Data Visualization with R

Claudia A Engel

Last updated: April 20, 2018

Contents

Prerequisites and Preparations

To get the most out of this workshop you should have:

- a **basic knowledge** of R and/or be familiar with the topics covered in the Introduction to R.
- have a recent version of R and RStudio installed.
- have installed the `tidyverse` package.

Recommended:

- Create a new RStudio project `R-data-viz` in a new folder `R-data-viz` and download both CSV files into a subdirectory called `data`:
  - Download `MS_stops.csv` from here: https://github.com/cengel/R-data-viz/raw/master/data/MS_stops.csv
  - Download `MS_county_stops.csv` from here: https://github.com/cengel/R-data-viz/raw/master/data/MS_county_stops.csv
  - If you have your working directory set to `R-data-viz` which contains a folder called `data` you can copy, paste, and run the following two lines in R:

```
download.file("https://github.com/cengel/R-data-viz/raw/master/data/MS_stops.csv",
              "data/MS_stops.csv")

download.file("https://github.com/cengel/R-data-viz/raw/master/data/MS_stops_by_county.csv",
              "data/MS_stops_by_county.csv")
```

- Open up a new R Script file `R-data-viz.R` for the code you'll create during the workshop.

References

Murrell, P. (2012): R Graphics, 2nd ed. Stanford only online access

Rahlf, T (2017): Data Visualisation with R. http://www.springer.com/us/book/9783319497501

Wickham, H. (2016): ggplot2: Elegant Graphics for Data Analysis. 2nd ed. http://link.springer.com/book/10.1007/978-3-319-24277-4

Acknowledgements

Part of the materials for this tutorial are adapted from http://datacarpentry.org and http://softwarecarpentry.org.

Data sample taken from https://openpolicing.stanford.edu/


```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : EOF within quoted string
```

Chapter 1

Data Visualization with ggplot2

Learning Objectives

- Bind a data frame to a plot
- Select variables to be plotted and variables to define the presentation such as size, shape, color, transparency, etc. by defining aesthetics (`aes`)
- Add a graphical representation of the data in the plot (points, lines, bars) adding "geoms" layers
- Produce univariate plots, barplots, boxplots, and line plots using ggplot.
- Produce bivariate plots, like scatter plots, hex plots, stacked bar, and bivariate line charts using ggplot.
- Modify the aesthetics for the entire plot as well as for individual "geoms" layers
- Modify plot elements (labels, text, scale, orientation)
- Group observations by a factor variable
- Break up plot into multiple panels (facetting)
- Apply ggplot themes and create and apply customized themes
- Save a plot created by ggplot as an image

We start by loading the required packages. **ggplot2** is included in the **tidyverse** package.

```
library(tidyverse)
```

Next we load the data into R:

```
stops_county <- read.csv('data/MS_stops_by_county.csv')
```

These data are a small subset extracted from the openpolicing.stanford.edu dataset and contain aggregated for each county in Missisippi. Let's take a look at the data.

```
head(stops_county)
str(stops_county)
```

1.1 Plotting with ggplot2

ggplot2 is a plotting package that makes it simple to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties, so we only need minimal changes if the underlying data change or if we decide to change from a bar plot to a scatterplot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

ggplot generally likes data in the 'long' format: i.e., a column for every dimension, and a row for every observation. Well structured data will save you lots of time when making figures with ggplot.

ggplot graphics are built step by step by adding new elements using the `+` sign.

To build a ggplot we need to:

- bind the plot to a specific data frame using the `data` argument

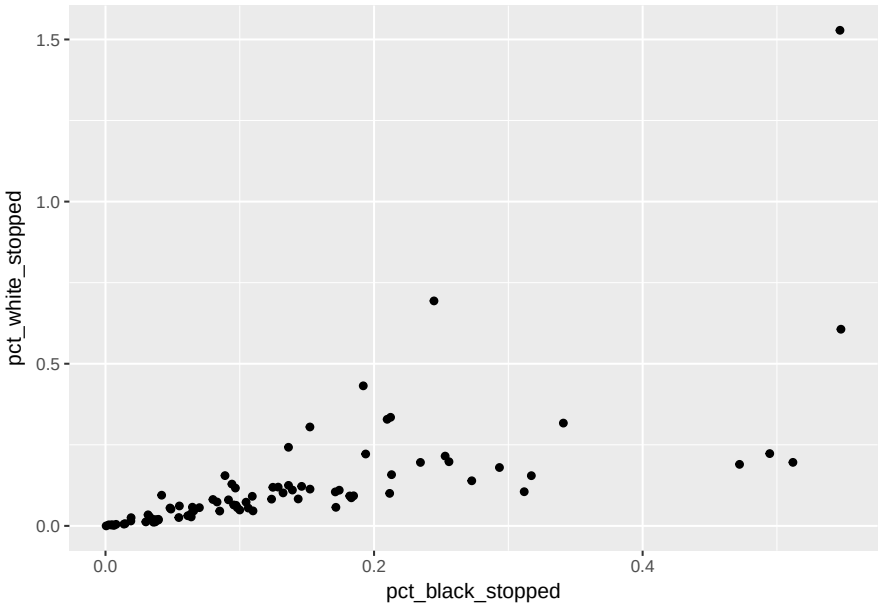
```
ggplot(data = stops_county)
```

- define aesthetics (`aes`), by selecting the variables to be plotted and the variables to define the presentation such as plotting size, shape color, etc.

```
ggplot(data = stops_county, aes(x = pct_black_stopped, y = pct_white_stopped))
```

- add "geoms" – a graphical representation of the data in the plot (points, lines, bars). To add a geom to the plot use `+` operator

```
ggplot(data = stops_county, aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point()
```



The `+` in the **ggplot2** package is particularly useful because it allows you to modify existing `ggplot` objects. This means you can easily set up plot "templates" and conveniently explore different types of plots, so the above plot can also be generated with code like this:

```
# Assign plot to a variable
MS_plot <- ggplot(data = stops_county, aes(x = pct_black_stopped, y = pct_white_stopped))

# Draw the plot
MS_plot + geom_point()
```

Notes:

- Any parameters you set in the `ggplot()` function can be seen by any geom layers that you add (i.e., these are universal plot settings). This includes the x and y axis you set up in `aes()`.
- Any parameters you set in the `geom_*()` function are treated independently of (and override) the settings defined globally in the `ggplot()` function.
- Geoms are plotted in the order they are added after each `+`, that means geoms last added will display on top of prior geoms.

- The `+` sign used to add layers **must be placed at the end of each line** containing a layer. If, instead, the `+` sign is added in the line before the other layer, **`ggplot2`** will not add the new layer and will return an error message.

```
# this is the correct syntax for adding layers
MS_plot +
  geom_point()

# this will not add the new layer and will return an error message
MS_plot
  + geom_point()
```

To learn more about **`ggplot`** after the workshop, you may want to check out this cheatsheet about **`ggplot`**.

1.2 Building your plots iteratively

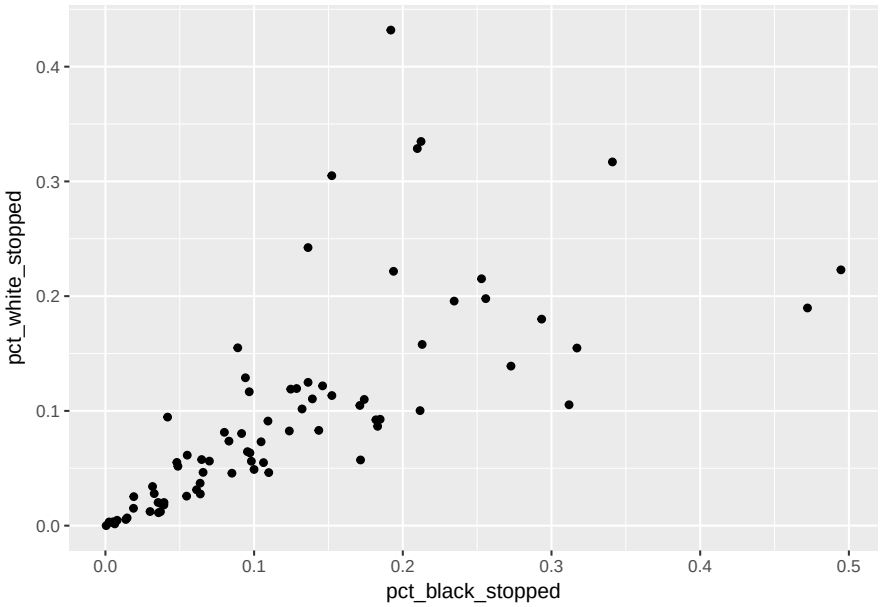
Building plots with ggplot can be of great help when you engage in exploratory data analysis. It is typically an iterative process, where you go back and forth between your data and their graphical representation, which helps you in the process of getting to know your data better.

Conveniently, `ggplot` works with pipes. The code below does the same thing as above:

```
stops_county %>%
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point()
```

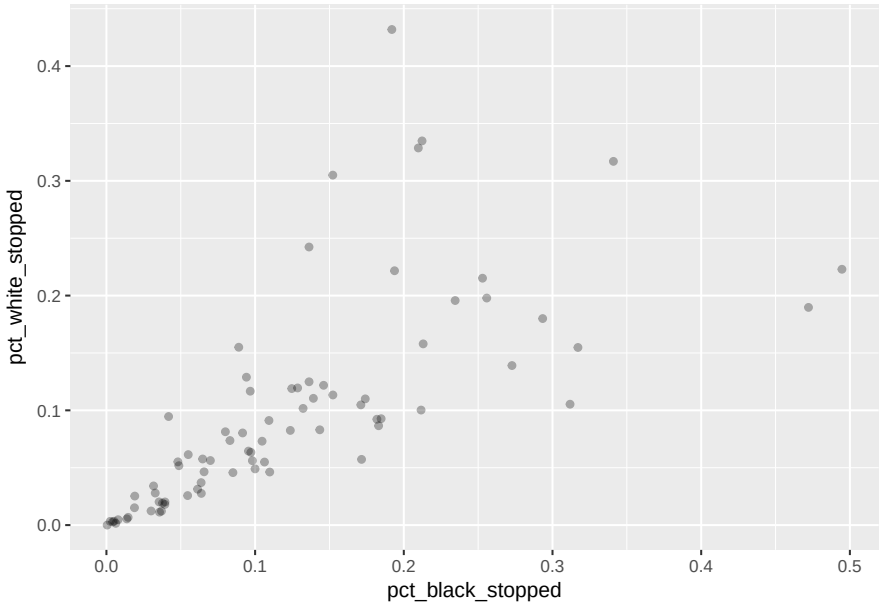
We pipe the content of the table into `ggplot()`, so we can omit the first (`data =` argument). Now let's use this to clean up a few odd outliers in our data before we pass them to ggplot.

```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point()
```



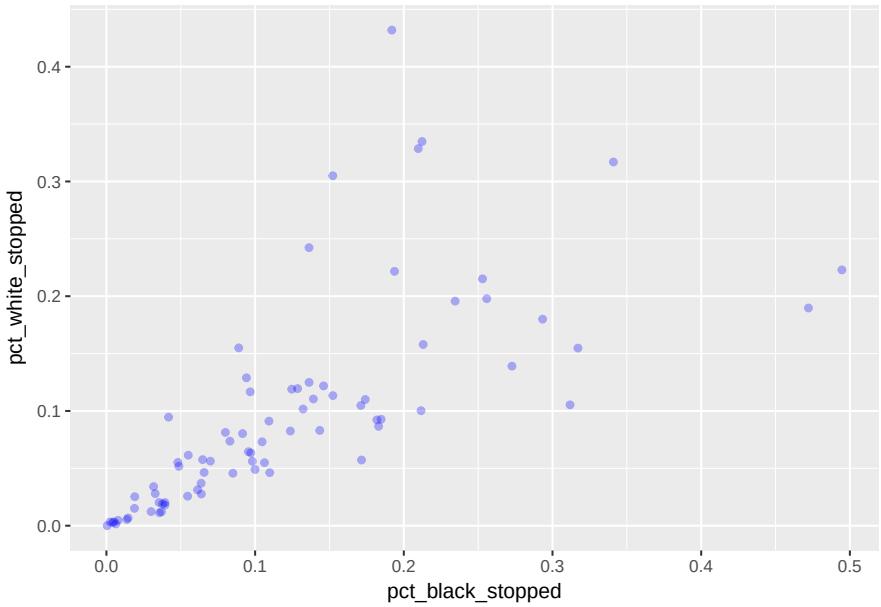
Then we can start modifying this plot to extract more information from it. For instance, we can add transparency (`alpha`) to avoid overplotting:

```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point(alpha = 0.3)
```



We can also add a color for all the points:

