ETC3555 2018 - Lab 2

Introduction to ggplot2

Cameron Roach

31 July, 2018

Preliminaries

Introduction

This lab will focus on producing plots using `R`'s `ggplot2` package. We will cover:

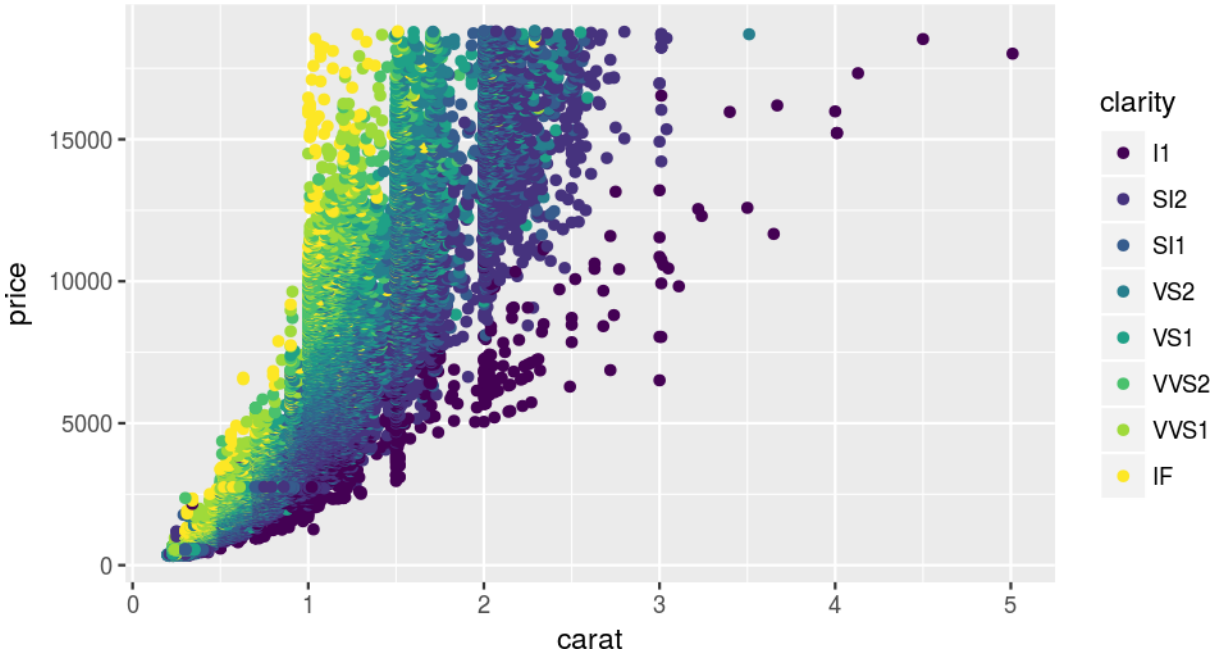
- scatter plots
- aesthetics
- useful geoms
- facetting plots
- adding smoothers
- formatting plots.

An online reference is available from ggplot2.tidyverse.org/reference/.

A short tutorial

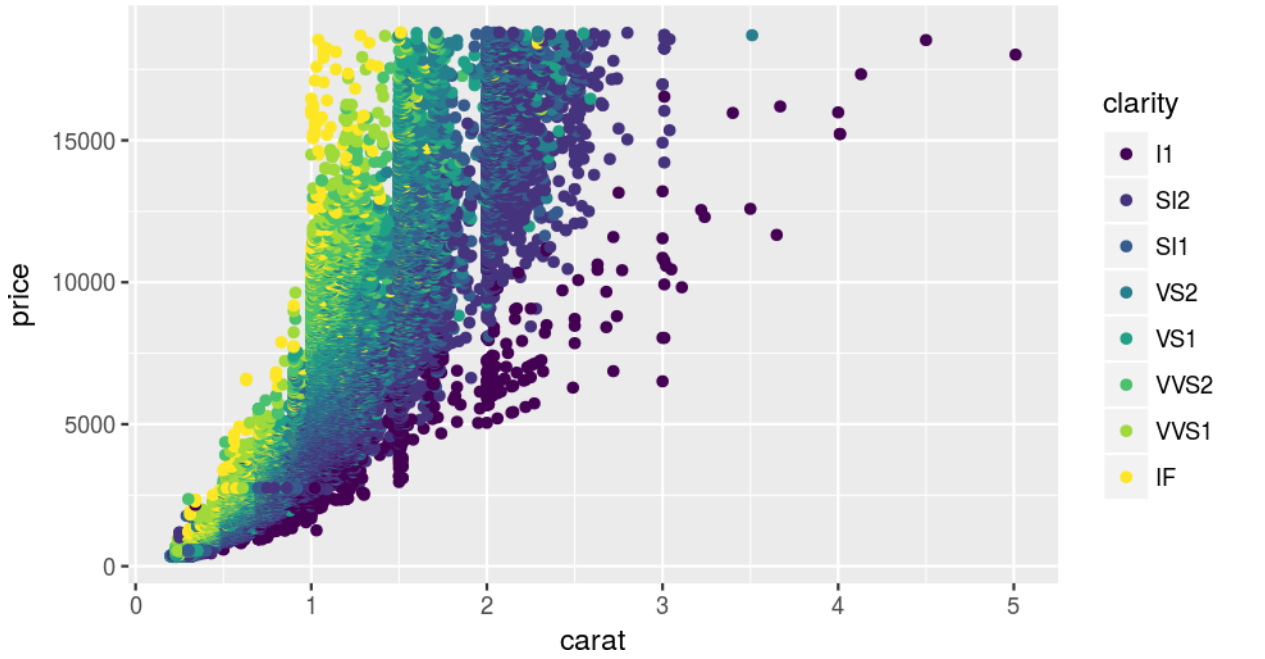
A typical `ggplot2` function takes the following form:

```
library(tidyverse)

ggplot(diamonds, aes(x = carat, y = price, colour = clarity)) +
  geom_point()
```



