DATA SCIENCE AND MACHINE LEARNING

Common GGPLOT VISUALIZATIONS

Dimitris Fouskakis
Professor in Applied Statistics,
Department of Mathematics,
School of Applied Mathematical & Physical Sciences,
National Technical University of Athens
Email: fouskakis@math.ntua.gr

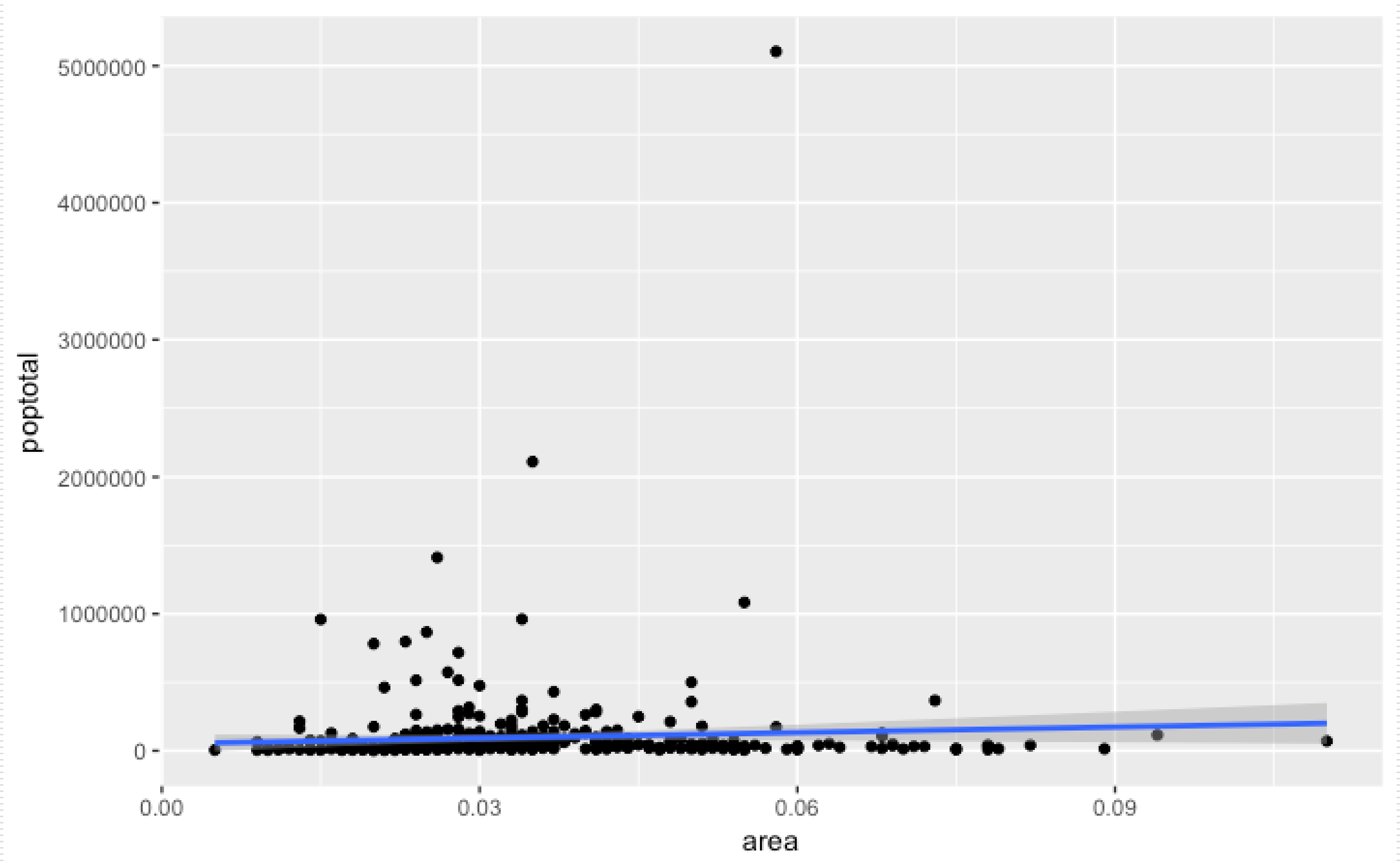
1. Correlation

- The most frequently used plot for data analysis is undoubtedly the **scatterplot**. Whenever you want to understand the nature of relationship between two continuous variables, invariably the first choice is the scatterplot.
- It can be drawn using geom_point(). Additionally, geom_smooth which draws a smoothing line (**based on loess - locally estimated scatterplot smoothing**) by default, can be tweaked to draw the line of best fit by setting method='lm'.

1. Correlation

```
library(ggplot2)
g <- ggplot(midwest, aes(x=area, y=poptotal)) +
geom_point() + geom_smooth(method="lm")
# set se=FALSE to turnoff confidence bands
plot(g)
```

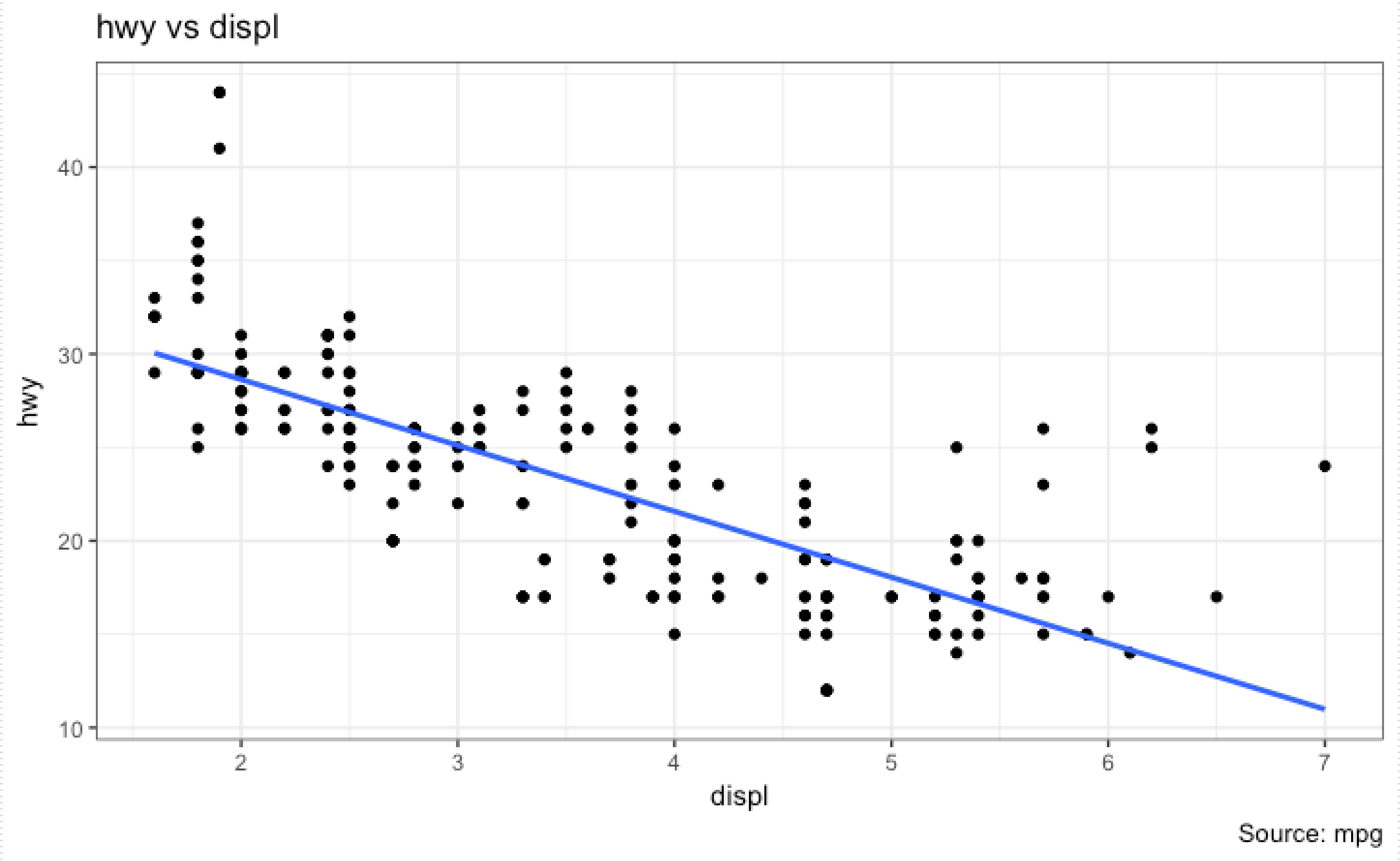
1. Correlation



1. Correlation

```
library(ggplot2)
data(mpg, package="ggplot2")  # load data
# mpg <- read.csv("http://goo.gl/uEeRGu")  # alt data
source

g <- ggplot(mpg, aes(x=displ, y=hwy)) +
      geom_point() +
      labs(title="hwy vs displ", caption = "Source: mpg") +
      geom_smooth(method="lm", se=FALSE) +
      theme_bw()  # apply bw theme
plot(g)
```

1. Correlation



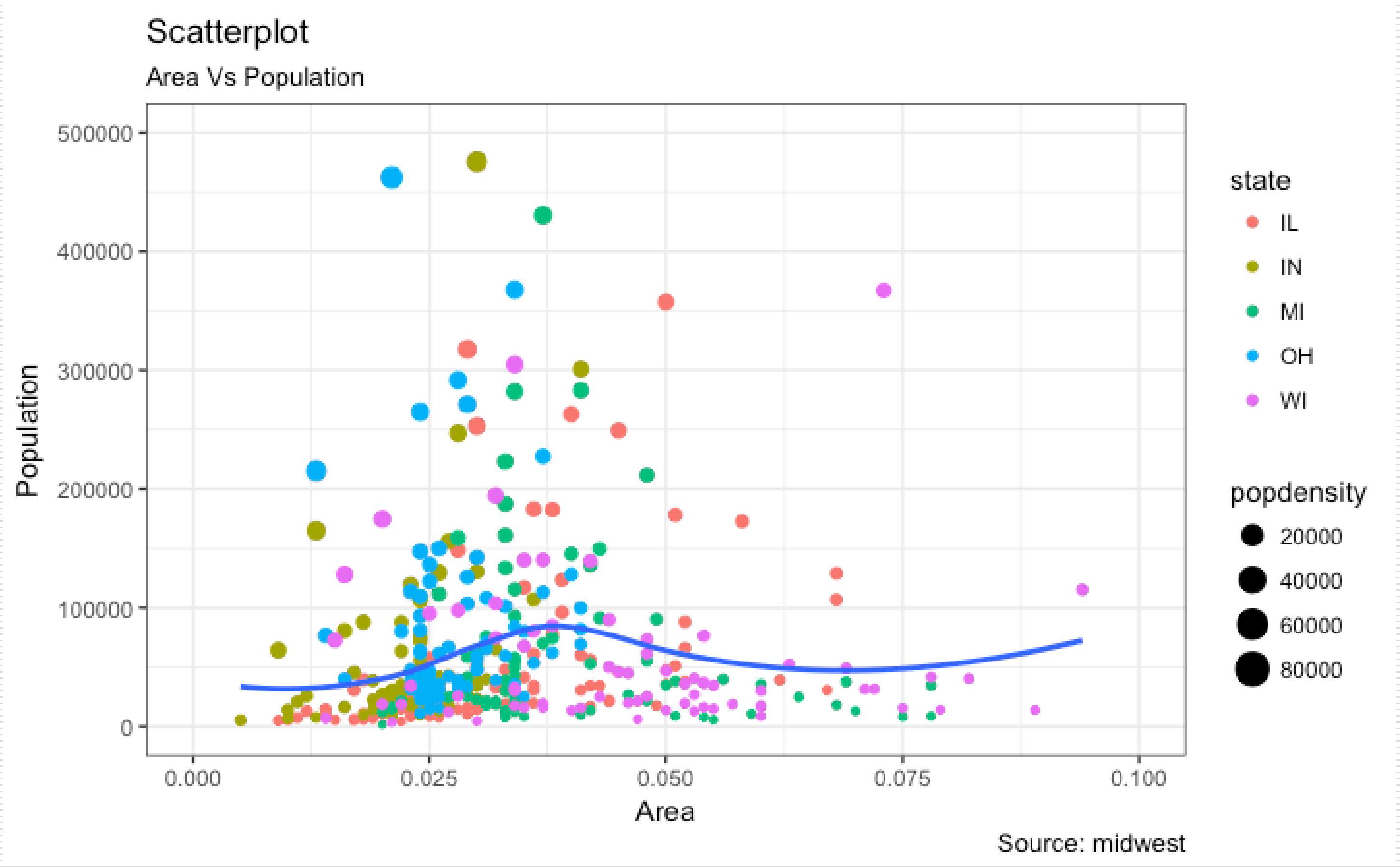
1. Correlation

```
# install.packages("ggplot2")
# load package and data
options(scipen=999)  # turn-off scientific notation like 1e+48
library(ggplot2)
theme_set(theme_bw())  # pre-set the bw theme.
data("midwest", package = "ggplot2")
# midwest <- read.csv("http://goo.gl/G1K41K")  # bkup data source
```

1. Correlation

```
# Scatterplot
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state, size=popdensity)) +
  geom_smooth(method="loess", se=F) +
  xlim(c(0, 0.1)) +
  ylim(c(0, 500000)) +
  labs(subtitle="Area Vs Population",
       y="Population",
       x="Area",
       title="Scatterplot",
       caption = "Source: midwest")
plot(gg)
```

1. Correlation



1. Correlation

- When presenting the results, sometimes we would encircle certain special group of points or region in the chart so as to draw the attention to those peculiar cases. This can be conveniently done using the geom_encircle() in ggalt package.
- Within geom_encircle(), set the data to a new dataframe that contains only the points (rows) or interest. Moreover, you can expand the curve so as to pass just outside the points. The color and size (thickness) of the curve can be modified as well.

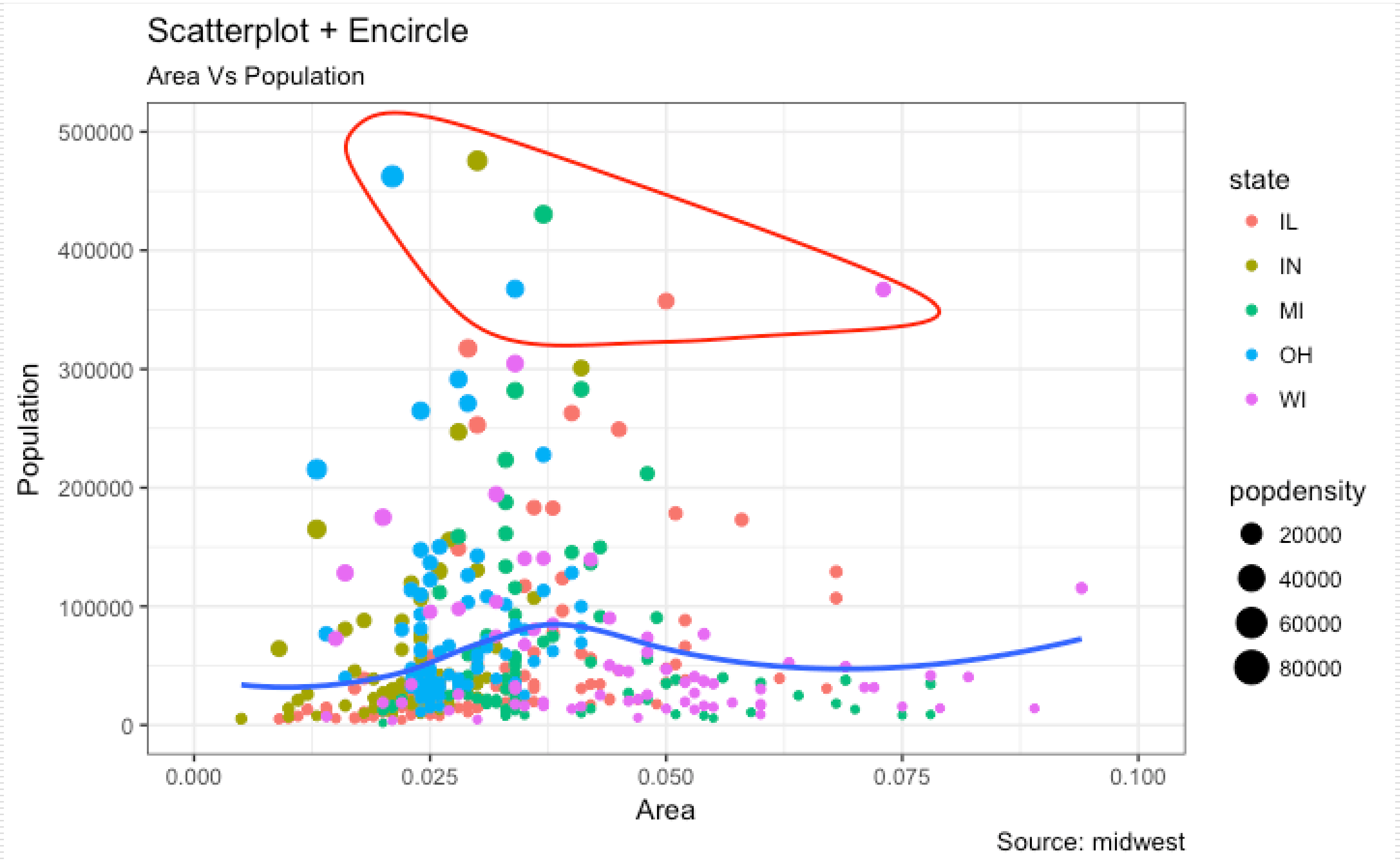
1. Correlation

```
# install devtools
install.packages("devtools")
# install 'ggalt' pkg
# devtools::install_github("hrbrmstr/ggalt")
options(scipen = 999)
library(ggplot2)
library(ggalt)
midwest_select <- midwest[midwest$poptotal > 350000 &
                          midwest$poptotal <= 500000 &
                          midwest$area > 0.01 &
                          midwest$area < 0.1, ]
```

1. Correlation

```
# Plot
ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state, size=popdensity)) +   # draw points
  geom_smooth(method="loess", se=F) +
  xlim(c(0, 0.1)) +
  ylim(c(0, 500000)) +   # draw smoothing line
  geom_encircle(aes(x=area, y=poptotal),
                data=midwest_select,
                color="red",
                size=2,
                expand=0.08) +   # encircle
  labs(subtitle="Area Vs Population",
       y="Population",
       x="Area",
       title="Scatterplot + Encircle",
       caption="Source: midwest")
```

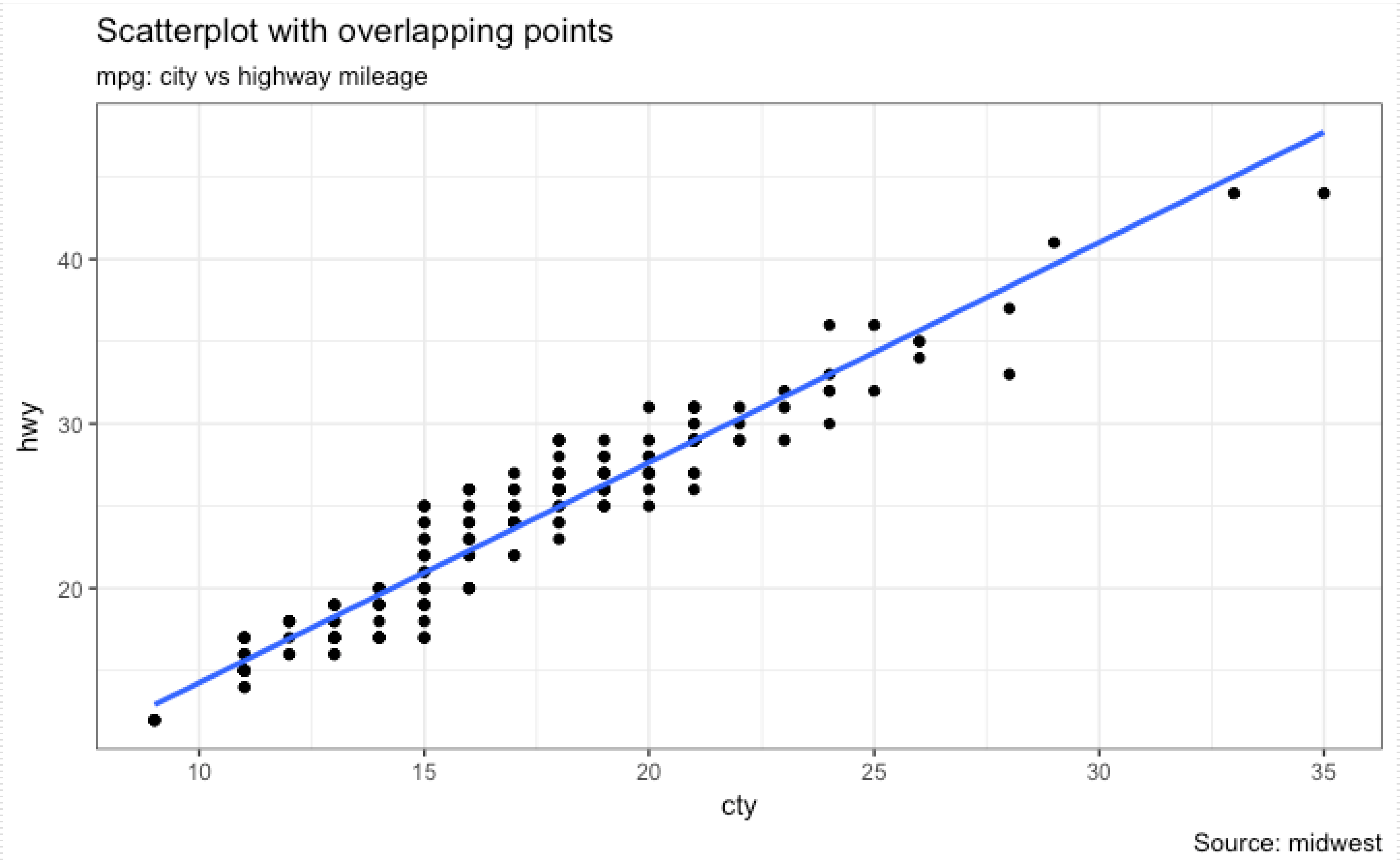
1. Correlation



1. Correlation

```
# load package and data
library(ggplot2)
data(mpg, package="ggplot2")
# alternate source: "http://goo.gl/uEeRGu")
theme_set(theme_bw())  # pre-set the bw theme.
g <- ggplot(mpg, aes(cty, hwy))
# Scatterplot
g + geom_point() +
  geom_smooth(method="lm", se=F) +
  labs(subtitle="mpg: city vs highway mileage",
       y="hwy",
       x="cty",
       title="Scatterplot with overlapping points",
       caption="Source: midwest")
```

1. Correlation



1. Correlation

- What we have here is a scatterplot of city and highway mileage in mpg dataset. We have seen a similar scatterplot and this looks neat and gives a clear idea of how the city mileage (cty) and highway mileage (hwy) are well correlated.
- But, this innocent looking plot is hiding something. Can you find out?