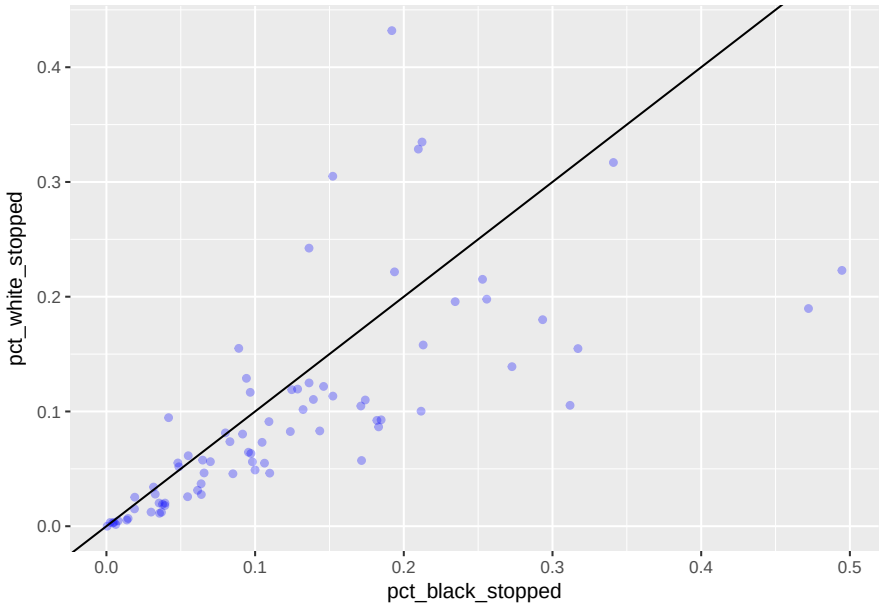
```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point(alpha = 0.3, color= "blue")
```



We can add another layer to the plot with +:

```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
```

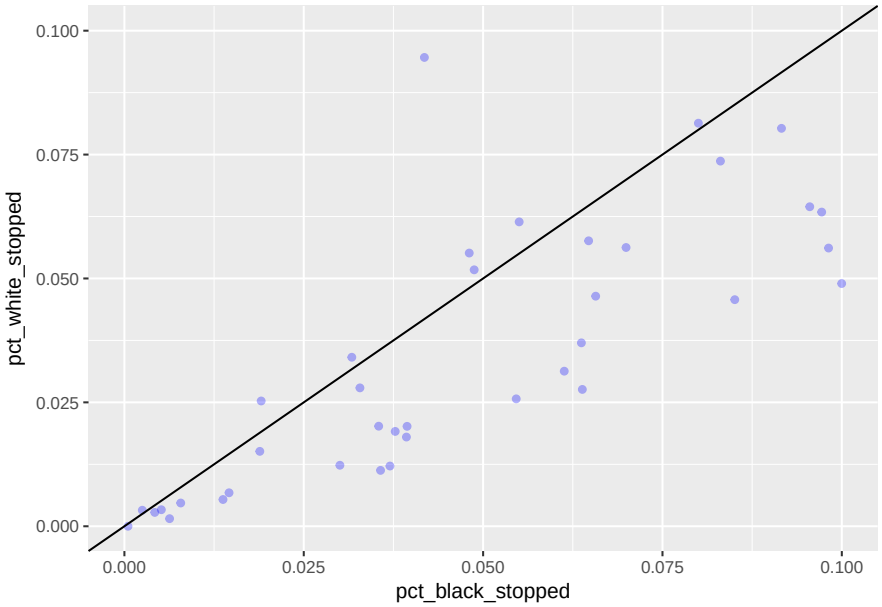
```
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point(alpha = 0.3, color= "blue") +
  geom_abline(intercept = 0)
```



If we wanted to "zoom" into the plot, we could filter to a smaller range of values before passing them to ggplot, but we can also tell ggplot to only plot the x and y values for certain ranges. For this we use `scale_x_continuous` and `scale_y_continuous`. You will receive a message from ggplot telling you how many rows it has removed from the plot.

```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
  ggplot(aes(x = pct_black_stopped, y = pct_white_stopped)) +
  geom_point(alpha = 0.3, color= "blue") +
  geom_abline(intercept = 0) +
  scale_x_continuous(limits = c(0, 0.1)) +
  scale_y_continuous(limits = c(0, 0.1))
```

```
#> Warning: Removed 40 rows containing missing values (geom_point).
```



Challenge

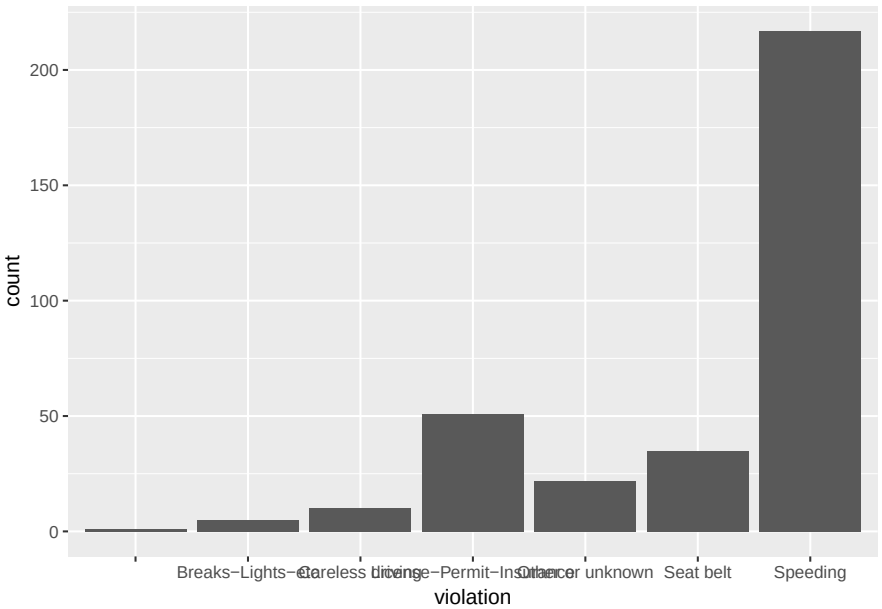
Modify the plot above to display different color for both points and abline, and show a different range of data. How might you change the size of the dots?

1.3 Barplot

There are two types of bar charts in ggplot, `geom_bar` and `geom_col`. `geom_bar` makes the height of the bar proportional to the number of cases in each group and counts the number of cases at each x position.

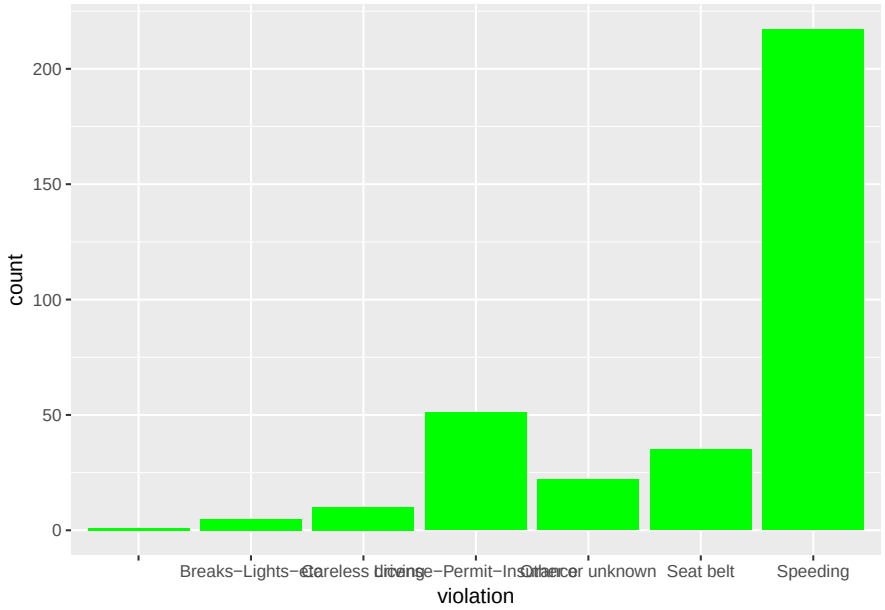
If we wanted to see how many violations we have of each type could say:

```
ggplot(trafficstops, aes(violation)) +
  geom_bar()
```



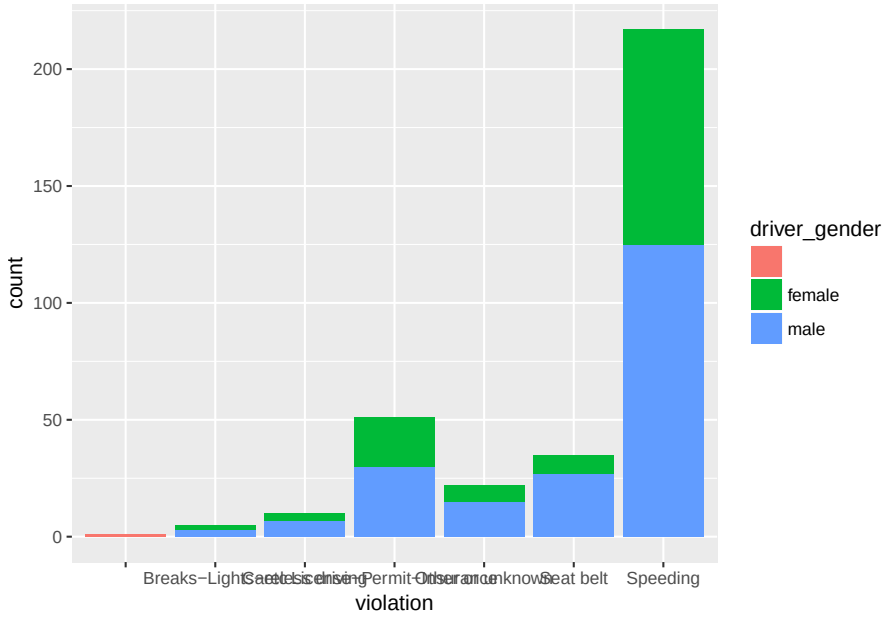
As we have seen we could color the bars, but instead of `color` we use `fill`. (What happens when you use `color`?)

```
ggplot(trafficstops, aes(violation)) +
  geom_bar(fill = "green")
```



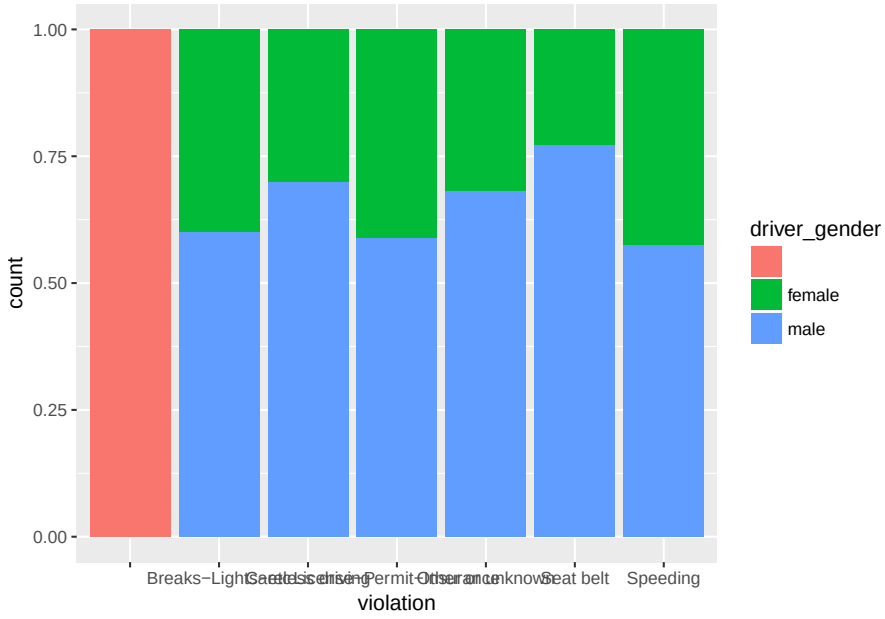
Instead of coloring everything the same we could also color by another category, say gender. For this we have to set the parameter within the `aes()` function, which takes care of mapping the values to different colors:

```
ggplot(trafficstops, aes(violation)) +
  geom_bar(aes(fill = driver_gender))
```