The initial `ggplot` call specifies the data frame to be used and the aesthetic mappings. Aesthetic mappings describe how the data will be mapped to various aesthetics in various geoms. Aesthetics typically include properties such as the x-axis variable, y-axis variable, point colours and shapes. However, each geom differs and users should refer to the `R` help files to see what aesthetics are available. If the data and aesthetics are declared inside the `ggplot` function they will be common to every geom that is added. Alternatively, if they are declared inside one of the geoms like so

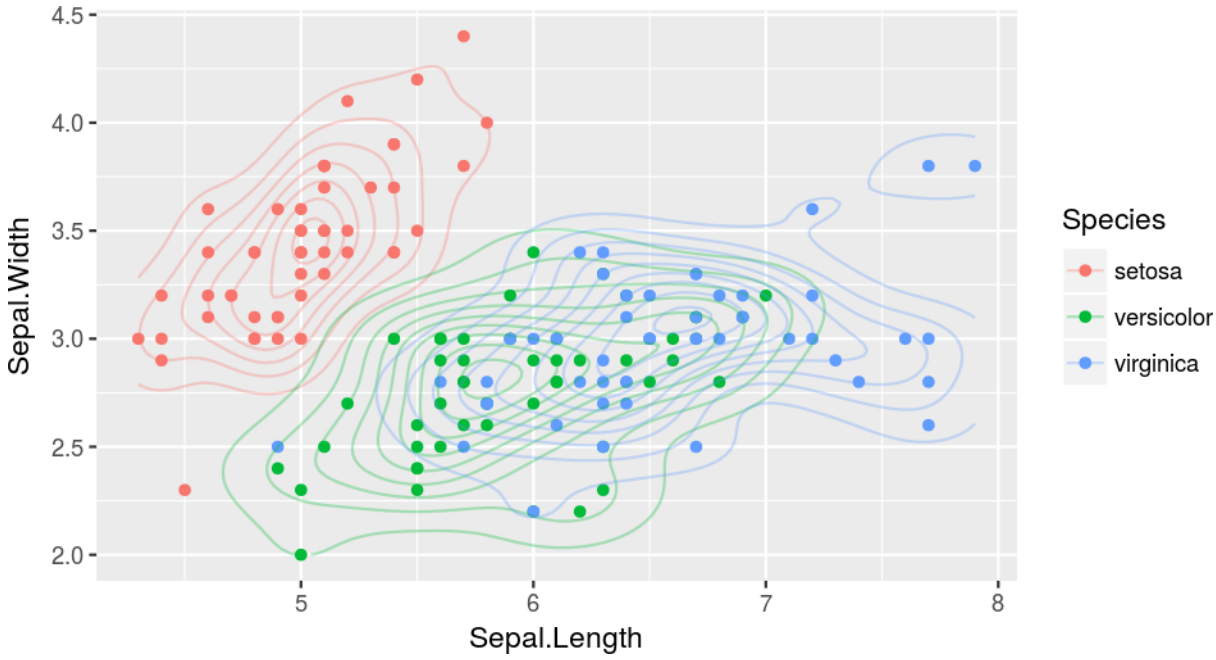
```
ggplot() +
  geom_point(data = diamonds, aes(x = carat, y = price, colour = clarity))
```



they will only apply to that geom. This can be useful when several data frames are to be used or when different geoms should have different aesthetics (e.g. colour points by clarity, but don't colour a smoother).

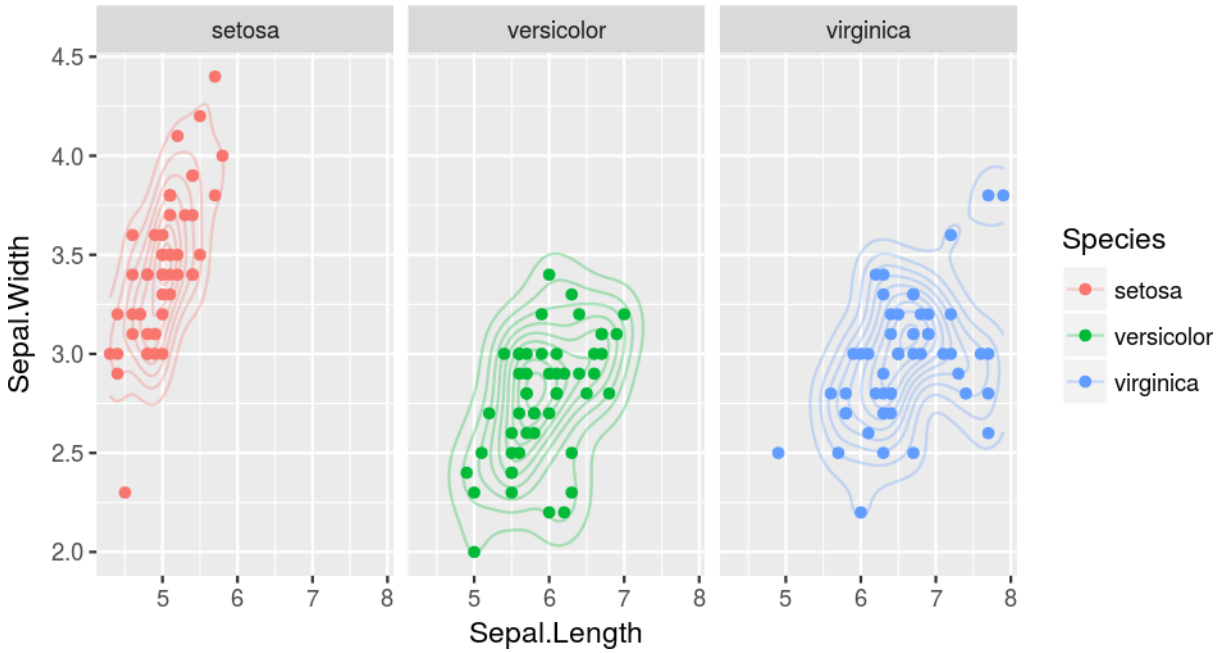
We can build up a ggplot object by sequentially adding layers. The following plot combines a simple scatter plot with 2D density contours:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +
  geom_point() +
  geom_density_2d(alpha = 0.3)
```



The contour plot is a bit unclear. We can use facetting to split the plot across several panels. The following code adds the `facet_wrap` function which creates a separate panel for each species.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +
  geom_point() +
  geom_density_2d(alpha = 0.3) +
  facet_wrap(~Species)
```



