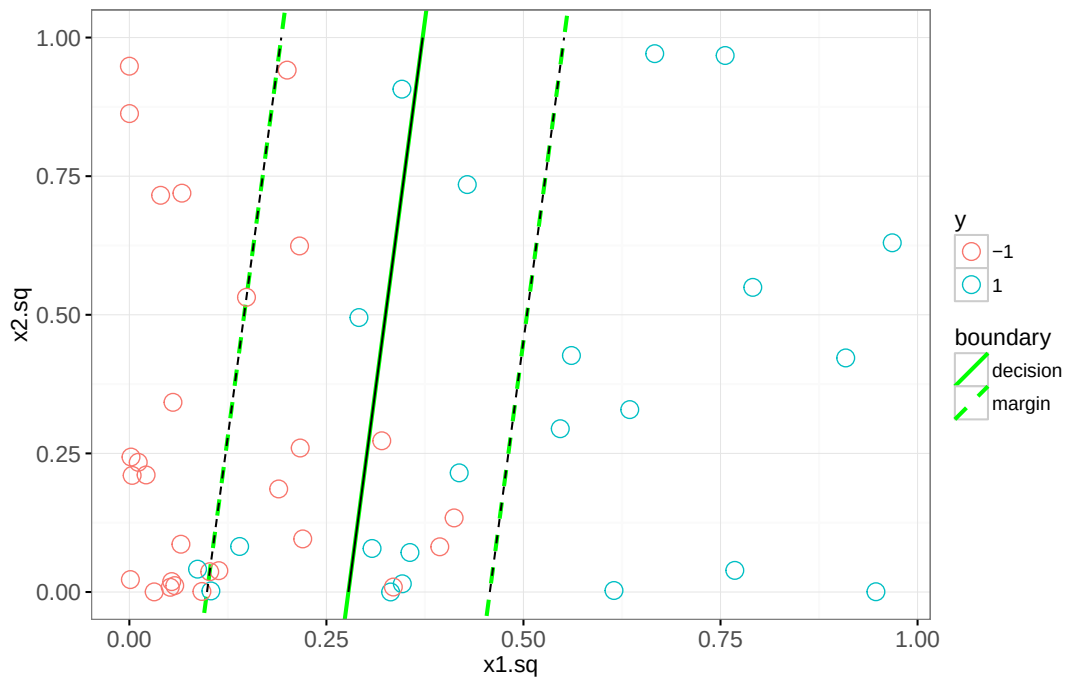
```



The plot above shows the true decision boundary using a grey `vline`. It also uses `geom_contour` to display the decision boundary (solid black line, predicted score 0) and the margin (dashed black line, predicted score -1 and 1). Since the decision boundary and margin are linear in this space, we can also use `geom_abline` to display them. To do that we need to do some math, and work out the equations for the slope and intercepts of those lines (as a function of the learned bias `b(fit)` and `weight.vec`, as well as the scale parameters

mu and sigma).

```
## The equation of the margin lines is  $x_2 = m_2 + s_2/w_2[c+b+w_1*m_1/s_1]$ 
##  $-s_2*w_1/(w_2*s_1)*x_1$  for  $c=1$  and  $-1$ .  $x$  is input feature,  $m$  is mean,  $s$ 
## is scale,  $w$  is learned weight.
fit.sc <- scaling(fit)$x.scale
if(is.null(fit.sc)){
  fit.sc <- list(
    "scaled:center"=c(0,0),
    "scaled:scale"=c(1,1))
}
mu <- fit.sc[["scaled:center"]]
sigma <- fit.sc[["scaled:scale"]]
weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
predF.linear <- function(fit, X){
  X.sc <- scale(X, mu, sigma)
  X.sc %*% weight.vec - b(fit)
}
abline.dt <- data.table(
  y=factor(c(-1,0,1)),
  boundary=c("margin", "decision", "margin"),
  intercept=mu[2]+sigma[2]/weight.vec[2]*(
    c(-1, 0, 1)+b(fit)+weight.vec[1]*mu[1]/sigma[1]),
  slope=-weight.vec[1]*sigma[2]/(weight.vec[2]*sigma[1]))
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
    color="green",
    size=1,
    data=abline.dt)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    size=4,
    data=subtrain.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=0,
    color="black",
    data=grid.sq.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=c(-1, 1),
    color="black",
    linetype="dashed",
    data=grid.sq.dt)
```



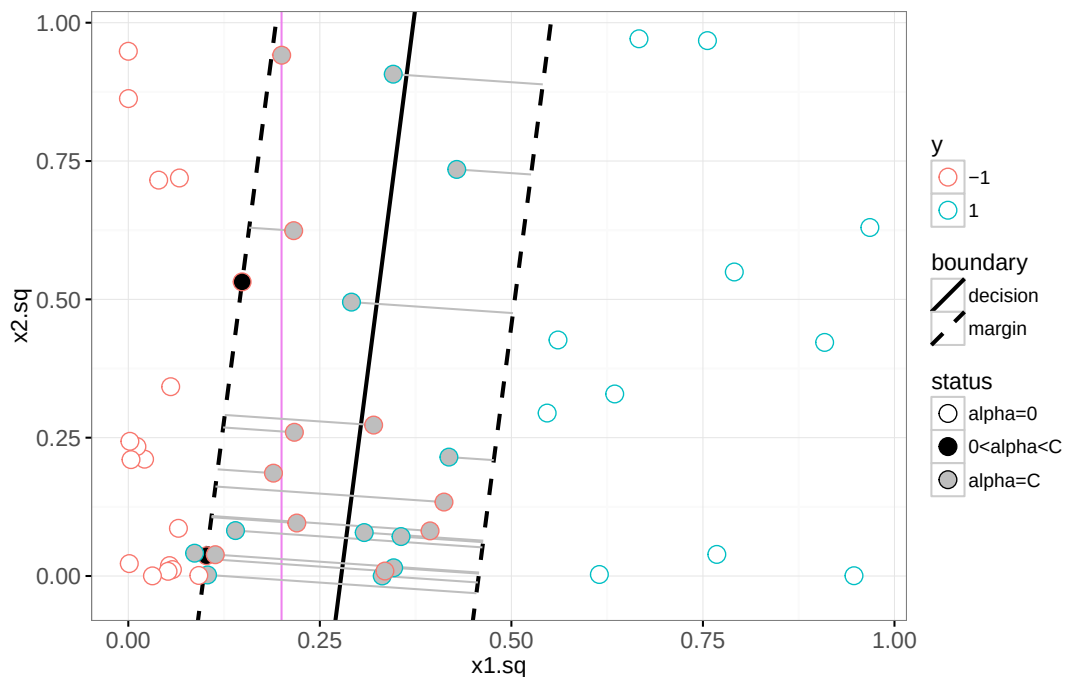
The plot above confirms that our computation of the slope and intercepts (green lines) agrees with the contours (black lines). In the plot below, we show the learned `alpha` coefficients, and add a `geom_segment` to visualize the slack.

```
subtrain.dt[, alpha := 0]
train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
subtrain.dt[, status := ifelse(
  alpha==0, "alpha=0",
  ifelse(alpha==1, "alpha=C", "0<alpha<C"))]
slack.slope <- weight.vec[2]*sigma[1]/(weight.vec[1]*sigma[2])
slack.dt <- subtrain.dt[alpha==1,]
slack.join <- abline.dt[slack.dt, on=list(y)]
slack.join[, x1.sq.margin := (
  x2.sq-slack.slope*x1.sq-intercept)/(slope-slack.slope)]
slack.join[, x2.sq.margin := slope*x1.sq.margin + intercept]
sv.colors <- c(
  "alpha=0"="white",
  "0<alpha<C"="black",
  "alpha=C"="grey")
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary), color="violet",
    data=x1sq.boundary)+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
```

```

    size=1,
    data=abline.dt)+
  geom_segment(aes(
    x1=sq, x2=sq,
    xend=x1.sq.margin, yend=x2.sq.margin),
    color="grey",
    data=slack.join)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  geom_point(aes(
    x1=sq, x2=sq, color=y, fill=status),
    shape=21,
    size=4,
    data=subtrain.dt)

```



The plot above shows the slack in grey segments, and the decision and margin lines in black. The Bayes decision boundary is shown in the background as a vertical violet line. The support vectors are the points with non-zero alpha coefficients. Black filled support vectors are on the margin, and grey support vectors are on the wrong side of the margin (and have non-zero slack). The plot below shows the model that was learned in the original feature space,

```

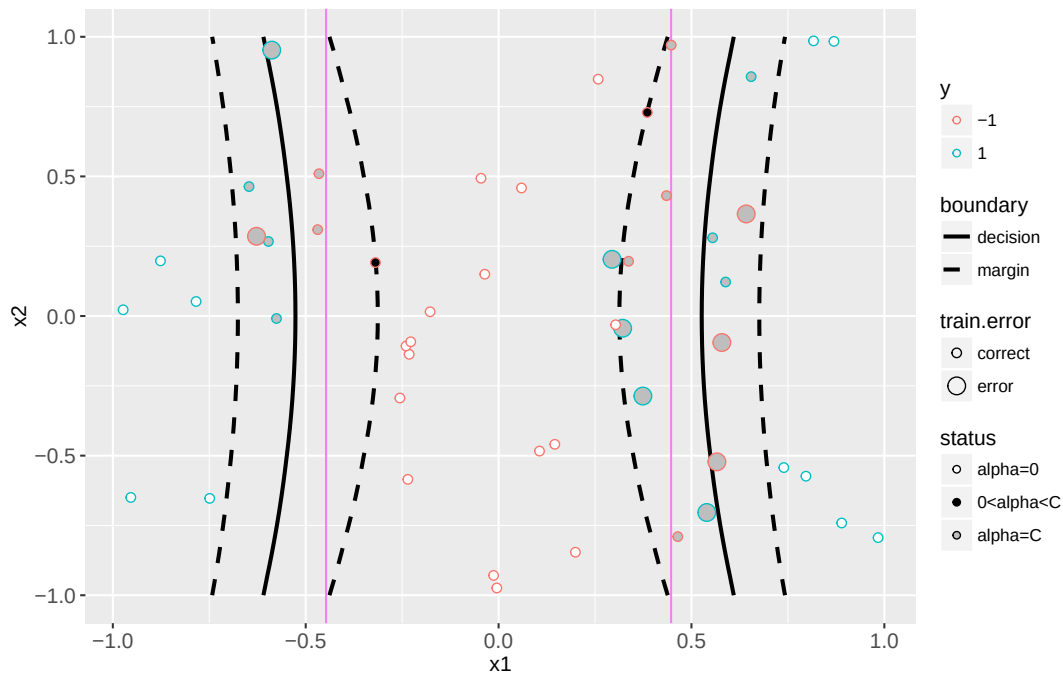
n.grid <- 41
x.vec <- seq(-1, 1, l=n.grid)
grid.dt <- data.table(expand.grid(
  x1=x.vec,
  x2=x.vec))
getBoundaryDF <- function(score.vec, level.vec=c(-1, 0, 1)){

```

```

stopifnot(length(score.vec) == n.grid * n.grid)
several.paths <- contourLines(
  x.vec, x.vec,
  matrix(score.vec, n.grid, n.grid),
  levels=level.vec)
contour.list <- list()
for(path.i in seq_along(several.paths)){
  contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
    path.i,
    level.num=as.numeric(level),
    level.fac=factor(level, level.vec),
    boundary=ifelse(level==0, "decision", "margin"),
    x1=x, x2=y))
}
do.call(rbind, contour.list)
}
grid.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
boundaries <- grid.dt[, getBoundaryDF(pred.f)]
ggplot()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary),
    color="violet",
    data=x1.boundaries)+
  geom_path(aes(
    x1, x2, group=path.i, linetype=boundary),
    size=1,
    data=boundaries)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  scale_size_manual(values=c(correct=2, error=4))+
  geom_point(aes(
    x1, x2, color=y,
    size=train.error,
    fill=status),
    shape=21,
    data=subtrain.dt)

```



The goal below will be to make an animint that shows how the decision boundary, margin, and slack change as a function of the cost parameter.