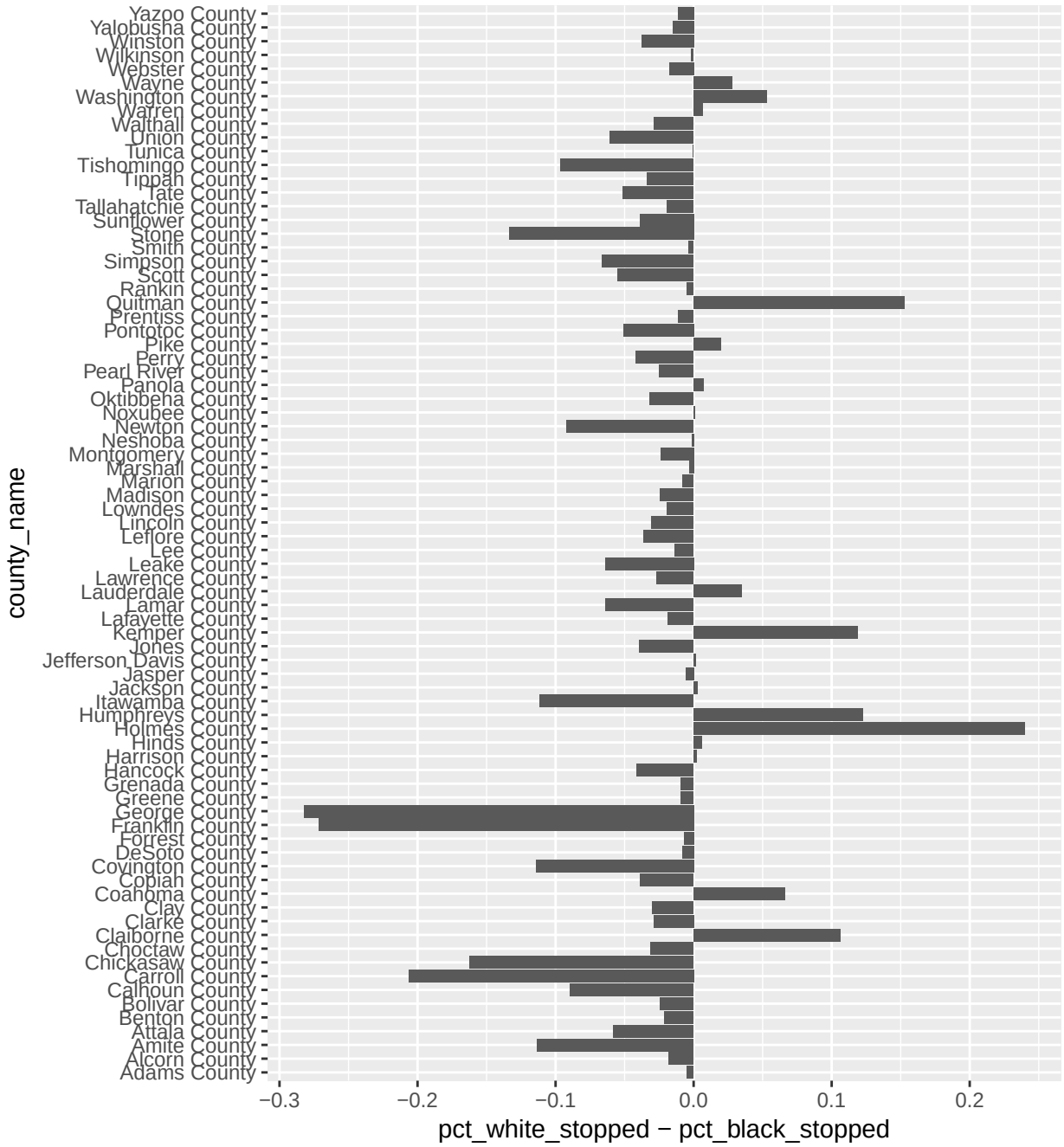
If we wanted to see the proportions within each category we can tell ggplot to stretch the bars between 0 and 1, we can set the position parameter to 'fill':

```
ggplot(trafficstops, aes(violation)) +
  geom_bar(aes(fill = driver_gender), position = "fill")
```



The other type of barchart, `geom_col`, is used if you want the heights of the bars to represent values in the data. It leaves the data as is. For example, we can use `geom_col` for a different way of visualizing the data shown in the scatterplot above. For readability I have also flipped the coordinates:

```
stops_county %>%
  filter(pct_white_stopped < 0.5 & pct_black_stopped < 0.5) %>%
  ggplot(aes(x = county_name, y = pct_white_stopped - pct_black_stopped)) +
    geom_col() +
    coord_flip()
```



Challenge

Make a barplot that shows for each race the proportion of stops for male and female drivers. How could you get rid of the NAs?

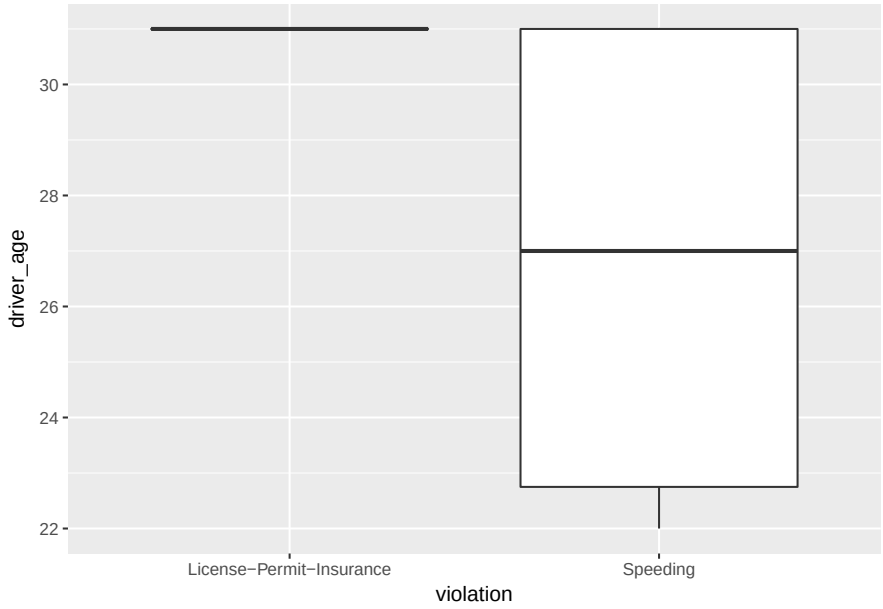
1.4 Boxplot

For this segment let's extract and work with the stops for Chickasaw County only.

```
Chickasaw_stops <- filter(trafficstops, county_name == "Chickasaw County")
```

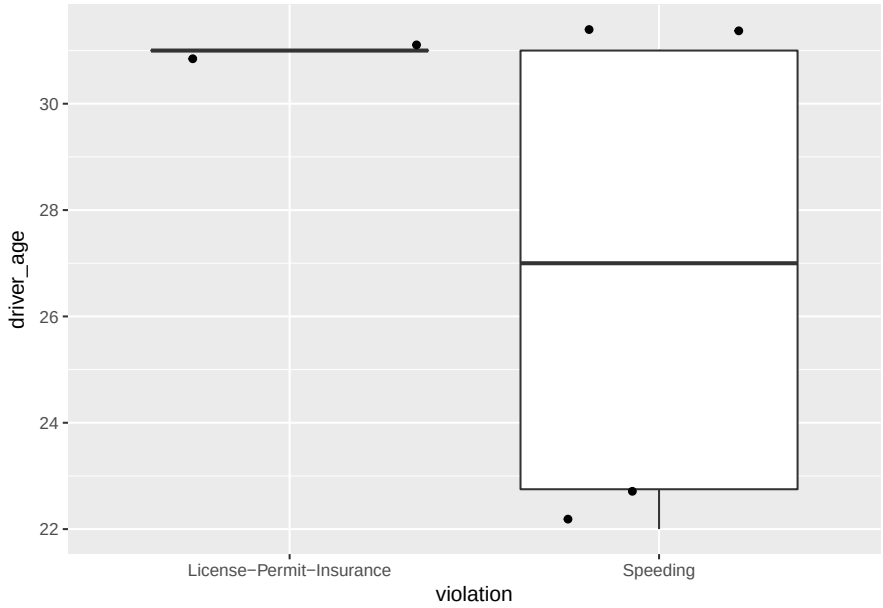
We can use boxplots to visualize the distribution of driver age within each violation:

```
ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
    geom_boxplot()
```



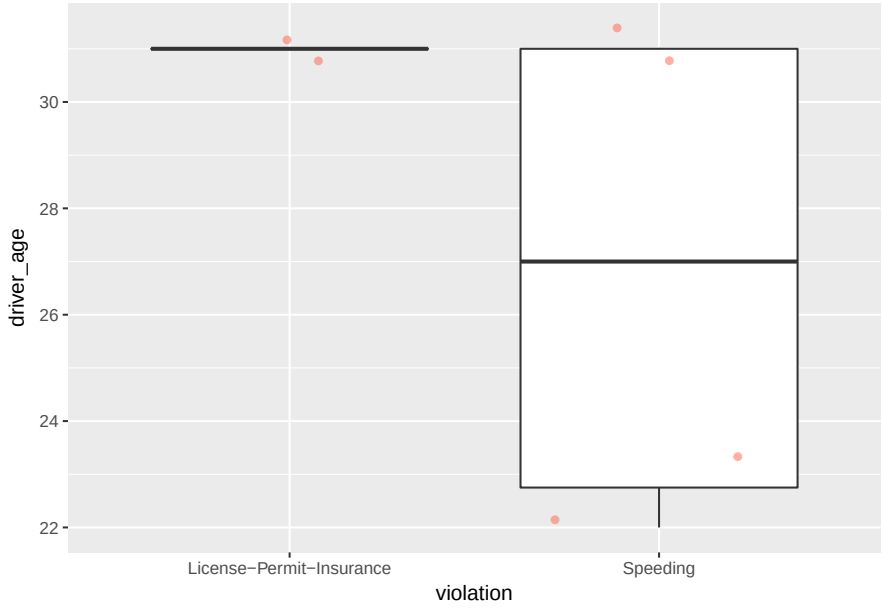
By adding points to boxplot, we can have a better idea of the number of measurements and of their distribution.

```
ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
    geom_boxplot() +
    geom_jitter()
```



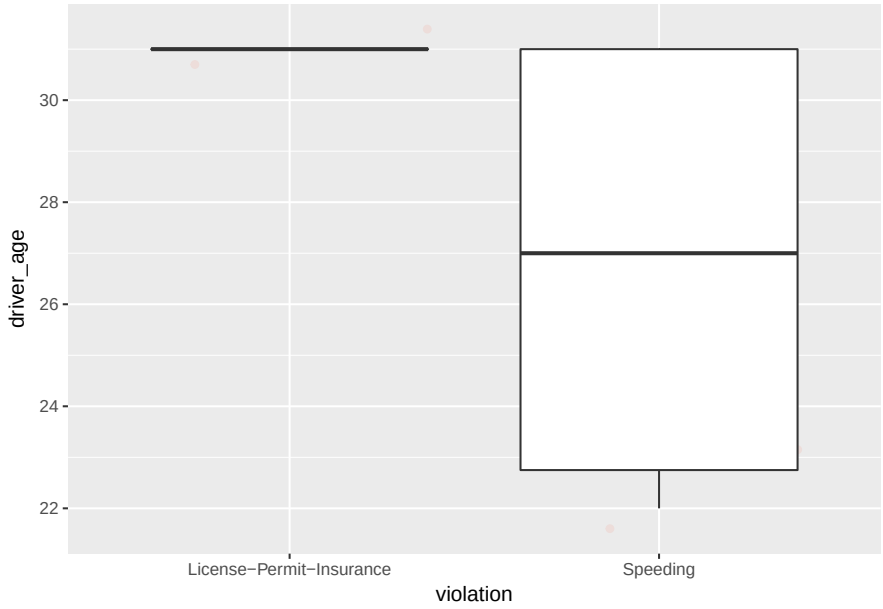
That looks quite messy. Let's clean it up by using the `alpha` parameter to make the dots more transparent and also change their color:

```
ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
    geom_boxplot() +
    geom_jitter(alpha = 0.5, color = "tomato")
```



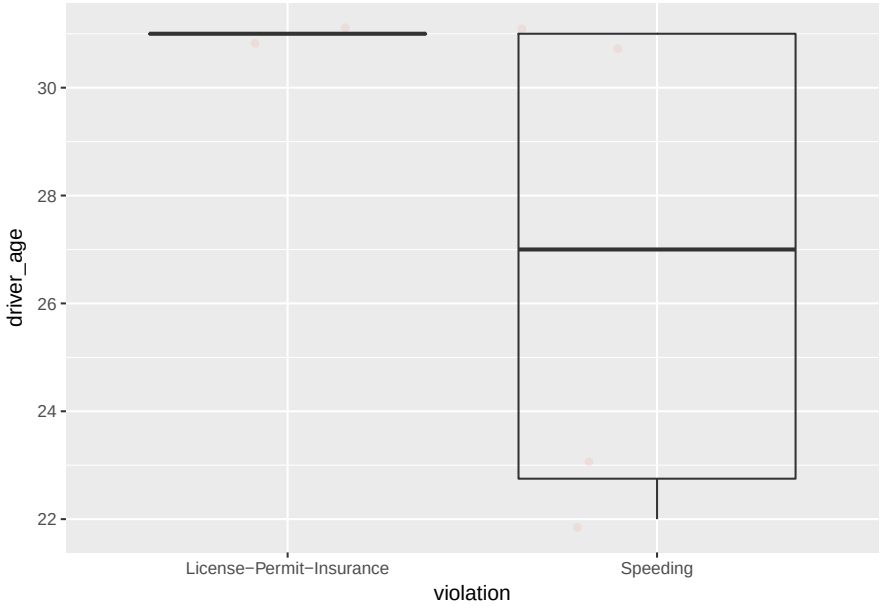
Notice how the boxplot layer is behind the jitter layer. We will change the plotting order to keep the boxplot visible.

```
ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
    geom_jitter(alpha = 0.1, color = "tomato") +
    geom_boxplot()
```



And finally we will change the transparency of the box plot so it does not cover the points:

```
ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
    geom_jitter(alpha = 0.1, color = "tomato") +
    geom_boxplot(alpha = 0)
```



Challenge

Boxplots are useful summaries, but hide the *shape* of the distribution. For example, if there is a bimodal distribution, it would not be observed with a boxplot. An alternative to the boxplot is the violin plot (sometimes known as a beanplot), where the shape (of the density of points) is drawn.

- Replace the box plot with a violin plot; see `geom_violin()`.

So far, we've looked at the distribution of age within violations Try making a new plot to explore the distribution of age for another variable:

- Create the age box plot for `driver_race`. Overlay the boxplot layer on a jitter layer to show actual measurements.