1. Correlation

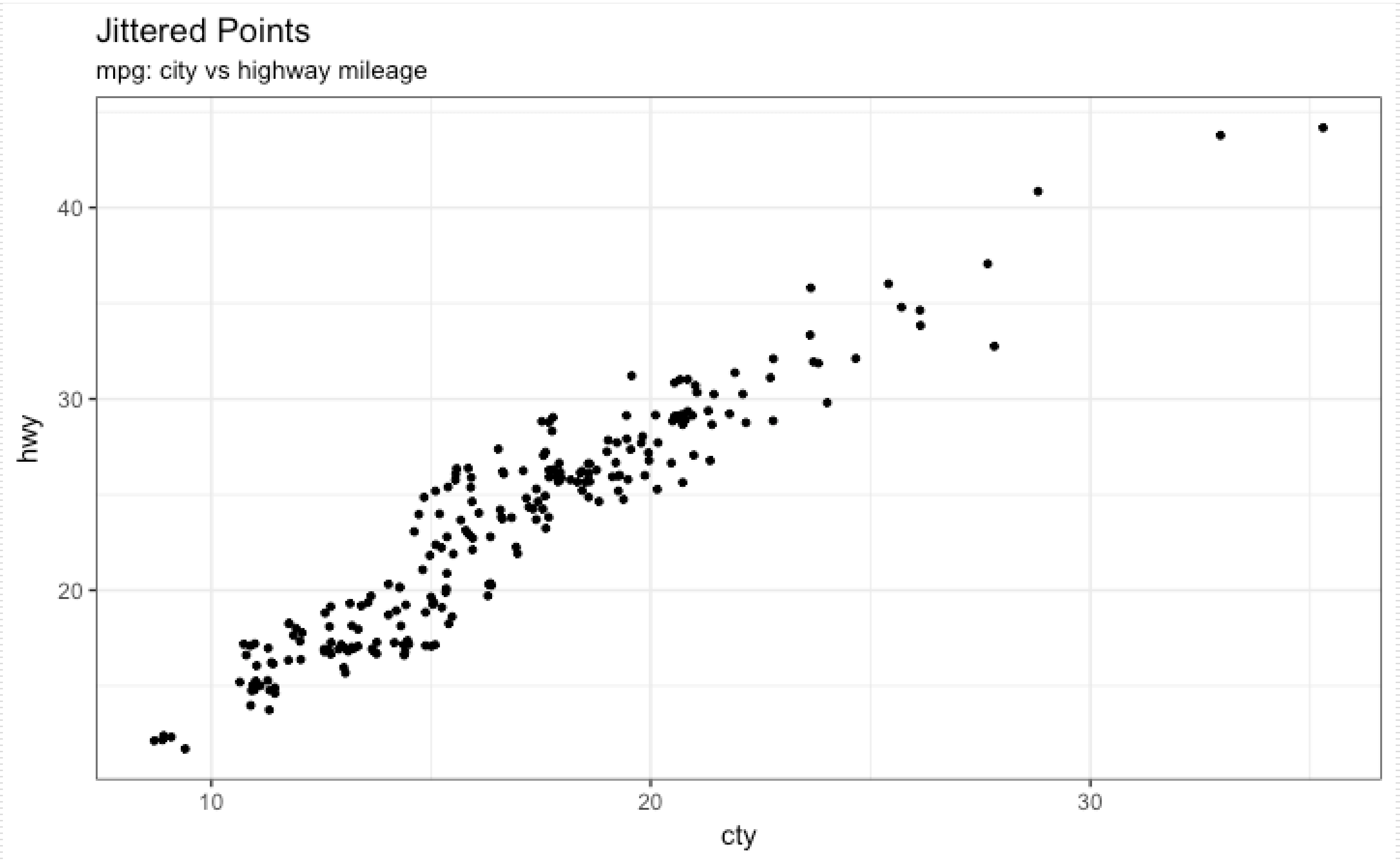
dim(mpg)

- The original data has 234 data points but the chart seems to display fewer points. What has happened? This is because there are many overlapping points appearing as a single dot. The fact that both cty and hwy are integers in the source dataset made it all the more convenient to hide this detail. So just be extra careful the next time you make scatterplot with integers.
- So how to handle this? There are few options. We can make a **jitter plot** with jitter_geom(). As the name suggests, the overlapping points are randomly jittered around its original position based on a threshold controlled by the width argument.

1. Correlation

```
# load package and data
library(ggplot2)
data(mpg, package="ggplot2")
# mpg <- read.csv("http://goo.gl/uEeRGu")
# Scatterplot
theme_set(theme_bw())  # pre-set the bw theme.
g <- ggplot(mpg, aes(cty, hwy))
g + geom_jitter(width = .5, size=1) +
  labs(subtitle="mpg: city vs highway mileage",
       y="hwy",
       x="cty",
       title="Jittered Points")
```

1. Correlation

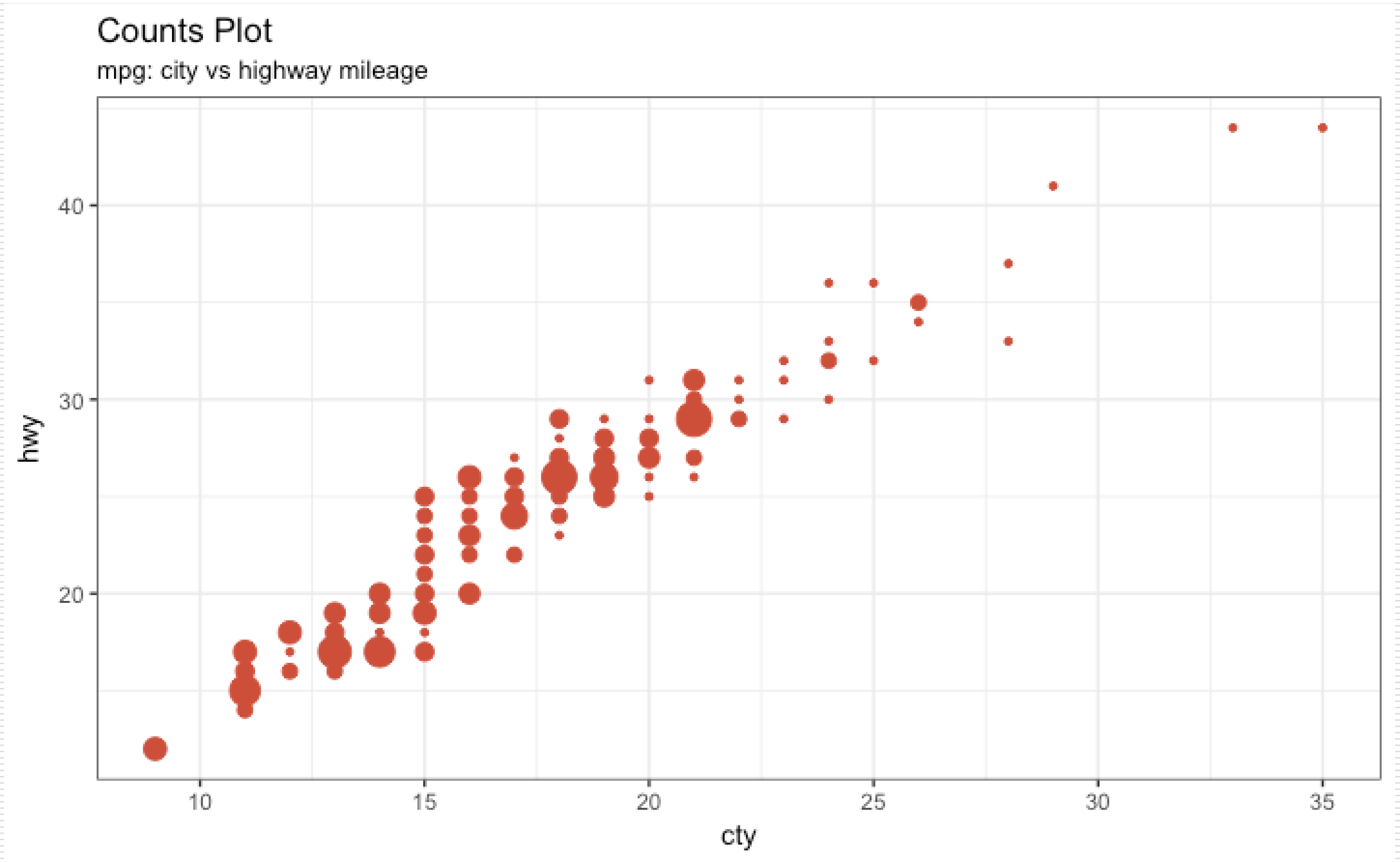


1. Correlation

- The second option to overcome the problem of data points overlap is to use what is called a **counts chart** with geom_count(). Wherever there is more points overlap, the size of the circle gets bigger.

```
# load package and data
library(ggplot2)
data(mpg, package="ggplot2")
# mpg <- read.csv("http://goo.gl/uEeRGu")

# Scatterplot
theme_set(theme_bw())  # pre-set the bw theme.
g <- ggplot(mpg, aes(cty, hwy))
g + geom_count(col="tomato3", show.legend=F) +
  labs(subtitle="mpg: city vs highway mileage",
       y="hwy",
       x="cty",
       title="Counts Plot")
```

1. Correlation



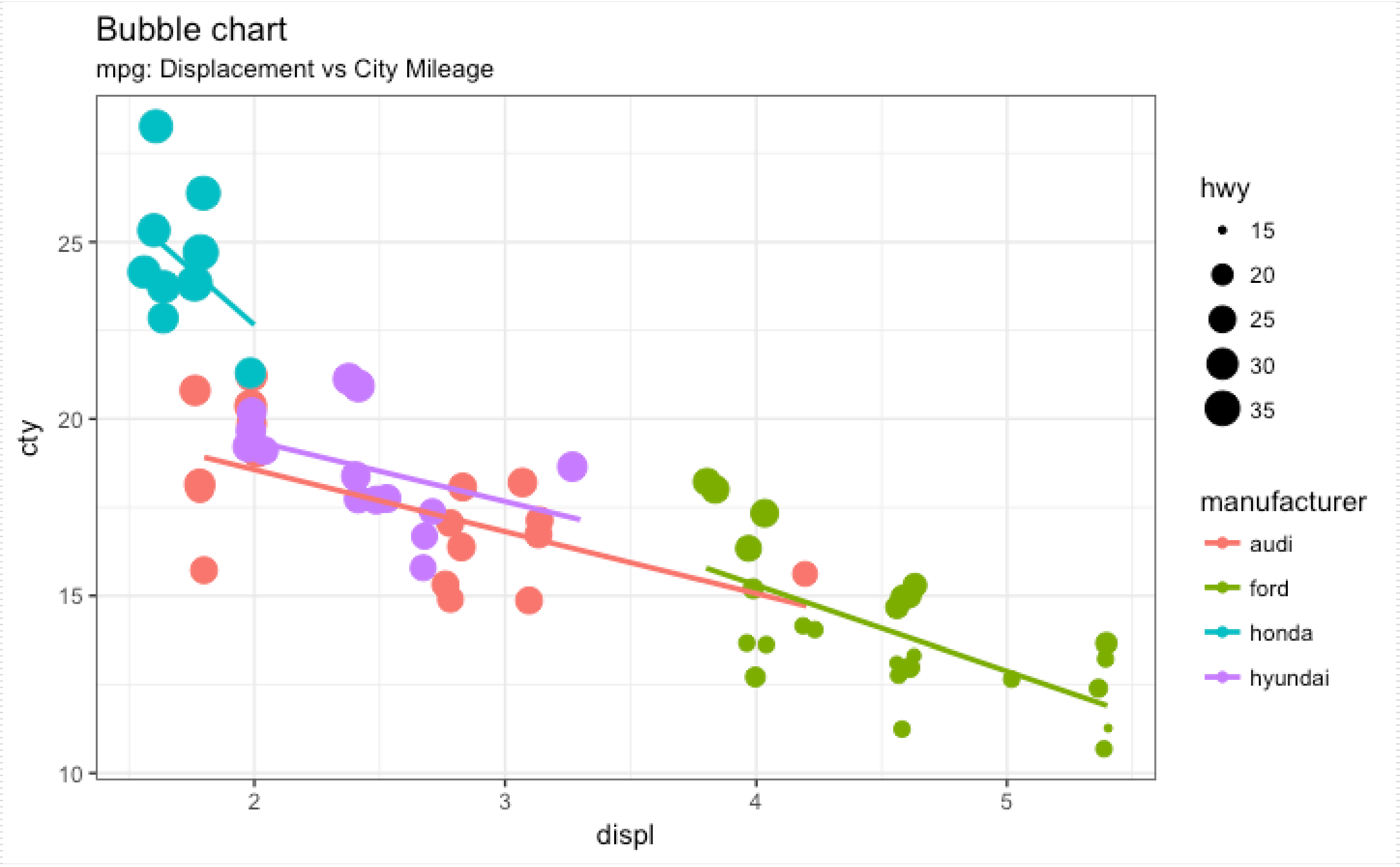
1. Correlation

- While scatterplot lets you compare the relationship between 2 continuous variables, **bubble chart** serves well if you want to understand relationship within the underlying groups based on:
  - a Categorical variable (by changing the color) and
  - another continuous variable (by changing the size of points).
- In simpler words, bubble charts are more suitable if you have 4-Dimensional data where two of them are numeric (X and Y) and one other categorical (color) and another numeric variable (size).

1. Correlation

```
# load package and data
library(ggplot2)
data(mpg, package="ggplot2")
# mpg <- read.csv("http://goo.gl/uEeRGu")

mpg_select <- mpg[mpg$manufacturer %in% c("audi", "ford", "honda", "hyundai"), ]

# Scatterplot
theme_set(theme_bw())  # pre-set the bw theme.
g <- ggplot(mpg_select, aes(displ, cty)) +
  labs(subtitle="mpg: Displacement vs City Mileage",
       title="Bubble chart")
g + geom_jitter(aes(col=manufacturer, size=hwy)) +
  geom_smooth(aes(col=manufacturer), method="lm", se=F)
```

1. Correlation



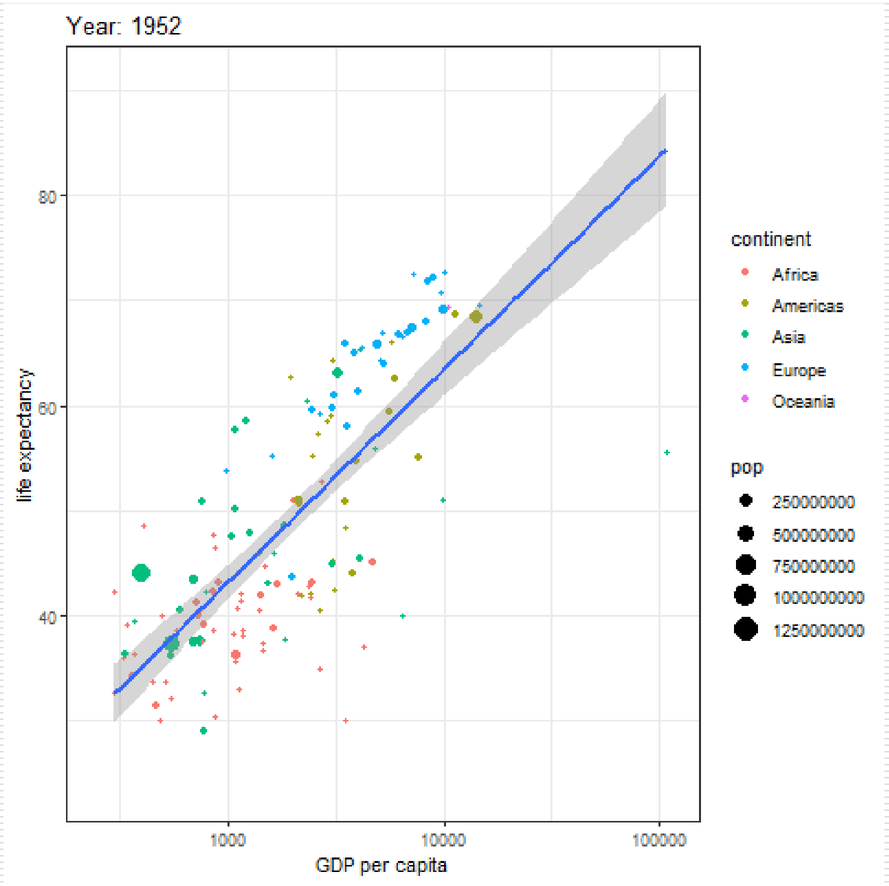
1. Correlation

- An **animated bubble chart** can be implemented using the gganimate package. It is same as the bubble chart, but, you have to show how the values change over a fifth dimension (typically time).
- The key thing to do is to set the aes(frame) to the desired column on which you want to animate. Rest of the procedure related to plot construction is the same. Once the plot is constructed, you can animate it using transition_time() by setting a chosen interval (e.g. ease_aes('linear'))

1. Correlation

```
> library(gganimate)
> library(gapminder)
> library(gifski)
> library(transformr)
> g <- ggplot(gapminder, aes(x = gdpPercap, y = lifeExp, frame = year)) +
  geom_point(aes(col=continent, size=pop)) +
            geom_smooth(aes(group = year),method = "lm") +
  scale_x_log10() +  # convert to log scale
  labs(title = 'Year: {frame_time}', x = 'GDP per capita', y = 'life
      expectancy') +
  transition_time(year) +
  ease_aes('linear')
 > animate(g, renderer = gifski_renderer())
```

1. Correlation



1. Correlation

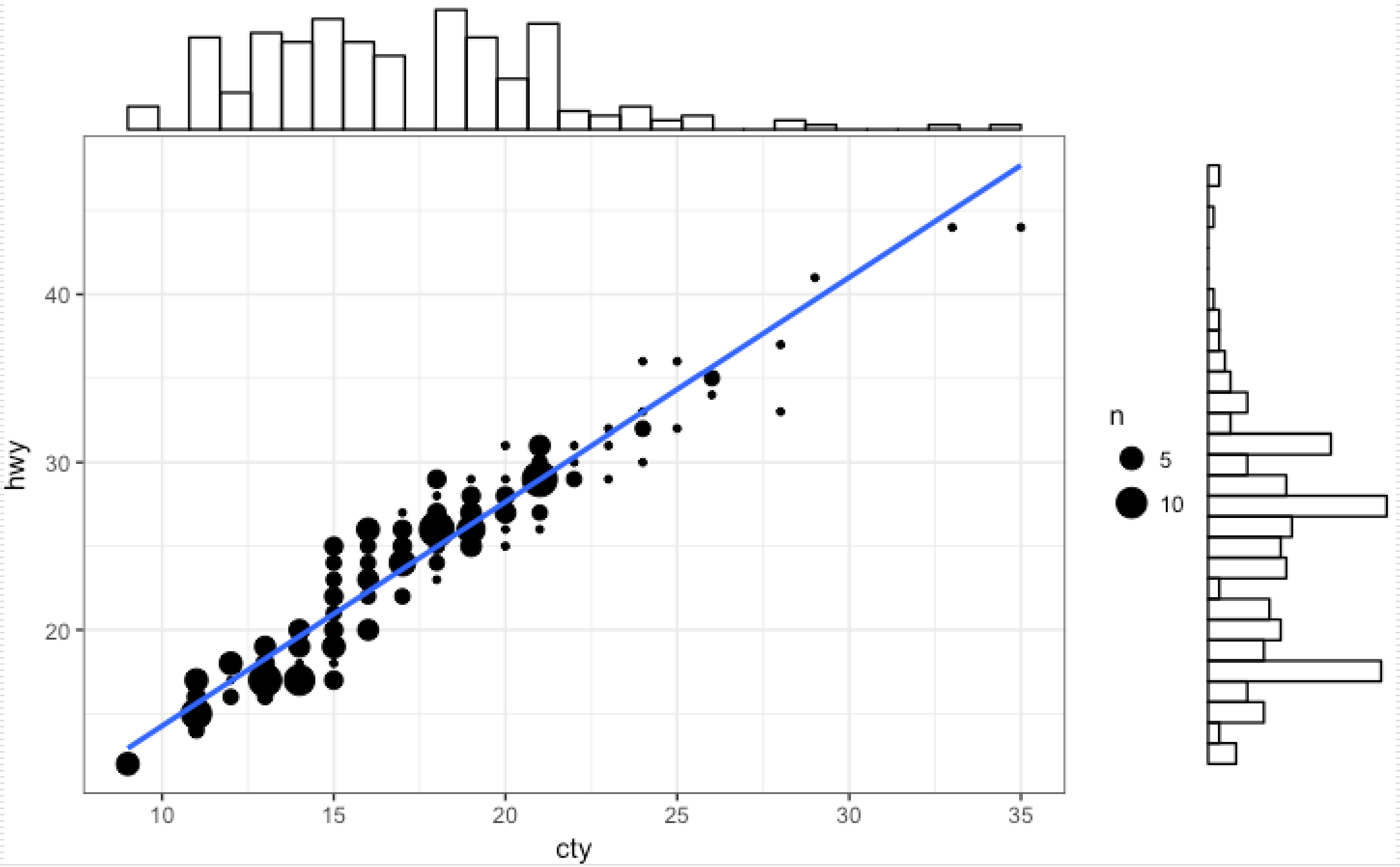
- If you want to show the relationship as well as the distribution in the same chart, use the marginal histogram. It has a histogram of the X and Y variables at the margins of the scatterplot.

- This can be implemented using the ggMarginal() function from the ‘ggExtra’ package. Apart from a histogram, you could choose to draw a marginal boxplot or density plot by setting the respective type option.