Exercises

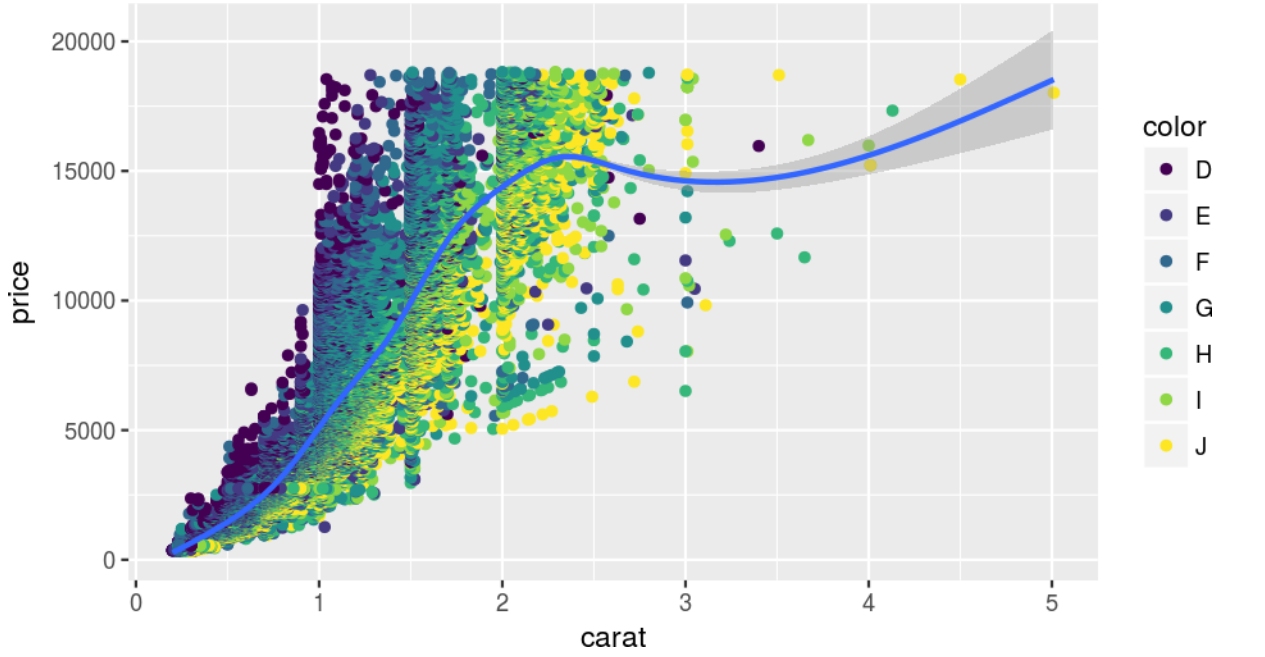
Exercise 1: Scatter plots and aesthetics

Load `tidyverse` and create a scatter plot using the `diamonds` data set. You can type `?diamonds` in the console to access a description of the data. Your scatter plot should have:

- carat on the x-axis
- price on the y-axis
- points coloured by diamond colour
- a smoother for the entire sample (i.e. not coloured by clarity).

You can add a smoother using the `geom_smooth` geom.

```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(aes(colour = color)) +
  geom_smooth()
```



Exercise 2: Geoms

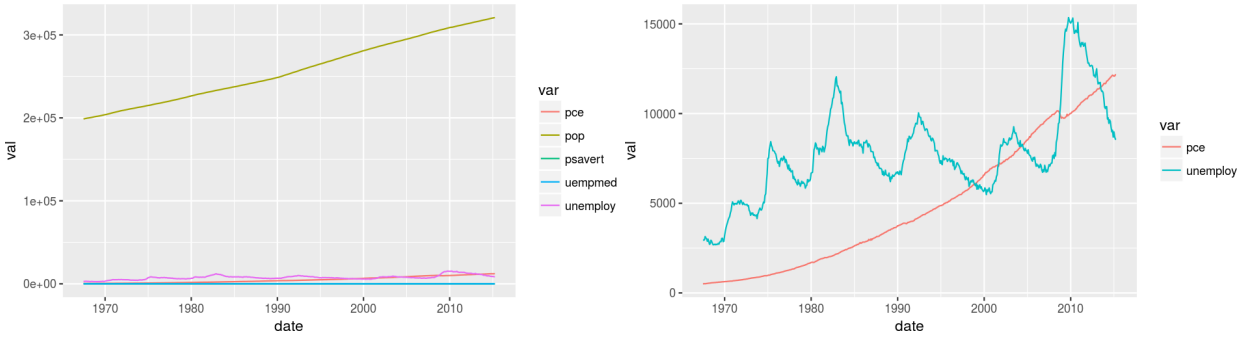
Line charts

The `geom_line` geom can be used to create connecting lines between observations in the data. Create a line chart using the `economics` data set with:

- date on the x-axis
- economic variable values on the y-axis
- lines coloured by economic variable.

Try plotting all variables initially. You will notice that plotting everything at once isn't particularly useful due to the different magnitudes of each economic indicator. We will explore a better way to visualise all the data at once using facetting in a later exercise. For the moment, recreate the plot but with only personal consumption expenditure and unemployment. Do you notice anything interesting in the time series?