```

modelInfo.list <- list()
predictions.list <- list()
slackSegs.list <- list()
modelLines.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
for(cost.param in round(10^seq(-1, 1, by=cost.by),1)){
  fit <- ksvm(
    squared.mat, y.vec, kernel="vanilladot", scaled=FALSE, C=cost.param)
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
  grid.sq.dt[, pred.f := predF(fit, cbind(x1.sq, x2.sq))]
  data.dt[, pred.y := predict(fit, cbind(x1.sq, x2.sq))]
  one.error <- data.dt[, list(errors=sum(y!=pred.y)), by=set]
  setErrors.list[[paste(cost.param)] <- data.table(
    cost.param, one.error)
  subtrain.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
}

```

[illegible]

```

inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
slackSegs <- do.call(rbind, slackSegs.list)
modelLines <- do.call(rbind, modelLines.list)
modelInfo <- do.call(rbind, modelInfo.list)
setErrors <- do.call(rbind, setErrors.list)
modelInfo.tall <- melt(modelInfo, id.vars="cost.param")
grid.sq.dt$boundary <- "true"
setErrors$variable <- "errors"
inputBoundaries[, boundary := ifelse(level.num==0, "decision", "margin")]
slackSegs$boundary <- "margin"
set.label.select <- data.table(
  cost.param=range(setErrors$cost.param),
  set=c("validation", "subtrain"),
  hjust=c(1, 0))
set.labels <- setErrors[set.label.select, on=list(cost.param, set)]
viz.linear.svm <- animint(
  selectModel=ggplot()+
    ggtitle("Select regularization parameter")+
    scale_x_continuous(limits=c(-1.5, 1.5))+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=modelInfo)+
    theme_bw()+
    facet_grid(variable ~ ., scales="free")+
    geom_line(aes(
      log10(cost.param), errors,
      group=set, color=set),
      data=setErrors)+
    geom_text(aes(
      log10(cost.param), errors-1, label=set,
      hjust=hjust,
      color=set),
      data=set.labels)+
    guides(color="none")+
    geom_line(aes(
      log10(cost.param), log10(value)),
      data=modelInfo.tall),
  inputSpace=ggplot()+
    ggtitle("Input space features")+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_vline(aes(
      xintercept=boundary,
      color="violet",
      data=x1.boundaries)+
    guides(color="none", fill="none", linetype="none")+

```



```

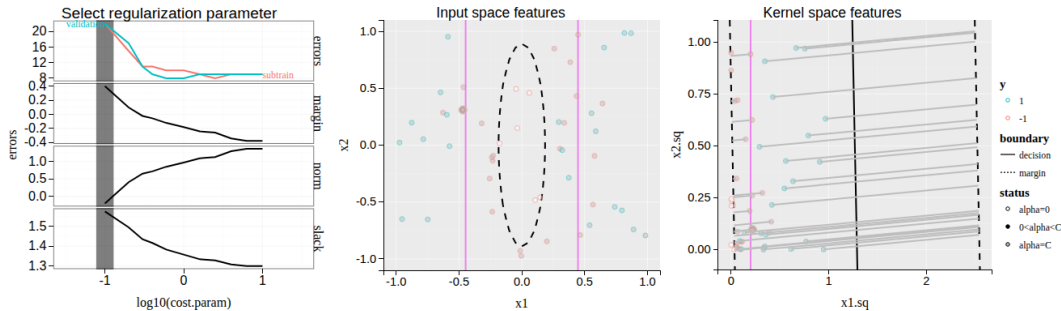
scale_linetype_manual(values=c(
  "-1"="dashed",
  "0"="solid",
  "1"="dashed"))+
geom_path(aes(
  x1, x2,
  group=path.i,
  linetype=level.fac),
  showSelected=c("boundary", "cost.param"),
  color="black",
  data=inputBoundaries)+
geom_point(aes(
  x1, x2, fill=status),
  showSelected=c("status", "y", "data.i", "cost.param"),
  size=5,
  color="grey",
  data=predictions)+
geom_point(aes(
  x1, x2, color=y, fill=status),
  showSelected=c("cost.param", "status", "y"),
  clickSelects="data.i",
  size=3,
  data=predictions),
kernelSpace=ggplot()+
  ggtitle("Kernel space features")+
  geom_vline(aes(
    xintercept=boundary), color="violet",
    data=x1sq.boundary)+
  ##coord_cartesian(xlim=c(0, 1), ylim=c(0, 1))+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
    showSelected="cost.param",
    color="black",
    data=modelLines)+
scale_linetype_manual(values=c(
  decision="solid",
  margin="dashed",
  true="solid"))+
geom_point(aes(
  x1.sq, x2.sq, fill=status),
  showSelected=c("data.i", "cost.param"),
  size=5,
  color="grey",
  data=predictions)+
geom_point(aes(
  x1.sq, x2.sq, color=y, fill=status),
  clickSelects="data.i",
  showSelected="cost.param",
  size=3,

```

```

    data=predictions)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  geom_segment(aes(
    x1=sq, x2=sq,
    xend=x1.sq.margin, yend=x2.sq.margin),
    showSelected=c("cost.param", "boundary"),
    color="grey",
    data=slackSegs))
viz.linear.svm

```



12.3 Non-linear polynomial kernel SVM

In the previous section we fit a linear kernel in the squared feature space, which resulted in learning a function which is non-linear in terms of the original feature space. In this section we directly fit a non-linear polynomial kernel in the original space.

```

predictions.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
orig.mat <- subtrain.dt[, cbind(x1, x2)]
for(cost.param in 10^seq(-1, 3, by=cost.by)){
  for(degree.num in seq(1, 6, by=1)){
    k <- polydot(degree.num, offset=0)
    fit <- ksvm(
      orig.mat, y.vec, kernel=k, scaled=FALSE, C=cost.param)
    grid.dt[, pred.f := predF(fit, cbind(x1, x2))]
    grid.dt[, pred.y := predict(fit, cbind(x1, x2))]
    grid.dt[, stopifnot(sign(pred.f) == pred.y)]
    data.dt[, pred.y := predict(fit, cbind(x1, x2))]
    one.error <- data.dt[, list(errors=sum(y != pred.y)), by=set]
    setErrors.list[[paste(cost.param, degree.num)]] <- data.table(
      cost.param, degree.num, one.error)
    boundaries <- getBoundaryDF(grid.dt$pred.f)
    if(is.data.frame(boundaries) && nrow(boundaries)){

```

```

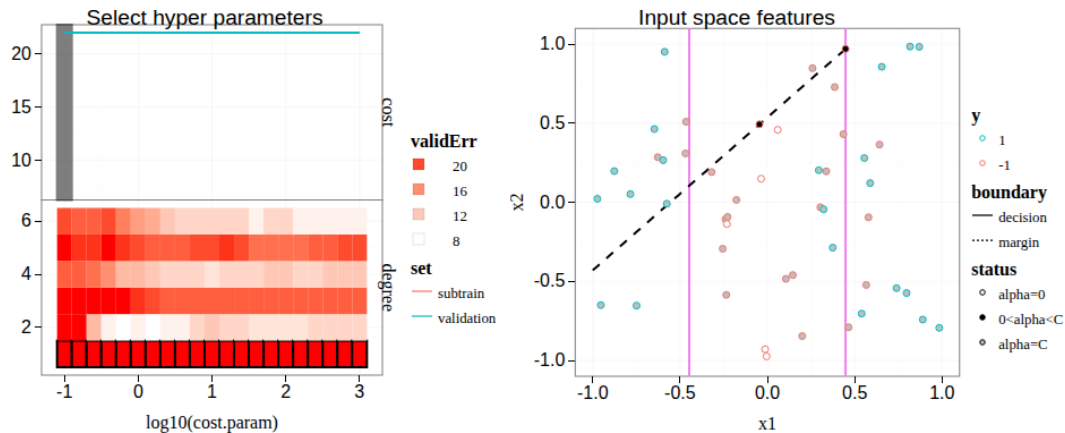
    cost.deg <- paste(cost.param, degree.num)
    inputBoundaries.list[[cost.deg]] <- data.table(
      cost.param, degree.num, boundaries)
  }
  subtrain.dt[, alpha := 0]
  train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
  subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
  subtrain.dt[, status := ifelse(
    alpha==0, "alpha=0",
    ifelse(alpha==cost.param, "alpha=C", "0<alpha<C"))]
  predictions.list[[paste(cost.param, degree.num)]] <- data.table(
    cost.param, degree.num, subtrain.dt)
}
}
inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
setErrors <- do.call(rbind, setErrors.list)
validationErrors <- setErrors[set=="validation"]
validationErrors$select <- "degree"
setErrors$select <- "cost"
animint(
  selectModel=ggplot()+
    ggtitle("Select hyper parameters")+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=setErrors[degree.num==1 & set=="subtrain",])+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  theme_animint(width=350, rowspan=1)+
  facet_grid(select ~ ., scales="free")+
  ylab("")+
  geom_line(aes(
    log10(cost.param), errors,
    key=set,
    group=set,
    color=set),
    showSelected="degree.num",
    data=setErrors)+
  scale_fill_gradient("validErr", low="white", high="red")+
  geom_tile(aes(
    log10(cost.param), degree.num, fill=errors),
    clickSelects="degree.num",
    data=validationErrors),
  inputSpace=ggplot()+
  theme_bw()+
  ggtitle("Input space features")+

```

```

scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
geom_vline(aes(
  xintercept=boundary),
  color="violet",
  data=x1.boundaries)+
scale_linetype_manual(values=c(
  margin="dashed",
  decision="solid"))+
geom_path(aes(
  x1, x2,
  group=path.i,
  linetype=boundary),
  showSelected=c("degree.num", "cost.param"),
  color="black",
  data=inputBoundaries)+
geom_point(aes(
  x1, x2, color=y, fill=status),
  showSelected=c("cost.param", "degree.num"),
  size=3,
  data=predictions))

```



12.4 Chapter summary and exercises

We used ggplots to visualize the Support Vector Machine model for binary classification. We used animint and interactivity to show how the SVM decision boundary changes as a function of the model hyper-parameters.

Exercises:

- Use HTML table layout for `viz.linear.svm`, so that the two feature space plots appear beside each other, and the “Select regularization parameter” plot appears above or below.
- Use `rbfdot` as the kernel function. Compute subtrain and validation error, then add a new panel to the “select hyper parameters” plot.
- Default scales use the same two colors for the `y` and `set` legends, which could be confusing.

Change the colors in one of the two legends so that they are different.

- Use `color` and `color_off` parameters to change the appearance of the `geom_tile` when selected or not, as explained in Chapter 6, section Specifying how selection state is displayed.

Next, [Chapter 13](#) explains how to visualize the Poisson regression model.



13

Poisson regression

This goal of this chapter is to create an interactive data visualization that explains [Poisson regression](#), a machine learning model for predicting an integer-valued output from inputs that are real-valued vectors. This is a “linear regression” model since it learns a linear function from the inputs to the output. Like least squares regression, Poisson regression can be formulated as a maximum likelihood problem. However, it differs from least squares linear regression since it uses a Poisson distribution to model the output labels, instead of a Gaussian distribution. This modeling choice is appropriate when output labels are non-negative integers.

Chapter outline:

- We begin by creating a plot that shows the probability mass function for a Poisson distribution mean parameter that can be interactively selected.
- We then add a second panel that shows the cumulative distribution function.
- We then add a second plot which shows the Poisson loss, with a second selector for label value.

13.1 Plot the probability mass function and select the Poisson mean parameter

The goal of this section is to create a data visualization that shows the probability mass function for a selected Poisson mean parameter.

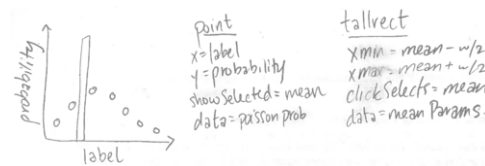


Figure 13.1: Poisson regression viz prob

```
library(data.table)
poisson.mean.diff <- 0.25
poisson.mean.vec <- seq(0, 5, by=poisson.mean.diff)
quantile.max <- 0.99
poisson.prob.list <- list()
for(poisson.mean in poisson.mean.vec){
```