1.5 Plotting time series data

To make things a little easer we first convert the date column we plan to use to Date format.

```
library(lubridate)
class(trafficstops$stop_date)
trafficstops$stop_date <- ymd(trafficstops$stop_date)
class(trafficstops$stop_date)
```

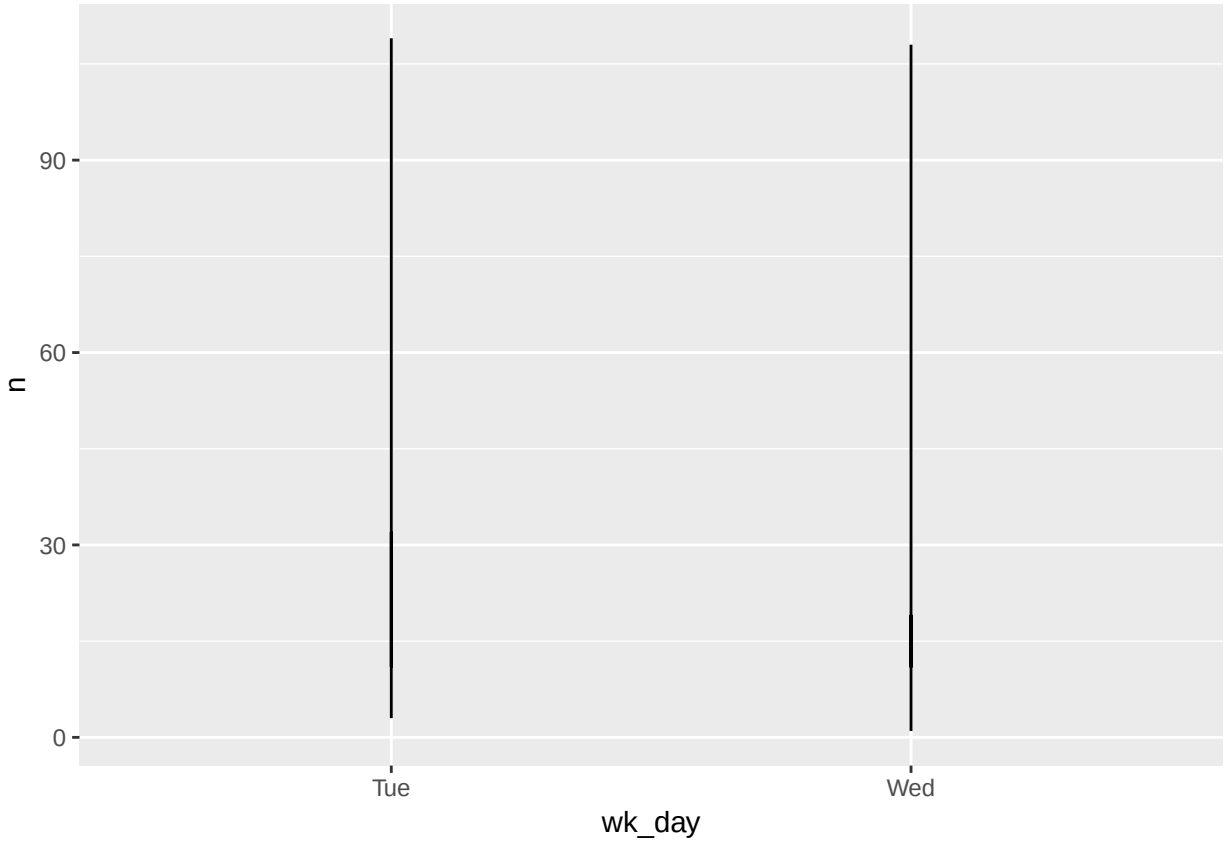
Let's calculate number of violation per weekday. For better understanding we will label the weekdays. First we need to group the data and count records within each group:

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label = TRUE)) %>%
  group_by(wk_day, violation) %>%
  tally
```

Timelapse data can be visualized as a line plot (with – you guessed it – `geom_line()`) mapping the days to the x axis and counts to the y axis. So we pipe the output from above into ggplot like this:

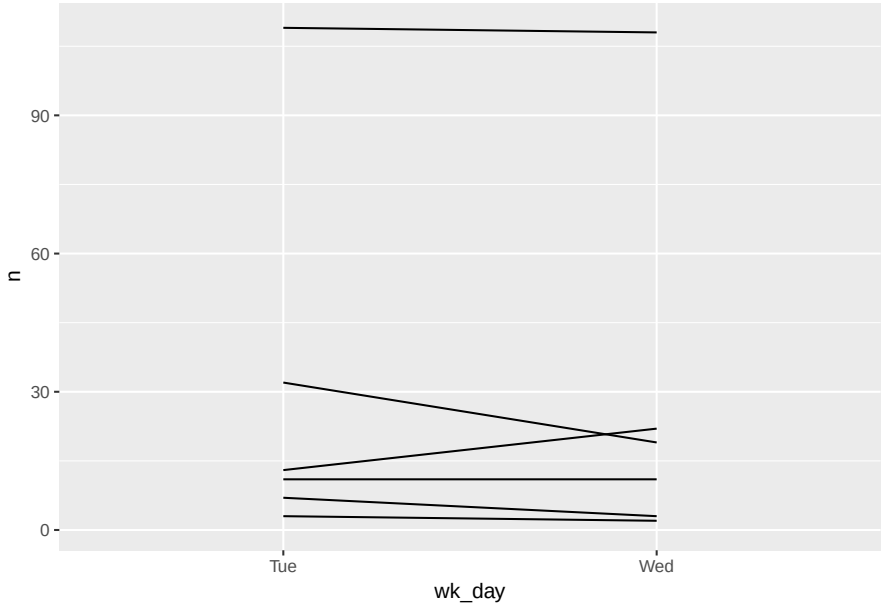
```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label = TRUE)) %>%
  group_by(wk_day, violation) %>%
```

```
  tally %>%
  ggplot(aes(x = wk_day, y = n)) +
     geom_line()
```



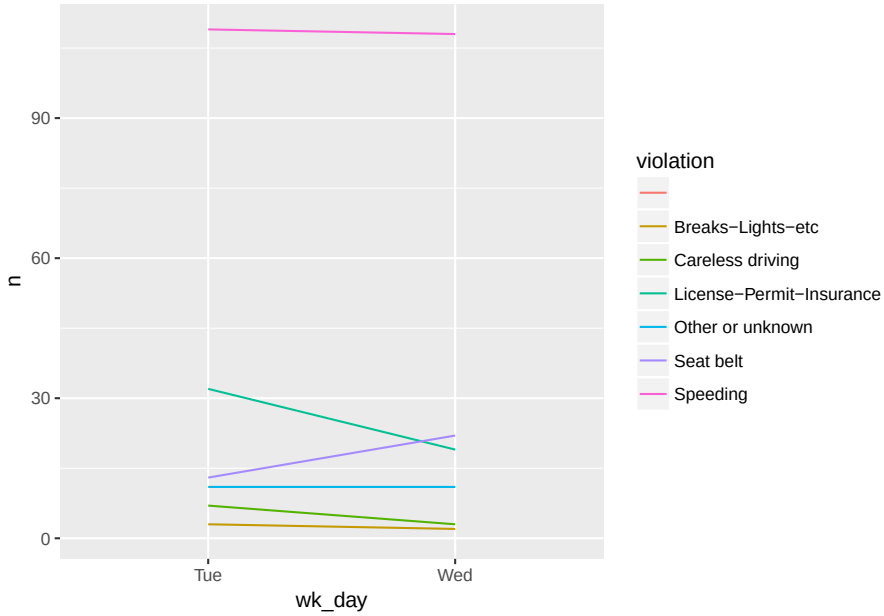
Unfortunately, this does not work because we plotted data for all the violations together. So what ggplot displays is the range of all values for each year in a vertial line. We need to tell ggplot to draw a line for each violation by modifying the aesthetic function to include `group = violation`:

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label = TRUE)) %>%
  group_by(wk_day, violation) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, group = violation)) +
     geom_line()
```



We will be able to distinguish violations in the plot if we add colors. (Colors groups automatically if the variable is numeric).

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label = TRUE)) %>%
  group_by(wk_day, violation) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, group = violation, color = violation)) +
     geom_line()
```

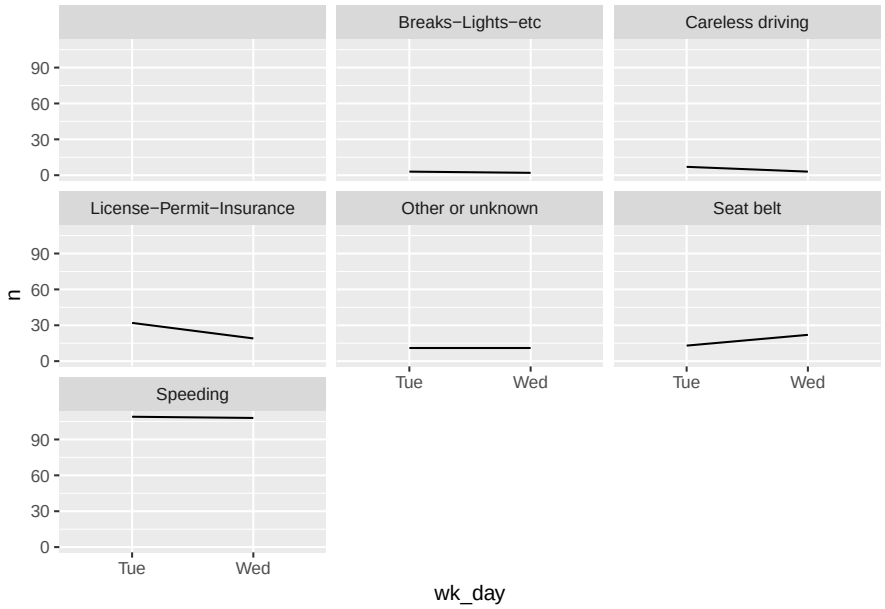


1.6 Faceting

ggplot has a special technique called *faceting* that allows to split one plot into multiple plots based on a factor included in the dataset. We will use it to make a time series plot for each violation:

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label = TRUE)) %>%
  group_by(wk_day, violation) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, group = violation)) +
     geom_line() +
     facet_wrap(~ violation)
```