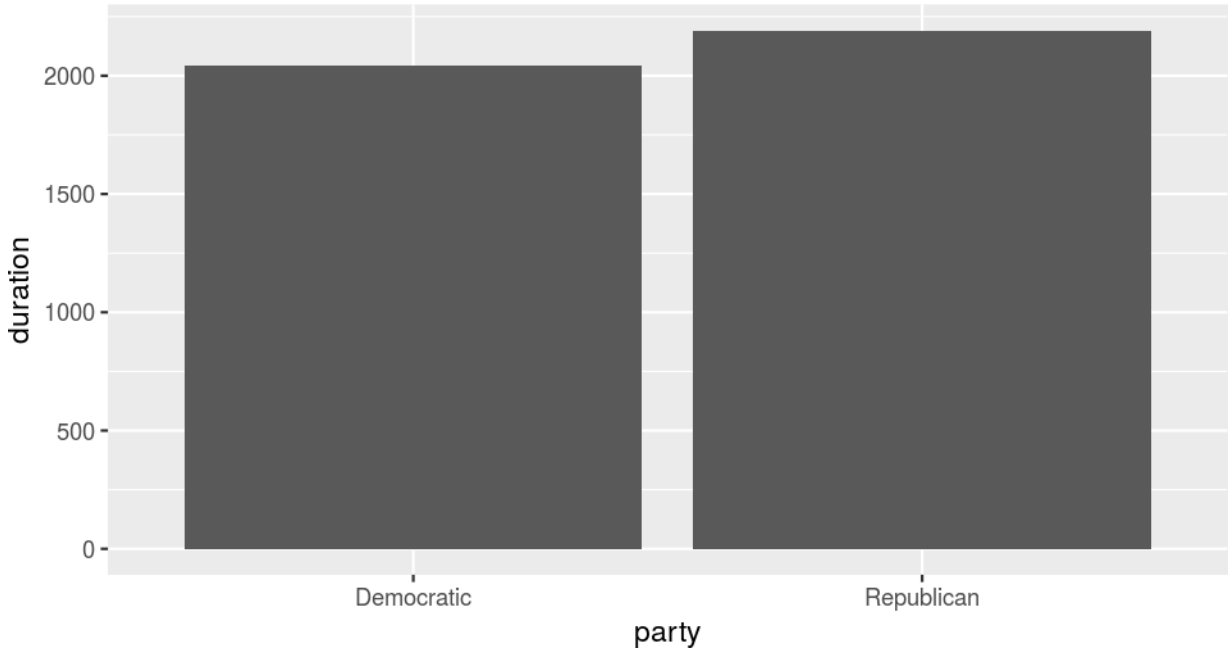
```
economics %>%
  gather(var, val, -date) %>%
  ggplot(aes(x = date, y = val, colour = var)) +
  geom_line()

economics %>%
  select(date, pce, unemploy) %>%
  gather(var, val, -date) %>%
  ggplot(aes(x = date, y = val, colour = var)) +
  geom_line()
```



Bar charts

Calculate the duration of each president using the `presidential` data set. Calculate the average presidential term for each party. Create a bar chart that shows the average presidential term in days for each party using the `geom_col` geom.

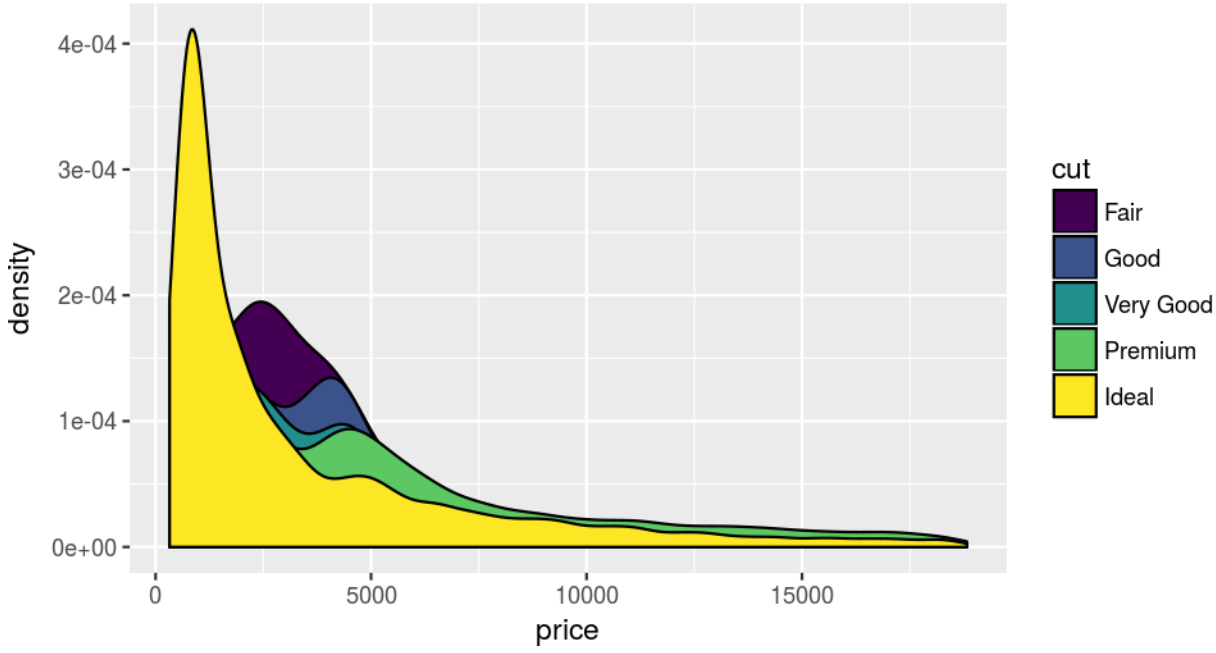
```
presidential %>%
  mutate(duration = end - start) %>%
  group_by(party) %>%
  summarise(duration = mean(duration)) %>%
  ggplot(aes(x = party, y = duration)) +
  geom_col()
```



Density plots

Use the `geom_density` geom to create density plots for the `diamonds` data set. Set the x-axis to price and the fill to cut quality.