```

label.max <- qpois(quantile.max, poisson.mean)
label <- 0:label.max
probability <- dpois(label, poisson.mean)
poisson.prob.list[[paste(poisson.mean)]] <- data.table(
  poisson.mean,
  label,
  probability,
  cum.prob=cumsum(probability))
}
poisson.prob <- do.call(rbind, poisson.prob.list)
poisson.prob

```

	poisson.mean	label	probability	cum.prob
1:	0.00	0	1.000000000	1.0000000
2:	0.25	0	0.778800783	0.7788008
3:	0.25	1	0.194700196	0.9735010
4:	0.25	2	0.024337524	0.9978385
5:	0.50	0	0.606530660	0.6065307

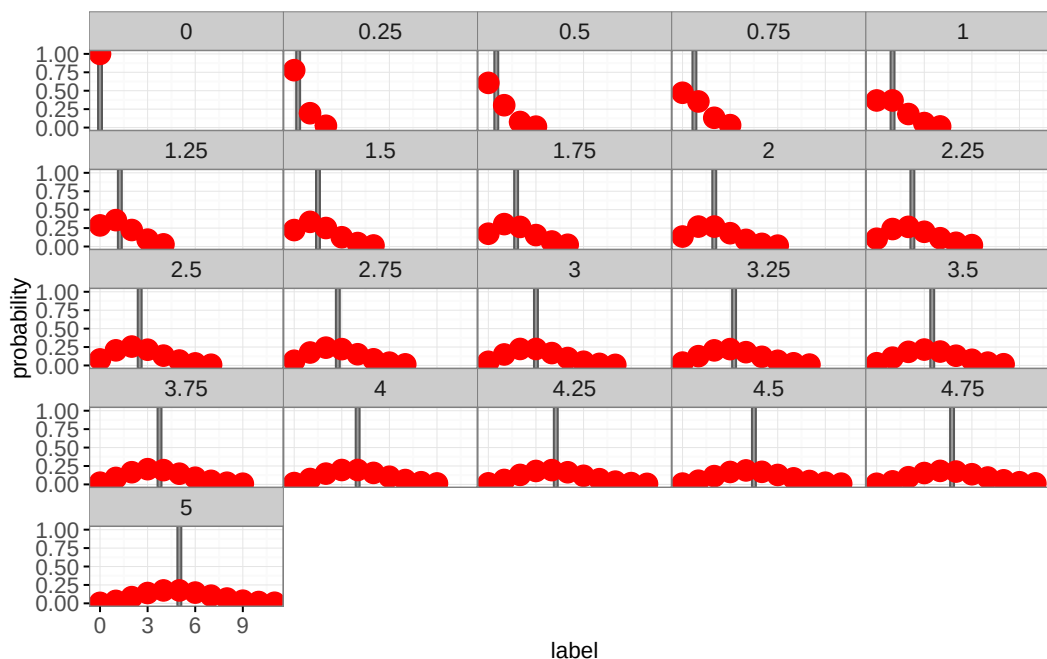
152:	5.00	7	0.104444863	0.8666283
153:	5.00	8	0.065278039	0.9319064
154:	5.00	9	0.036265577	0.9681719
155:	5.00	10	0.018132789	0.9863047
156:	5.00	11	0.008242177	0.9945469

The static data viz below shows one facet for each Poisson distribution.

```

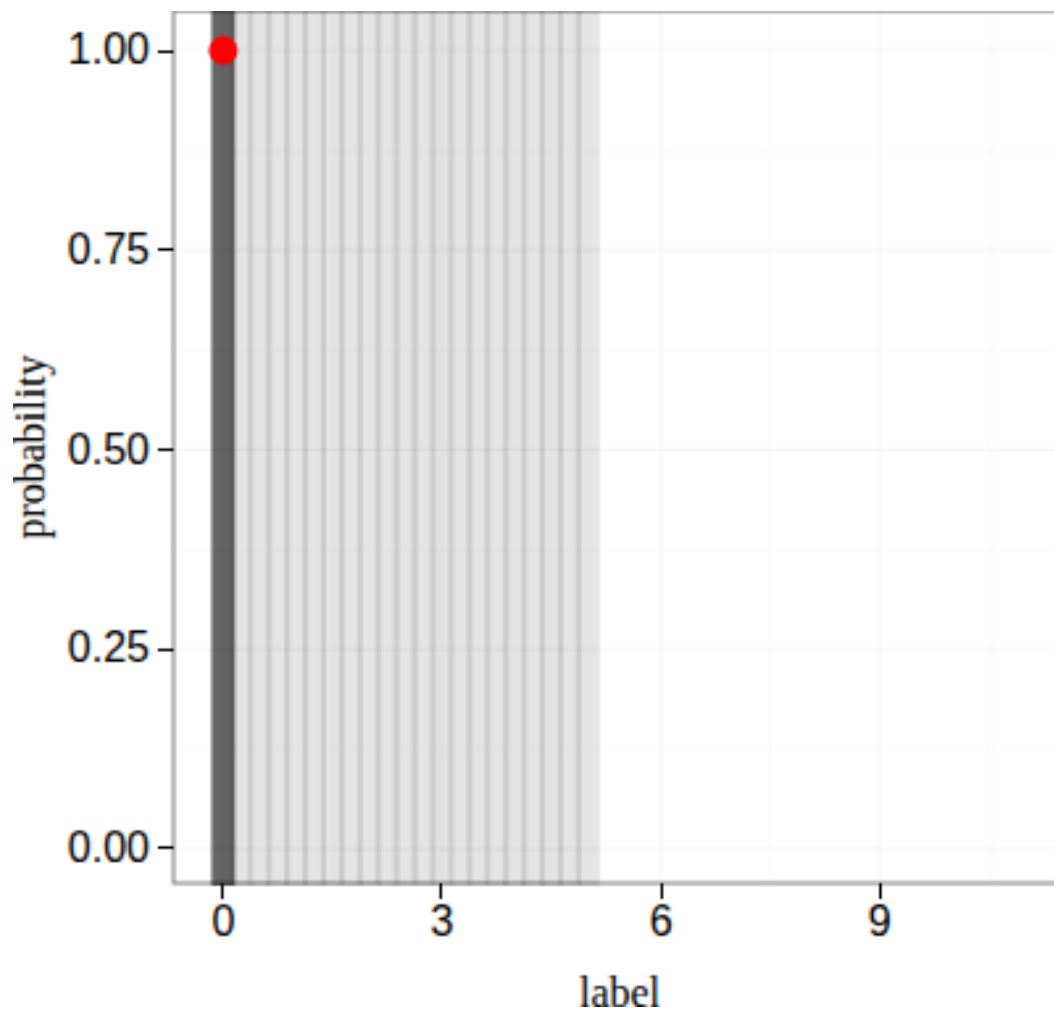
mean.tallrects <- data.table(
  poisson.mean=poisson.mean.vec,
  min=poisson.mean.vec - poisson.mean.diff/2,
  max=poisson.mean.vec + poisson.mean.diff/2)
library(animint2)
prob.mass <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  geom_tallrect(aes(
    xmin=min, xmax=max),
    clickSelects="poisson.mean",
    alpha=0.6,
    data=mean.tallrects)+
  geom_point(aes(
    label, probability,
    tooltip=sprintf("prob(label = %d) = %f", label, probability)),
    color="red",
    showSelected="poisson.mean",
    size=5,
    data=poisson.prob)
prob.mass+
  facet_wrap("poisson.mean")

```

Note that we used `alpha=0.6` with `geom_tallrect`, which means that the tallrect for the selected mean has 0.6 opacity, and the other tallrects have 0.1 opacity. Note also that we used `usecolor="red"` and `size=5` with `geom_point` so that it is easier to see the points on a grey background, and to hover the cursor over the points to see the tooltip. We next create an interactive version with `animint`.

```
animint(prob.mass)
```



You can click the viz above to change the mean of the Poisson distribution. You can also hover the cursor over a data point to see its probability. Note that for integer values of the Poisson mean, there are two labels that are the most probable (the mode of the Poisson distribution). For example the Poisson distribution with a mean of 3 attains its maximum probability of about 0.224 at label values of 2 and 3.

13.2 Add a panel for the cumulative distribution function

To add a panel for the cumulative distribution function, we will re-make the ggplot based on the sketch below.

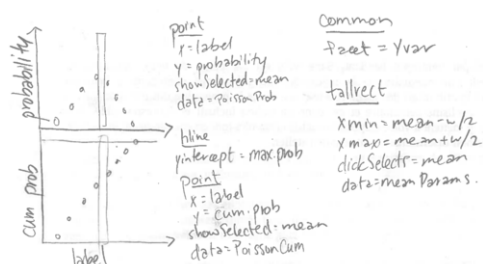
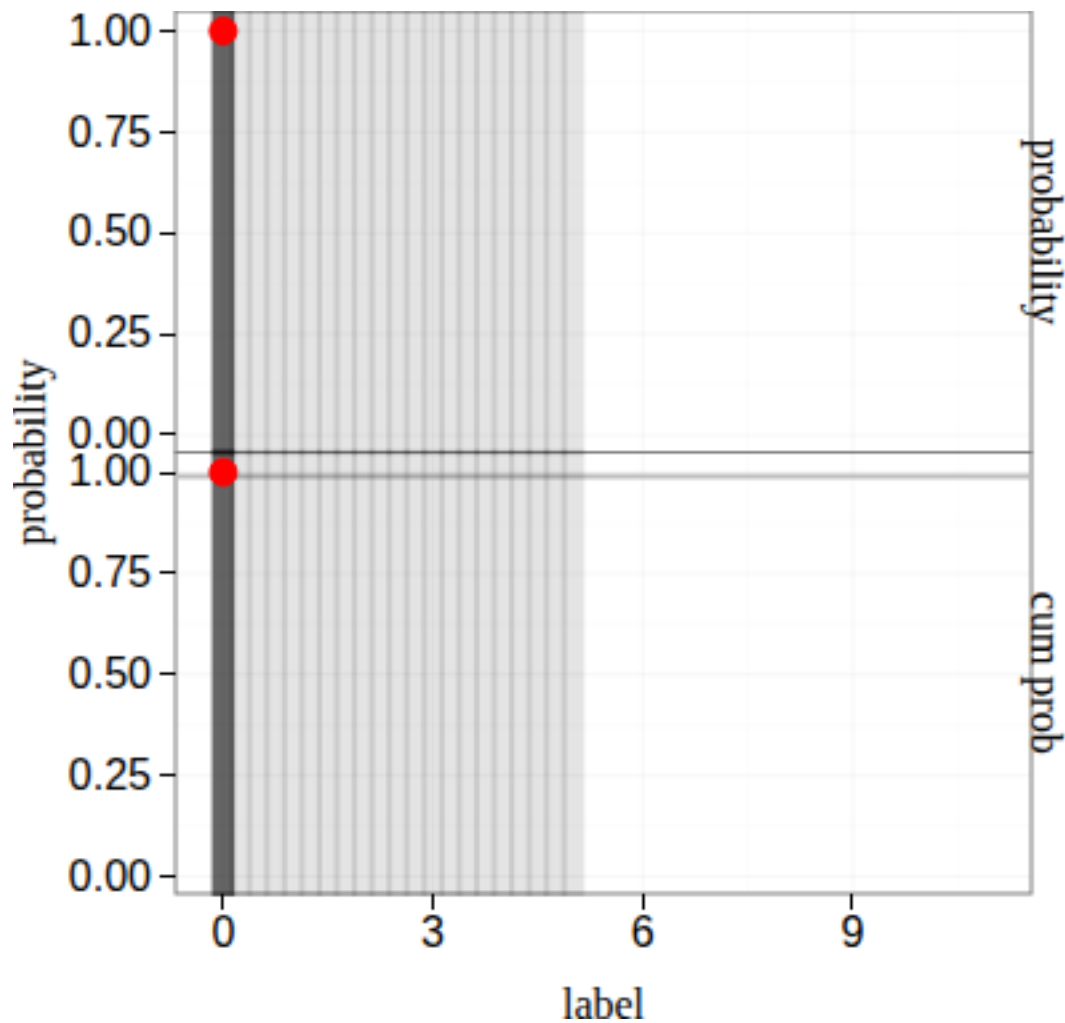


Figure 13.2: Poisson regression viz cum prob

When we specify the data sets, we will use the `addColumn` then `facet` idiom to add a `panel` variable.