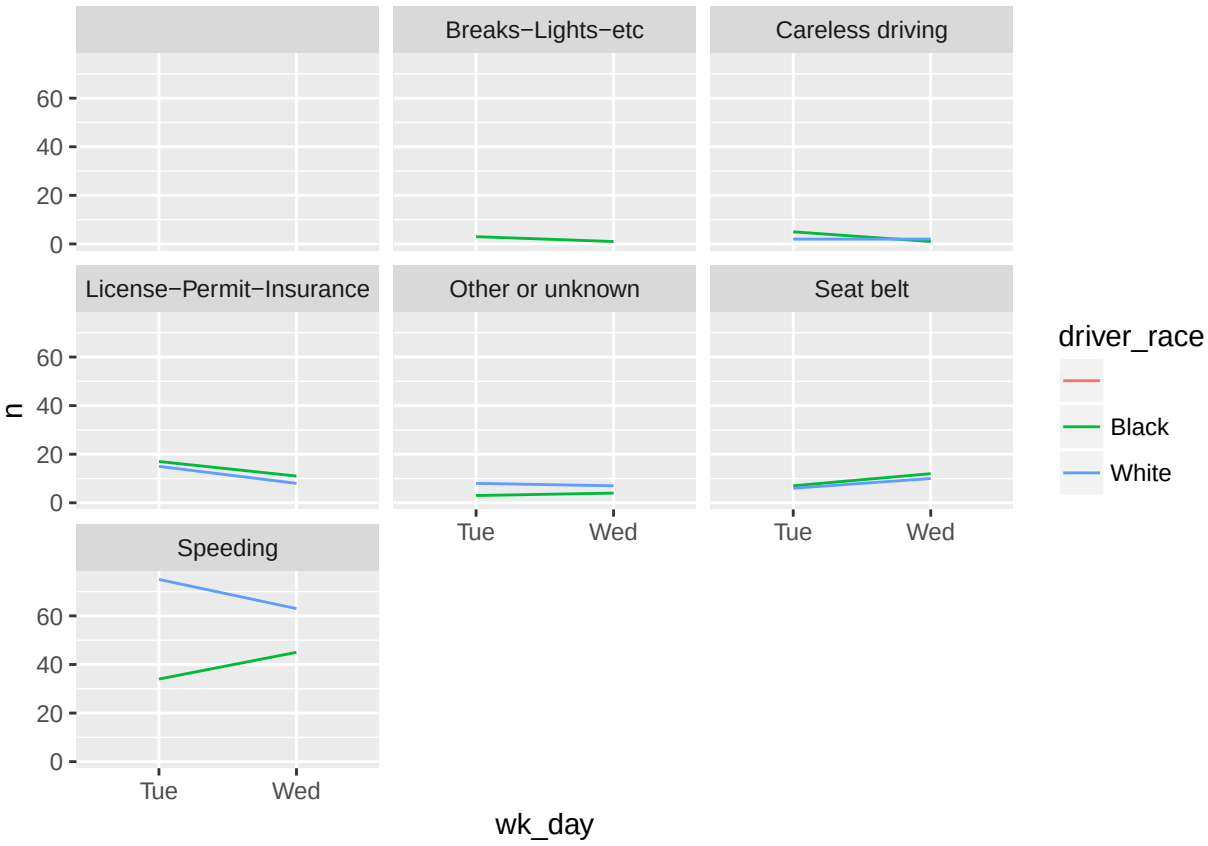
```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



Now we would like to split the line in each plot by the race of the driver. To do that we need to make counts in the data frame grouped by `day`, `violation`, and `driver_race`. We then make the faceted plot by splitting further by race using `color` and `group` (within a single plot):

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label=TRUE)) %>%
  group_by(wk_day, violation, driver_race) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, color = driver_race, group = driver_race)) +
  geom_line() +
  facet_wrap(~ violation)
```

```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



Note that there is an alternative, the `facet_grid` geometry, which allows you to explicitly specify how you want your plots to be arranged via formula notation (`rows ~ columns`; a `.` can be used as a placeholder that indicates only one row or column).

Challenge

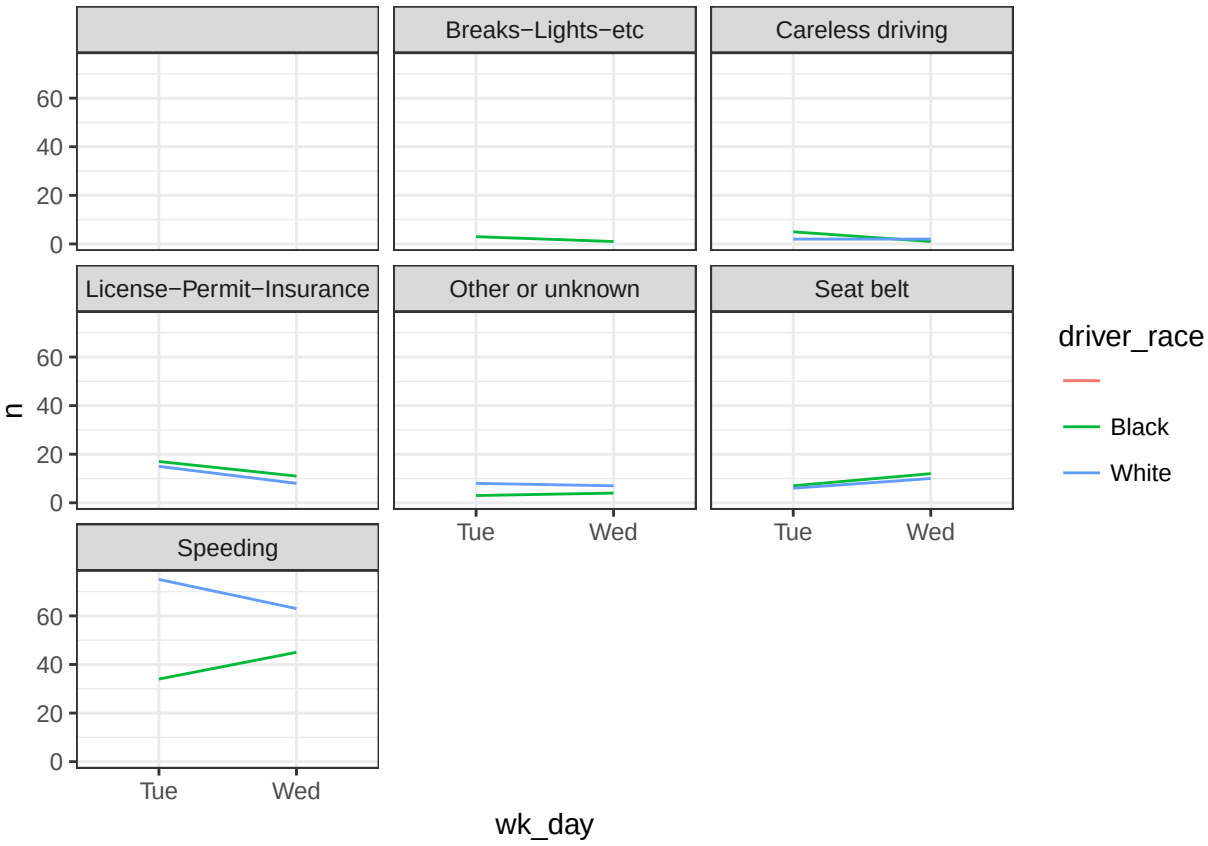
Use what you just learned to create a plot that depicts how the average age of each driver for the two recorded ethnicities changes through the week. Hint: make sure you remove the records with driver_age under 16. How would you go about visualizing both lines and points on the plot? How would you split your plot into one per each violation type?

1.7 ggplot2 themes

ggplot2 comes with several other themes which can be useful to quickly change the look of your visualization, for example `theme_bw()` changes the plot background to white:

```
trafficstops %>%
  mutate(wk_day = wday(stop_date, label=TRUE, abbr=TRUE)) %>%
  group_by(wk_day, violation, driver_race) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, color = driver_race, group = driver_race)) +
  geom_line() +
  facet_wrap(~ violation) +
  theme_bw()
```

```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



The complete list of themes is available at http://docs.ggplot2.org/current/ggtheme.html. `theme_minimal()` and `theme_light()` are popular, and `theme_void()` can be useful as a starting point to create a new hand-crafted theme.

The ggthemes package provides a wide variety of options (including an Excel 2003 theme). The **ggplot2** extensions website provides a list of packages that extend the capabilities of **ggplot2**, including additional themes.

1.8 Customization

There are endless possibilities to customize your plot, particularly when you are ready for publication or presentation. Let's look into just a few examples. Before we do that we will assign our plot above to a variable.

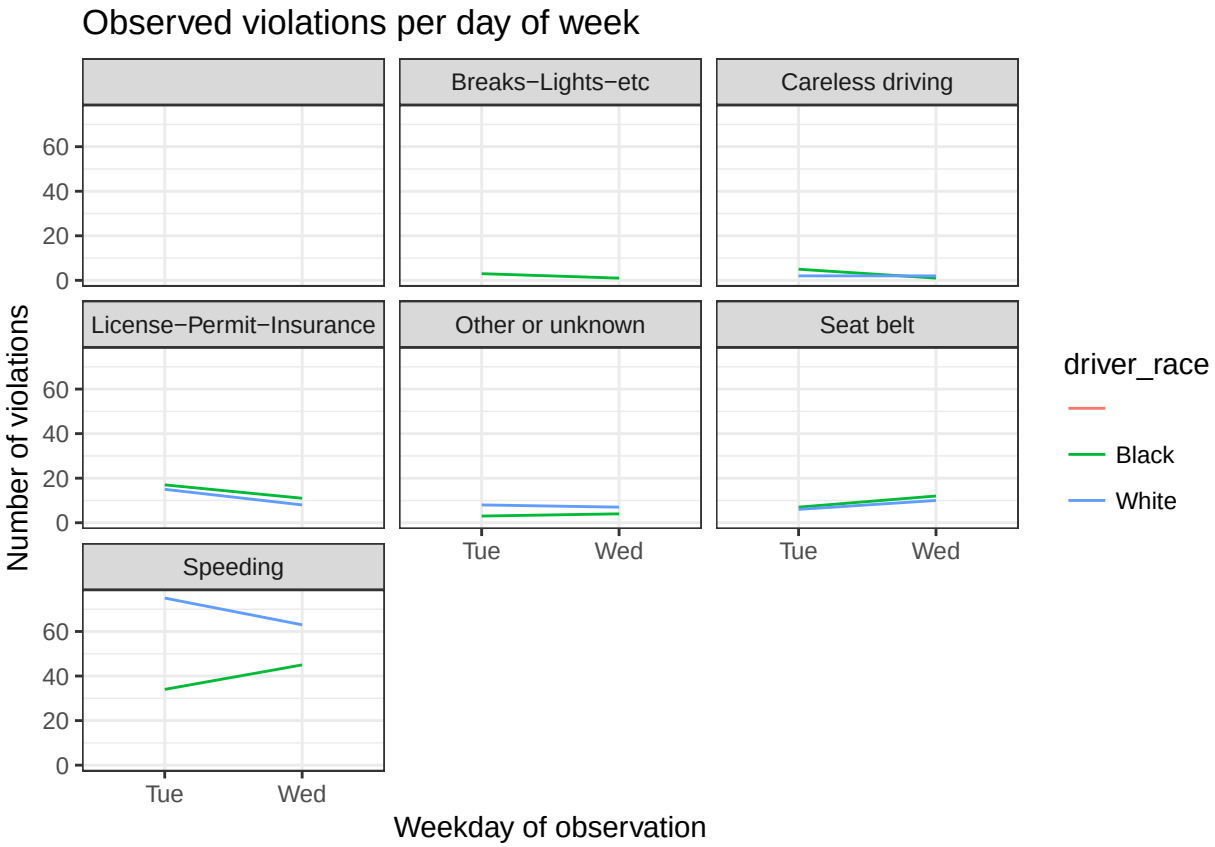
```
stops_facet_plot <- trafficstops %>%
  mutate(wk_day = wday(stop_date, label=TRUE, abbr=TRUE)) %>%
  group_by(wk_day, violation, driver_race) %>%
  tally %>%
  ggplot(aes(x = wk_day, y = n, color = driver_race, group = driver_race)) +
  geom_line() +
  facet_wrap(~ violation)
```

Now, let's change names of axes to something more informative than 'wk_day' and 'n' and add a title to the figure:

```
stops_facet_plot +
  labs(title = 'Observed violations per day of week',
       x = 'Weekday of observation',
```

```
        y = 'Number of violations') +
  theme_bw()
```

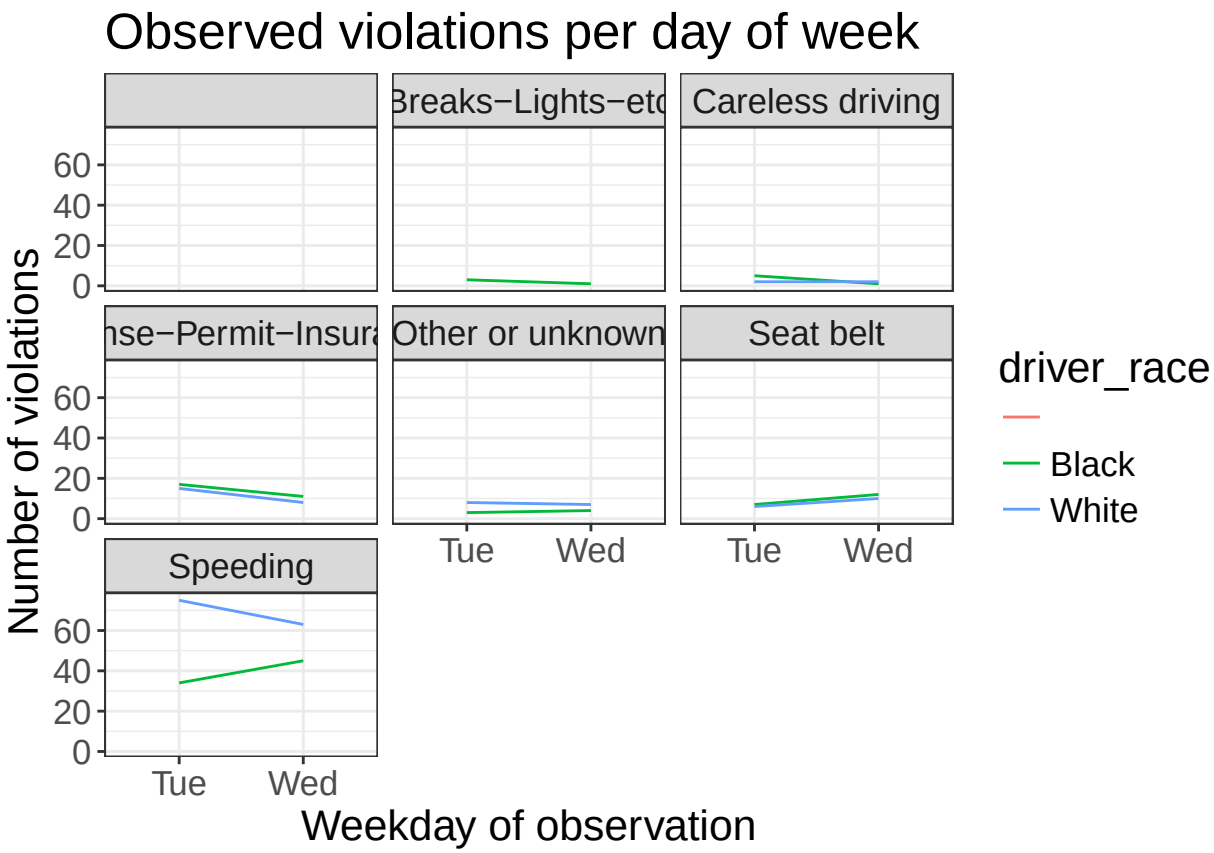
```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



The axes have more informative names, but their readability can be improved by increasing the font size:

```
stops_facet_plot +
  labs(title = 'Observed violations per day of week',
       x = 'Weekday of observation',
       y = 'Number of violations') +
  theme_bw() +
  theme(text = element_text(size=16))
```

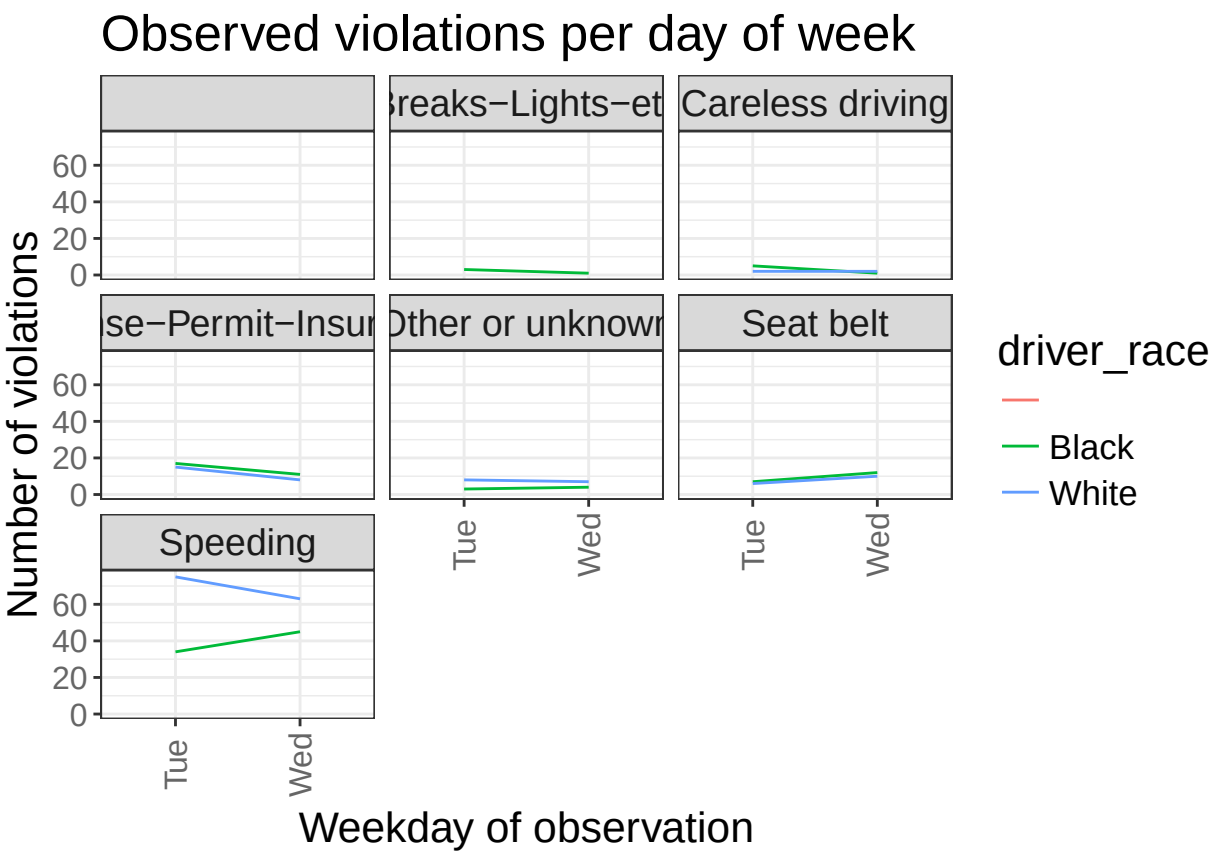
```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



After our manipulations, you may notice that the values on the x-axis are still not properly readable. Let's change the orientation of the labels and adjust them vertically and horizontally so they don't overlap. You can use a 90 degree angle, or experiment to find the appropriate angle for diagonally oriented labels:

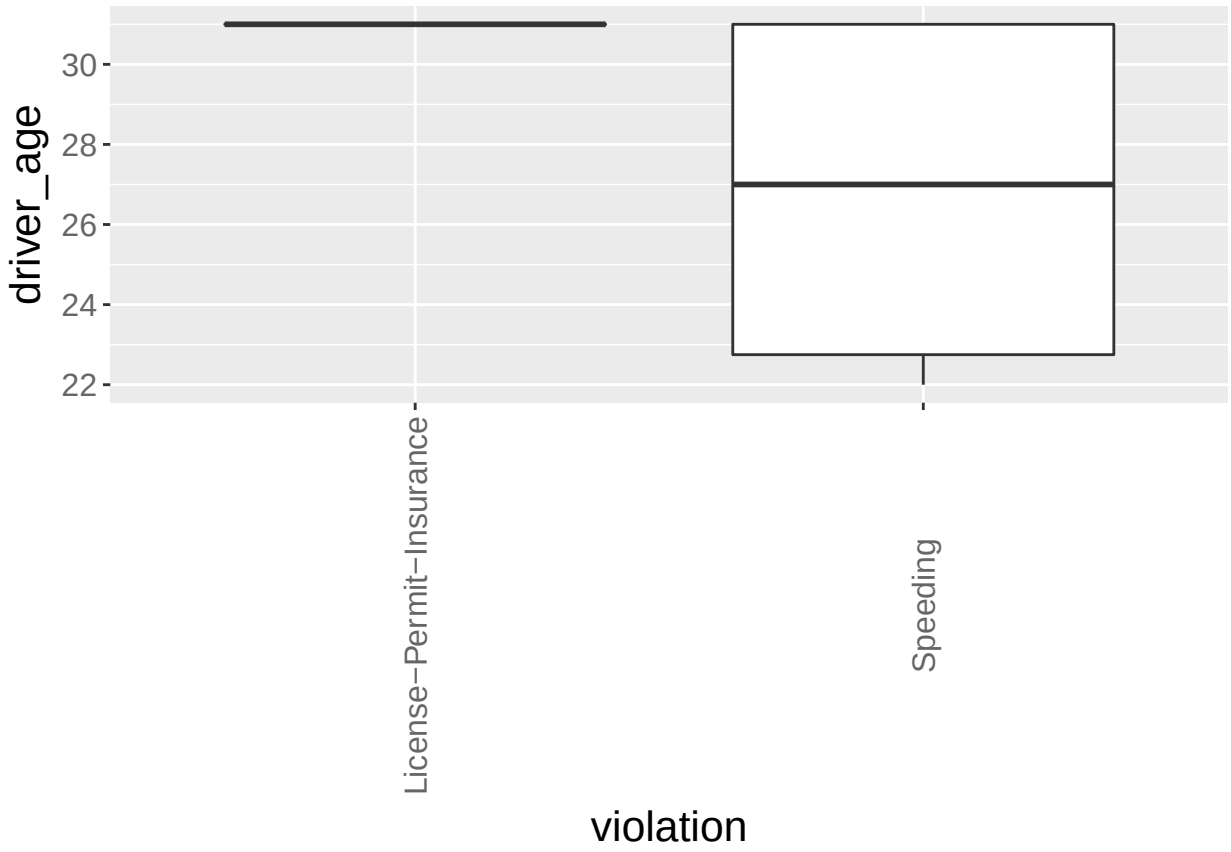
```
stops_facet_plot +
  labs(title = 'Observed violations per day of week',
         x = 'Weekday of observation',
         y = 'Number of violations') +
  theme_bw() +
  theme(axis.text.x = element_text(colour="grey40", size=12, angle=90, hjust=.5, vjust=.5),
        axis.text.y = element_text(colour="grey40", size=12),
        strip.text = element_text(size=14),
        text = element_text(size=16))
```

```
#> geom_path: Each group consists of only one observation. Do you need to
#> adjust the group aesthetic?
```



If you like the changes you created better than the default theme, you can save them as an object to be able to easily apply them to other plots you may create:

```
grey_theme <- theme(axis.text.x = element_text(colour="grey40", size=12, angle=90, hjust=.5, vjust=.5),
                    axis.text.y = element_text(colour="grey40", size=12), text=element_text(size=16))

ggplot(data = Chickasaw_stops, aes(x = violation, y = driver_age)) +
  geom_boxplot() +
  grey_theme
```



Note that it is also possible to change the fonts of your plots. If you are on Windows, you may have to install the **extrafont** package, and follow the instructions included in the README for this package.

> Challenge
>
> With all of this information in hand, please take another five minutes to either improve one of the plots generated in this exercise or create a beautiful graph of your own. Use the RStudio `ggplot2` cheat sheet for inspiration.
>
> Here are some ideas:
>
> - See if you can change the thickness of the lines.
> - Can you find a way to change the name of the legend? What about its labels?
> - Try using a different color palette (see http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/).

After creating your plot, you can save it out to a file in your prefered format. You can change the dimension (and resolution) of your plot by adjusting the appropriate arguments (`width`, `height` and `dpi`):

```
my_plot <- stops_facet_plot +
  labs(title = 'Observed violations per day of week',
         x = 'Weekday of observation',
         y = 'Number of violations') +
  theme_bw() +
  theme(axis.text.x = element_text(colour="grey40", size=12, angle=90, hjust=.5, vjust=.5),
        axis.text.y = element_text(colour="grey40", size=12),
        strip.text = element_text(size=14),
        text = element_text(size=16))

ggsave("name_of_file.png", my_plot, width=15, height=10)
```

Note: The parameters `width` and `height` also determine the font size in the saved plot.

Chapter 2

Alternatives: R base graphis and `lattice` graphs

Learning Objectives

- Make a plot with base plot package
- Make a plot with other base plot commands
- Make a simple plot with R lattice package
- Explain the difference to the ggplot approach and evaluate pros and cons

---

2.1 R base graphics

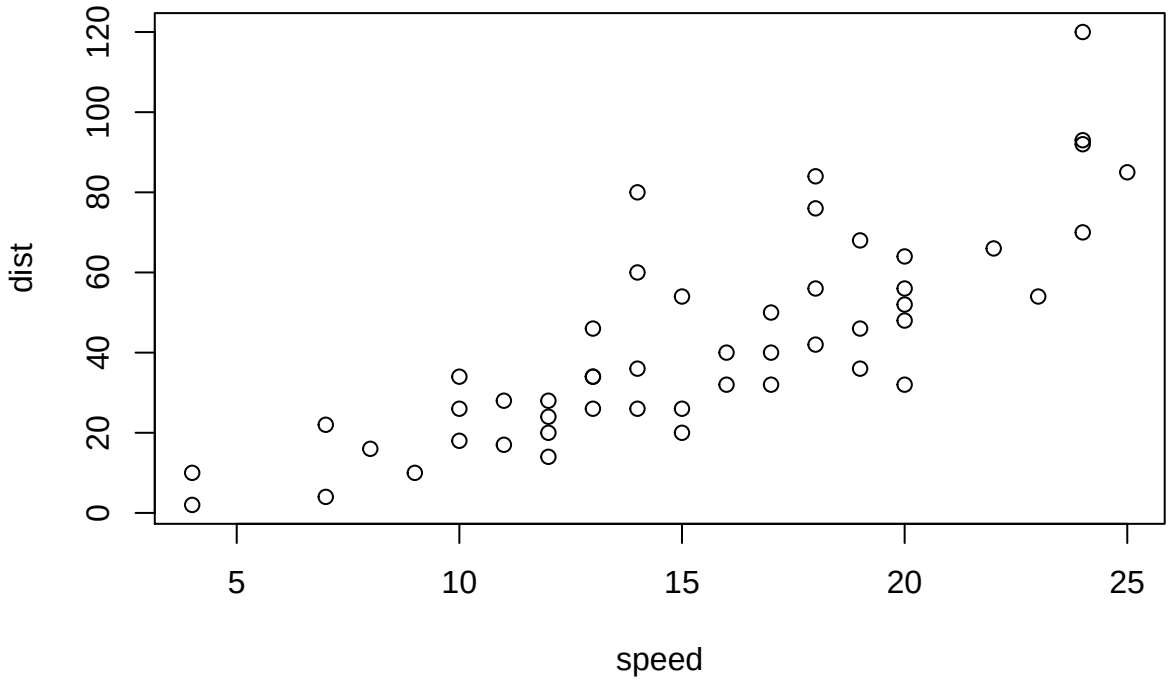
We will use a simple dataset that comes with the base R install.

```
head(cars)
```

```
#>   speed dist
#> 1     4    2
#> 2     4   10
#> 3     7    4
#> 4     7   22
#> 5     8   16
#> 6     9   10
```

R base comes with a simple plot command that can be applied to the data like that:

```
plot(cars)
```



Other things one can do with plot.

Walk through `plot` arguments and options.

How to do multiple plots.

https://www.stat.auckland.ac.nz/~paul/RGraphics/chapter1.pdf and code: https://www.stat.auckland.ac.nz/~paul/RGraphics/chapter1.html

Challenge: make a plot

Challenge: how would you do this with ggplot?