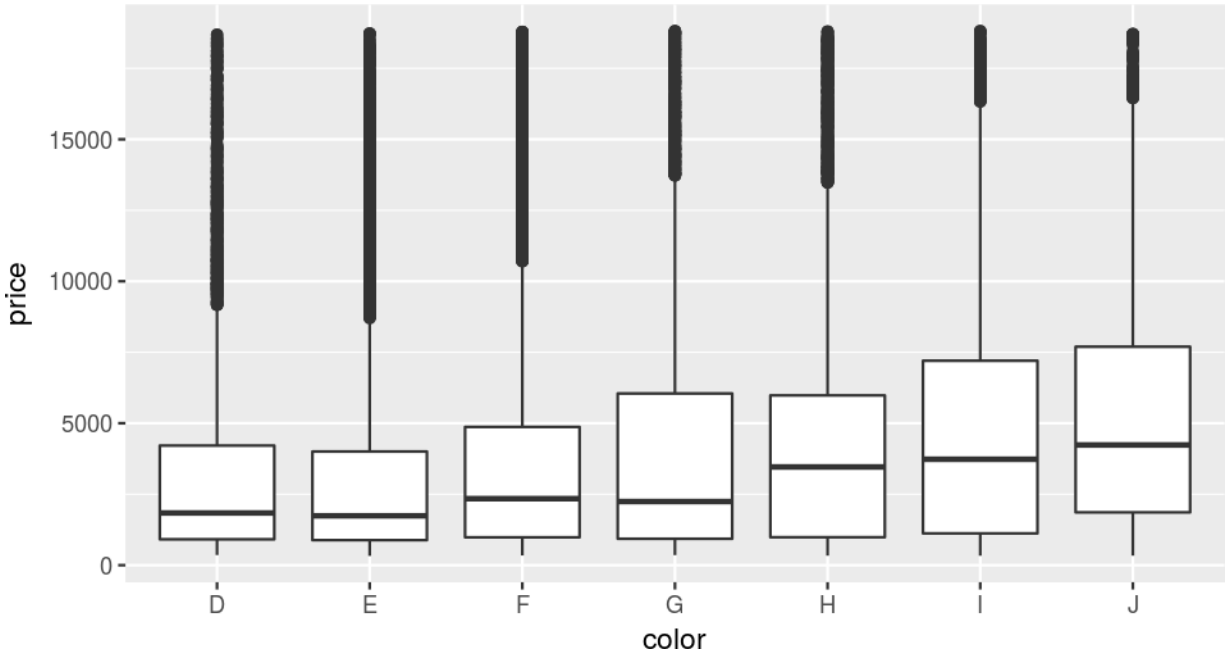
```
diamonds %>%
  ggplot(aes(x = price, fill = cut)) +
  geom_density()
```



Box plots

Use the `geom_boxplot` geom to create box plots for the `diamonds` data set. Set the x-axis to diamond colour and the y-axis to price.

```
diamonds %>%
  ggplot(aes(x = color, y = price)) +
  geom_boxplot()
```



Exercise 3: Facetting

The facetting options in `ggplot2` allow us to quickly create separate panels based on a variable. Read the documentation for `facet_wrap` and `facet_grid` and understand when you would use one or the other. Create a facetted plot using the economics data set so that each economic variable is plotted in its own panel. When facetting make sure to:

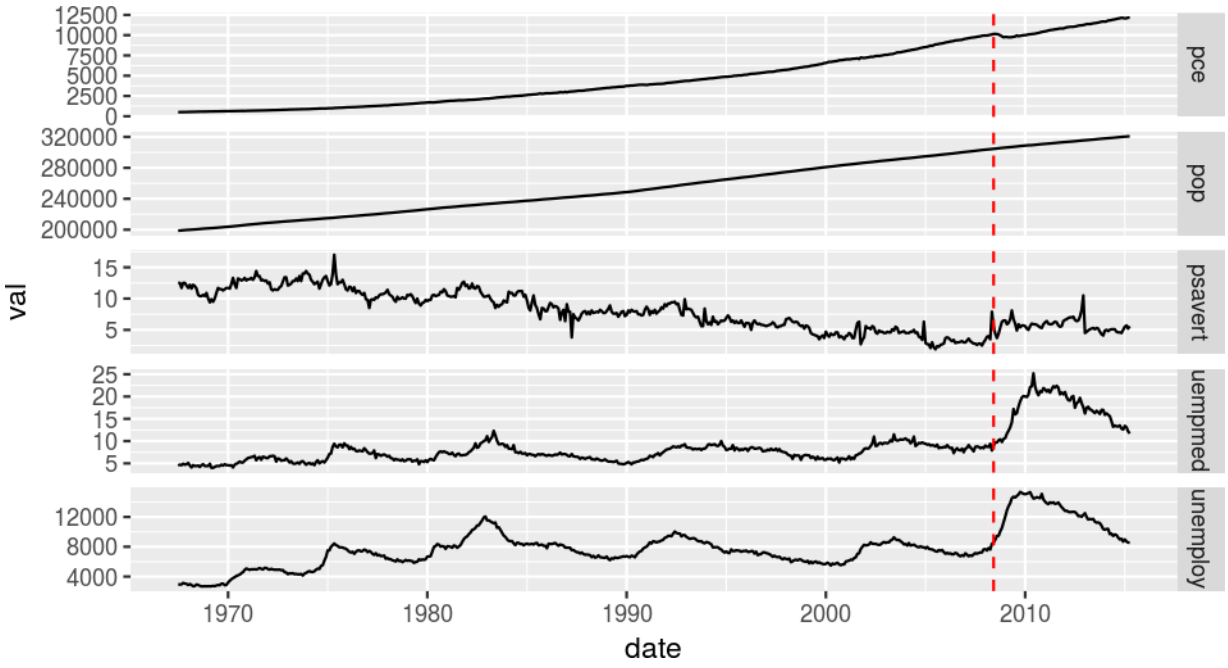
- use the `scales` argument to ensure each variable has its own y-scale
- use the `ncol` argument so that only one column of panels is created
- push the strip position to the right using the `strip.position` argument.