```
addColumn <- function(dt, panel){
  data.table(dt, panel=factor(panel, c("probability", "cum prob")))
}
quantile.max.dt <- data.table(quantile.max)
animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_hline(aes(
      yintercept=quantile.max,
      color="grey",
      data=addColumn(quantile.max.dt, "cum prob"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=mean.tallrects)+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      color="red",
      size=5,
      data=addColumn(poisson.prob, "probability"))+
    geom_point(aes(
      label, cum.prob,
      tooltip=sprintf(
        "prob(label <= %d) = %f", label, cum.prob)),
      showSelected="poisson.mean",
      color="red",
      size=5,
      data=addColumn(poisson.prob, "cum prob")))
```



Note how we used `addPanel` to add a `panel` variable to all the data sets for each geom except `geom_tallrect`. Using `panel` as a facet variable has the effect of drawing each geom in only one panel, except the `geom_tallrect` which is drawn in each panel.

Note that we also used a `geom_hline` to show 0.99, the cumulative distribution function threshold that was used to determine the set of points to plot for each Poisson distribution. This is an example of “show your arbitrary choices,” one of the general principles of designing good interactive data visualizations.

13.3 Add a plot of the Poisson loss and a selector for label value

Next we will compute the Poisson loss for several values of the output label.

```
PoissonLoss <- function(label, seg.mean){
  stopifnot(is.numeric(label))
}
```

```

stopifnot(is.numeric(seg.mean))
if(any(seg.mean < 0)){
  stop("PoissonLoss undefined for negative segment mean")
}
if(length(seg.mean)==1)seg.mean <- rep(seg.mean, length(label))
if(length(label)==1)label <- rep(label, length(seg.mean))
stopifnot(length(seg.mean) == length(label))
not.integer <- round(label) != label
is.negative <- label < 0
loss <- ifelse(
  not.integer | is.negative, Inf,
  ifelse(seg.mean == 0, ifelse(label == 0, 0, Inf),
    seg.mean - label * log(seg.mean)
    ## This term makes all the minima zero.
    -ifelse(label == 0, 0, label - label*log(label))))
loss
}

```

Below we compute the loss for several label values, using the list of data tables idiom.

```

label.vec <- unique(poisson.prob$label)
label.range <- range(label.vec)
mean.vec <- seq(label.range[1], label.range[2], l=100)
loss.min.list <- list()
loss.fun.list <- list()
for(label in label.vec){
  loss <- PoissonLoss(label, mean.vec)
  loss.fun.list[[paste(label)]] <- data.table(
    label, poisson.mean=mean.vec, loss)
  loss.min.list[[paste(label)]] <- data.table(
    label, loss=0)
}
loss.fun <- do.call(rbind, loss.fun.list)
loss.min <- do.call(rbind, loss.min.list)

```

We also make a data table to display text labels for the selected mean and label values.

```

mean.text <- data.table(
  label=max(poisson.prob$label)/2,
  probability=0.95,
  poisson.mean=poisson.mean.vec)
loss.max <- 10
label.text <- data.table(
  poisson.mean=max(mean.tallrects$max),
  loss=loss.max*0.95,
  label=label.vec)

```

Next we make a data viz with an additional panel.

```

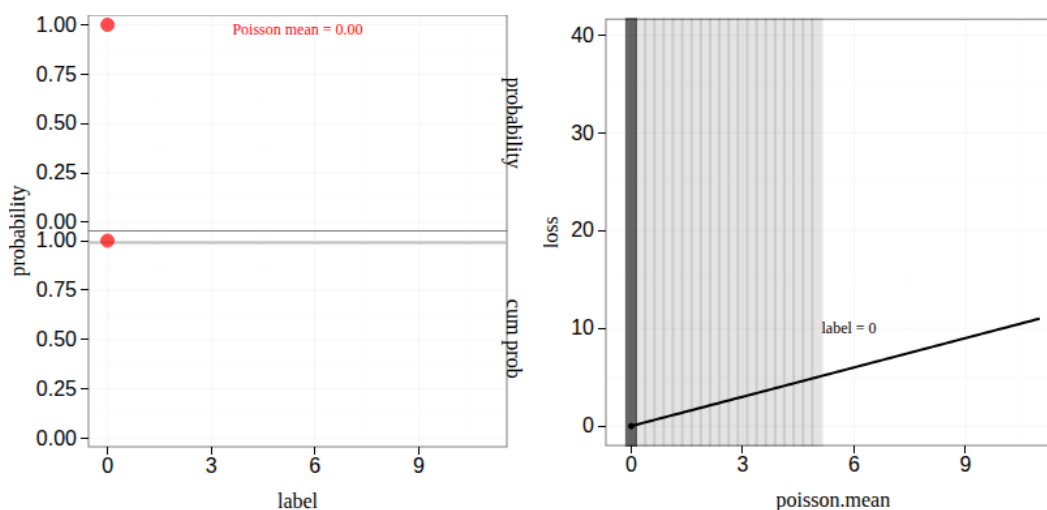
(viz.loss <- animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_text(aes(
      label, probability, label=sprintf(
        "Poisson mean = %.2f", poisson.mean)),
      color="red",
      showSelected="poisson.mean",
      data=addPanel(mean.text, "probability"))+
    geom_hline(aes(
      yintercept=quantile.max,
      color="grey",
      data=addPanel(quantile.max.dt, "cum prob"))+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      clickSelects="label",
      color="red",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "probability"))+
    geom_point(aes(
      label, cum.prob,
      tooltip=sprintf(
        "prob(label <= %d) = %f", label, cum.prob)),
      color="red",
      showSelected="poisson.mean",
      clickSelects="label",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "cum prob")),
  loss=ggplot()+
    theme_bw()+
    geom_text(aes(
      poisson.mean, loss,
      label=sprintf("label = %d", label)),
      showSelected="label",
      hjust=0,
      data=label.text)+
    geom_line(aes(
      poisson.mean, loss),
      showSelected="label",
      data=loss.fun)+
    geom_point(aes(
      label, loss),

```

```

    showSelected="label",
    data=loss.min)+
  geom_tallrect(aes(
    xmin=min, xmax=max),
    clickSelects="poisson.mean",
    alpha=0.6,
    data=mean.tallrects)))

```



The data viz above shows the probability on the left and the Poisson loss on the right.

```

viz.log.loss <- viz.loss
addX <- function(dt, x.var) data.table(dt, x.var=factor(
  x.var, c("poisson mean", "log(poisson mean)")))
finite.loss <- loss.fun[is.finite(loss)]
finite.loss[, log.poisson.mean := log(poisson.mean)]
finite.log.loss <- finite.loss[is.finite(log.poisson.mean)]
mean.tallrects[, log.min := ifelse(min < 0, -Inf, log(min))]

```

Warning in log(min): NaNs produced

```

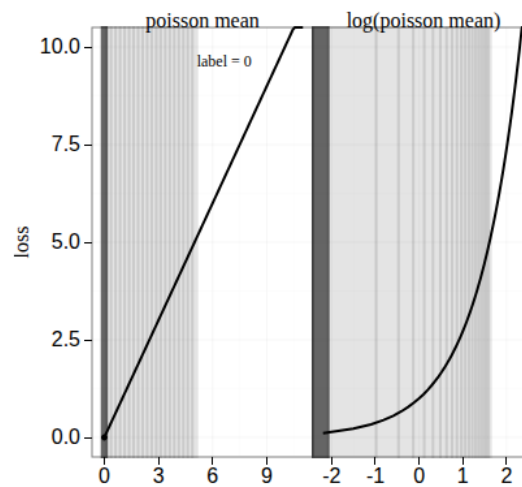
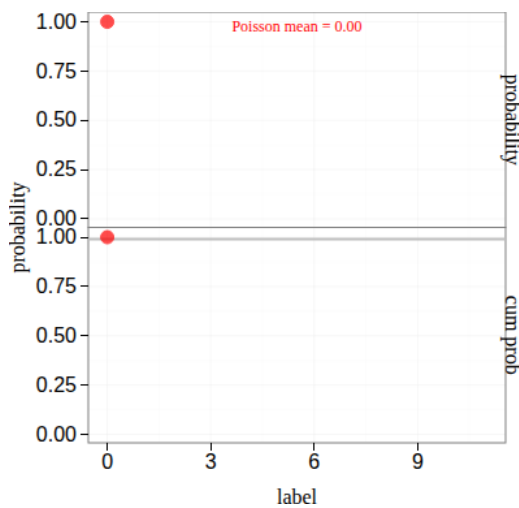
viz.log.loss$loss <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
  coord_cartesian(ylim=c(0, loss.max))+
  geom_text(aes(
    poisson.mean, loss, label=sprintf(
      "label = %d", label)),
    showSelected="label",
    hjust=0,
    data=addX(label.text, "poisson mean"))+
  geom_line(aes(

```

```

    poisson.mean, loss),
    showSelected="label",
    data=addX(finite.loss, "poisson mean"))+
geom_point(aes(
    label, loss),
    showSelected="label",
    data=addX(loss.min, "poisson mean"))+
geom_tallrect(aes(
    xmin=min, xmax=max),
    clickSelects="poisson.mean",
    alpha=0.6,
    data=addX(mean.tallrects, "poisson mean"))+
geom_line(aes(
    log.poisson.mean, loss),
    showSelected="label",
    data=addX(finite.log.loss, "log(poisson mean)"))+
geom_point(aes(
    log(label), loss),
    showSelected="label",
    data=addX(loss.min[0<label,], "log(poisson mean)"))+
geom_tallrect(aes(
    xmin=log.min, xmax=log(max)),
    clickSelects="poisson.mean",
    alpha=0.6,
    data=addX(mean.tallrects, "log(poisson mean)"))
viz.log.loss

```



13.4 Chapter summary and exercises

We explained how to visualize the Poisson distribution and loss, which are used for the Poisson regression model.

Exercises:

- The code above used `addPanel` and `addX` helper functions with several geoms to create multi-panel plots, which results in repetition. To avoid that repetition, create a new data viz which uses a single geom with a larger data set. For example, the red points in the two panels of the first plot could be defined using one `geom_point` with a larger data set (Hint: use `data.table::melt` with `measure.vars=c("cum.prob", "probability")`).
- Create a similar sequence of data visualizations for the [Binomial regression](#) model.

Next, [Chapter 14](#) explains how to use the named `clickSelects/showSelected` to visualize the PeakSegJoint machine learning model with data-driven selector variables.



14

Named clickSelects/showSelected

This chapter explains how to use named `clickSelects/showSelected` variables for creating data-driven selector names. This feature makes it easier to write animint code, and makes it faster to compile.

Chapter outline:

- We begin by downloading the PSJ data set and computing the data to plot.
- We show one method of defining an animint with many selectors, using for loops. This method is technically correct, but computationally inefficient.
- We then explain the preferred method for defining an animint with many selectors, using named `clickSelects/showSelected`. This method is more computationally efficient, and easier to code.

14.1 Download data set

The example data come from the [PeakSegJoint package](#). The code below downloads the data set.

```
if(!requireNamespace("animint2data"))  
  remotes::install_github("animint/animint2data")
```

Loading required namespace: animint2data

```
data(PSJ, package="animint2data")
```

14.2 Compute data to plot

The section below computes some common data that we will use in two data visualizations below.

```
res.error <- PSJ$error.total.chunk  
ann.colors <- c(  
  noPeaks="#f6f4bf",  
  peakStart="#ffafaf",
```

```

    peakEnd="#ff4c4c",
    peaks="#a445ee")
## prob.regions are the black segments that show which regions are
## mapped to which segmentation problems.
library(data.table)
all.regions <- data.table(do.call(rbind, PSJ$regions.by.problem))
prob.regions.names <- c(
  "bases.per.problem", "problem.i", "problem.name",
  "chromStart", "chromEnd")
prob.regions <- unique(data.frame(all.regions[, prob.regions.names])
prob.regions$sample.id <- "problems"
all.modelSelection <- data.table(do.call(
  rbind, PSJ$modelSelection.by.problem))
modelSelection.errors <- all.modelSelection[!is.na(errors)]
penalty.range <- all.modelSelection[, c(
  min(max.log.lambda), max(min.log.lambda))]
penalty.mid <- mean(penalty.range)
coverage.counts <- table(PSJ$coverage$sample.id)
facet.rows <- length(coverage.counts)+1
dvec <- diff(log(res.error$bases.per.problem))
dval <- exp(mean(dvec))
dval2 <- (dval-1)/2 + 1
res.error$min.bases.per.problem <- res.error$bases.per.problem/dval2
res.error$max.bases.per.problem <- res.error$bases.per.problem*dval2
modelSelection.labels <- unique(all.modelSelection[, data.table(
  problem.name=problem.name,
  bases.per.problem=bases.per.problem,
  problemStart=problemStart,
  problemEnd=problemEnd,
  min.log.lambda=penalty.mid,
  peaks=max(peaks)+0.5)])

```

14.3 Define data viz using for loops

The R code below constructs a data viz using for loops.