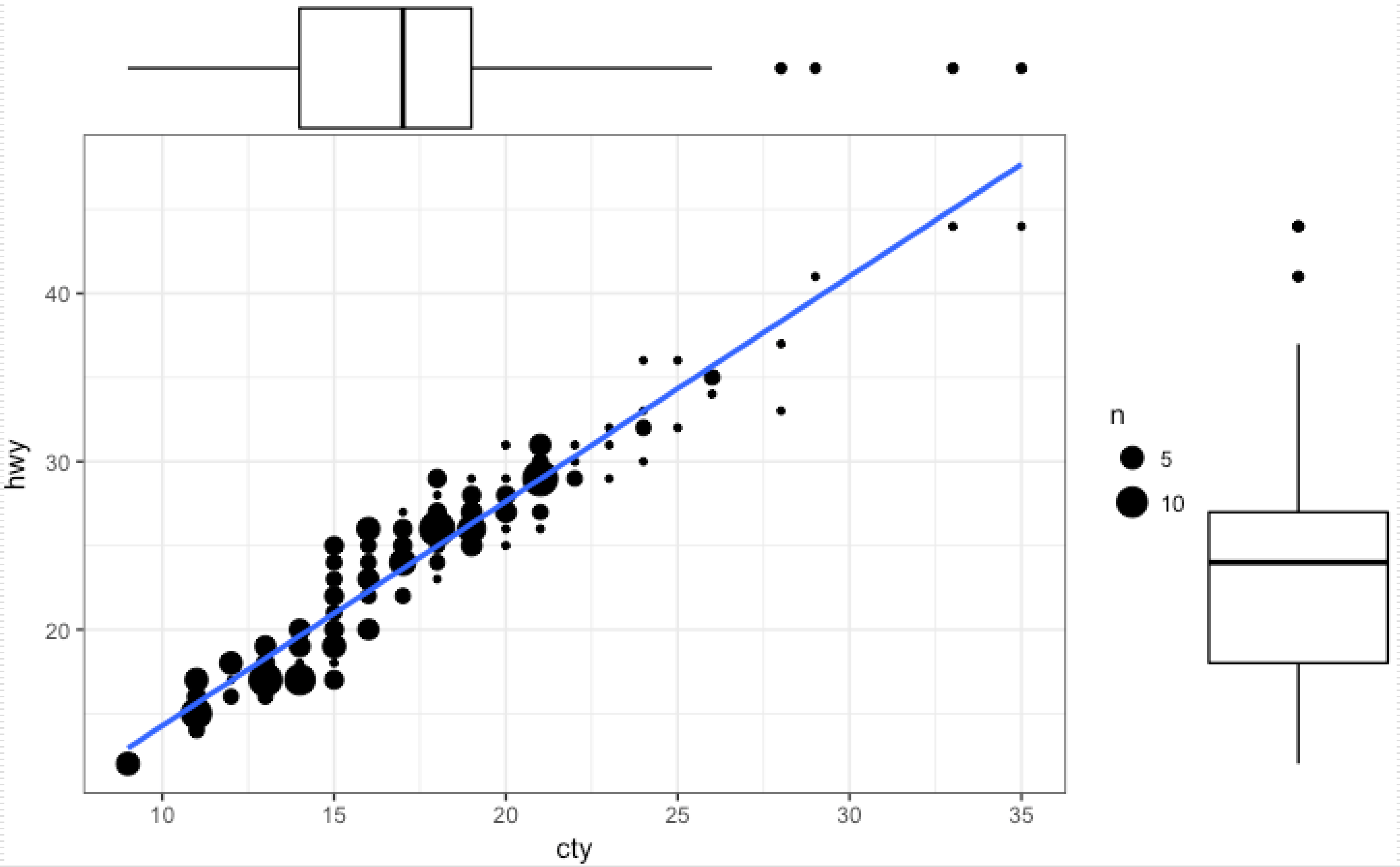
1. Correlation

```
# install ggExtra
# load package and data
library(ggplot2)
library(ggExtra)
data(mpg, package="ggplot2")
# mpg <- read.csv("http://goo.gl/uEeRGu")
# Scatterplot
theme_set(theme_bw())  # pre-set the bw theme.
mpg_select <- mpg[mpg$hwy >= 35 & mpg$cty > 27, ]
g <- ggplot(mpg, aes(cty, hwy)) +
  geom_count() +
  geom_smooth(method="lm", se=F)
ggMarginal(g, type = "histogram", fill="transparent")
ggMarginal(g, type = "boxplot", fill="transparent")
# ggMarginal(g, type = "density", fill="transparent")
```

1. Correlation



1. Correlation



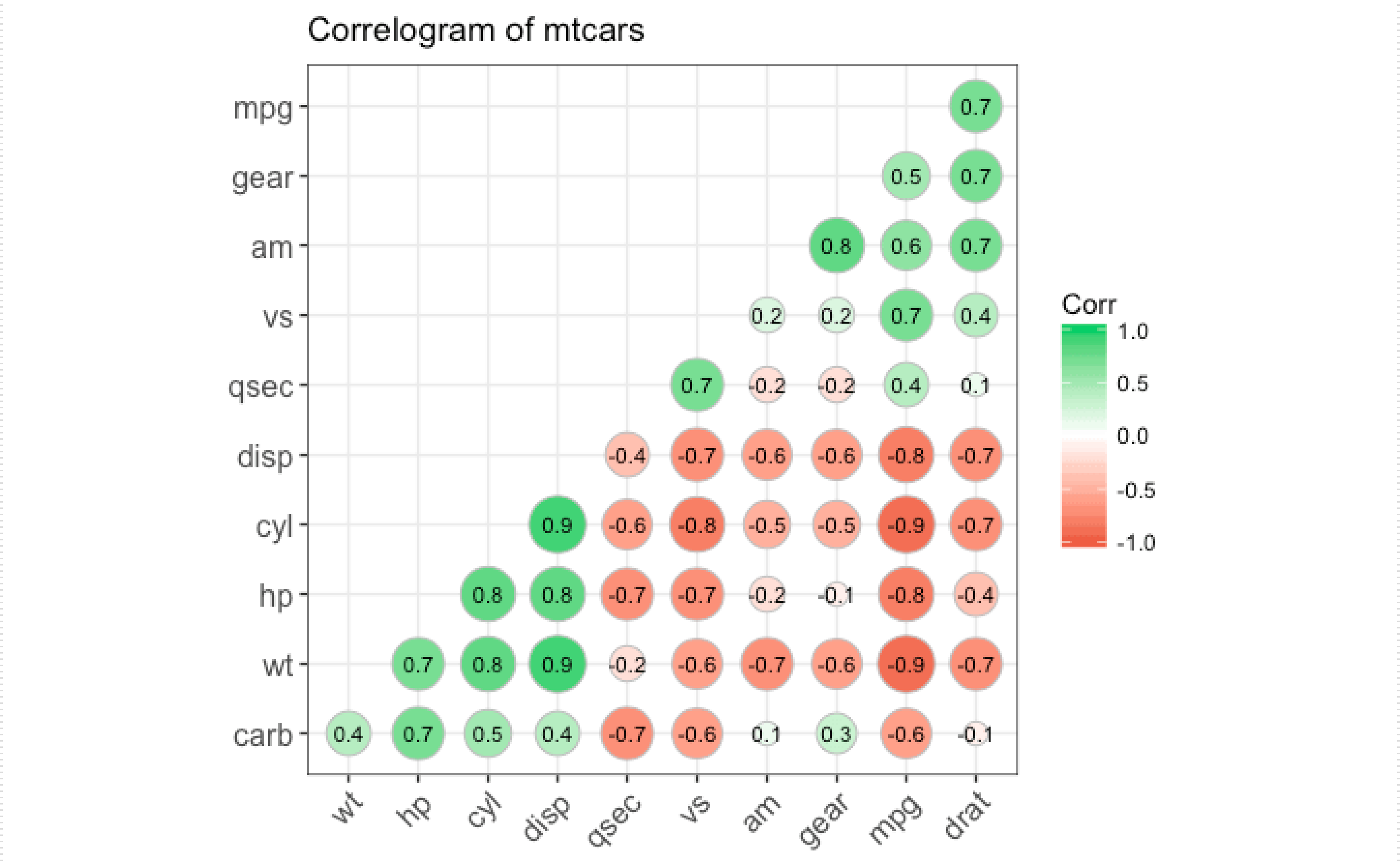
1. Correlation

- **Correlogram** let’s you examine the corellation of multiple continuous variables present in the same dataframe. This is conveniently implemented using the ggcorrplot package.

1. Correlation

```
# devtools::install_github("kassambara/ggcorrplot")
library(ggplot2)
library(ggcorrplot)
# Correlation matrix
data(mtcars)
corr <- round(cor(mtcars), 1)
# Plot
ggcorrplot(corr, hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 3,
           method="circle",
           colors = c("tomato2", "white", "springgreen3"),
           title="Correlogram of mtcars",
           ggtheme=theme_bw)
```

1. Correlation



	wt	hp	cyl	disp	qsec	vs	am	gear	mpg	drat
mpg										0.7
gear									0.5	0.7
am								0.8	0.6	0.7
vs							0.2	0.2	0.7	0.4
qsec						0.7	-0.2	-0.2	0.4	0.1
disp					-0.4	-0.7	-0.6	-0.6	-0.8	-0.7
cyl				0.9	-0.6	-0.8	-0.5	-0.5	-0.9	-0.7
hp			0.8	0.8	-0.7	-0.7	-0.2	-0.1	-0.8	-0.4
wt		0.7	0.8	0.9	-0.2	-0.6	-0.7	-0.6	-0.9	-0.7
carb	0.4	0.7	0.5	0.4	-0.7	-0.6	0.1	0.3	-0.6	-0.1

2. Deviations

- **Diverging Bars** is a bar chart that can handle both negative and positive values. This can be implemented by a smart tweak with geom_bar(). But the usage of geom_bar() can be quite confusing. That's because, it can be used to make a bar chart as well as a histogram. Let's explain.
- By default, geom_bar() has the stat set to count. That means, when you provide just a continuous X variable (and no Y variable), it tries to make a histogram out of the data.
- In order to make a bar chart create bars instead of histogram, you need to do two things.

2. Deviations

- Set stat=identity
- Provide both x and y inside aes() where, x is either character or factor and y is numeric.
- In order to make sure you get diverging bars instead of just bars, make sure, your categorical variable has 2 categories that changes values at a certain threshold of the continuous variable. In below example, the mpg from mtcars dataset is normalised by computing the z score. Those vehicles with mpg above zero are marked green and those below are marked red.

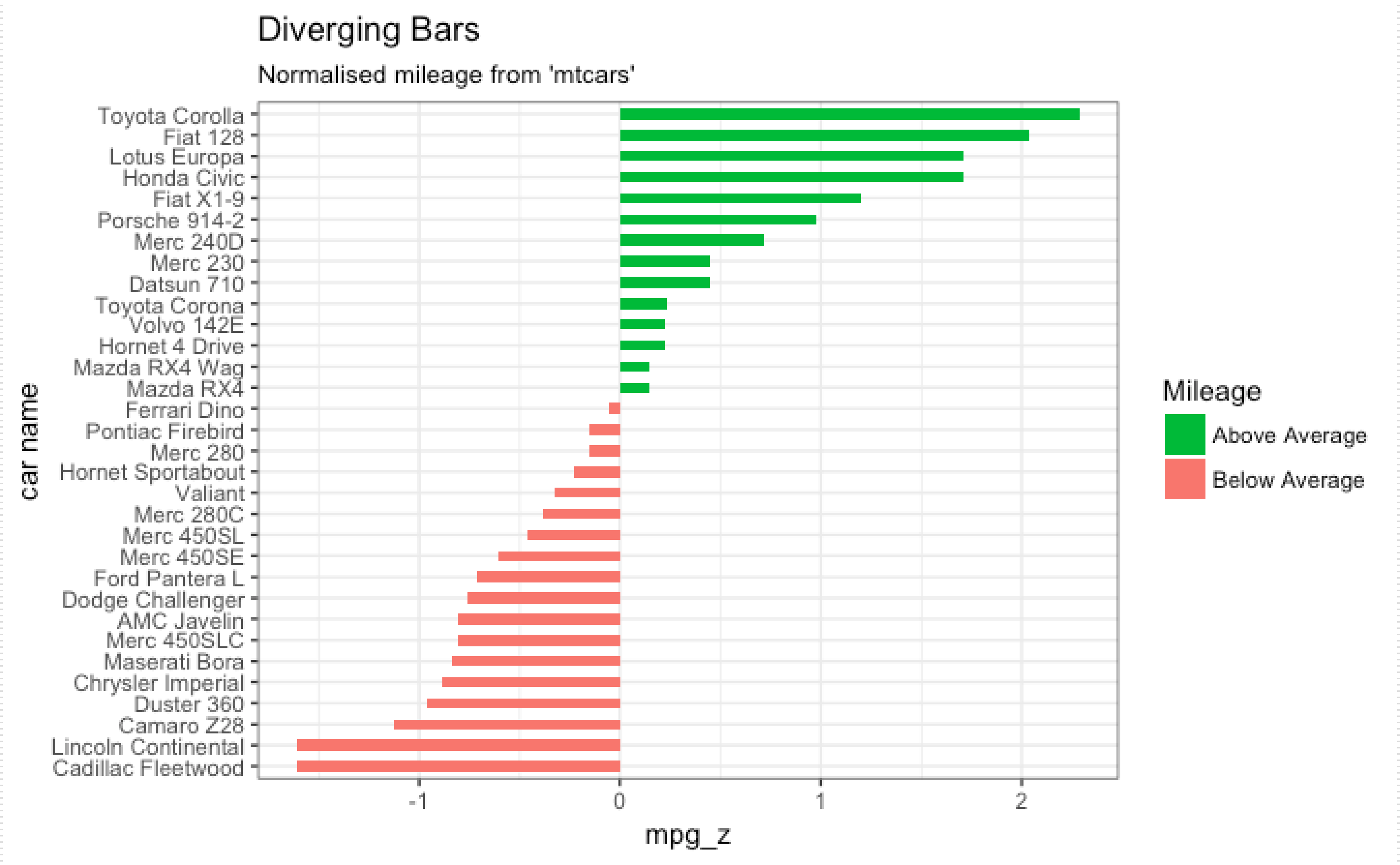
2. Deviations

```
library(ggplot2)
theme_set(theme_bw())
# Data Prep
data("mtcars")  # load data
mtcars$`car name` <- rownames(mtcars)  # create new
column for car names
mtcars$mpg_z <- round((mtcars$mpg -
mean(mtcars$mpg))/sd(mtcars$mpg), 2)
# compute normalized mpg
mtcars$mpg_type <- ifelse(mtcars$mpg_z < 0, "below",
"above")  # above / below avg flag
mtcars <- mtcars[order(mtcars$mpg_z), ]  # sort
```

2. Deviations

```
mtcars$`car name` <- factor(mtcars$`car name`, levels = 
mtcars$`car name`)  # convert to factor to retain sorted order in 
plot.
# Diverging Barcharts
ggplot(mtcars, aes(x=`car name`, y=mpg_z, label=mpg_z)) + 
  geom_bar(stat='identity', aes(fill=mpg_type), width=.5)  +
  scale_fill_manual(name="Mileage", 
                    labels = c("Above Average", "Below Average"), 
values = c("above"="#00ba38", "below"="#f8766d")) + 
  labs(subtitle="Normalised mileage from 'mtcars'", 
       title= "Diverging Bars") + 
  coord_flip()
```

2. Deviations



2. Deviations

- **Area charts** are typically used to visualize how a particular metric (such as % returns from a stock) performed compared to a certain baseline. Other types of %returns or %change data are also commonly used. The geom_area() implements this.

2. Deviations

```
#install package lubridate (to get/set years component of a date-time)
data("economics", package = "ggplot2")
# Compute % Returns
economics$returns_perc <- c(0, diff(economics$psavert)/economics$psavert[-length(economics$psavert)])
# Create break points and labels for axis ticks
brks <- economics$date[seq(1, length(economics$date), 12)]
lbls <- lubridate::year(economics$date[seq(1, length(economics$date), 12)])
```

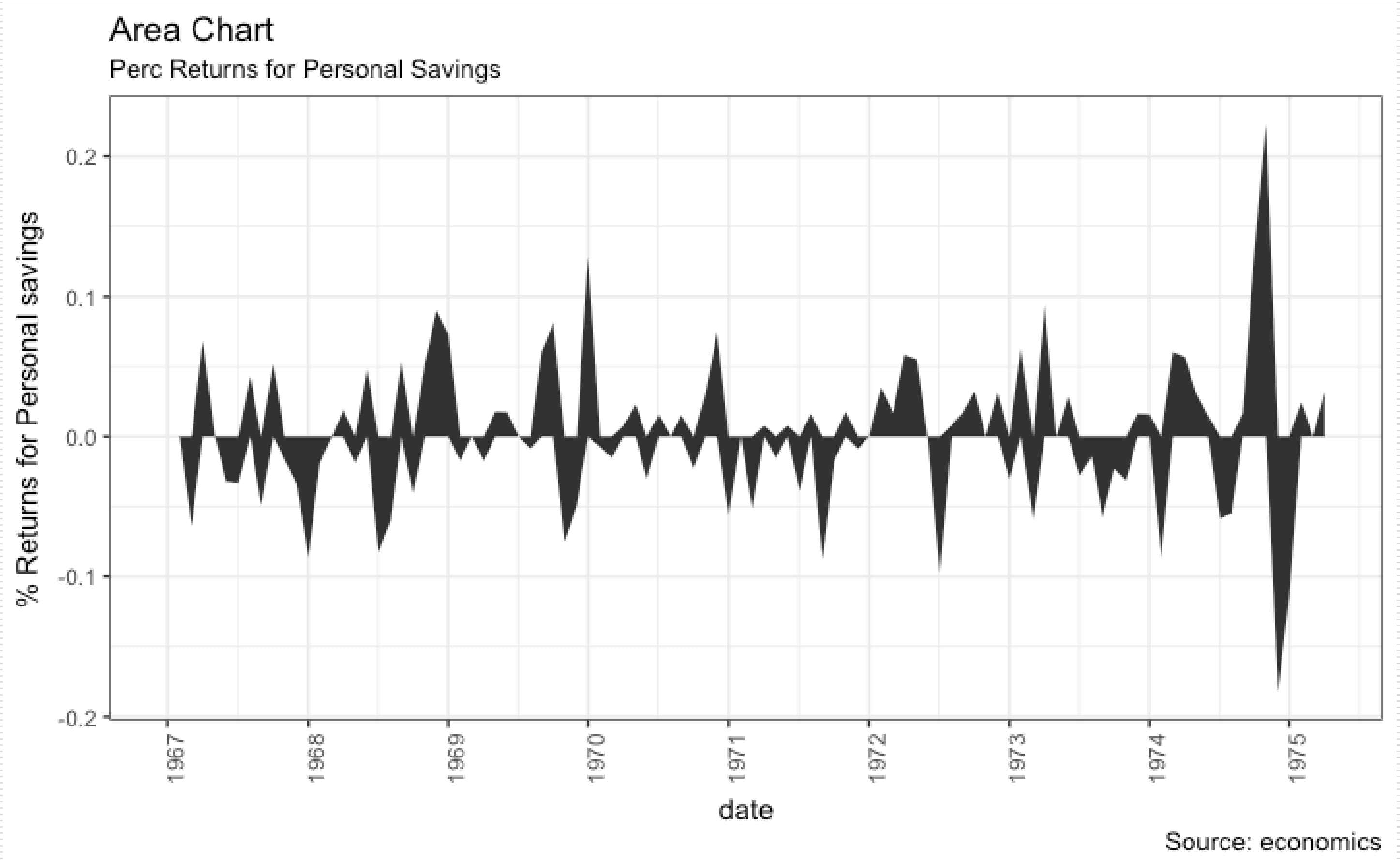
Successive Differences

personal savings rate

2. Deviations

```
# Plot
ggplot(economics[1:100, ], aes(date, returns_perc)) +
  geom_area() +
  scale_x_date(breaks=brks, labels=lbls) +
  theme(axis.text.x = element_text(angle=90)) +
  labs(title="Area Chart",
       subtitle = "Perc Returns for Personal Savings",
       y="% Returns for Personal savings",
       caption="Source: economics")
```

2. Deviations



3. Ranking

- **Ordered Bar Chart** is a Bar Chart that is ordered by the Y axis variable. Just sorting the dataframe by the variable of interest isn't enough to order the bar chart. In order for the bar chart to retain the order of the rows, the X axis variable (i.e. the categories) has to be converted into a factor.
- Let's plot the mean city mileage for each manufacturer from mpg dataset. First, aggregate the data and sort it before you draw the plot. Finally, the X variable is converted to a factor.