Other base package plot commands: `histogram`, `boxplot`, `stripchart`, `barplot`, `mosaicplot`, `dotchart` and more.

http://www.cyclismo.org/tutorial/R/plotting.html and http://www.cyclismo.org/tutorial/R/intermediatePlotting.html

https://www.harding.edu/fmccown/r/

http://courses.atlas.illinois.edu/spring2017/STAT/STAT200/RProgramming/Plotting.html

Challenge: use some of these commands

2.2 `lattice` package

Lattice is another major graphic package in R. [… more about it here…] It works like this:

```
xyplot(dist ~ speed, cars)
```

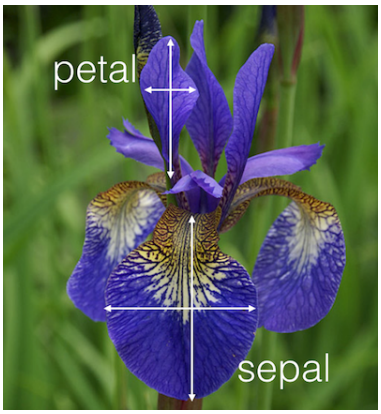
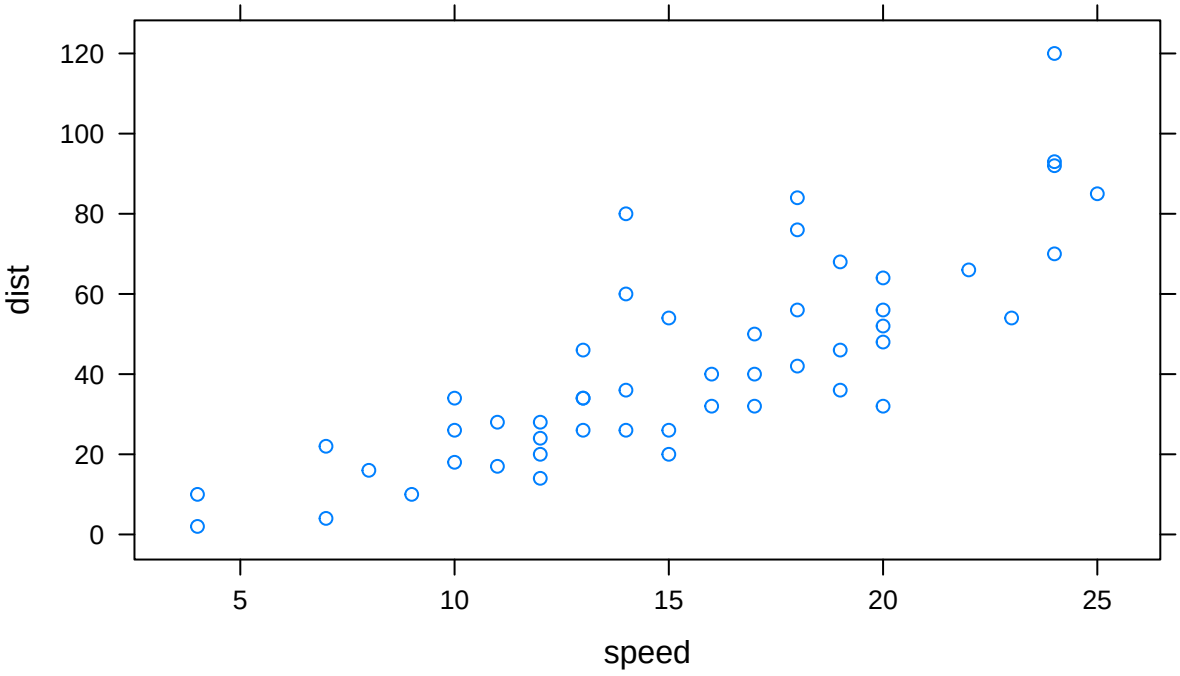


Figure 2.1: Iris Petal and Sepal (Source: kaggle.com)



Iris data example (again, other data??) :

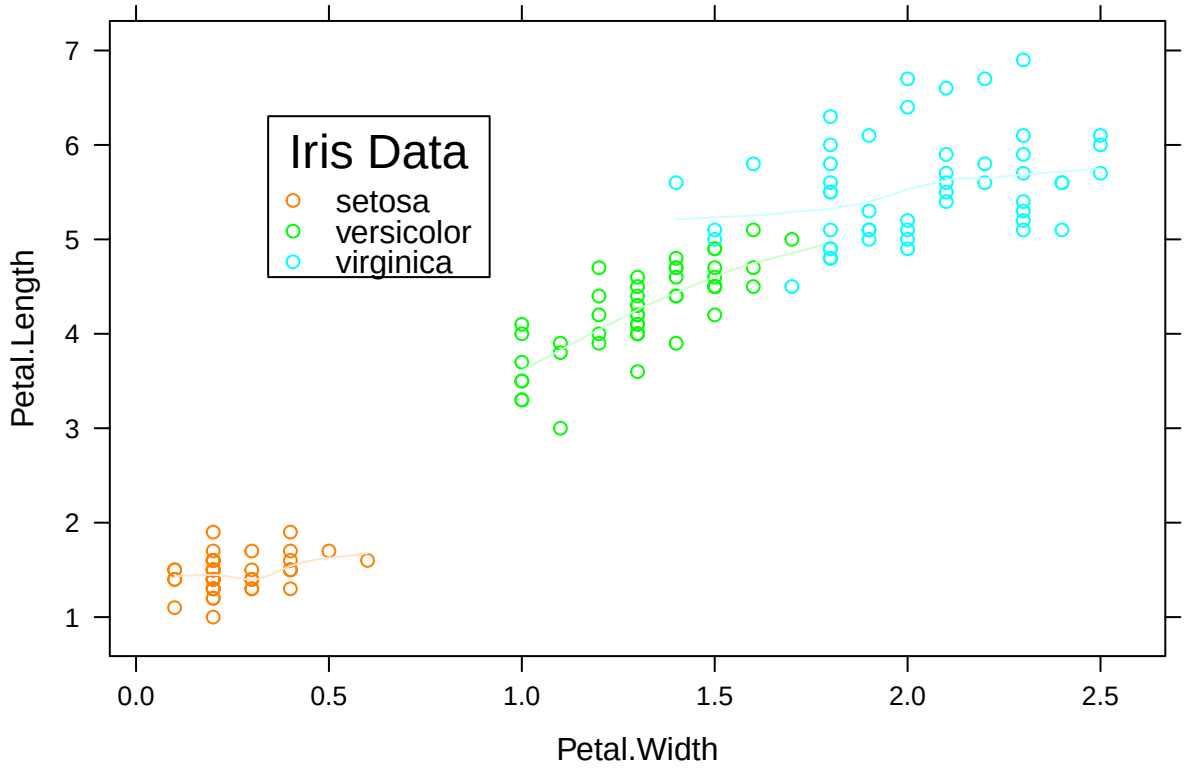
```
head(iris)
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
#> 1          5.1         3.5          1.4         0.2  setosa
#> 2          4.9         3.0          1.4         0.2  setosa
#> 3          4.7         3.2          1.3         0.2  setosa
#> 4          4.6         3.1          1.5         0.2  setosa
#> 5          5.0         3.6          1.4         0.2  setosa
#> 6          5.4         3.9          1.7         0.4  setosa
```

Contains 50 samples from 3 species with 4 measurements: length and width of petals and length and width of sepals.

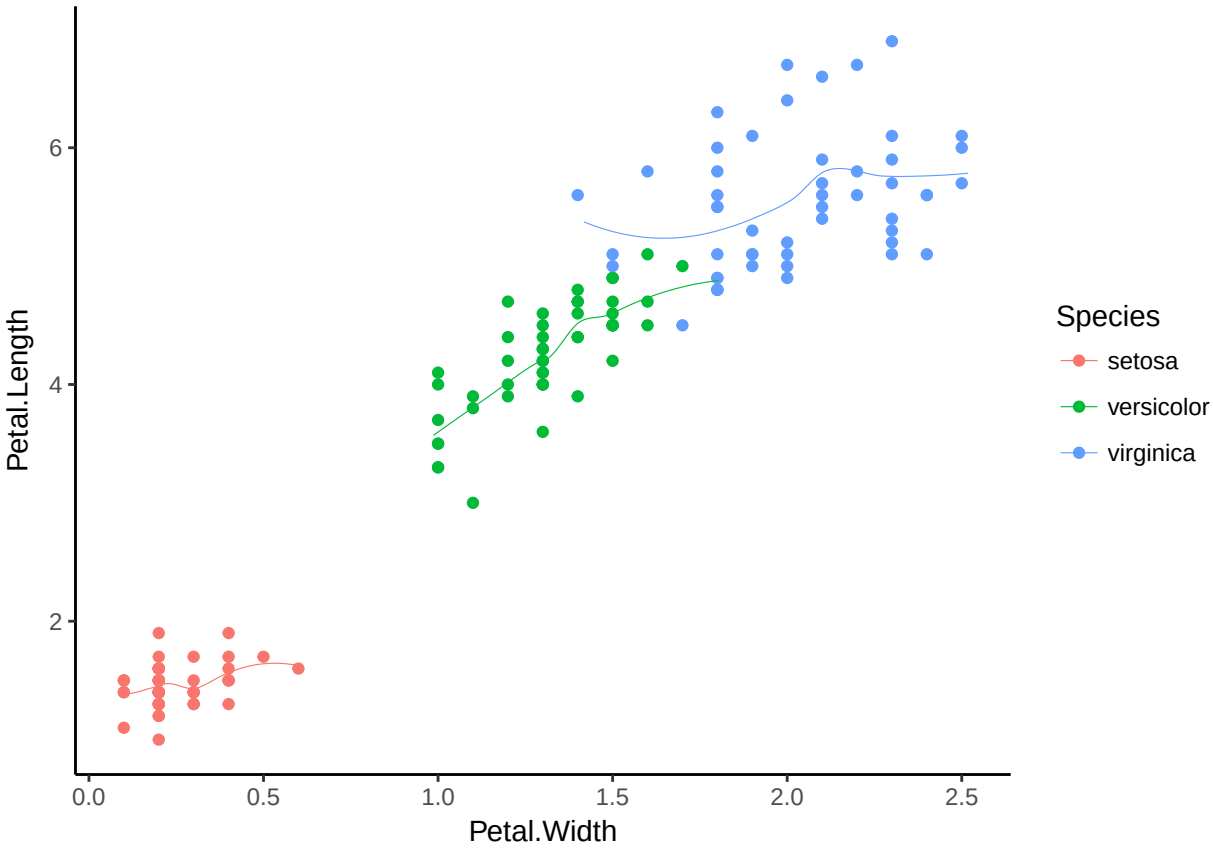
With `lattice`

```
xyplot(Petal.Length~Petal.Width, data = iris, groups=Species,
  panel = panel.superpose, type = c("p", "smooth"), span=.75,
  col.line = trellis.par.get("strip.background")$col,
  col.symbol = trellis.par.get("strip.shingle")$col,
  key = list(title="Iris Data", x=.15, y=.85, corner=c(0,1), border=TRUE,
        points = list(col=trellis.par.get("strip.shingle")$col[1:3],
                  pch = trellis.par.get("superpose.symbol")$pch[1:3],
                  cex = trellis.par.get("superpose.symbol")$cex[1:3]),
                  text = list(levels(iris$Species))))
```



The same with ggplot:

```
ggplot(data=iris, aes(x=Petal.Width, y=Petal.Length, color=Species)) +
  geom_point() +
  stat_smooth(aes(jitter(Petal.Width), jitter(Petal.Length)), size=.2, se=F) +
  theme_classic()
```



pros - cons?

More about `lattice`:

https://www.stat.auckland.ac.nz/~paul/RGraphics/chapter4.pdf R code: https://www.stat.auckland.ac.nz/~paul/RGraphics/chapter4.html

Chapter 3

Domain specific graphs

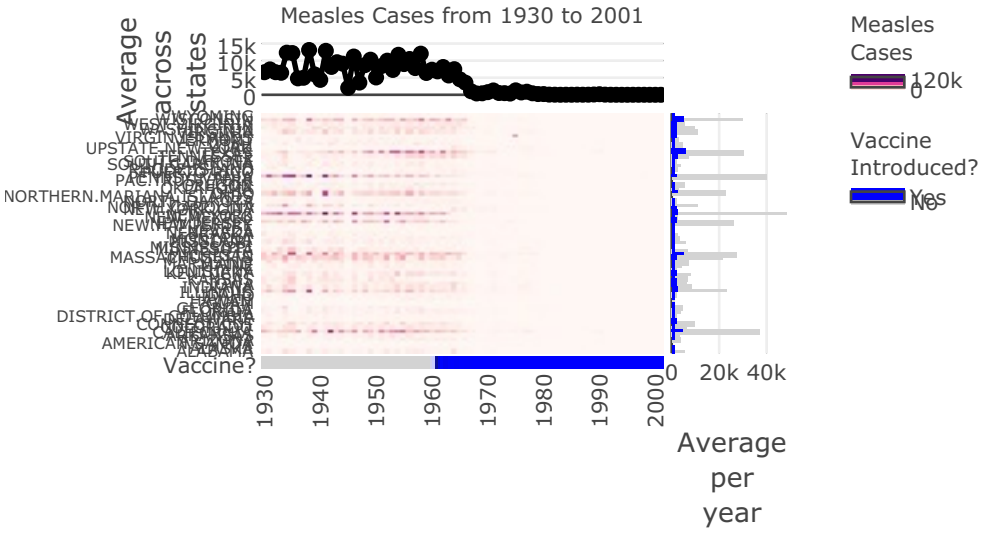
Learning Objectives

- Be aware of specialized graph packages and know where to look for them
- Understand the basic structure of iheatmapr, tmap, and visNetwork examples
- Modify parameters of provided graph examples

`ggplot` can get you a long way, but if you need to do a particular, more complex graph it is worth checking if there might be an R package for that. Typically it would do one type of visualization and do that really well. Below are a few examples.