Once you have your plot laid out correctly you can add a vertical line using this geom

```
geom_vline(xintercept = dmy("01/06/2008"), linetype = "dashed", colour = "red")
```

Make sure you have loaded the `lubridate` package which contains the `dmy` function. The `lubridate` package is part of the tidyverse (although it is not loaded by default) and contains several helper functions when dealing with dates and times. Here, the `dmy` function automatically converts our string to a date object so that the vertical line may be plotted in the correct place.

```
library(lubridate)  # for dmy function

economics %>%
  gather(var, val, -date) %>%
  ggplot(aes(x = date, y = val)) +
  geom_line() +
  geom_vline(xintercept = dmy("01/06/2008"), linetype = "dashed", colour = "red") +
  facet_wrap(~var, scales = "free_y", ncol = 1, strip.position = "right")
```



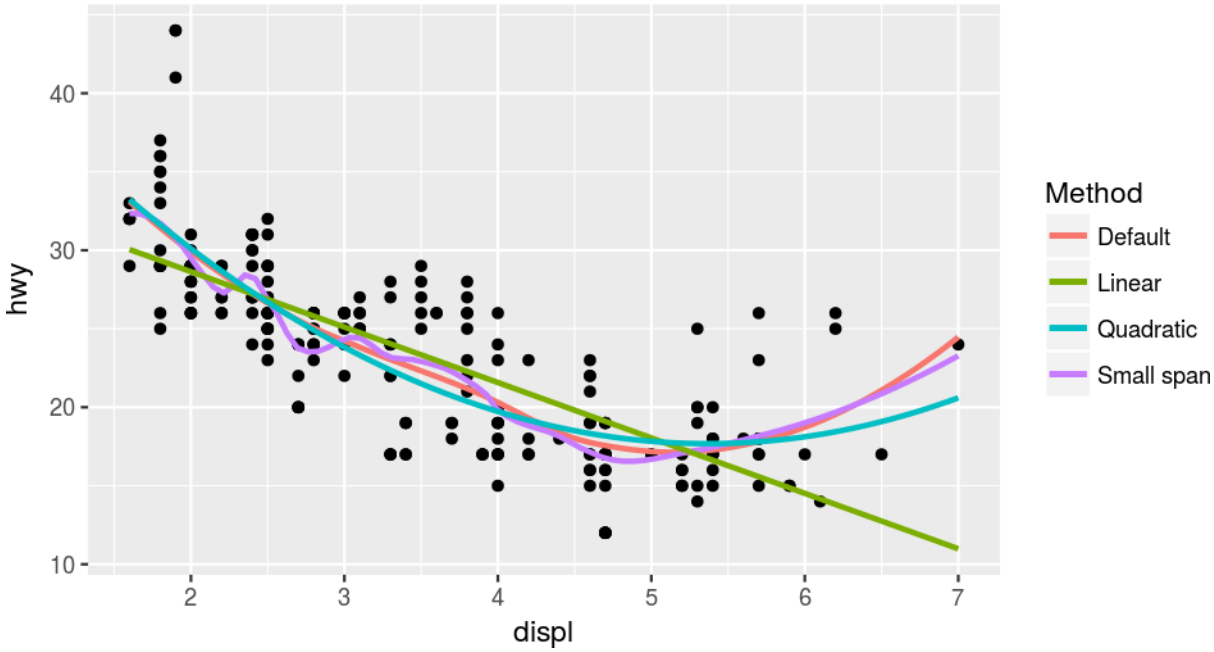
Exercise 4: Smoothers

We previously saw that you can add a smoother to our plots. Create a scatter plot using the `mpg` data set with engine displacement on the x-axis and highway miles per gallon on the y-axis (type `?mpg` to see a description of the data). Fit four smoothers using `geom_smooth`:

- loess with default values
- loess with `span` of 0.3
- a linear smoother
- a quadratic smoother.

Use the `method` and `formula` arguments to convert the smoothed line to a linear fit with an appropriate formula. Which smoother do you prefer? Can you figure out a way to plot all the smoothed lines at once and include a legend? (Hint: try adding a colour aesthetic to each `geom_smooth`).