3. Ranking

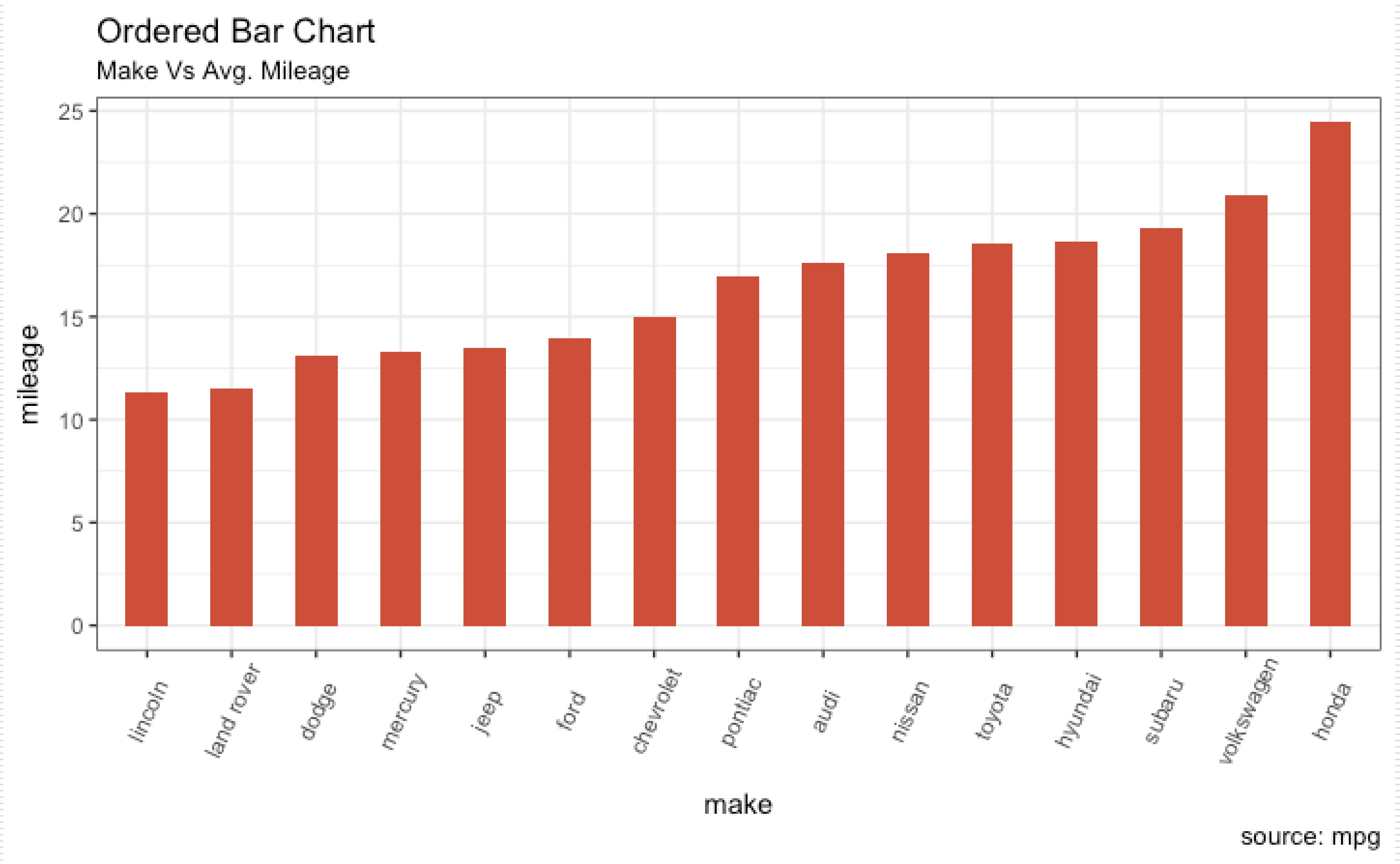
```
# Prepare data: group mean city mileage by manufacturer.
cty_mpg <- aggregate(mpg$cty, by=list(mpg$manufacturer), FUN=mean)  # aggregate
colnames(cty_mpg) <- c("make", "mileage")  # change column names
cty_mpg <- cty_mpg[order(cty_mpg$mileage), ]  # sort
cty_mpg$make <- factor(cty_mpg$make, levels = cty_mpg$make)  # to retain the order in plot.
head(cty_mpg, 4)
#>          make  mileage
#> 9     lincoln 11.33333
#> 8  land rover 11.50000
#> 3       dodge 13.13514
#> 10    mercury 13.25000
```

3. Ranking

- The X variable is now a factor, let’s plot.

```
library(ggplot2)
theme_set(theme_bw())
# Draw plot
ggplot(cty_mpg, aes(x=make, y=mileage)) +
  geom_bar(stat="identity", width=.5, fill="tomato3") +
  labs(title="Ordered Bar Chart",
       subtitle="Make Vs Avg. Mileage",
       caption="source: mpg") +
  theme(axis.text.x = element_text(angle=65, vjust=0.6))
```

3. Ranking


Ordered Bar Chart
Make Vs Avg. Mileage
mileage
25
20
15
10
5
0
lincoln
land rover
dodge
mercury
jeep
ford
chevrolet
pontiac
audi
nissan
toyota
hyundai
subaru
volkswagen
honda
make
source: mpg

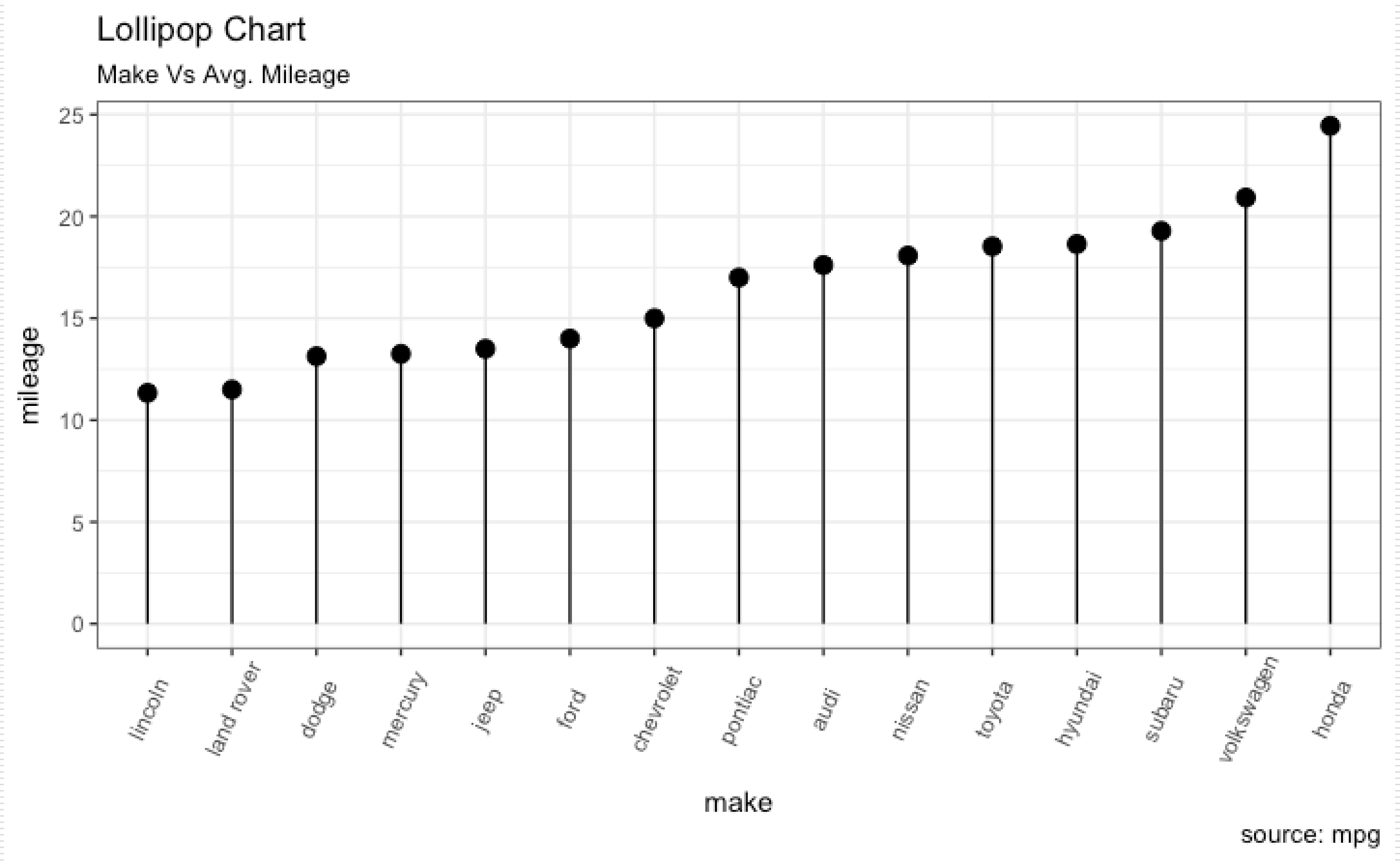
3. Ranking

- **Lollipop charts** conveys the same information as in bar charts. By reducing the thick bars into thin lines, it reduces the clutter and lays more emphasis on the value. It looks nice and modern.

3. Ranking

```
library(ggplot2)
theme_set(theme_bw())
# Plot
ggplot(cty_mpg, aes(x=make, y=mileage)) +
  geom_point(size=3) +
  geom_segment(aes(x=make,
                   xend=make,
                   y=0,
                   yend=mileage)) +
  labs(title="Lollipop Chart",
       subtitle="Make Vs Avg. Mileage",
       caption="source: mpg") +
  theme(axis.text.x = element_text(angle=65, vjust=0.6))
```

3. Ranking


Lollipop Chart
Make Vs Avg. Mileage
mileage
0
5
10
15
20
25
lincoln
land rover
dodge
mercury
jeep
ford
chevrolet
pontiac
audi
nissan
toyota
hyundai
subaru
volkswagen
honda
make
source: mpg

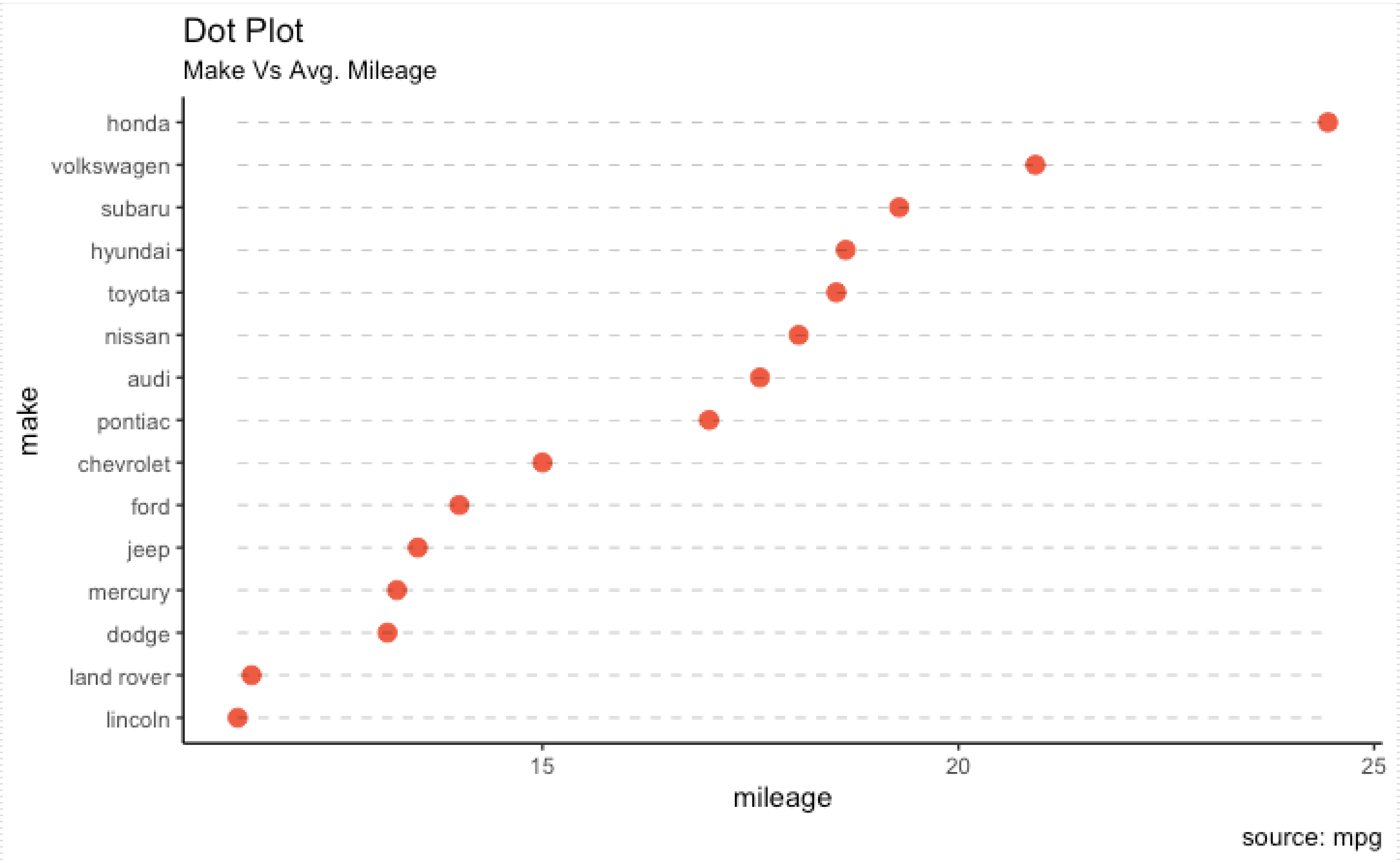
3. Ranking

- **Dot plots** are very similar to lollipops, but without the line and is flipped to horizontal position. It emphasizes more on the rank ordering of items with respect to actual values and how far apart are the entities with respect to each other.

3. Ranking

```
#install scales
library(ggplot2)
library(scales)
theme_set(theme_classic())
# Plot
ggplot(cty_mpg, aes(x=make, y=mileage)) +
  geom_point(col="tomato2", size=3) +   # Draw points
  geom_segment(aes(x=make,
                   xend=make,
                   y=min(mileage),
                   yend=max(mileage)),
               linetype="dashed",
               size=0.1) +   # Draw dashed lines
  labs(title="Dot Plot",
       subtitle="Make Vs Avg. Mileage",
       caption="source: mpg") +
  coord_flip()
```

3. Ranking


Dot Plot
Make Vs Avg. Mileage
honda
volkswagen
subaru
hyundai
toyota
nissan
audi
pontiac
chevrolet
ford
jeep
mercury
dodge
land rover
lincoln
make
15
20
25
mileage
source: mpg

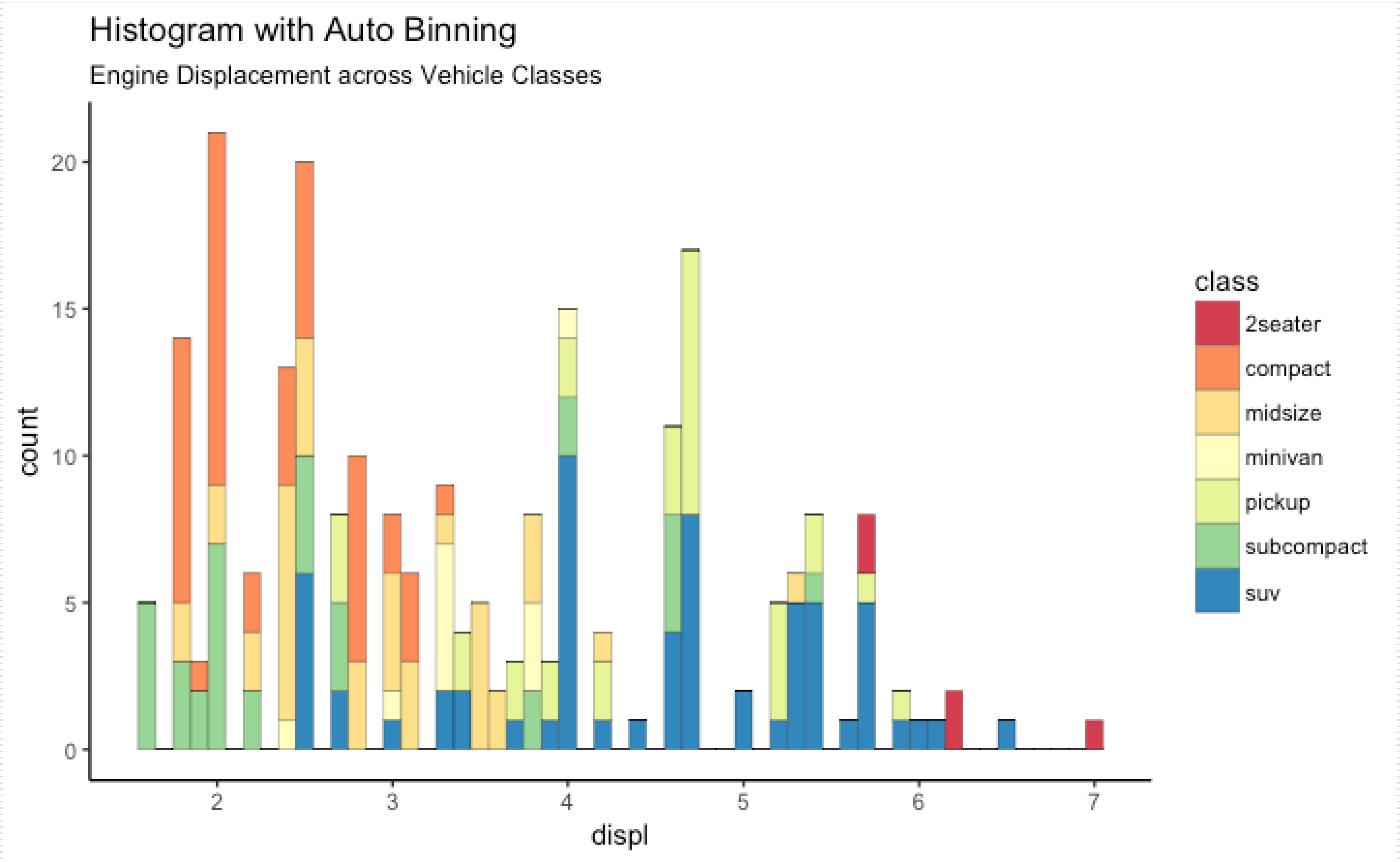
4. Distribution

- **Histogram** on a continuous variable can be accomplished using either geom_bar() or geom_histogram(). When using geom_histogram(), you can control the **number of bars** using the bins option. Else, you can set the **range** covered by each bin using binwidth. The value of binwidth is on the same scale as the continuous variable on which histogram is built. Since, geom_histogram gives facility to control both number of bins as well as binwidth, it is the preferred option to create histogram on continuous variables.

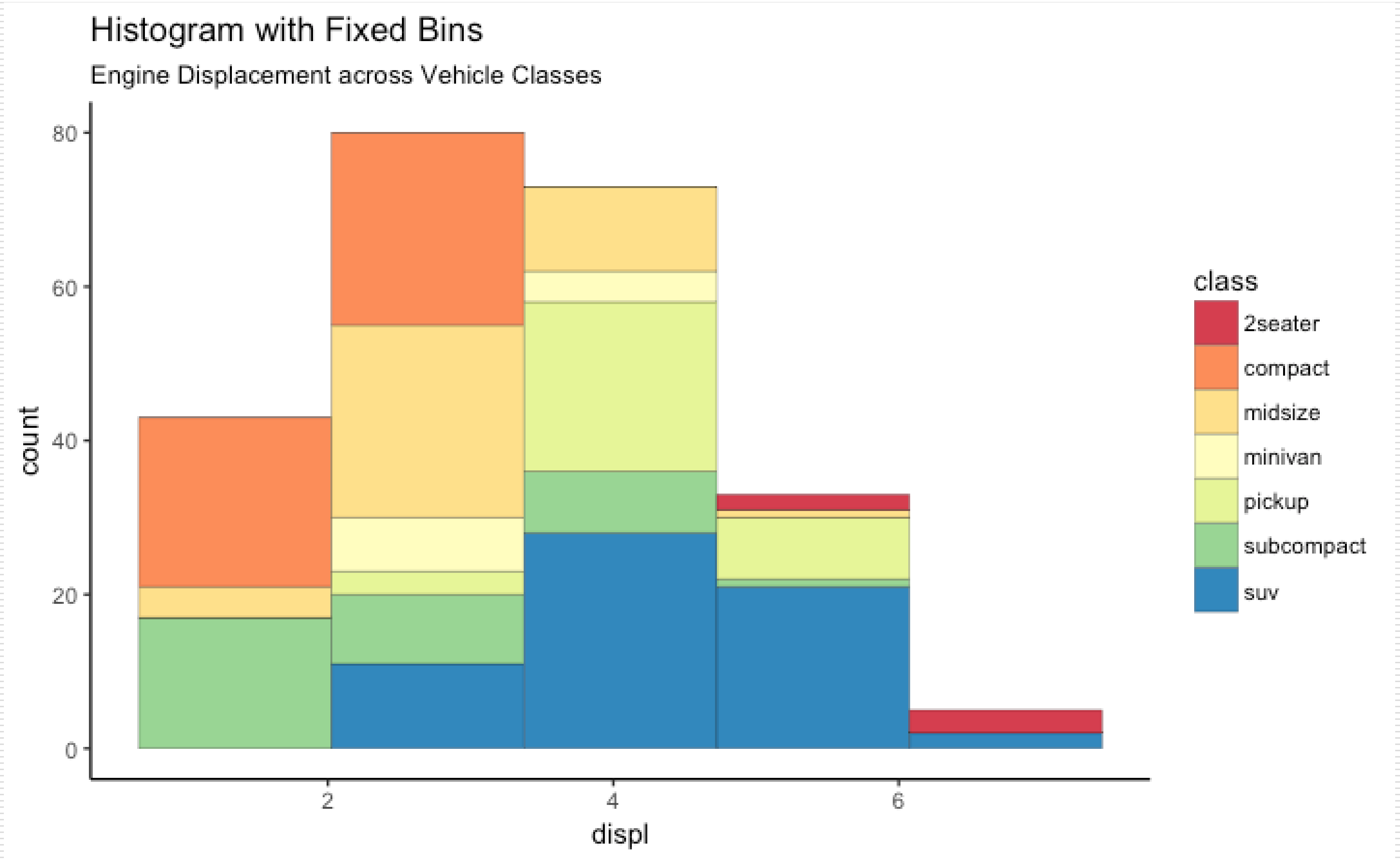
4. Distribution

```
library(ggplot2)
theme_set(theme_classic())
# Histogram on a Continuous (Numeric) Variable
g <- ggplot(mpg, aes(displ)) + scale_fill_brewer(palette = "Spectral")
g + geom_histogram(aes(fill=class),
                   binwidth = .1,
                   col="black",
                   size=.1) +  # change binwidth
  labs(title="Histogram with Auto Binning",
       subtitle="Engine Displacement across Vehicle Classes")
g + geom_histogram(aes(fill=class),
                   bins=5,
                   col="black",
                   size=.1) +   # change number of bins
  labs(title="Histogram with Fixed Bins",
       subtitle="Engine Displacement across Vehicle Classes")
```