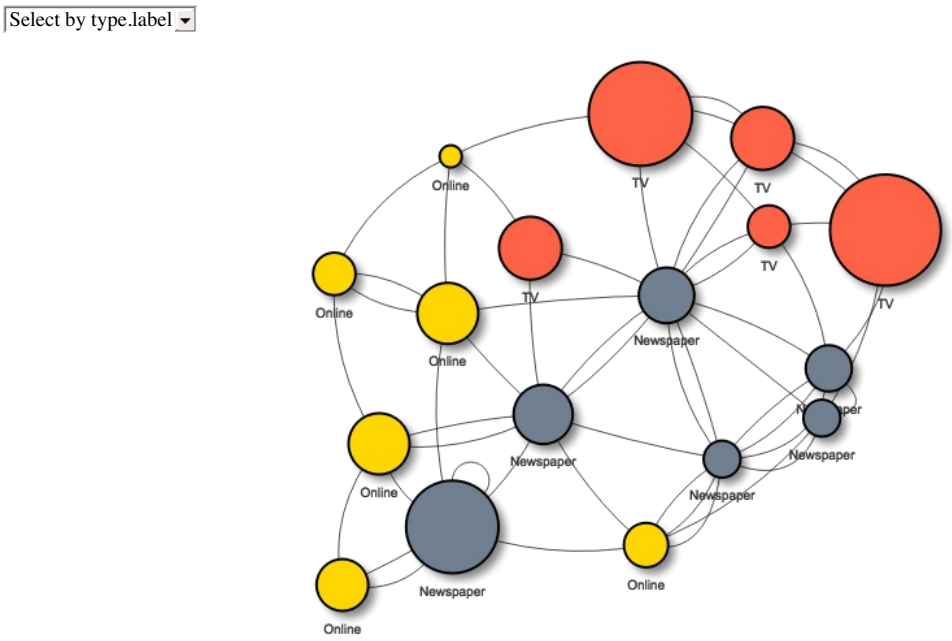
3.1 Heatmaps (e.g. `iheatmapr`)

```
library(iheatmapr)
data(measles, package = "iheatmapr")

main_heatmap(measles, name = "Measles<br>Cases", x_categorical = FALSE,
             layout = list(font = list(size = 8))) %>%
  add_col_groups(ifelse(1930:2001 < 1961,"No","Yes"),
                 side = "bottom", name = "Vaccine<br>Introduced?",
                 title = "Vaccine?",
                 colors = c("lightgray","blue")) %>%
  add_col_labels(ticktext = seq(1930,2000,10),font = list(size = 8)) %>%
  add_row_labels(size = 0.3,font = list(size = 6)) %>%
  add_col_summary(layout = list(title = "Average<br>across<br>states"),
                  yname = "summary")  %>%
  add_col_title("Measles Cases from 1930 to 2001", side= "top") %>%
  add_row_summary(groups = TRUE,
                  type = "bar",
                  layout = list(title = "Average<br>per<br>year",
                                font = list(size = 8)))
```



3.2 Networks (e.g. `visNetwork`)

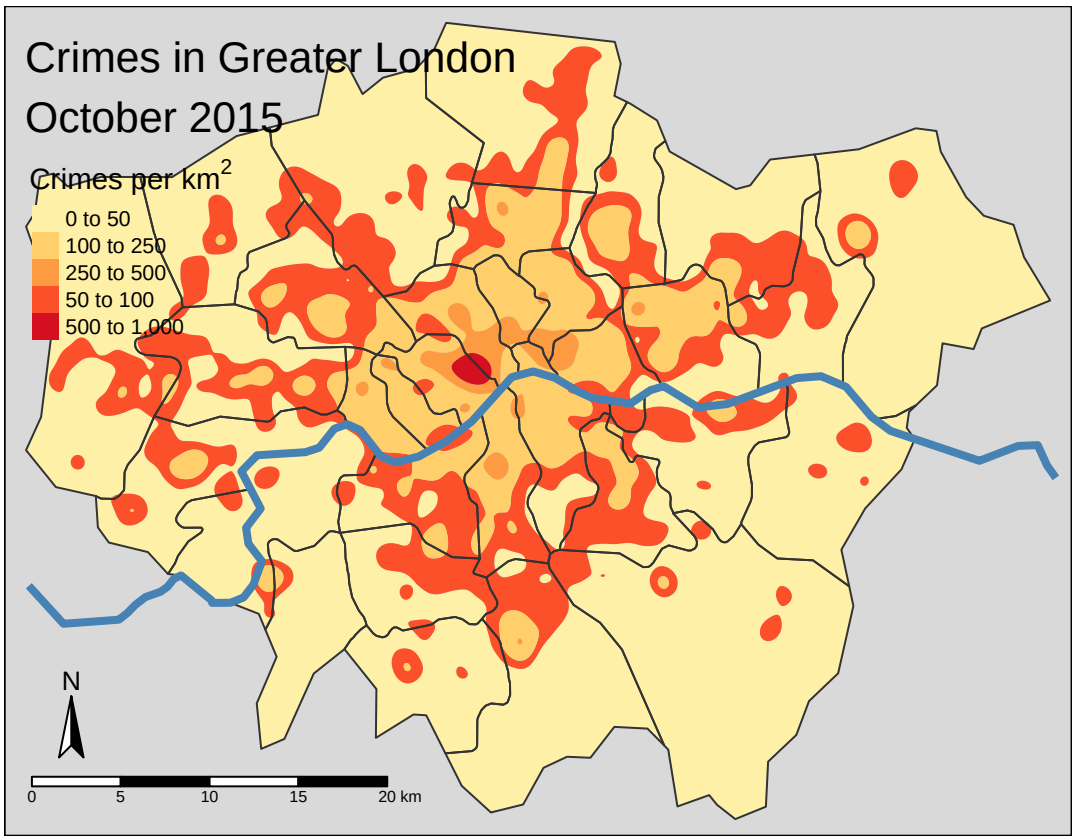
```
library('visNetwork')
nodes <- read.csv("https://github.com/cengel/R-data-viz/raw/master/demo-data/network/Dataset1-Media-Exa
links <- read.csv("https://github.com/cengel/R-data-viz/raw/master/demo-data/network/Dataset1-Media-Exa
nodes$shape <- "dot"
nodes$shadow <- TRUE # Nodes will drop shadow
nodes$title <- nodes$media # Text on click
nodes$label <- nodes$type.label # Node label
nodes$size <- nodes$audience.size # Node size
nodes$borderWidth <- 2 # Node border width

nodes$color.background <- c("slategrey", "tomato", "gold")[nodes$media.type]
nodes$color.border <- "black"
nodes$color.highlight.background <- "orange"
nodes$color.highlight.border <- "darkred"

visNetwork(nodes, links) %>%
  visOptions(highlightNearest = TRUE,
             selectedBy = "type.label")
```



3.3 Maps (e.g. `tmap`)

For this demo we first need to download the data. If you have your working directory set to `R-data-viz` and it contains a folder called `data`, this will download and extract the map data into a subfolder of your `data` folder, called `map`:

```
download.file("https://github.com/cengel/R-data-viz/raw/master/demo-data/map.zip",
              "data/map.zip")
unzip("data/map.zip", exdir="data")
```



Now, here is the map.

```
library(tmap)
library(tmaptools)
suppressPackageStartupMessages(library(sf))

london <- st_read("data/map", "London", quiet = TRUE)
crime_densities <- st_read("data/map", "Crimes", quiet = TRUE)
thames <- st_read("data/map", "Thames", quiet = TRUE)

tm_shape(crime_densities) +
  tm_fill(col = "level", palette = "YlOrRd",
    title = expression("Crimes per " * km^2)) +
tm_shape(london) + tm_borders() +
tm_shape(thames) + tm_lines(col = "steelblue", lwd = 4) +
tm_compass(position = c("left", "bottom")) +
tm_scale_bar(position = c("left", "bottom")) +
tm_style_gray(title = "Crimes in Greater London\nOctober 2015")
```

Good starting places to look for additional examples are the R Graph Gallery: https://www.r-graph-gallery.com/all-graphs/ and the CRAN Task View: https://CRAN.R-project.org/view=Graphics.

Chapter 4

Interactive graphs

Learning Objectives

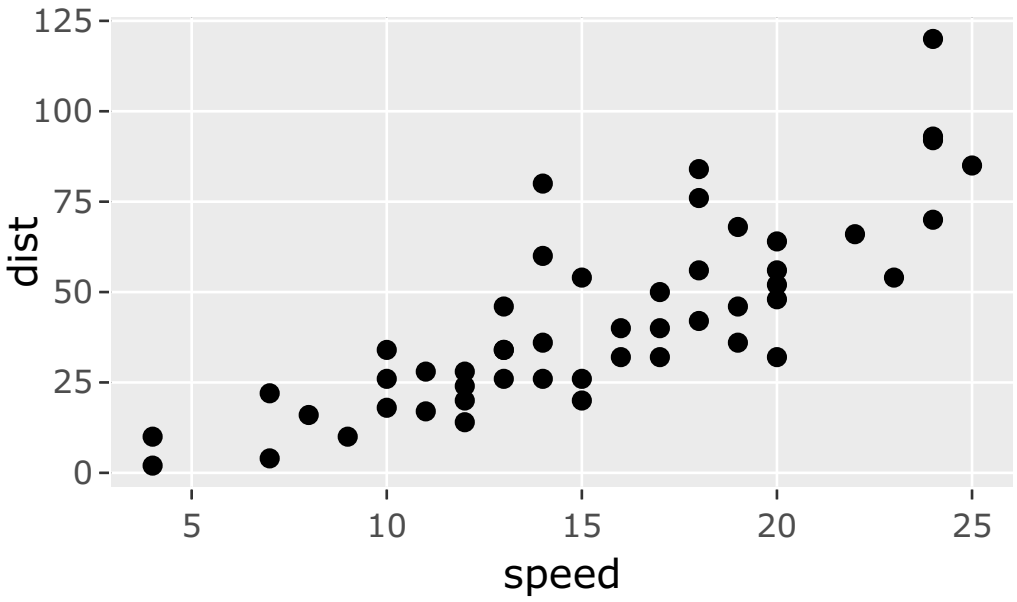
- Be aware of R interactive graphing capabilities
- Understand the basic structure of a shiny app

4.1 plotly

```
library(plotly)
```

Using the `ggplotly` function:

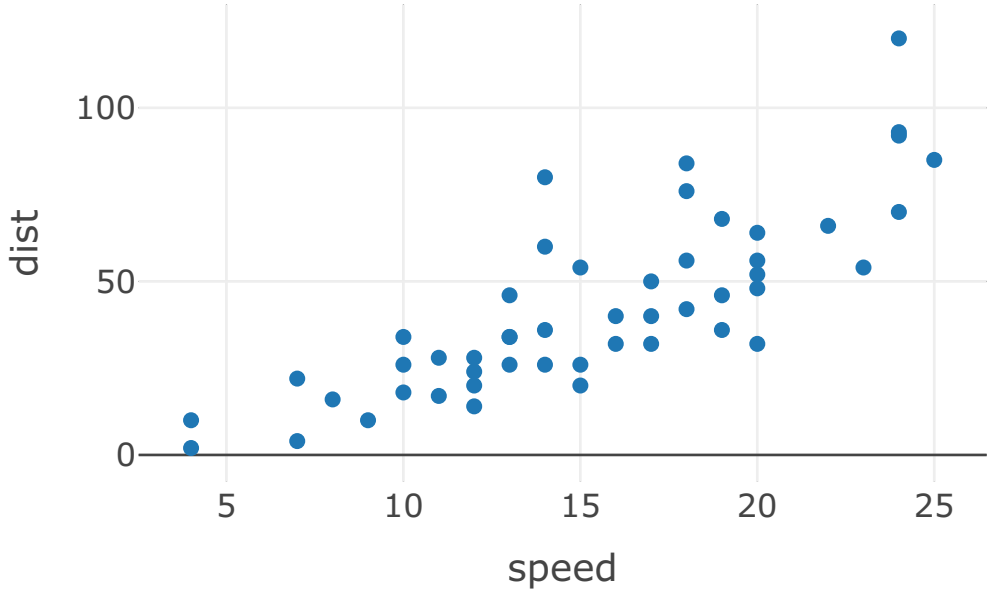
```
p <- ggplot(cars, aes(speed, dist)) + geom_point()
ggplotly(p)
```



(dev. version of ggplot2 now has a `ggplotly()` function)

Using the `plot_ly` function:

```
library(plotly)
plot_ly(cars, x = ~speed, y = ~dist, type = "scatter", mode = "markers")
```



4.2 shiny

https://cengel.shinyapps.io/shiny_demo/

```
library(shiny)
library(ggplot2)
inputPanel(
    sliderInput("speed_range",
        label = "Range of Speed:",
        min = min(cars$speed),
        max = max(cars$speed),
        value = range(cars$speed)))

renderPlot({
    cars_sel <- subset(cars, speed >= input$speed_range[1] &
                           speed < input$speed_range[2])
    ggplot(cars_sel, aes(speed, dist)) + geom_point()})
```

For a more comprehensive example see: https://cengel.shinyapps.io/RioSlaveMarket/

Bibliography