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(aes(colour = "Default"), se = FALSE) +
  geom_smooth(aes(colour = "Small span"), se = FALSE, span = 0.3) +
  geom_smooth(aes(colour = "Linear"), se = FALSE, method = "lm",
              formula = y ~ x) +
  geom_smooth(aes(colour = "Quadratic"), se = FALSE, method = "lm",
              formula = y ~ poly(x, 2)) +
  labs(colour = "Method")
```



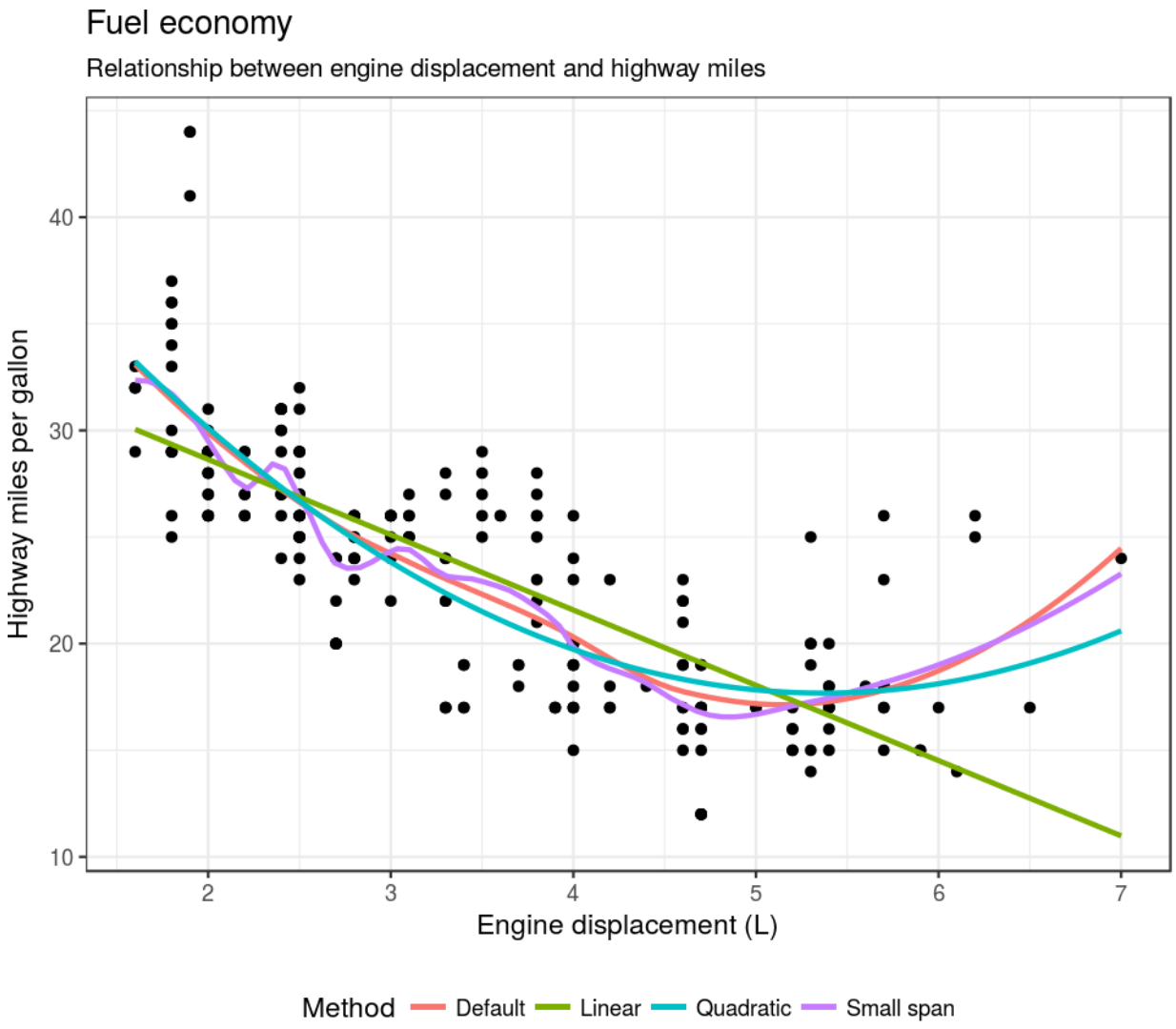
Exercise 5: Formatting plots

Some useful functions for formatting plots are:

- `labs` specifies chart titles and axis labels
- `xlim` and `ylim` changes the plot limits
- `theme` tweaks the plot layout
- `theme_bw`, `theme_dark`, etc. are complete themes that override all display settings of a plot.

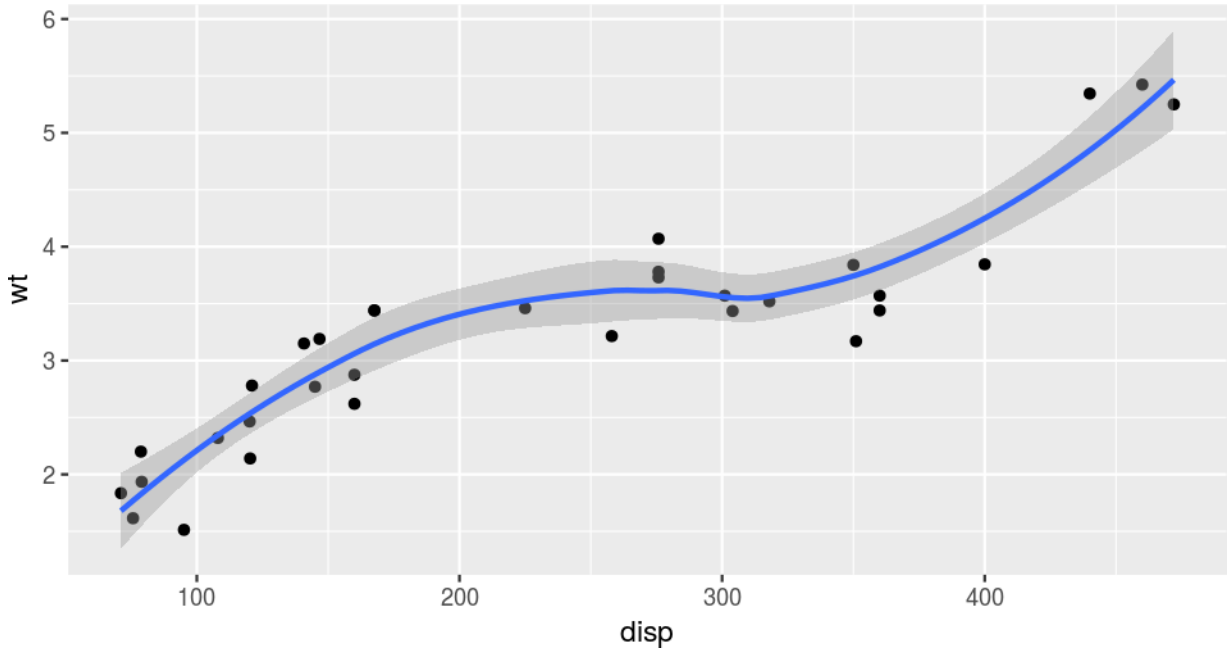
Use `labs` to add a title, subtitle, caption and better axis labels to the `mpg` plot you created in the previous exercise. Title and subtitle should explain the plot whereas caption should give the data source. Add the `theme_bw` layer to the plot. Finally, use `theme` to move the legend from the default position to the bottom of the plot (hint: use the `legend.position` argument).

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(aes(colour = "Default"), se = FALSE) +
  geom_smooth(aes(colour = "Small span"), se = FALSE, span = 0.3) +
  geom_smooth(aes(colour = "Linear"), se = FALSE, method = "lm",
              formula = y ~ x) +
  geom_smooth(aes(colour = "Quadratic"), se = FALSE, method = "lm",
              formula = y ~ poly(x, 2)) +
  labs(title = "Fuel economy",
       subtitle = "Relationship between engine displacement and highway miles",
       x = "Engine displacement (L)",
       y = "Highway miles per gallon",
       caption = "Data: http://fueleconomy.gov",
       colour = "Method") +
  theme_bw() +
  theme(legend.position = "bottom")
```