4. Distribution



4. Distribution

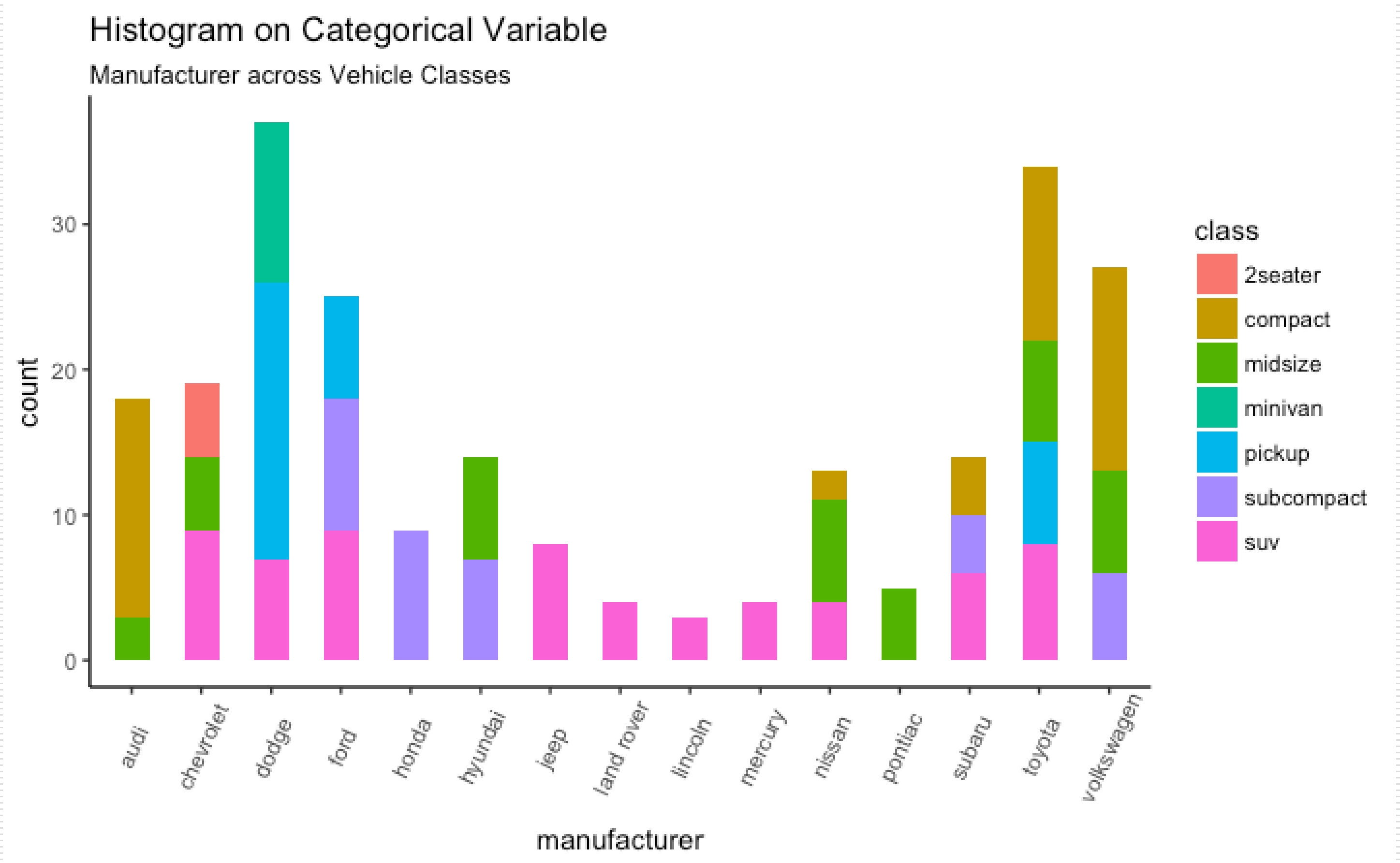


4. Distribution

- **Bar Chart** on a categorical variable would result in a frequency chart showing bars for each category. By adjusting width, you can adjust the thickness of the bars.

```
library(ggplot2)
theme_set(theme_classic())

# Histogram on a Categorical variable
g <- ggplot(mpg, aes(manufacturer))
g + geom_bar(aes(fill=class), width = 0.5) +
  theme(axis.text.x = element_text(angle=65, vjust=0.6)) +
  labs(title="Histogram on Categorical Variable",
       subtitle="Manufacturer across Vehicle Classes")
```

4. Distribution



4. Distribution

- By default, geom_bar() has the stat set to count. That means, when you provide just a continuous X variable (and no Y variable), it tries to make a histogram out of the data.
- In order to make a bar chart create bars instead of histogram, you need to do two things.
- Set stat=identity
- Provide both x and y inside aes() where, x is either character or factor and y is numeric.
- A bar chart can be drawn from a categorical column variable or from a separate frequency table. By adjusting width, you can adjust the thickness of the bars. If your data source is a frequency table, that is, if you don’t want ggplot to compute the counts, you need to set the stat=identity inside the geom_bar().

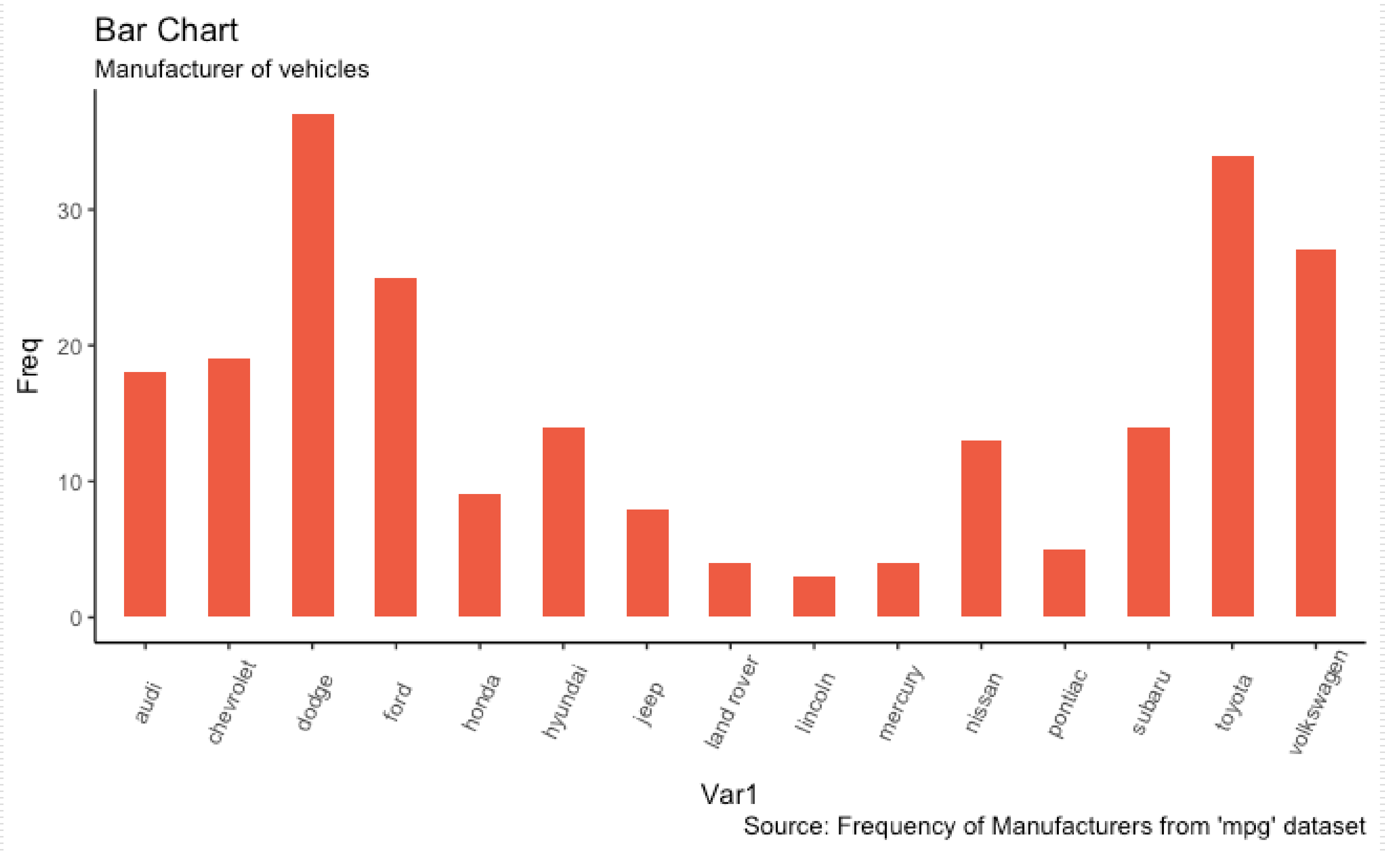
4. Distribution

```
# prep frequency table
freqtable <- table(mpg$manufacturer)
df <- as.data.frame.table(freqtable)
head(df)
#>         Var1 Freq
#> 1       audi   18
#> 2  chevrolet   19
#> 3      dodge   37
#> 4       ford   25
#> 5      honda    9
#> 6    hyundai   14
# plot
library(ggplot2)
theme_set(theme_classic())
```

4. Distribution

```
# Plot
g <- ggplot(df, aes(Var1, Freq))
g + geom_bar(stat="identity", width = 0.5, fill="tomato2") 
+
      labs(title="Bar Chart",
            subtitle="Manufacturer of vehicles",
            caption="Source: Frequency of Manufacturers from 
'mpg' dataset") +
      theme(axis.text.x = element_text(angle=65, 
vjust=0.6))
```

4. Distribution

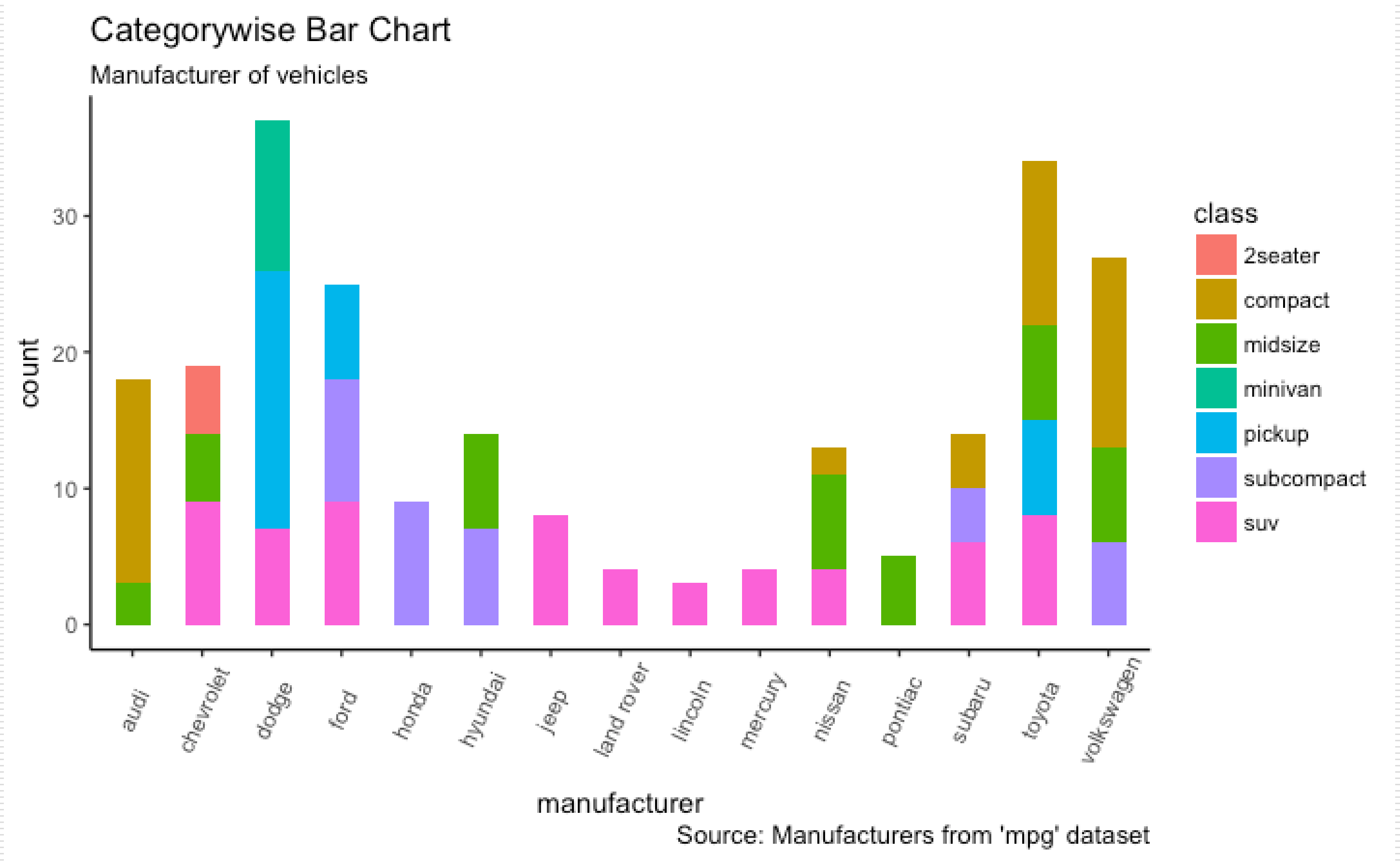


4. Distribution

- It can be computed directly from a column variable as well. In this case, only X is provided and stat=identity is not set.

```
# From on a categorical column variable
g <- ggplot(mpg, aes(manufacturer))
g + geom_bar(aes(fill=class), width = 0.5) +
  theme(axis.text.x = element_text(angle=65, vjust=0.6))
+
  labs(title="Categorywise Bar Chart",
       subtitle="Manufacturer of vehicles",
       caption="Source: Manufacturers from 'mpg' dataset")
```

4. Distribution

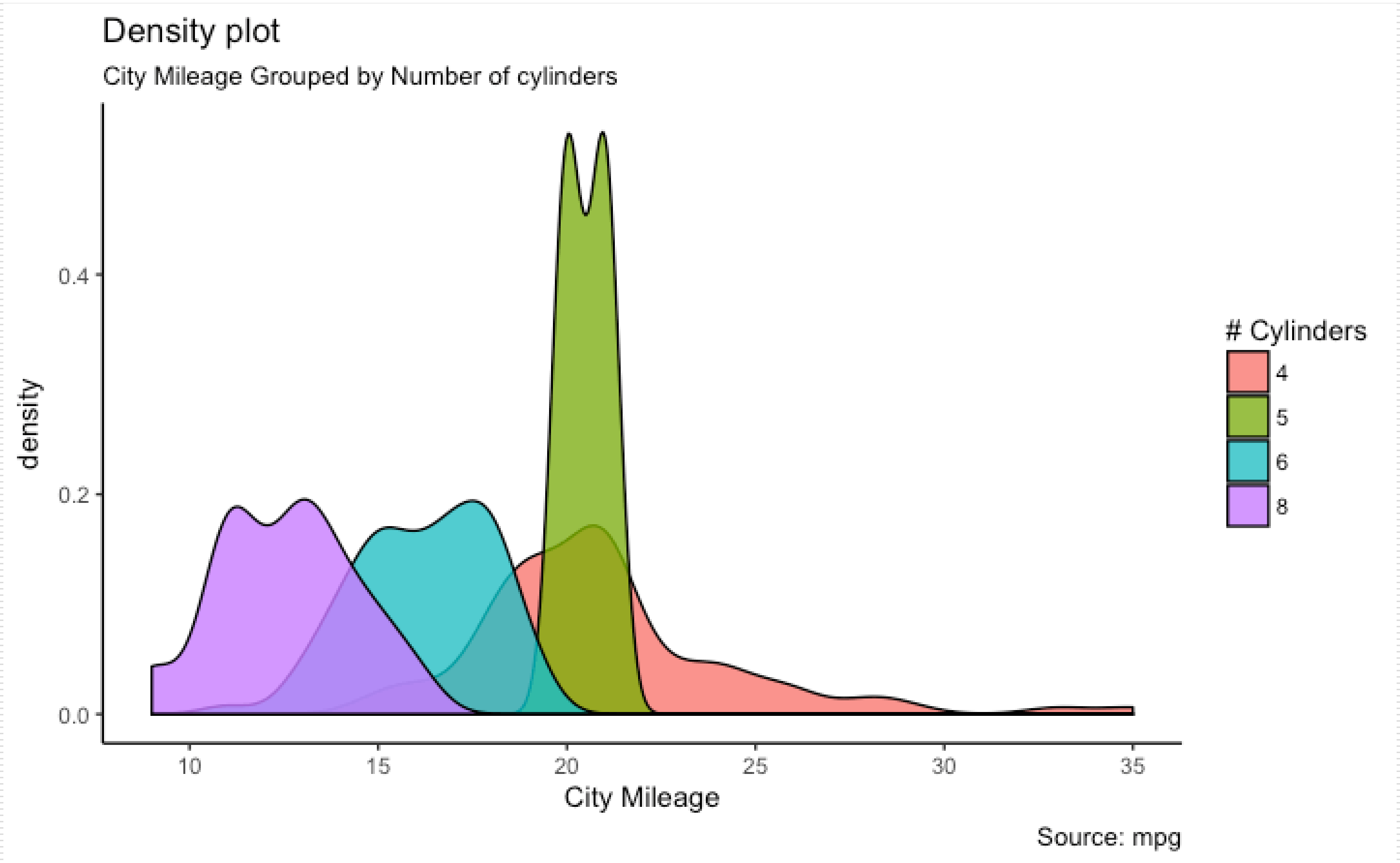


4. Distribution

- Density Plot

```
library(ggplot2)
theme_set(theme_classic())

# Plot
g <- ggplot(mpg, aes(cty))
g + geom_density(aes(fill=factor(cyl)), alpha=0.8) +
    labs(title="Density plot",
         subtitle="City Mileage Grouped by Number of cylinders",
         caption="Source: mpg",
         x="City Mileage",
         fill="# Cylinders")
```

4. Distribution



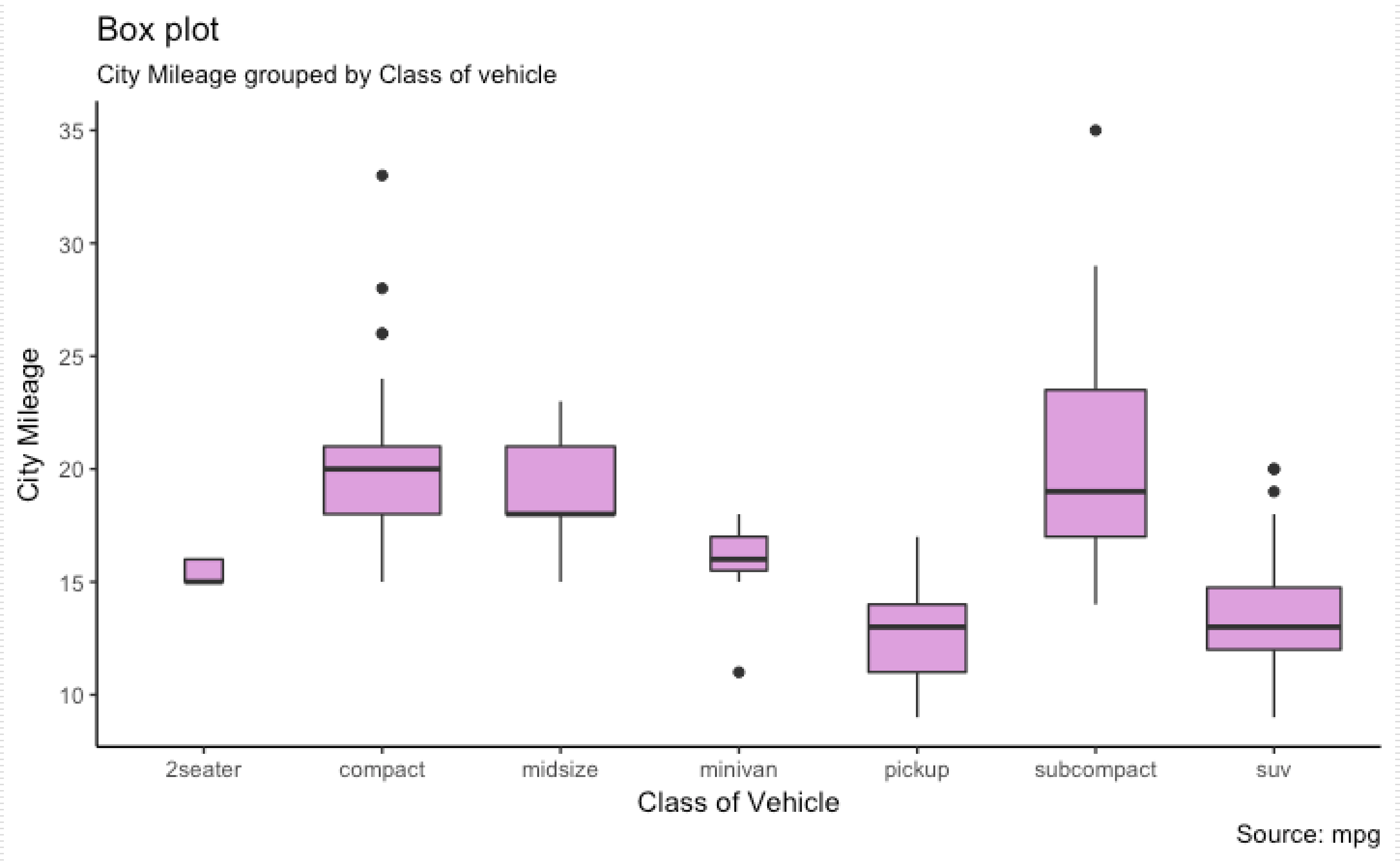
4. Distribution

- **Box plot** is an excellent tool to study the distribution. It can also show the distributions within multiple groups, along with the median, range and outliers if any.
- The dark line inside the box represents the median. The top of box is 75%ile and bottom of box is 25%ile. The end points of the lines (aka whiskers) is at a distance of 1.5*IQR, where IQR or Inter Quartile Range is the distance between 25th and 75th percentiles. The points outside the whiskers are marked as dots and are normally considered as extreme points.
- Setting varwidth=T adjusts the width of the boxes to be proportional to the number of observation it contains.