```

library(animint2)
print(timing.for.construct <- system.time({
  viz.for <- list(
    coverage=ggplot()+
      geom_segment(aes(
        chromStart/1e3, problem.i,
        xend=chromEnd/1e3, yend=problem.i),
        showSelected="bases.per.problem",
        clickSelects="problem.name",
        data=prob.regions)+

```

```

ggtitle("select problem")+
geom_text(aes(
  chromStart/1e3, problem.i,
  label=sprintf("%d problems mean size %.1f kb",
    problems, mean.bases/1e3)),
  showSelected="bases.per.problem",
  data=PSJ$problem.labels,
  hjust=0)+
geom_segment(aes(
  problemStart/1e3, problem.i,
  xend=problemEnd/1e3, yend=problem.i),
  showSelected="bases.per.problem",
  clickSelects="problem.name",
  size=5,
  data=PSJ$problems)+
scale_y_continuous(
  "aligned read coverage",
  breaks=function(limits){
    floor(limits[2])
  })+
scale_linetype_manual(
  "error type",
  limits=c(
    "correct",
    "false negative",
    "false positive"),
  values=c(
    correct=0,
    "false negative"=3,
    "false positive"=1))+
scale_x_continuous(paste(
  "position on chr11",
  "(kilo bases = kb)"))+
coord_cartesian(xlim=c(118167.406, 118238.833))+
geom_tallrect(aes(
  xmin=chromStart/1e3, xmax=chromEnd/1e3,
  fill=annotation),
  alpha=0.5,
  color="grey",
  data=PSJ$filled.regions)+
scale_fill_manual(values=ann.colors)+
theme_bw()+
theme_animint(width=1500, height=facet.rows*100)+
theme(panel.margin=grid::unit(0, "cm"))+
facet_grid(sample.id ~ ., labeller=function(df){
  df$sample.id <- sub("McGill0", "", sub(" ", "\n", df$sample.id))
  df
}, scales="free")+
geom_line(aes(

```

```

    base/1e3, count),
    data=PSJ$coverage,
    color="grey50"),
resError=ggplot()+
  ggtitle("select problem size")+
  ylab("minimum percent incorrect regions")+
  geom_tallrect(aes(
    xmin=min.bases.per.problem,
    xmax=max.bases.per.problem),
    clickSelects="bases.per.problem",
    alpha=0.5,
    data=res.error)+
  scale_x_log10()+
  geom_line(aes(
    bases.per.problem, errors/regions*100,
    color=chunks, size=chunks),
    data=data.frame(res.error, chunks="this"))+
  geom_line(aes(
    bases.per.problem, errors/regions*100,
    color=chunks, size=chunks),
    data=data.frame(PSJ$error.total.all, chunks="all")),
modelSelection=ggplot()+
  geom_segment(aes(
    min.log.lambda, peaks,
    xend=max.log.lambda, yend=peaks),
    showSelected=c("bases.per.problem", "problem.name"),
    data=data.frame(all.modelSelection, what="peaks"),
    size=5)+
  geom_text(aes(
    min.log.lambda, peaks,
    label=sprintf(
      "%.1f kb in problem %s",
      (problemEnd-problemStart)/1e3, problem.name)),
    showSelected=c("problem.name", "bases.per.problem"),
    data=data.frame(modelSelection.labels, what="peaks"))+
  geom_segment(aes(
    min.log.lambda, as.integer(errors),
    xend=max.log.lambda, yend=as.integer(errors)),
    showSelected=c("bases.per.problem", "problem.name"),
    data=data.frame(modelSelection.errors, what="errors"),
    size=5)+
  ggtitle("select number of samples with 1 peak")+
  ylab("")+
  facet_grid(what ~ ., scales="free"),
title="Animint compiler with for loops",
first=PSJ$first)
## For every problem there is a selector (called problem.dot) for the
## number of peaks in that problem. So in this for loop we add a few
## layers with aes_string(clickSelects=problem.dot) or

```

```

## aes_string(showSelected=problem.dot) to the coverage and
## modelSelection plots.
for(problem.dot in names(PSJ$modelSelection.by.problem)){
  regions.dt <- PSJ$regions.by.problem[[problem.dot]]
  regions.dt[[problem.dot]] <- regions.dt$peaks
  if(!is.null(regions.dt)){
    viz.for$coverage <- viz.for$coverage+
      geom_tallrect(aes(
        xmin=chromStart/1e3,
        xmax=chromEnd/1e3,
        linetype=status),
        showSelected=c(problem.dot, "bases.per.problem"),
        data=data.frame(regions.dt),
        fill=NA,
        color="black")
  }
  if(problem.dot %in% names(PSJ$peaks.by.problem)){
    peaks <- PSJ$peaks.by.problem[[problem.dot]]
    peaks[[problem.dot]] <- peaks$peaks
    prob.peaks.names <- c(
      "bases.per.problem", "problem.i", "problem.name",
      "chromStart", "chromEnd", problem.dot)
    prob.peaks <- unique(data.frame(peaks[, prob.peaks.names])
    prob.peaks$sample.id <- "problems"
    viz.for$coverage <- viz.for$coverage +
      geom_segment(aes(
        chromStart/1e3, 0,
        xend=chromEnd/1e3, yend=0),
        clickSelects="problem.name",
        showSelected=c(problem.dot, "bases.per.problem"),
        data=peaks, size=7, color="deepskyblue")+
      geom_segment(aes(
        chromStart/1e3, problem.i,
        xend=chromEnd/1e3, yend=problem.i),
        clickSelects="problem.name",
        showSelected=c(problem.dot, "bases.per.problem"),
        data=prob.peaks, size=7, color="deepskyblue")
  }
  modelSelection.dt <- PSJ$modelSelection.by.problem[[problem.dot]]
  modelSelection.dt[[problem.dot]] <- modelSelection.dt$peaks
  viz.for$modelSelection <- viz.for$modelSelection+
    geom_tallrect(aes(
      xmin=min.log.lambda,
      xmax=max.log.lambda),
      clickSelects=problem.dot,
      showSelected=c("problem.name", "bases.per.problem"),
      data=modelSelection.dt, alpha=0.5)
  }
}
})

```

```

user  system elapsed
2.622  0.002  2.624

```

Note the timing of the code above. It takes a long time just to evaluate the R code that defines this data viz, since it has so many geoms. Next, we compile the data visualization.

```

print(timing.for.compile <- system.time({
  animint2dir(viz.for, "Ch14-for")
}))

```

Warning: Using size for a discrete variable is not advised.

Warning: Using size for a discrete variable is not advised.

Warning in checkSingleShowSelectedValue(meta\$selectors): showSelected variables with only 1 level: chr11.118184422.118184700peaks, chr11.118192951.118193582peaks, chr11.118203893.118204314peaks

```

user  system elapsed
242.604  0.264 244.054

```

Note that the compilation also takes a long time, since there are so many geoms. The data viz can be viewed on Ch14-for/index.html. In the next section we will create the same data viz, but more efficiently.

14.4 Define data viz using named *clickSelects/showSelected*

In this section we use named *clickSelects/showSelected* to create a more efficient version of the previous data visualization. In general, any data visualization defined using for loops in R code can be made more efficient by instead using this method.

```

sample.peaks <- data.table(do.call(rbind, PSJ$peaks.by.problem))
prob.peaks.names <- c(
  "bases.per.problem", "problem.i", "problem.name", "peaks",
  "chromStart", "chromEnd")
problem.peaks <- unique(sample.peaks[, ..prob.peaks.names])
problem.peaks$sample.id <- "problems"
peakvar <- function(position){
  paste0(gsub("[-:]", ".", position), "peaks")
}
all.regions[, selector := peakvar(problem.name)]
sample.peaks[, selector := peakvar(problem.name)]
problem.peaks[, selector := peakvar(problem.name)]
all.modelSelection[, selector := peakvar(problem.name)]
print(timing.named.construct <- system.time({
  viz.named <- list(
    coverage=ggplot()+
      ggtitle("select problem")+
      geom_segment(aes(
        chromStart/1e3, problem.i,

```



```

    xend=chromEnd/1e3, yend=problem.i),
    showSelected="bases.per.problem",
    clickSelects="problem.name",
    data=prob.regions)+
geom_text(aes(
  chromStart/1e3, problem.i,
  label=sprintf(
    "%d problems mean size %.1f kb",
    problems, mean.bases/1e3)),
  showSelected="bases.per.problem",
  data=PSJ$problem.labels,
  hjust=0)+
geom_segment(aes(
  problemStart/1e3, problem.i,
  xend=problemEnd/1e3, yend=problem.i),
  showSelected="bases.per.problem",
  clickSelects="problem.name",
  size=5,
  data=PSJ$problems)+
scale_y_continuous(
  "aligned read coverage",
  breaks=function(limits){
    floor(limits[2])
  })+
scale_linetype_manual(
  "error type",
  limits=c(
    "correct",
    "false negative",
    "false positive"),
  values=c(
    correct=0,
    "false negative"=3,
    "false positive"=1))+
scale_x_continuous(paste(
  "position on chr11",
  "(kilo bases = kb)"))+
coord_cartesian(xlim=c(118167.406, 118238.833))+
geom_tallrect(aes(
  xmin=chromStart/1e3, xmax=chromEnd/1e3,
  fill=annotation),
  alpha=0.5,
  color="grey",
  data=PSJ$filled.regions)+
scale_fill_manual(values=ann.colors)+
theme_bw()+
theme_animint(width=1500, height=facet.rows*100)+
theme(panel.margin=grid::unit(0, "cm"))+
facet_grid(sample.id ~ ., labeller=function(df){

```

```

    df$sample.id <- sub("McGill10", "", sub(" ", "\n", df$sample.id))
  df
}, scales="free")+
geom_line(aes(
  base/1e3, count),
  data=PSJ$coverage,
  color="grey50")+
geom_tallrect(aes(
  xmin=chromStart/1e3,
  xmax=chromEnd/1e3,
  linetype=status),
  showSelected=c("selector"="peaks", "bases.per.problem"),
  data=all.regions,
  fill=NA,
  color="black")+
geom_segment(aes(
  chromStart/1e3, 0,
  xend=chromEnd/1e3, yend=0),
  clickSelects="problem.name",
  showSelected=c("selector"="peaks", "bases.per.problem"),
  data=sample.peaks, size=7, color="deepskyblue")+
geom_segment(aes(
  chromStart/1e3, problem.i,
  xend=chromEnd/1e3, yend=problem.i),
  clickSelects="problem.name",
  showSelected=c("selector"="peaks", "bases.per.problem"),
  data=problem.peaks, size=7, color="deepskyblue"),
resError=ggplot()+
  ggtitle("select problem size")+
  ylab("minimum percent incorrect regions")+
  geom_tallrect(aes(
    xmin=min.bases.per.problem,
    xmax=max.bases.per.problem),
    clickSelects="bases.per.problem",
    alpha=0.5,
    data=res.error)+
  scale_x_log10()+
  geom_line(aes(
    bases.per.problem, errors/regions*100,
    color=chunks, size=chunks),
    data=data.frame(res.error, chunks="this"))+
  geom_line(aes(
    bases.per.problem, errors/regions*100,
    color=chunks, size=chunks),
    data=data.frame(PSJ$error.total.all, chunks="all")),
modelSelection=ggplot()+
  geom_segment(aes(
    min.log.lambda, peaks,
    xend=max.log.lambda, yend=peaks),

```

```

    showSelected=c("problem.name", "bases.per.problem"),
    data=data.frame(all.modelSelection, what="peaks"),
    size=5)+
  geom_text(aes(
    min.log.lambda, peaks,
    label=sprintf(
      "%.1f kb in problem %s",
      (problemEnd-problemStart)/1e3, problem.name)),
    showSelected=c("problem.name", "bases.per.problem"),
    data=data.frame(modelSelection.labels, what="peaks"))+
  geom_segment(aes(
    min.log.lambda, as.integer(errors),
    xend=max.log.lambda, yend=as.integer(errors)),
    showSelected=c("problem.name", "bases.per.problem"),
    data=data.frame(modelSelection.errors, what="errors"),
    size=5)+
  ggtitle("select number of samples with 1 peak")+
  ylab("")+
  geom_tallrect(aes(
    xmin=min.log.lambda,
    xmax=max.log.lambda,
    clickSelects=c("selector"="peaks"),
    showSelected=c("problem.name", "bases.per.problem"),
    data=all.modelSelection, alpha=0.5)+
    facet_grid(what ~ ., scales="free"),
    title="Animint compiler with named clickSelects/showSelected",
    first=PSJ$first)
### For every problem there is a selector (called problem.name) for
### the number of peaks in that problem. The animint2dir compiler
### creates a selection variable for every unique value of
### clickSelects/showSelected names (and it uses corresponding values
### to set/update the selected value/geoms).
  )))

```

```

user  system elapsed
0.025   0.000   0.025

```

It is clear that it takes much less time to evaluate the R code above which uses the named `clickSelects/showSelected`. We compile it below.