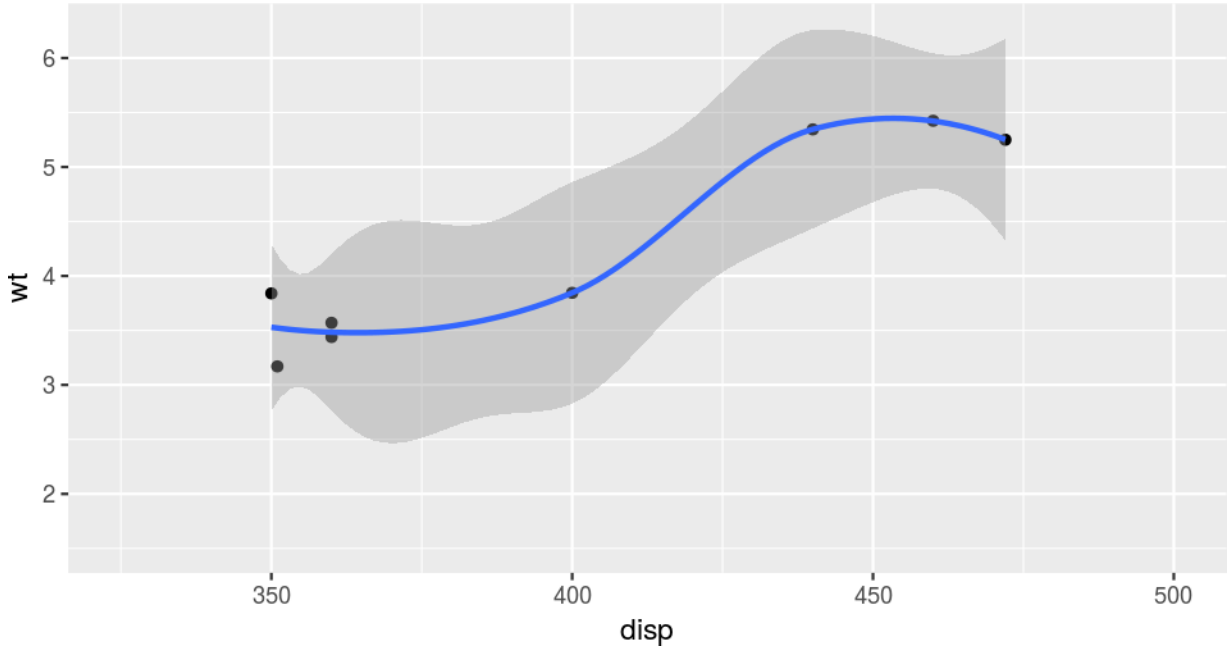


If you ever use the `xlim` or `ylim` options be careful. Any values outside the limits will be replaced by NAs in the plot - hence smoothers will be affected. If you wish to apply a smoother to all the data *and then* zoom in you should use `coord_cartesian`. Take the following plot and limit the x-axis between 325 and 500 using `xlim` and `coord_cartesian`. How does the smoother change? Which method is correct?

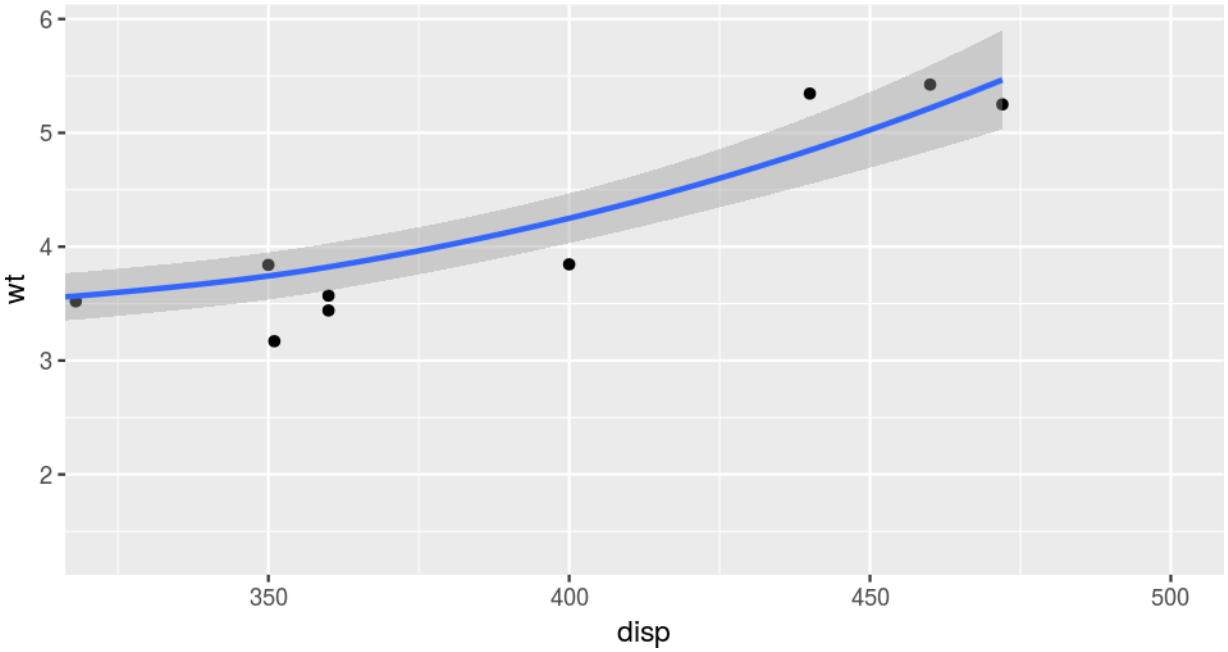
```
p <- ggplot(mtcars, aes(disp, wt)) +
  geom_point() +
  geom_smooth()

p
```




```
p + xlim(325, 500)
```




```
p + coord_cartesian(xlim = c(325, 500))
```


6
5
4
3
2
wt
350
400
450
500
disp