4. Distribution

```
library(ggplot2)
theme_set(theme_classic())
# Plot
g <- ggplot(mpg, aes(class, cty))
g + geom_boxplot(varwidth=T, fill="plum") +
   labs(title="Box plot",
        subtitle="City Mileage grouped by Class of vehicle",
        caption="Source: mpg",
        x="Class of Vehicle",
        y="City Mileage")
```

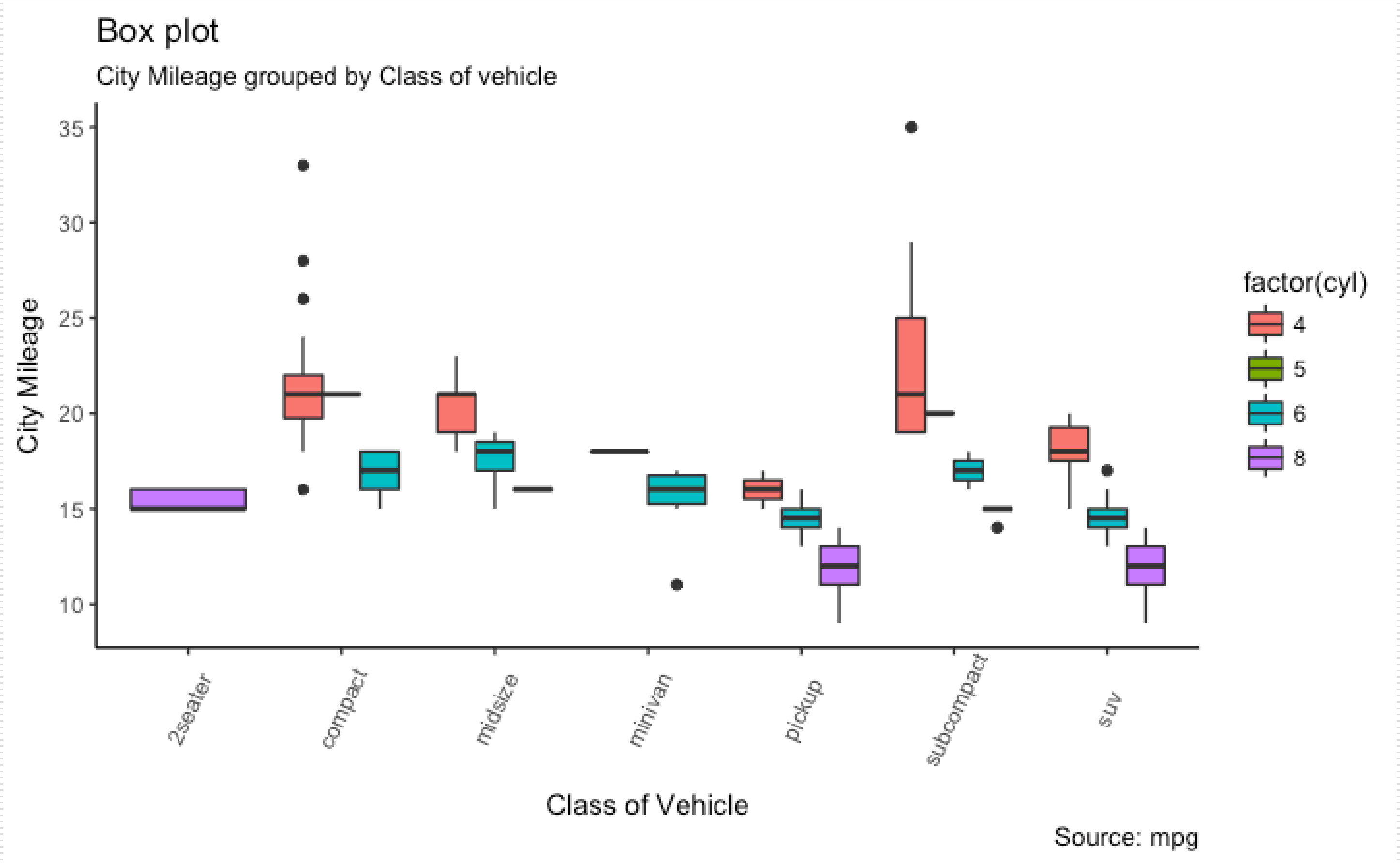
4. Distribution



4. Distribution

```
library(ggthemes)
g <- ggplot(mpg, aes(class, cty))
g + geom_boxplot(aes(fill=factor(cyl))) +
  theme(axis.text.x = element_text(angle=65, vjust=0.6))
+
  labs(title="Box plot",
       subtitle="City Mileage grouped by Class of vehicle",
       caption="Source: mpg",
       x="Class of Vehicle",
       y="City Mileage")
```

4. Distribution



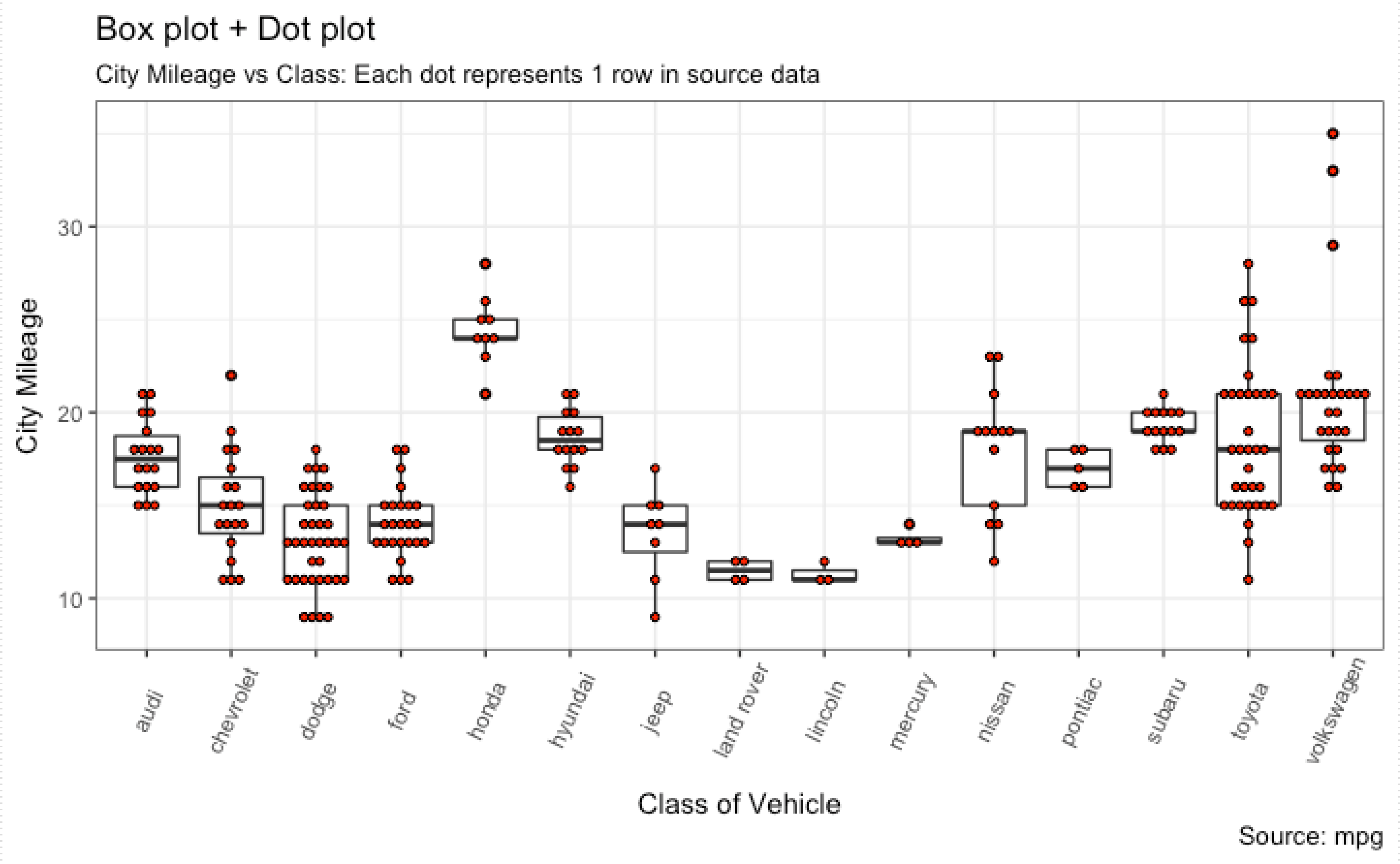
4. Distribution

- On top of the information provided by a **box plot, the dot plot** can provide more clear information in the form of summary statistics by each group. The dots are staggered such that each dot represents one observation. So, in below chart, the number of dots for a given manufacturer will match the number of rows of that manufacturer in source data.

4. Distribution

```
library(ggplot2)
theme_set(theme_bw())
# plot
g <- ggplot(mpg, aes(manufacturer, cty))
g + geom_boxplot() +
  geom_dotplot(binaxis='y',
               stackdir='center',
               dotsize = .5,
               fill="red") +
  theme(axis.text.x = element_text(angle=65, vjust=0.6)) +
  labs(title="Box plot + Dot plot",
       subtitle="City Mileage vs Class: Each dot represents 1 row in source data", caption="Source: mpg", x="Class of Vehicle",
       y="City Mileage")
```

4. Distribution

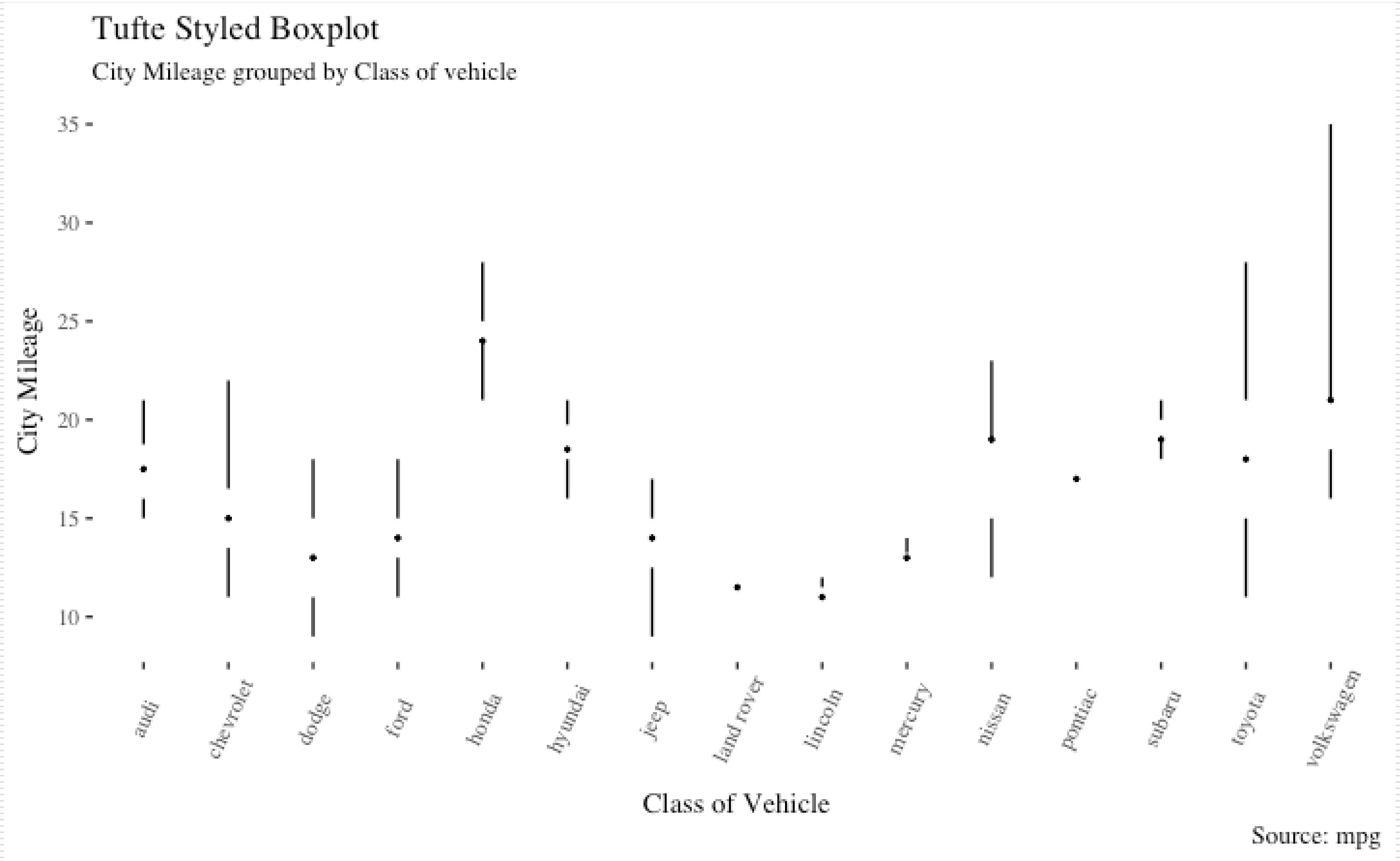


4. Distribution

- **Tufte box plot**, provided by ggthemes package is inspired by the works of Edward Tufte. Tufte’s Box plot is just a box plot made minimal and visually appealing.

```
library(ggthemes)
library(ggplot2)
theme_set(theme_tufte())  # from ggthemes
# plot
g <- ggplot(mpg, aes(manufacturer, cty))
g + geom_tufteboxplot() +
    theme(axis.text.x = element_text(angle=65, vjust=0.6)) +
    labs(title="Tufte Styled Boxplot",
         subtitle="City Mileage grouped by Class of vehicle",
         caption="Source: mpg",
         x="Class of Vehicle", y="City Mileage")
```

4. Distribution

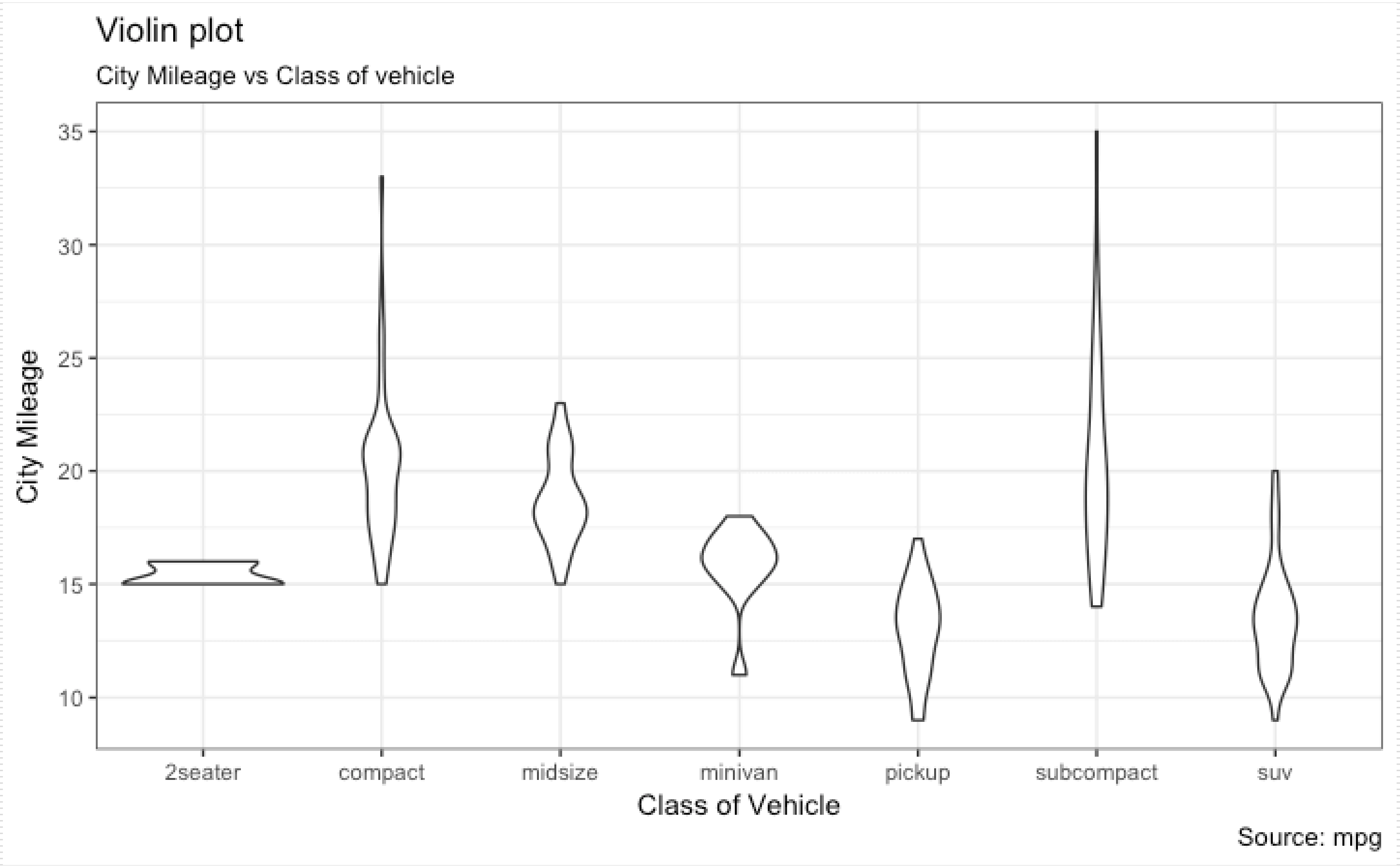


4. Distribution

- A **violin plot** is similar to box plot but shows the density within groups. Not much info provided as in boxplots. It can be drawn using geom_violin().

```
library(ggplot2)
theme_set(theme_bw())
# plot
g <- ggplot(mpg, aes(class, cty))
g + geom_violin() +
  labs(title="Violin plot",
       subtitle="City Mileage vs Class of vehicle",
       caption="Source: mpg",
       x="Class of Vehicle", y="City Mileage")
```

4. Distribution



5. Composition

- **Pie chart**, a classic way of showing the categorical composition of the total population. Is a slightly tricky to implement in ggplot2 using the coord_polar().

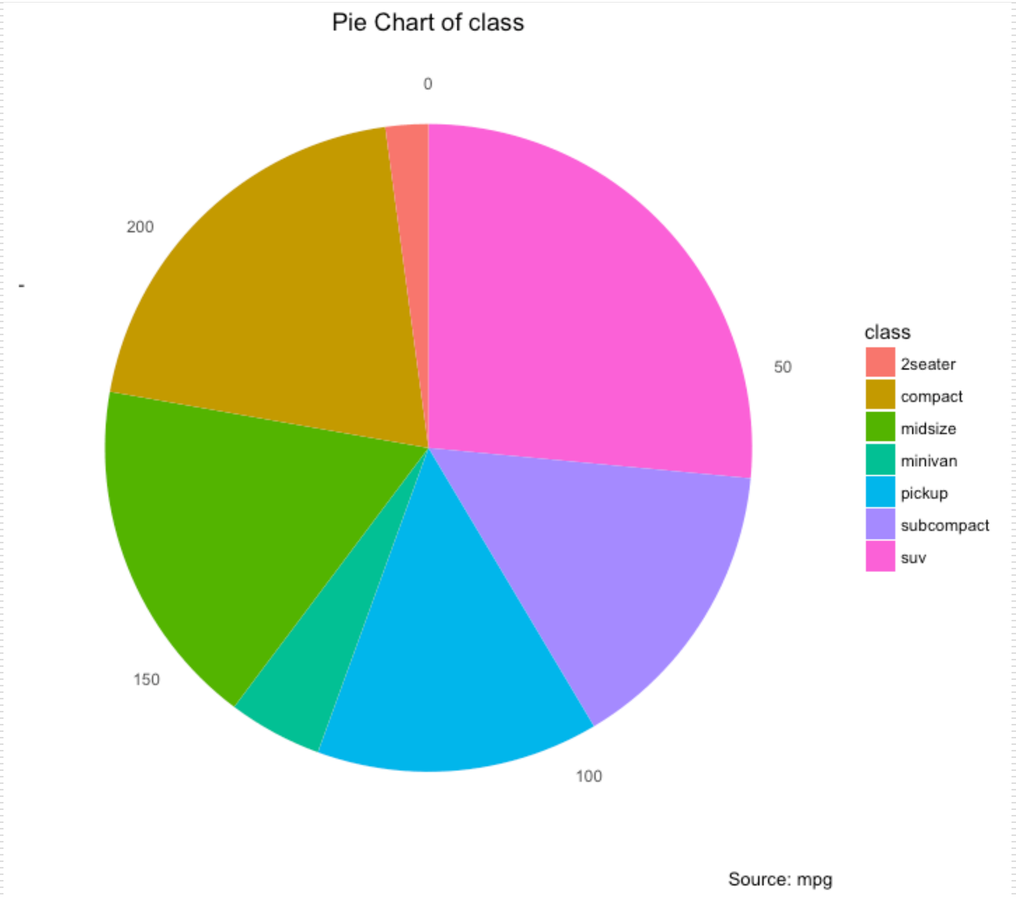
5. Composition

```
library(ggplot2)
theme_set(theme_classic())

# Source: Frequency table
df <- as.data.frame(table(mpg$class))
colnames(df) <- c("class", "freq")
pie <- ggplot(df, aes(x = "", y=freq, fill = factor(class))) +
  geom_bar(width = 1, stat = "identity") +
  theme(axis.line = element_blank(),
        plot.title = element_text(hjust=0.5)) +
  labs(fill="class",
       x=NULL,
       y=NULL,
       title="Pie Chart of class",
       caption="Source: mpg")
pie + coord_polar(theta = "y", start=0)
```

5. Composition

```
# Source: Categorical variable.
# mpg$class
pie <- ggplot(mpg, aes(x = "", fill = factor(class))) +
  geom_bar(width = 1) +
  theme(axis.line = element_blank(),
        plot.title = element_text(hjust=0.5)) +
  labs(fill="class",
       x=NULL,
       y=NULL,
       title="Pie Chart of class",
       caption="Source: mpg")

pie + coord_polar(theta = "y", start=0)
```