```

print(timing.named.compile <- system.time({
  animint2dir(viz.named, "Ch14-named")
}))

```

Warning: Using size for a discrete variable is not advised.
Warning: Using size for a discrete variable is not advised.

```

user  system elapsed
1.931   0.052   7.758

```

The animint produced above can be viewed on Ch14-named/index.html. Note that it should appear to be the same as the other data viz above. The timings above show that

named *clickSelects/showSelected* are much faster than for loops, in both the definition and compilation steps.

14.5 Disk usage comparison

In this section we compute the disk usage of both methods.

```
viz.dirs.vec <- c("Ch14-for", "Ch14-named")
viz.dirs.text <- paste(viz.dirs.vec, collapse=" ")
(cmd <- paste("du -ks", viz.dirs.text))
```

```
[1] "du -ks Ch14-for Ch14-named"
```

```
kb.dt <- fread(cmd=cmd)
setnames(kb.dt, c("kilobytes", "path"))
kb.dt
```

	kilobytes	path
1:	4488	Ch14-for
2:	1744	Ch14-named

The table above shows that the data viz defined using for loops takes about twice as much disk space as the data viz that used named *clickSelects/showSelected*.

14.6 Chapter summary and exercises

The table below summarizes the disk usage and timings presented in this chapter. It is clear that named *clickSelects/showSelected* are more efficient in both respects, and should be used instead of for loops.

```
data.frame(
  kilobytes=kb.dt$kilobytes,
  construct.seconds=c(
    timing.for.construct[["elapsed"]],
    timing.named.construct[["elapsed"]]),
  compile.seconds=c(
    timing.for.compile[["elapsed"]],
    timing.named.compile[["elapsed"]]),
  row.names=c("for", "named"))
```

	kilobytes	construct.seconds	compile.seconds
for	4488	2.624	244.054
named	1744	0.025	7.758

Exercises:

- Use named `clickSelects/showSelected` to create a visualization of some data from your domain of expertise.

Next, [Chapter 15](#) explains how to visualize root-finding algorithms.



15

Newton's root-finding method

Roots of a function $f(x)$ are values x such that $f(x)=0$. Some functions f have an explicit expression for their roots. For example:

- the linear function $f(x)=b*x+c=0$ has a single root $x=-c/b$, if b is not zero.
- the quadratic function $f(x)=a*x^2+b*x+c=0$ has two roots $x=(-b\pm\sqrt{b^2-4*a*c})/(2*a)$, if the discriminant is positive $b^2-4*a*c>0$.
- the sin function $f(x)=\sin(a*x)=0$ has an infinite number of roots: $\pi*z/a$ for all integers z .

However, there are some functions which have no explicit expression for their roots. For example, the roots of the Poisson loss $f(x)=a*x+b*\log(x)+c$ have no explicit expression in terms of common mathematical functions. (actually it has a solution in terms of the [Lambert W function](#) but that function is not commonly available) This goal of this chapter is to create an interactive data visualization that explains [Newton's method](#) for finding the roots of such functions.

Chapter outline:

- We begin by implementing the Newton method for the Poisson loss, to find the root which is larger than the minimum. We create several static and one interactive data visualization.
- We then suggest an exercise for finding the root which is smaller than the minimum.

15.1 Larger root in mean space

We begin by defining coefficients of a Poisson Loss function with two roots.

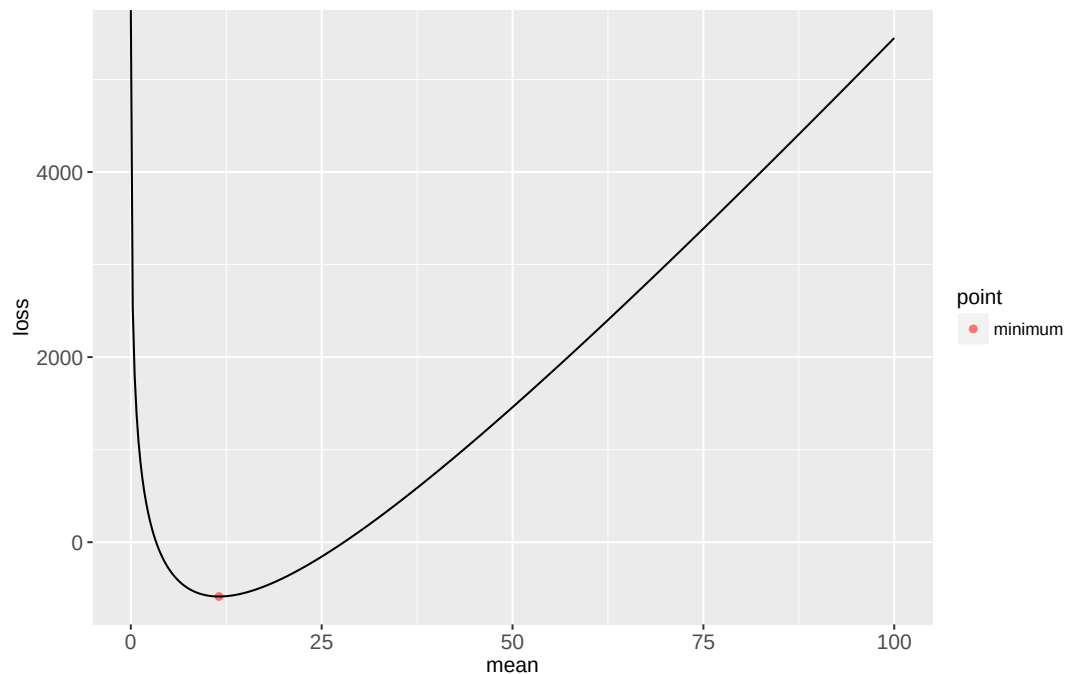
```
Linear <- 95
Log <- -1097
Constant <- 1000
loss.fun <- function(Mean){
  Linear*Mean + Log*log(Mean) + Constant
}
(mean.at.optimum <- -Log/Linear)
```

```
[1] 11.54737
```

```
(loss.at.optimum <- loss.fun(mean.at.optimum))
```

```
[1] -586.764
```

```
library(data.table)
loss.dt <- data.table(mean=seq(0, 100, l=400))
loss.dt[, loss := loss.fun(mean)]
opt.dt <- data.table(
  mean=mean.at.optimum,
  loss=loss.at.optimum,
  point="minimum")
library(animint2)
gg.loss <- ggplot()+
  geom_point(aes(mean, loss, color=point), data=opt.dt)+
  geom_line(aes(mean, loss), data=loss.dt)
print(gg.loss)
```



Our goal is to find the two roots of this function. Newton's root finding method starts from an arbitrary candidate root, and then repeatedly uses linear approximations to find more accurate candidate roots. To compute the linear approximation, we need the derivative:

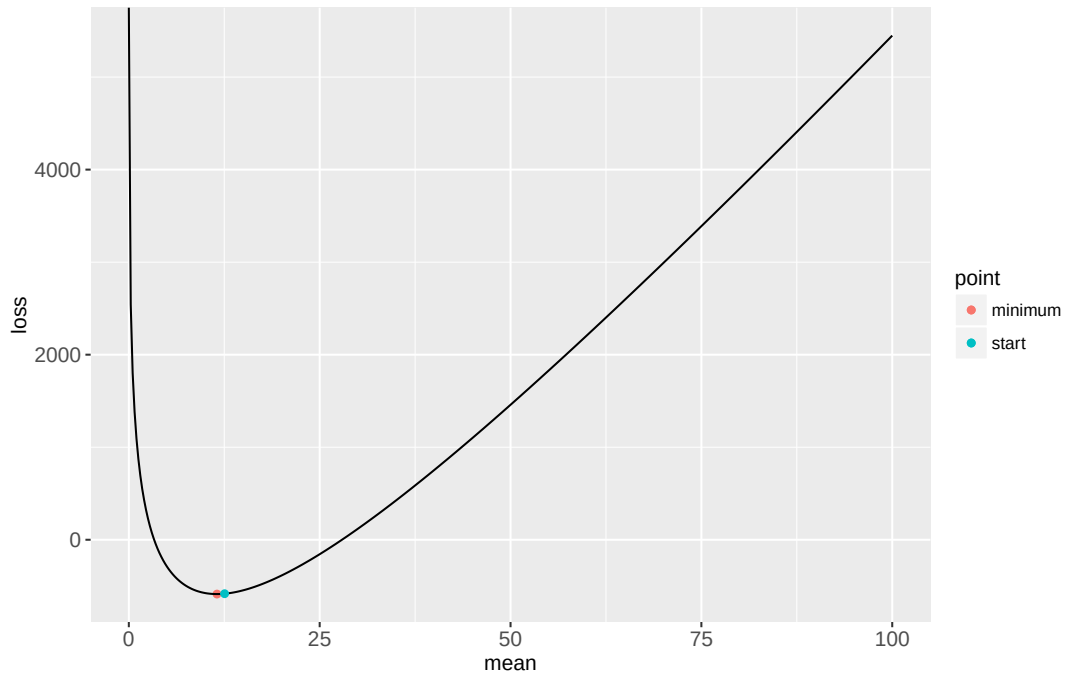
```
loss.deriv <- function(Mean){
  Linear + Log/Mean
}
```

We begin the root finding at a point larger than the minimum,

```
possible.root <- mean.at.optimum+1
gg.loss+
  geom_point(aes(
```



```
mean, loss, color=point),
data=data.table(
  point="start",
  mean=possible.root,
  loss=loss.fun(possible.root)))
```



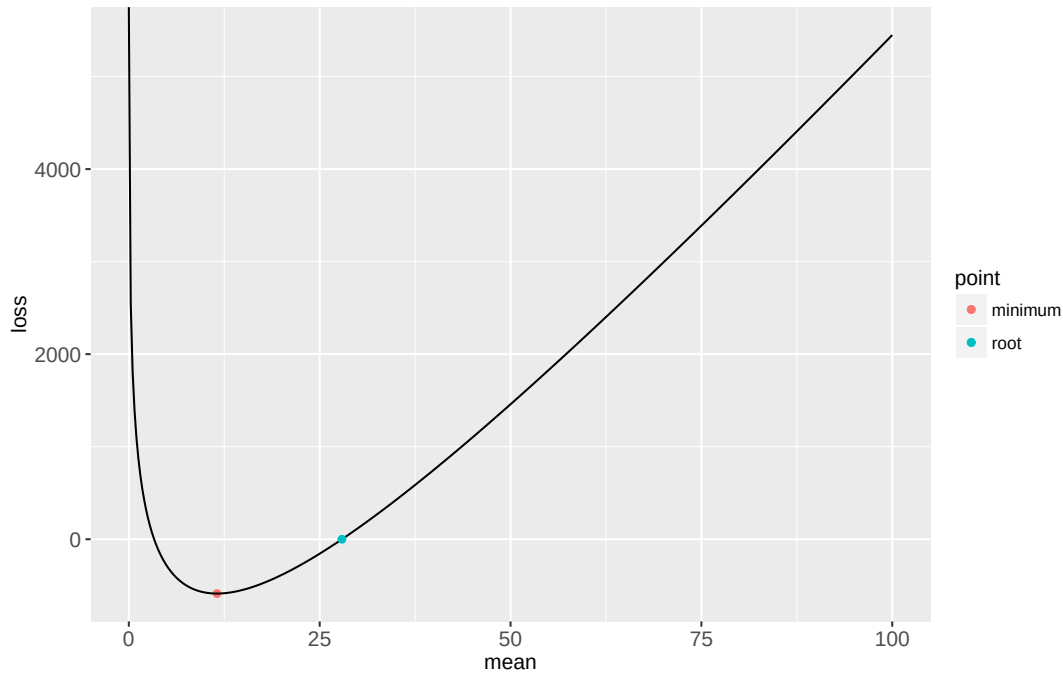
We then use the following implementation of Newton's method to find a root,