5. Composition

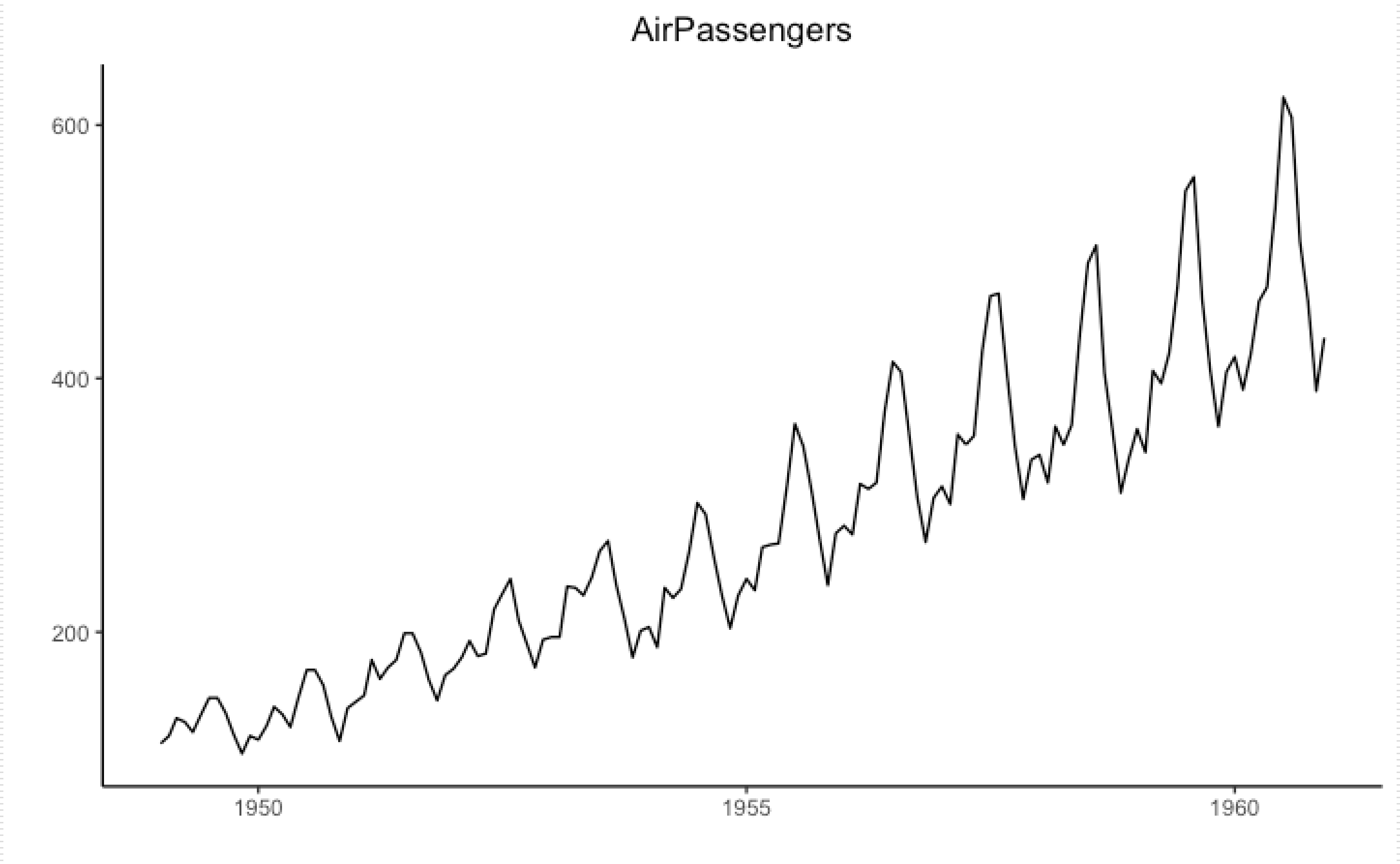


6. Change

- **Time Series Plot** From a Time Series Object.

```
## From Timeseries object (ts)
#install ggfortify & zoo
library(ggplot2)
library(ggfortify)
theme_set(theme_classic())
# Plot
autoplot(AirPassengers) +
  labs(title="AirPassengers") +
  theme(plot.title = element_text(hjust=0.5))
```

6. Change

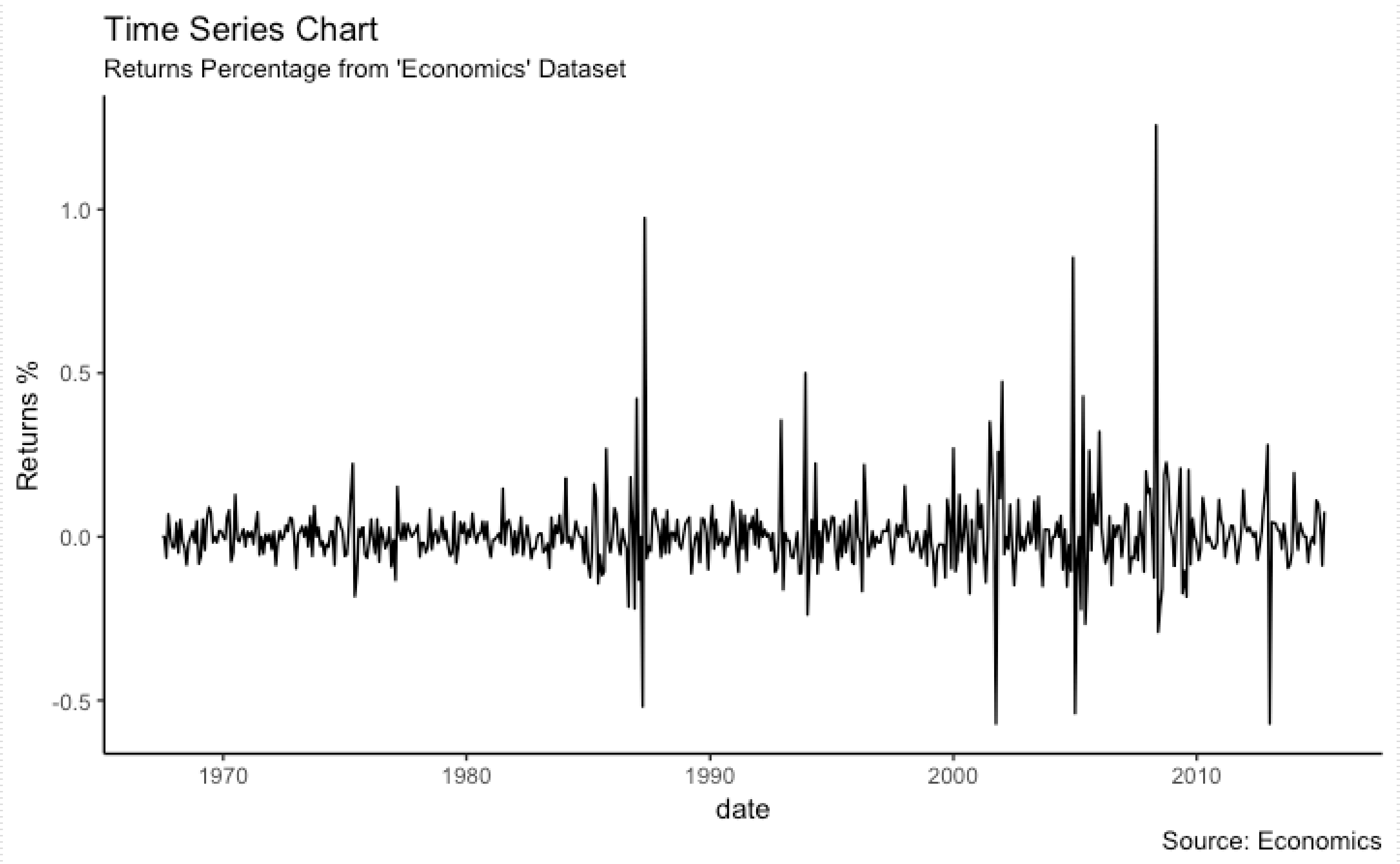


6. Change

- Using geom_line(), **a time series (or line chart)** can be drawn from a data.frame as well. The X axis breaks are generated by default. In below example, the breaks are formed once every 10 years.

```
#Default X Axis Labels
library(ggplot2)
theme_set(theme_classic())

# Allow Default X Axis Labels
ggplot(economics, aes(x=date)) +
  geom_line(aes(y=returns_perc)) +
  labs(title="Time Series Chart",
       subtitle="Returns Percentage from 'Economics' Dataset",
       caption="Source: Economics",
       y="Returns %")
```

6. Change


Time Series Chart
Returns Percentage from 'Economics' Dataset
Returns %
1.0
0.5
0.0
-0.5
1970
1980
1990
2000
2010
date
Source: Economics

6. Change

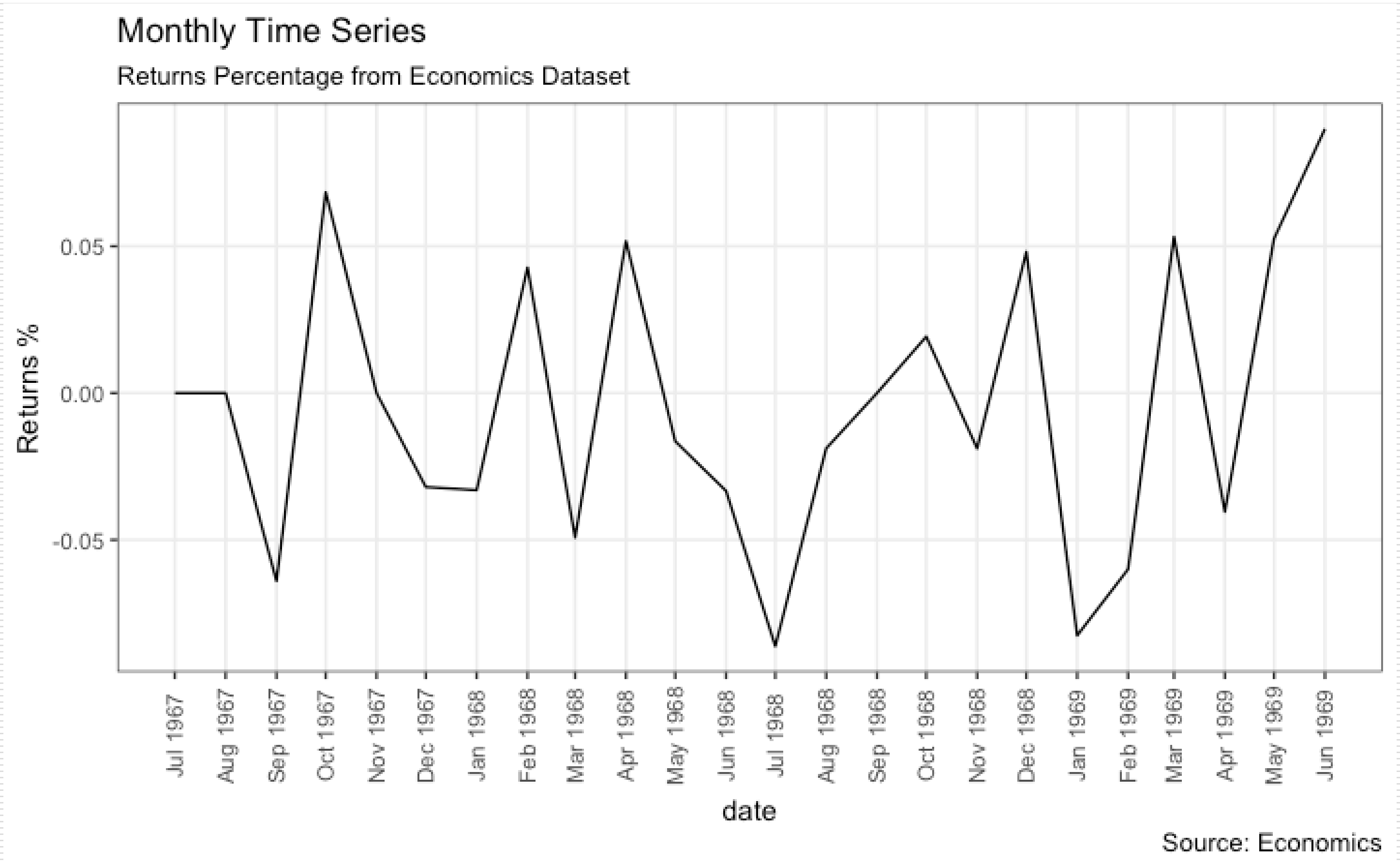
- If you want to set your own time intervals (breaks) in X axis, you need to set the breaks and labels using scale_x_date().

```
library(ggplot2)
library(lubridate)
theme_set(theme_bw())
economics_m <- economics[1:24, ]
# labels and breaks for X axis text
lbls <- paste0(month.abb[lubridate::month(economics_m$date)], " ", 
lubridate::year(economics_m$date))
brks <- economics_m$date
```

6. Change

```
# plot
ggplot(economics_m, aes(x=date)) +
  geom_line(aes(y=returns_perc)) +
  labs(title="Monthly Time Series",
       subtitle="Returns Percentage from Economics Dataset",
       caption="Source: Economics",
       y="Returns %") +  # title and caption
  scale_x_date(labels = lbls,
               breaks = brks) +  # change to monthly ticks and labels
  theme(axis.text.x = element_text(angle = 90, vjust=0.5),  # rotate x axis text
        panel.grid.minor = element_blank())  # turn off minor grid
```

6. Change



Monthly Time Series
Returns Percentage from Economics Dataset
Returns %
0.05
0.00
-0.05
Jul 1967
Aug 1967
Sep 1967
Oct 1967
Nov 1967
Dec 1967
Jan 1968
Feb 1968
Mar 1968
Apr 1968
May 1968
Jun 1968
Jul 1968
Aug 1968
Sep 1968
Oct 1968
Nov 1968
Dec 1968
Jan 1969
Feb 1969
Mar 1969
Apr 1969
May 1969
Jun 1969
date
Source: Economics

6. Change

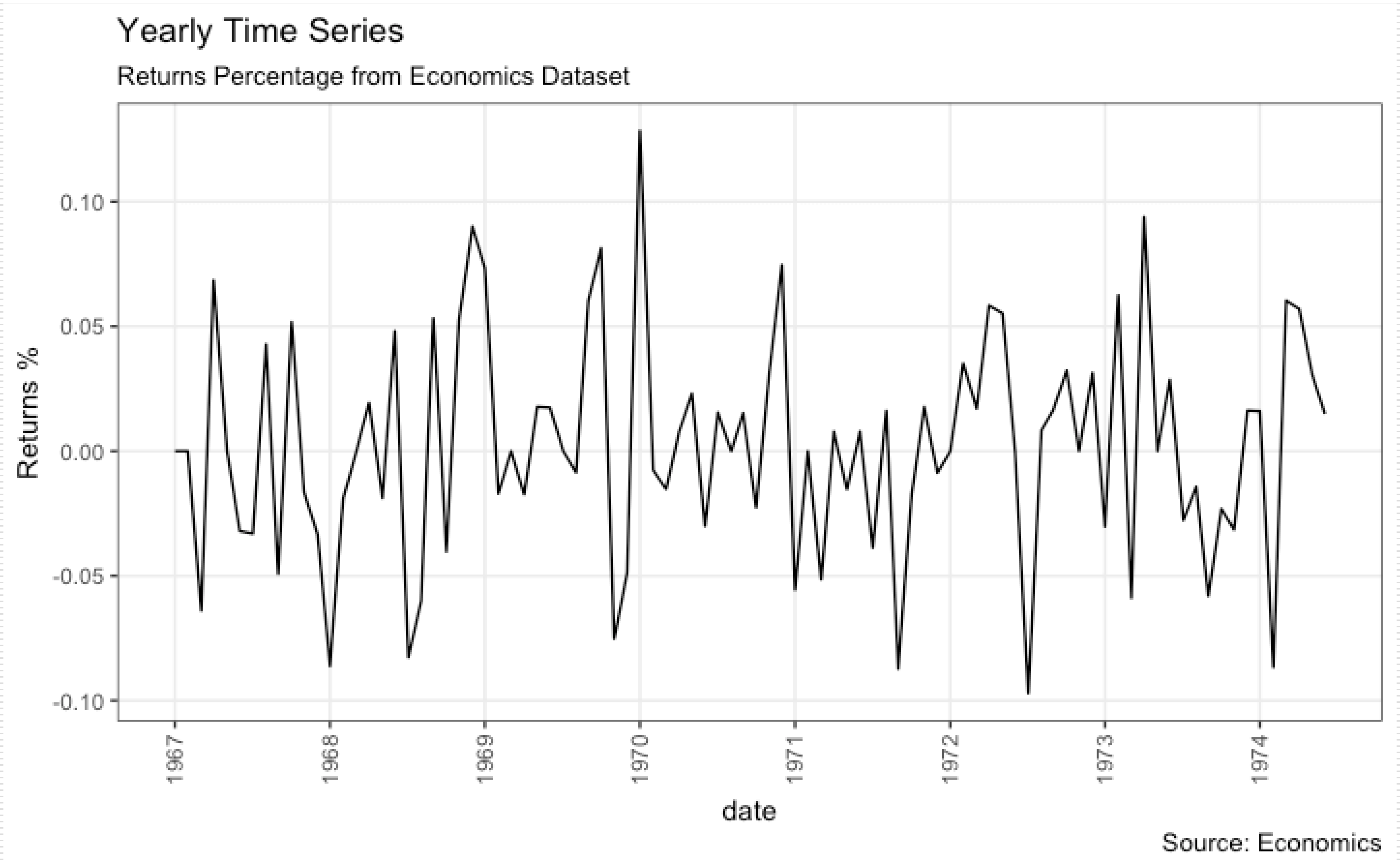
- Time Series Plot For a *Yearly* Time Series

```
library(ggplot2)
library(lubridate)
theme_set(theme_bw())

economics_y <- economics[1:90, ]

# labels and breaks for X axis text
brks <- economics_y$date[seq(1, length(economics_y$date), 12)]
lbls <- lubridate::year(brks)
```

6. Change

```
# plot
ggplot(economics_y, aes(x=date)) +
  geom_line(aes(y=returns_perc)) +
  labs(title="Yearly Time Series",
       subtitle="Returns Percentage from Economics Dataset",
       caption="Source: Economics",
       y="Returns %") +  # title and caption
  scale_x_date(labels = lbls,
               breaks = brks) +  # change to yearly ticks and labels
  theme(axis.text.x = element_text(angle = 90, vjust=0.5),  # rotate x axis text
        panel.grid.minor = element_blank())  # turn off minor grid
```

6. Change



Yearly Time Series
Returns Percentage from Economics Dataset
Returns %
0.10
0.05
0.00
-0.05
-0.10
1967
1968
1969
1970
1971
1972
1973
1974
date
Source: Economics

6. Change

- In this example, I construct the ggplot from a **long data format**. That means, the column names and respective values of all the columns are stacked in just 2 variables (variable and value respectively). If you were to convert this data to wide format, it would look like the economics dataset.

- In below example, the geom_line is drawn for value column and the aes(col) is set to variable. This way, with just one call to geom_line, **multiple colored lines are drawn**, one each for each unique value in variable column. The scale_x_date() changes the X axis breaks and labels, and scale_color_manual changes the color of the lines.

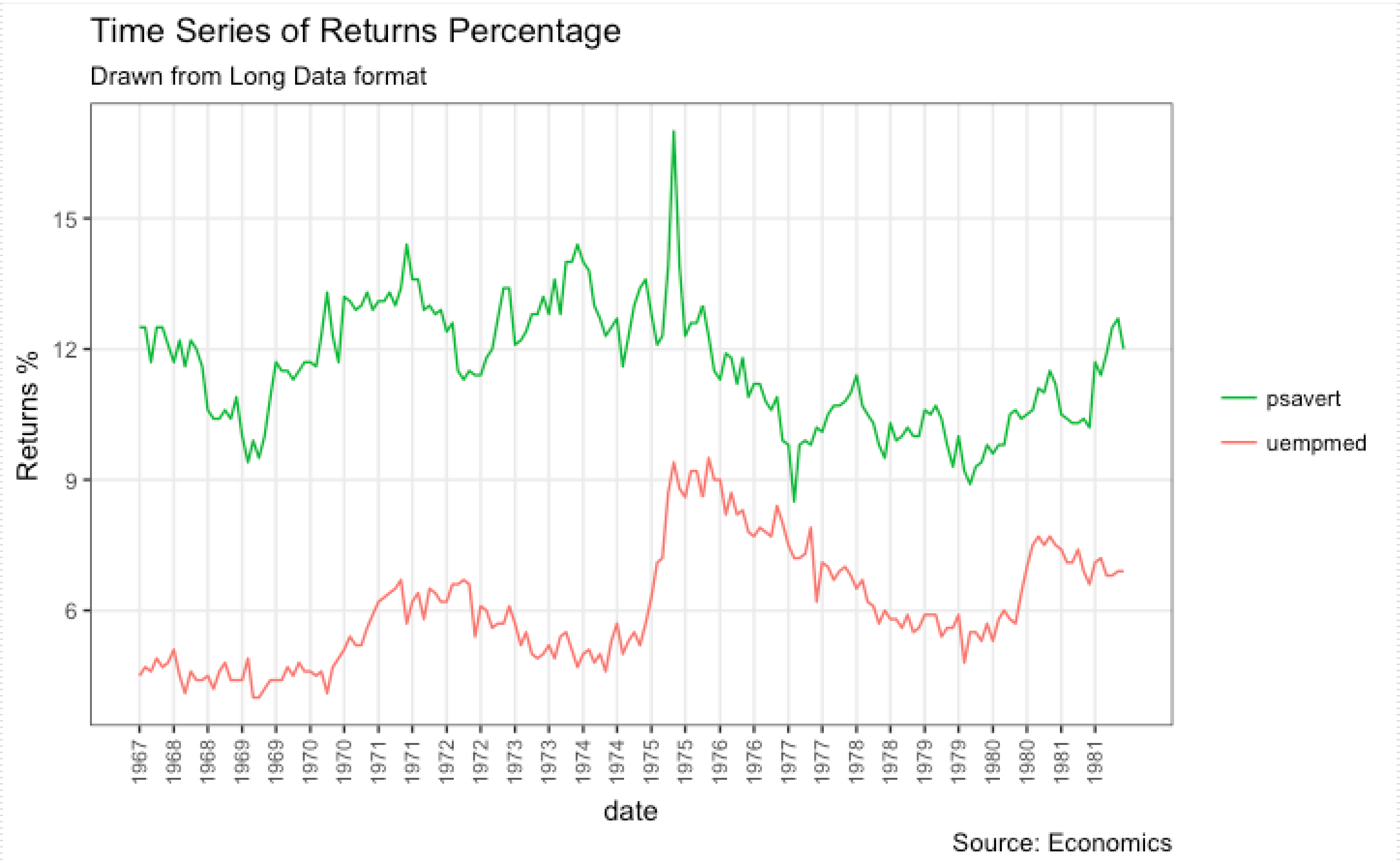
6. Change

```
data(economics_long, package = "ggplot2")
head(economics_long)
#>         date variable value      value01
#>       <date>   <fctr> <dbl>        <dbl>
#> 1 1967-07-01      pce 507.4 0.0000000000
#> 2 1967-08-01      pce 510.5 0.0002660008
#> 3 1967-09-01      pce 516.3 0.0007636797
#> 4 1967-10-01      pce 512.9 0.0004719369
#> 5 1967-11-01      pce 518.1 0.0009181318
#> 6 1967-12-01      pce 525.8 0.0015788435
library(ggplot2)
library(lubridate)
theme_set(theme_bw())

df <- economics_long[economics_long$variable %in% c("psavert", "uempmed"), ]
df <- df[lubridate::year(df$date) %in% c(1967:1981), ]
```

6. Change

```
# labels and breaks for X axis text
brks <- df$date[seq(1, length(df$date), 12)]
lbls <- lubridate::year(brks)
# plot
ggplot(df, aes(x=date)) +
  geom_line(aes(y=value, col=variable)) +
  labs(title="Time Series of Returns Percentage",
       subtitle="Drawn from Long Data format",
       caption="Source: Economics",
       y="Returns %",
       color=NULL) +  # title and caption
  scale_x_date(labels = lbls, breaks = brks) +  # change to yearly ticks and labels
  scale_color_manual(labels = c("psavert", "uempmed"),
                     values = c("psavert"="#00ba38", "uempmed"="#f8766d")) +  # line color
  theme(axis.text.x = element_text(angle = 90, vjust=0.5, size = 8),  # rotate x axis text
        panel.grid.minor = element_blank())  # turn off minor grid
```