```
iteration <- 1
solution.list <- list()
thresh.dt <- data.table(thresh=1e-6)
while(thresh.dt$thresh < abs({
  fun.value <- loss.fun(possible.root)
})){
  cat(sprintf("mean=%e loss=%e\n", possible.root, fun.value))
  deriv.value <- loss.deriv(possible.root)
  new.root <- possible.root - fun.value/deriv.value
  solution.list[[iteration]] <- data.table(
    iteration, possible.root, fun.value, deriv.value, new.root)
  iteration <- iteration+1
  possible.root <- new.root
}
```

```
mean=1.254737e+01 loss=-5.828735e+02
mean=8.953188e+01 loss=4.574958e+03
mean=3.424363e+01 loss=3.768946e+02
mean=2.825783e+01 loss=1.901051e+01
mean=2.791944e+01 loss=7.929111e-02
```

```
mean=2.791802e+01 loss=1.425562e-06
```

```
root.dt <- data.table(point="root", possible.root, fun.value)
gg.loss+
  geom_point(aes(possible.root, fun.value, color=point),
             data=root.dt)
```



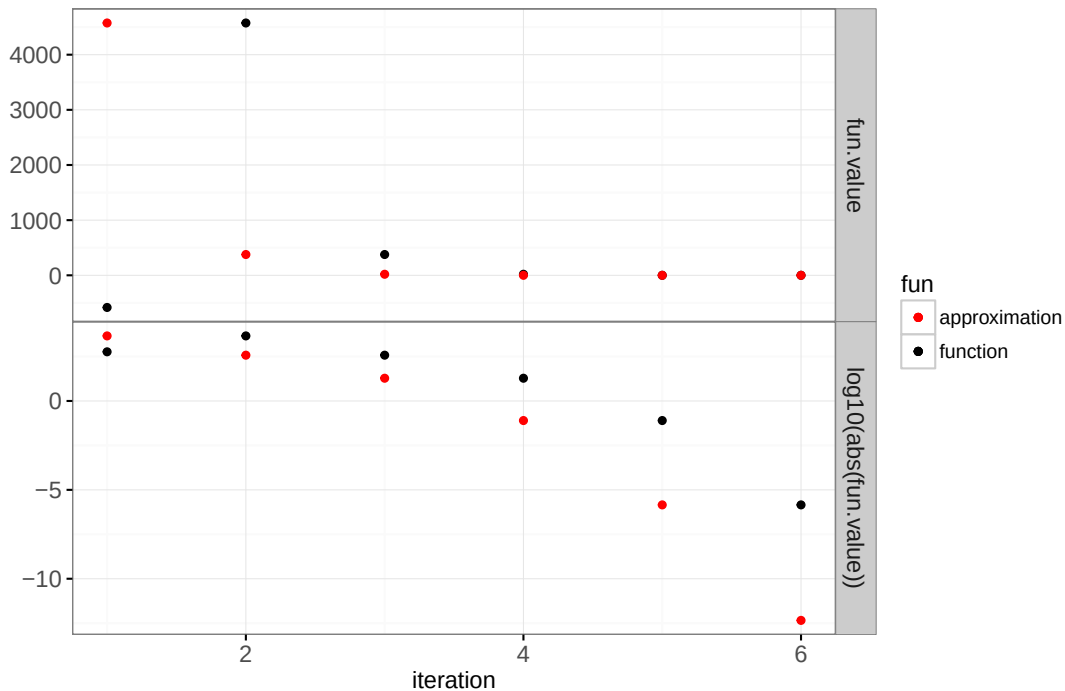
The plot above shows the root that was found. The stopping criterion was an absolute cost value less than $1e-6$ so we know that this root is at least that accurate. The following plot shows the accuracy of the root as a function of the number of iterations.

```
solution <- do.call(rbind, solution.list)
solution$new.value <- c(solution$fun.value[-1], fun.value)
gg.it <- ggplot()+
  geom_point(aes(
    iteration, fun.value, color=fun),
    data=data.table(solution, y="fun.value", fun="function"))+
  geom_point(aes(
    iteration, log10(abs(fun.value)), color=fun),
    data=data.table(
      solution, y="log10(abs(fun.value))", fun="function"))+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_point(aes(
    iteration, new.value, color=fun),
    data=data.table(solution, y="fun.value", fun="approximation"))+
  geom_point(aes(
    iteration, log10(abs(new.value)), color=fun),
    data=data.table(
```

```

    solution, y="log10(abs(fun.value))", fun="approximation"))+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y ~ ., scales="free")+
  ylab("")
print(gg.it)

```



The plot above shows a horizontal line for the stopping criterion threshold, on the log scale. It is clear that the red dot in the last iteration is much below that threshold.

The plot below shows each step of the algorithm. The left panels show the linear approximation at the candidate root, along with the root of the linear approximation. The right panels show the root of the linear approximation, along with the corresponding function value (the new candidate root).

```

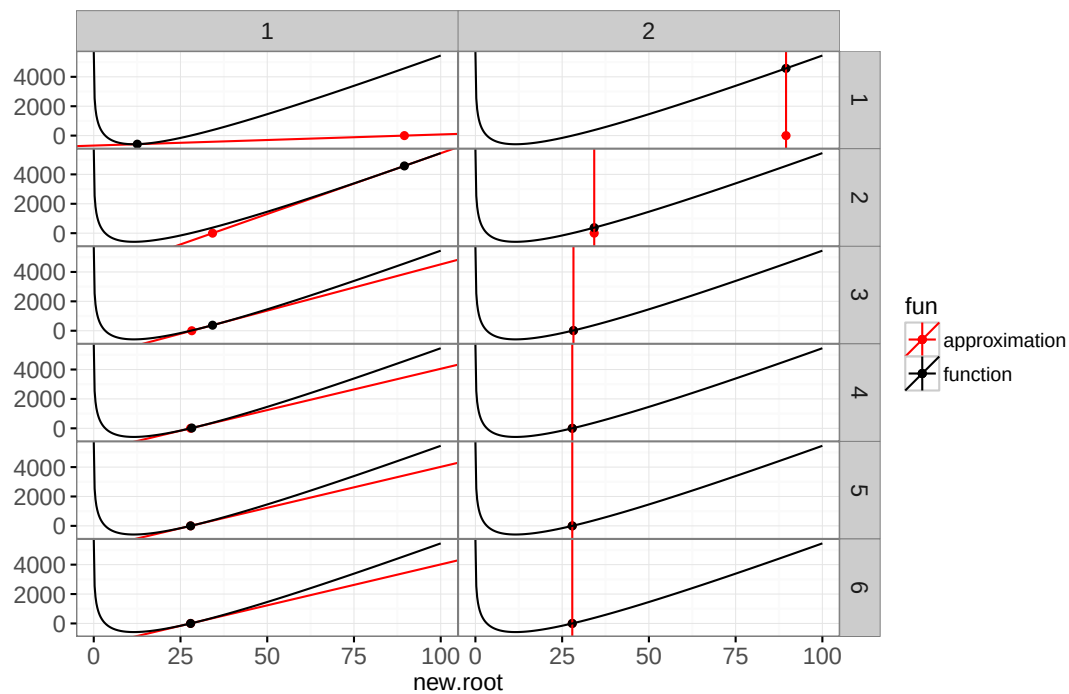
## y - fun.value = deriv.value * (x - possible.root)
## y = deriv.value*x + fun.value-possible.root*deriv.value
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(iteration ~ step)+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_abline(aes(
    slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
    color=fun),
    data=data.table(solution, fun="approximation", step=1))+
  geom_point(aes(

```

```

    new.root, 0, color=fun),
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    new.root, new.value, color=fun),
    data=data.table(solution, fun="function", step=2))+
  geom_vline(aes(
    xintercept=new.root, color=fun),
    data=data.table(solution, fun="approximation", step=2))+
  geom_point(aes(
    possible.root, fun.value, color=fun),
    data=data.table(solution, fun="function", step=1))+
  geom_line(aes(
    mean, loss, color=fun),
    data=data.table(loss.dt, fun="function"))+
  ylab("")

```



It is clear that the algorithm quickly converges to the root. The following is an animated interactive version of the same data viz.

```

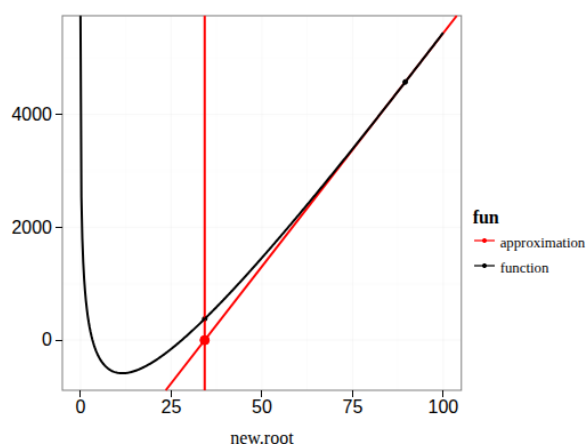
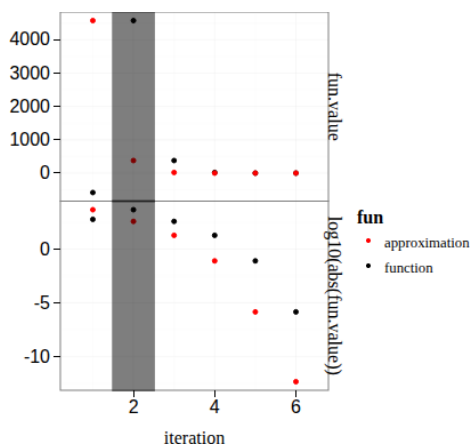
animint(
  time=list(variable="iteration", ms=2000),
  iterations=gg.it+
  theme_animint(width=300, colspan=1)+
  geom_tallrect(aes(
    xmin=iteration-0.5,
    xmax=iteration+0.5,
    clickSelects="iteration",

```

```

    alpha=0.5,
    data=solution),
loss=ggplot()+
  theme_bw()+
  scale_color_manual(values=c(
    "function"="black",
    "approximation"="red"))+
  geom_abline(aes(
    slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
    color=fun),
    showSelected="iteration",
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    new.root, 0, color=fun),
    showSelected="iteration",
    size=4,
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    new.root, new.value, color=fun),
    showSelected="iteration",
    data=data.table(solution, fun="function"))+
  geom_vline(aes(
    xintercept=new.root, color=fun),
    showSelected="iteration",
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    possible.root, fun.value, color=fun),
    showSelected="iteration",
    data=data.table(solution, fun="function"))+
  geom_line(aes(
    mean, loss, color=fun),
    data=data.table(loss.dt, fun="function"))+
  ylab(""))

```



15.2 Comparison with Lambert W solution

The code below uses the Lambert W function to compute a root, and compares its solution to the one we computed using Newton's method.

```
inside <- Linear*exp(-Constant/Log)/Log
root.vec <- Log/Linear*c(
  LambertW::W(inside, 0),
  LambertW::W(inside, -1))
```

Registered S3 methods overwritten by 'ggplot2':

method	from
drawDetails.zeroGrob	animint2
grobHeight.absoluteGrob	animint2
grobHeight.zeroGrob	animint2
grobWidth.absoluteGrob	animint2
grobWidth.zeroGrob	animint2
grobX.absoluteGrob	animint2
grobY.absoluteGrob	animint2
heightDetails.titleGrob	animint2
heightDetails.zeroGrob	animint2
makeContext.dotstackGrob	animint2
print.ggplot2_bins	animint2
print.rel	animint2
widthDetails.titleGrob	animint2
widthDetails.zeroGrob	animint2

```
loss.fun(c(
  Newton=root.dt$possible.root,
  Lambert=root.vec[2]))
```

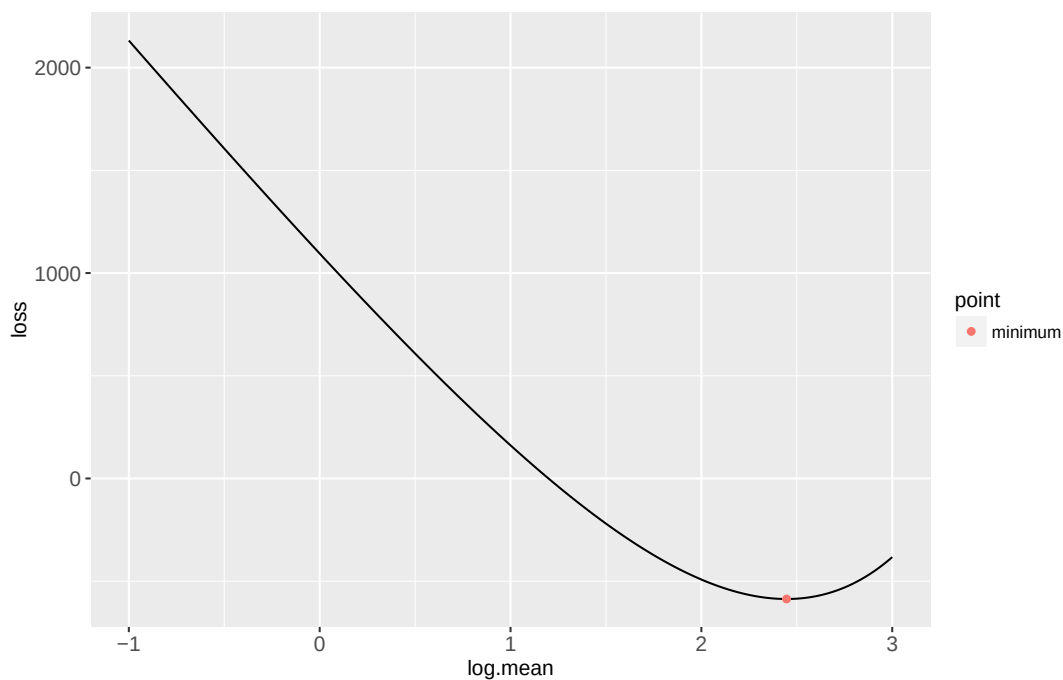
Newton	Lambert
-4.547474e-13	0.000000e+00

For these data, the Lambert W function yields a root which is slightly more accurate than our implementation of Newton's method.

15.3 Root-finding in the log space

The previous section showed an algorithm for finding the root which is larger than the minimum. In this section we explore an algorithm for finding the other root (smaller than the minimum). Note that the Poisson loss is highly non-linear as mean goes to zero, so the linear approximation in the Newton root finding will not work very well. Instead, we equivalently perform root finding in the log space:

```
log.loss.fun <- function(log.mean){
  Linear*exp(log.mean) + Log*log.mean + Constant
}
log.loss.dt <- data.table(
  log.mean=seq(-1, 3, l=400)
)[
  , loss := log.loss.fun(log.mean)]
ggplot()+
  geom_line(aes(
    log.mean, loss),
    data=log.loss.dt)+
  geom_point(aes(
    log(mean), loss, color=point),
    data=opt.dt)
```



Exercise: derive and implement the Newton method for this function, in order to find the root that is smaller than the minimum. Create an animint similar to the previous section.

15.4 Chapter summary and exercises

In this chapter we explored several visualizations of the Newton method for finding roots of smooth functions.

Exercises:

- Add a title to each plot.

- Add a size legend to the first plot (black points larger than red), so that we can see when red and black have the same value.
- Add a `geom_hline` to emphasize the $\text{loss}=0$ value in the second plot.
- Add a `geom_hline` to emphasize the stopping threshold in the first plot.
- Turn off one of the two legends, to save space.
- How to specify smooth transitions between iterations?
- Instead of using iteration as the animation/time variable, create a new one in order to show two distinct states/steps for each iteration, i.e. the `step` variable in the faceted plot above.
- What happens to the rate of convergence when you try to find the larger root in the log space, or the smaller root in the original space? Theoretically it should not converge as fast, since the functions are more nonlinear for those roots. Make a data visualization that allows you to select the starting value, and shows how many iterations it takes to converge to within the threshold.
- Create another plot that allows you to select the threshold. Plot the number of iterations as a function of threshold.
- Derive the loss function for [Binomial regression](#), and visualize the corresponding Newton root finding method.
- Refactor `gg.it` code to use only one `geom_point`, instead of the four geoms in the current code. Hint: use `rbind()` to create a single table with all of the data.

Next, [Chapter 16](#) explains how to visualize change-point detection models.

A

Appendix, useful idioms

This appendix describes several idioms which are useful for creating animints.

A.1 Space saving facets

To emphasize the plotted data in faceted ggplots, eliminate the space between facets using the following idiom.

```
ggplot()+  
  geom_point(aes(Petal.Width, Sepal.Width), iris)+  
  theme_bw()+  
  theme(panel.margin=grid::unit(0, "lines"))+  
  facet_grid(. ~ Species)
```

There are three parts of this idiom:

- `panel.margin=0` eliminates space between panels.
- `theme_bw` activates a black and white theme (black panel borders and white panel backgrounds). This is necessary in order to see the boundaries between panels, since the ggplot default `theme_grey` uses grey panel backgrounds and no panel borders.
- `facet_*` creates a multi-panel ggplot.

Note that we use the grid unit `lines`, which equals the height of one line of text at the default size. This is the only grid unit which animint knows how to translate. It is not recommended to use other units such as `cm`.

A.2 List of data tables

The list of data tables idiom is very useful for creating interactive data visualizations of arbitrary complexity. The general form looks like

```
library(data.table)  
outer.data.list <- list()  
inner.data.list <- list()  
for(outer in outer.vec){  
  outer.dt <- computeOuter(outer)
```

```

outer.data.list[[paste(outer)]] <- data.table(outer, outer.dt)
for(inner in inner.vec){
  inner.dt <- computeInner(outer.dt, inner)
  inner.data.list[[paste(outer, inner)]] <-
    data.table(outer, inner, inner.dt)
}
}
outer.data <- do.call(rbind, outer.data.list)
inner.data <- do.call(rbind, inner.data.list)

```

Some comments:

- The first part of the idiom involves initializing empty lists. Here there are two, `outer.data.list` and `inner.data.list`. However there can be as many as necessary.
- The second part of the idiom is a bunch of nested for loops that assign data tables to elements of those lists.
- Functions like `computeOuter` and `computeInner` can be used, or you can just do the computations directly inside the for loop.
- To ensure that your code will run as fast as possible, use matrix-vector or vector-scalar operations in the innermost for loop. If you only do scalar-scalar operations in your innermost for loop, then you can definitely improve the performance of your code by removing that for loop and re-writing the computation in terms of vector-scalar operations.
- The `paste` function is used to assign a `data.table` to a named list element. Although in principle one could use either `data.frame` or `data.table`, in practice `data.table` is often much faster during the last combination step.
- The last part of the idiom uses `do.call` with `rbind` to combine the data tables stored during the for loops.

A.3 addColumn then facet

This idiom is useful for creating multi-panel ggplots with aligned axes. First, define a function which takes as input a data table and one or more values which will be used to add factors to that data table.

```

addColumn <- function(df, time.period){
  data.frame(df, time.period=factor(time.period, c("1975", "1960-2010")))
}
animint(
  ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=addColumn(WorldBank1975, "1975"))+
  geom_path(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    group=country),
    data=addColumn(WorldBankBefore1975, "1975"))+

```

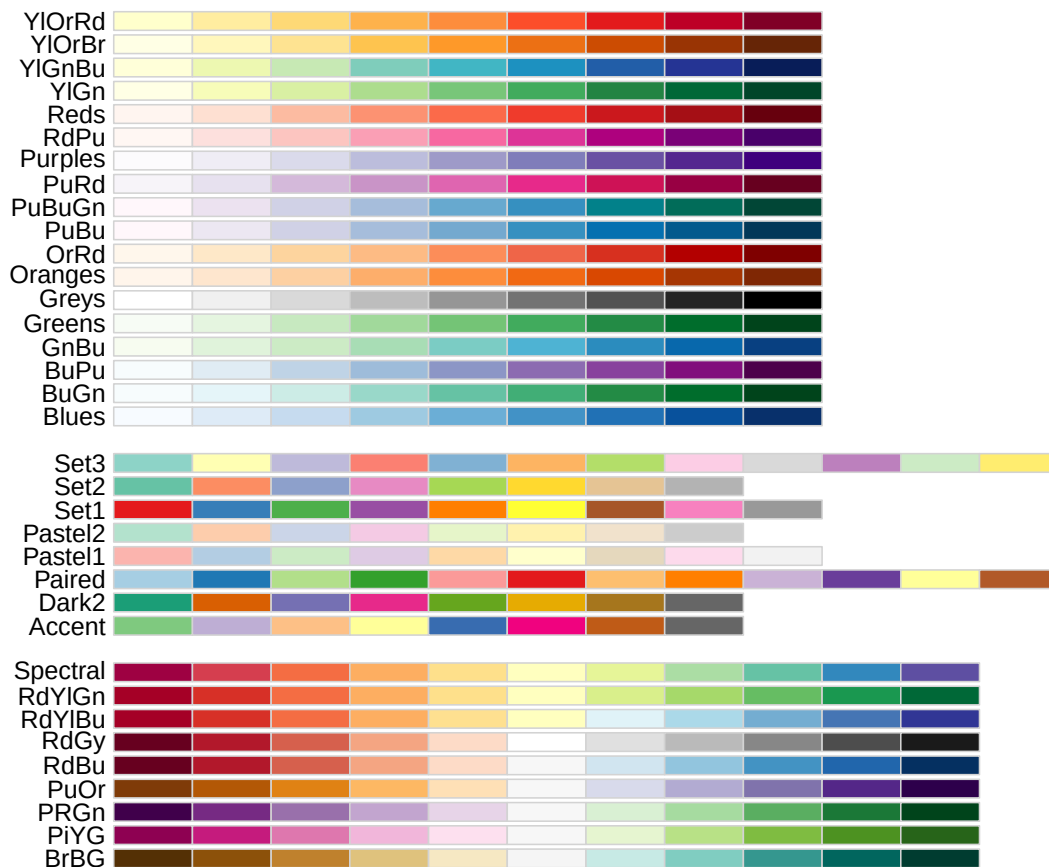
```
geom_line(aes(
  x=year, y=fertility.rate, color=region, group=country),
  data=addColumn(WorldBank, "1960-2010"))+
facet_grid(. ~ time.period, scales="free")+
xlab(""))
```

Note that `scales="free"` and `xlab("")` are used since the x axes now have very different units (year and life expectancy).

A.4 Manual color legends

Color and fill legends in `ggplot2` can be manually specified via `scale_color_manual` and `scale_fill_manual`. Typically we will choose one of the ColorBrewer palettes:

```
RColorBrewer::display.brewer.all()
```



For example to get the R code for the Set1 palette, we can write

```
dput(RColorBrewer::brewer.pal(Inf, "Set1"))
```

Warning in RColorBrewer::brewer.pal(Inf, "Set1"): n too large, allowed maximum for palette Set1 is 30
Returning the palette you asked for with that many colors

```
c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",  
"#A65628", "#F781BF", "#999999")
```

We can then copy that R code from the terminal and paste it into our text editor

```
data(WorldBank, package="animint2")  
region.colors <-  
  c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",  
    "#A65628", "#F781BF", "#999999")  
names(region.colors) <- levels(WorldBank$region)  
region.colors
```

```
      East Asia & Pacific (all income levels)
              "#E41A1C"
      Europe & Central Asia (all income levels)
              "#377EB8"
      Latin America & Caribbean (all income levels)
              "#4DAF4A"
      Middle East & North Africa (all income levels)
              "#984EA3"
      North America
              "#FF7F00"
      South Asia
              "#FFFF33"
      Sub-Saharan Africa (all income levels)
              "#A65628"
              <NA>
              "#F781BF"
              <NA>
              "#999999"
```

Then we can use it with `scale_color_manual`

```
library(animint2)  
ggplot()+  
  scale_color_manual(values=region.colors)+  
  geom_point(aes(  
    x=life.expectancy, y=fertility.rate, color=region),  
    data=WorldBank)
```

Warning: Removed 1490 rows containing missing values (geom_point).