6. Change


Time Series of Returns Percentage
Drawn from Long Data format
Returns %
15
12
9
6
1967
1968
1968
1969
1969
1970
1970
1971
1971
1972
1972
1973
1973
1974
1974
1975
1975
1976
1976
1977
1977
1978
1978
1979
1979
1980
1980
1981
1981
date
psavert
uempmed
Source: Economics

6. Change

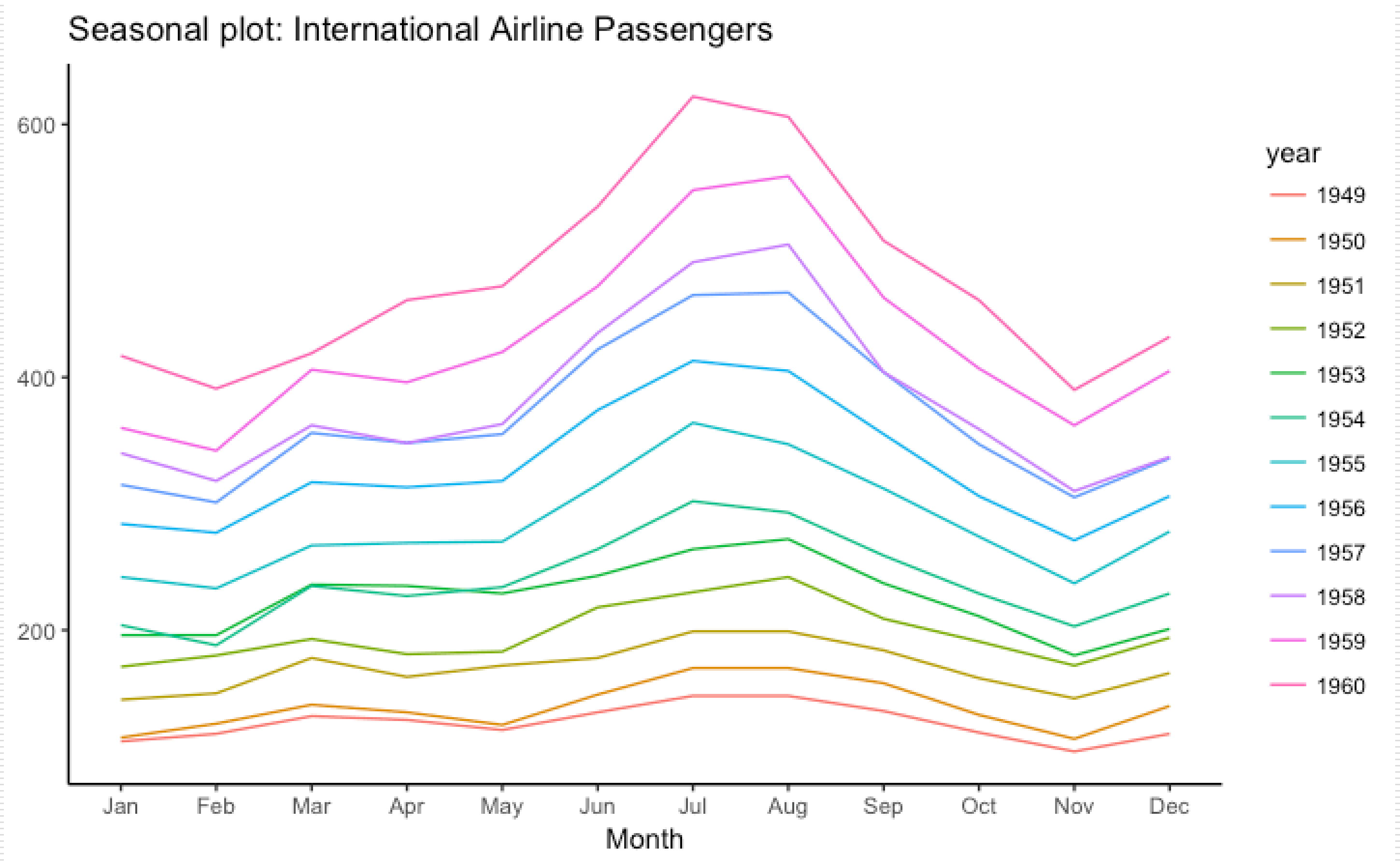
- If you are working with a time series object of class ts or xts, you can view the seasonal fluctuations through a **seasonal plot** drawn using forecast::ggseasonplot. Below is an example using the native AirPassengers and nottem time series.

- You can see the traffic increase in air passengers over the years along with the repetitive seasonal patterns in traffic. Whereas Nottingham does not show an increase in overall temperatures over the years, but they definitely follow a seasonal pattern.

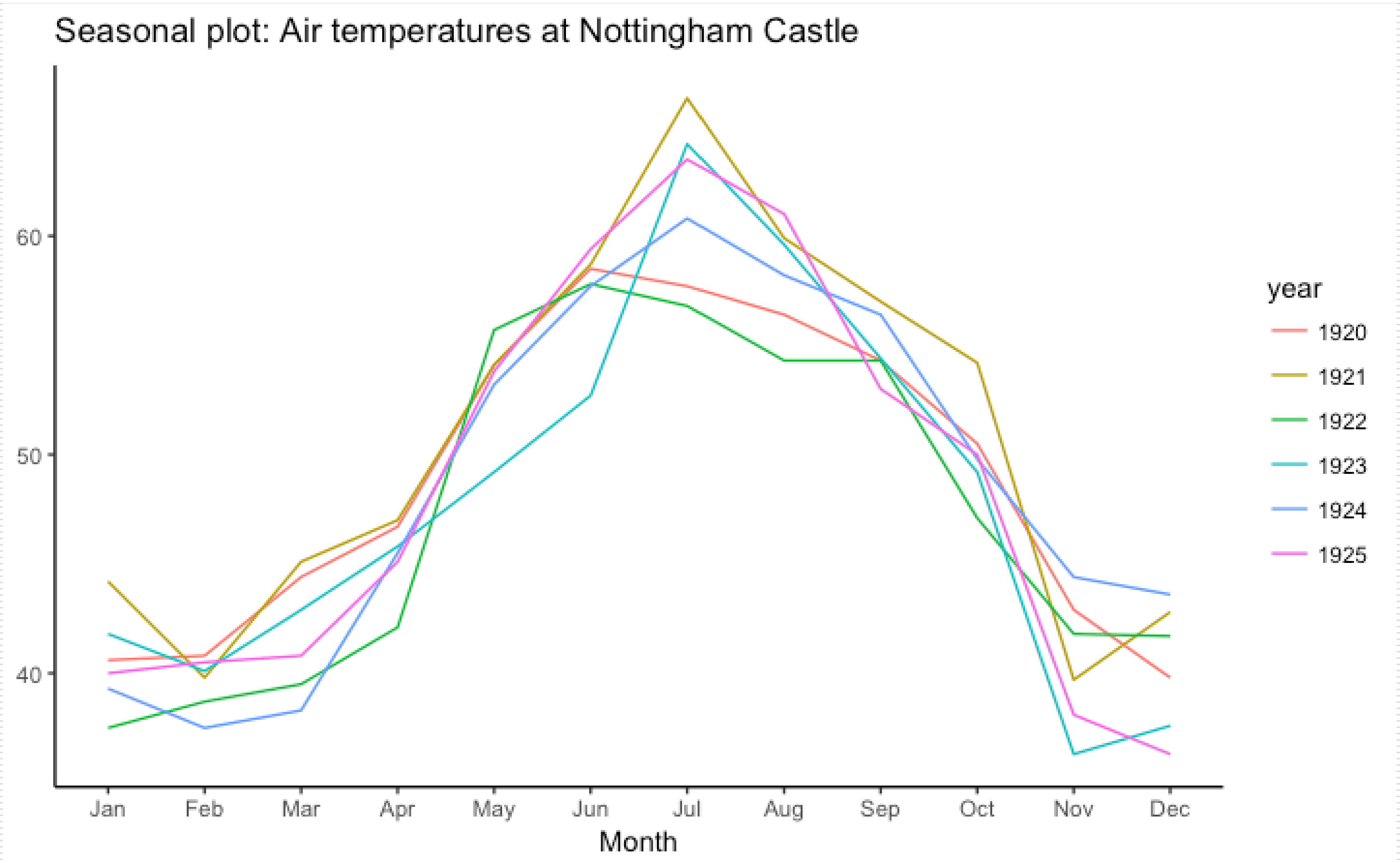
6. Change

```
#install forecast
library(ggplot2)
library(forecast)
theme_set(theme_classic())
# Subset data
nottem_small <- window(nottem, start=c(1920, 1), end=c(1925, 12))  # subset a smaller timewindow
# Plot
ggseasonplot(AirPassengers) + labs(title="Seasonal plot: International Airline Passengers")
ggseasonplot(nottem_small) + labs(title="Seasonal plot: Air temperatures at Nottingham Castle")
```

6. Change


Seasonal plot: International Airline Passengers
600
400
200
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
year
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960

6. Change


Seasonal plot: Air temperatures at Nottingham Castle
60
50
40
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
year
1920
1921
1922
1923
1924
1925

7. Groups

- It is possible to show the distinct **clusters** or groups using geom_encircle(). If the dataset has multiple weak features, you can compute the principal components and draw a scatterplot using PC1 and PC2 as X and Y axis.
- The geom_encircle() can be used to encircle the desired groups. The only thing to note is the data argument to geom_circle(). You need to provide a subsetted dataframe that contains only the observations (rows) that belong to the group as the data argument.

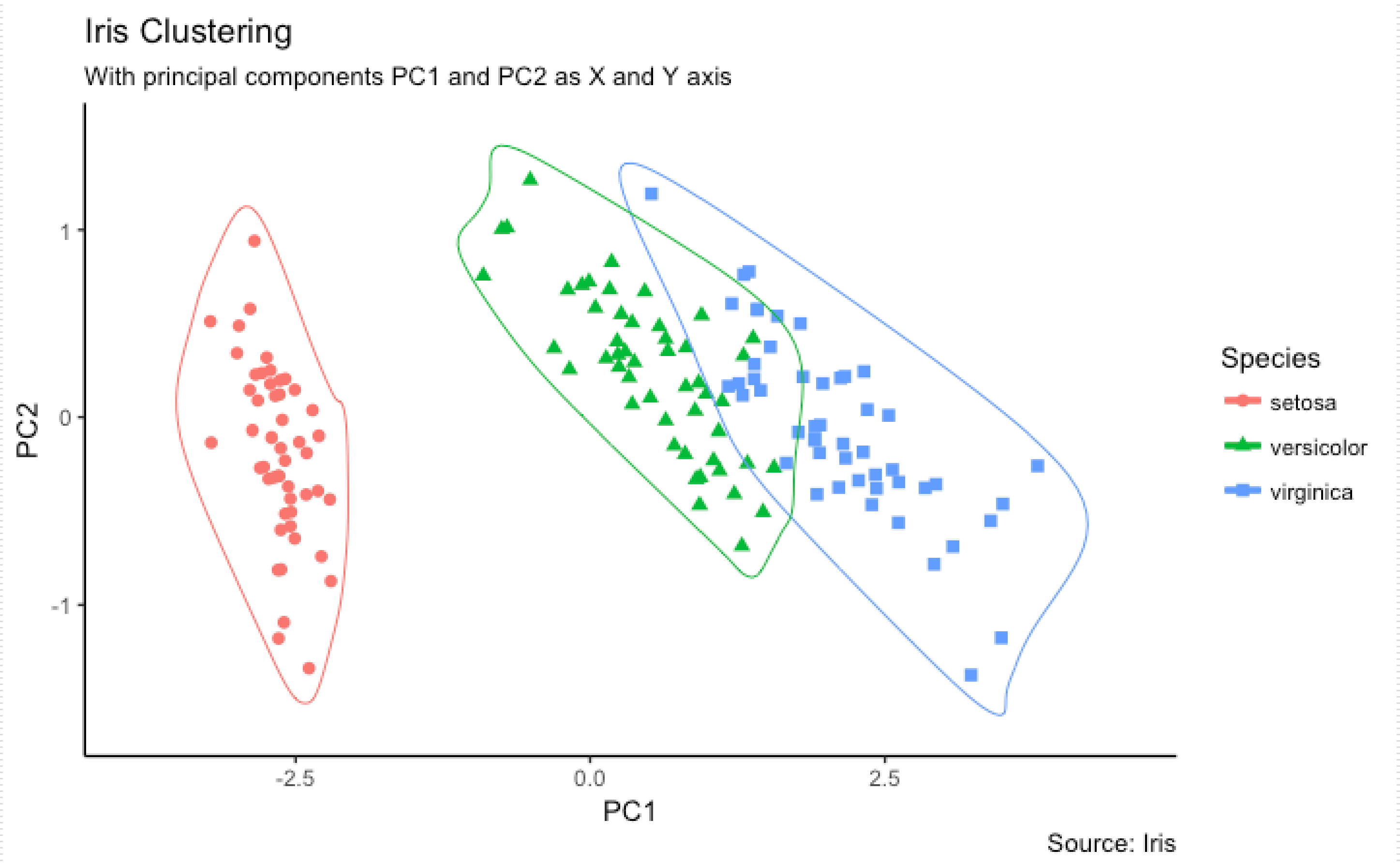
7. Groups

```
# devtools::install_github("hrbrmstr/ggalt")
library(ggplot2)
library(ggalt)
library(ggfortify)
theme_set(theme_classic())

# Compute data with principal components ------------------
df <- iris[c(1, 2, 3, 4)]
pca_mod <- prcomp(df)  # compute principal components

# Data frame of principal components ----------------------
df_pc <- data.frame(pca_mod$x, Species=iris$Species)  # dataframe of principal
components
df_pc_vir <- df_pc[df_pc$Species == "virginica", ]  # df for 'virginica'
df_pc_set <- df_pc[df_pc$Species == "setosa", ]  # df for 'setosa'
df_pc_ver <- df_pc[df_pc$Species == "versicolor", ]  # df for 'versicolor'
```

7. Groups

```
# Plot ----------------------------------------------------
ggplot(df_pc, aes(PC1, PC2, col=Species)) +
  geom_point(aes(shape=Species), size=2) +   # draw points
  labs(title="Iris Clustering",
       subtitle="With principal components PC1 and PC2 as X and Y axis",
       caption="Source: Iris") +
  coord_cartesian(xlim = 1.2 * c(min(df_pc$PC1), max(df_pc$PC1)),
                  ylim = 1.2 * c(min(df_pc$PC2), max(df_pc$PC2))) +   # change axis limits
  geom_encircle(data = df_pc_vir, aes(x=PC1, y=PC2)) +   # draw circles
  geom_encircle(data = df_pc_set, aes(x=PC1, y=PC2)) +
  geom_encircle(data = df_pc_ver, aes(x=PC1, y=PC2))
```

7. Groups


Iris Clustering
With principal components PC1 and PC2 as X and Y axis
PC2
1
0
-1
-2.5
0.0
2.5
PC1
Species
setosa
versicolor
virginica
Source: Iris

8. Spatial

```
# Don't bother installing if you already have them
install.packages(c("ggplot2", "devtools", "dplyr", "stringr"))
# some standard map packages.
install.packages(c("maps", "mapdata"))
# the github version of ggmap, which recently pulled in a
small fix I had for a bug
devtools::install_github("dkahle/ggmap")

library(ggplot2)
library(ggmap)
library(maps)
library(mapdata)
```

8. Spatial

- The maps package contains a lot of outlines of continents, countries, states, and counties that have been with R for a long time.
- The mapdata package contains a few more, higher-resolution outlines.
- The maps package comes with a plotting function, but, we will opt to use ggplot2 to plot the maps in the maps package.
- Recall that ggplot2 operates on data frames. Therefore we need some way to translate the maps data into a data frame format the ggplot can use.

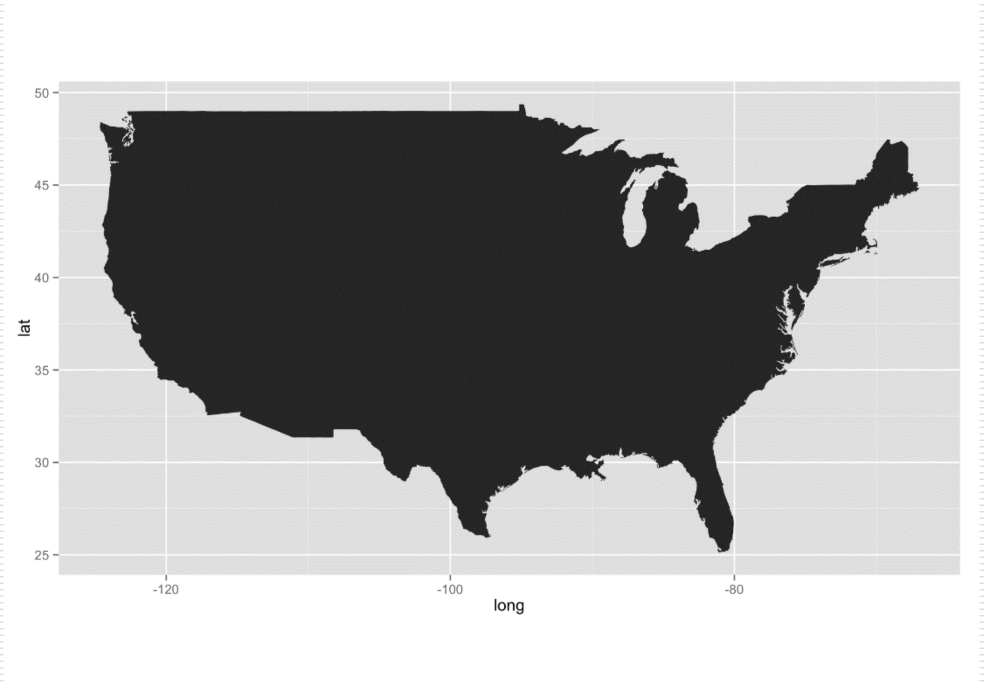
8. Spatial

```
usa <- map_data("usa")
```

- Info:
  - long is longitude. Things to the west of the prime meridian are negative.
  - lat is latitude.
  - order. This just shows in which order ggplot should "connect the dots"
  - region and subregion tell what region or subregion a set of points surrounds.
  - group. This is very important! ggplot2's functions can take a group argument which controls (amongst other things) whether adjacent points should be connected by lines. If they are in the same group, then they get connected, but if they are in different groups then they don't.

8. Spatial

```
ggplot() + geom_polygon(data = usa, aes(x=long, y = lat, group
= group)) + coord_fixed(1.3)
```



8. Spatial

- What is this coord_fixed()?
- This is very important when drawing maps.
- It fixes the relationship between one unit in the y direction and one unit in the x direction.
- Then, even if you change the outer dimensions of the plot (i.e. by changing the window size or the size of the pdf file you are saving it to (in ggsave for example)), the aspect ratio remains unchanged.
- In the above case, I decided that if every y unit was 1.3 times longer than an x unit, then the plot came out looking good.

8. Spatial

- Play with aesthetics:

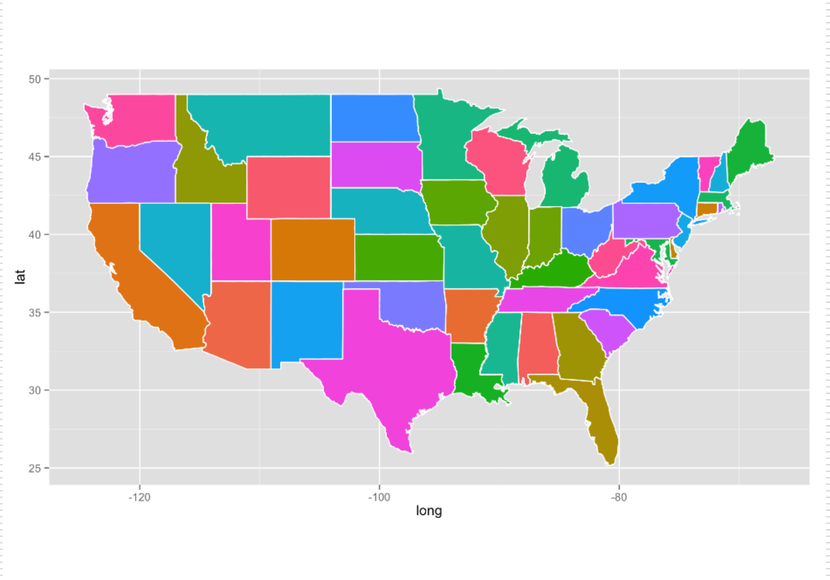
```
ggplot() + geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill =
NA, color = "red") + coord_fixed(1.3)

ggplot() + geom_polygon(data = usa, aes(x=long, y = lat, group = group), fill =
"violet", color = "blue") + coord_fixed(1.3)

# add points in specific places
labs <- data.frame( long = c(-122.064873, -122.306417), lat = c(36.951968,
47.644855), names = c("SWFSC-FED", "NWFSC"), stringsAsFactors = FALSE)
gg1 <- ggplot() +  geom_polygon(data = usa, aes(x=long, y = lat, group =
group), fill = "violet", color = "blue") +  coord_fixed(1.3)
gg1 + geom_point(data = labs, aes(x = long, y = lat), color = "yellow", size =
4)
```

8. Spatial

states <- map_data("state")

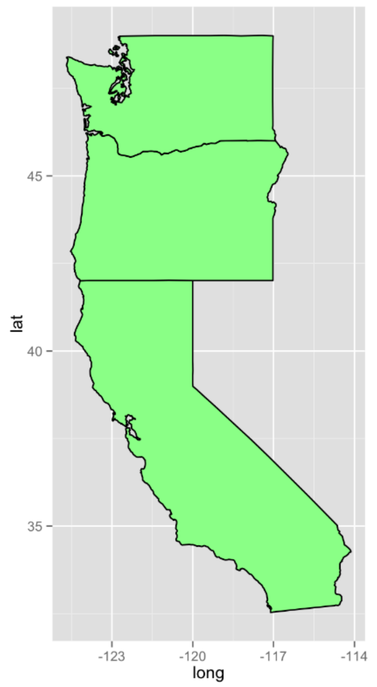
ggplot(data = states) + geom_polygon(aes(x = long, y = lat, fill = region, group = group), color = "white") + coord_fixed(1.3) +

guides(fill=FALSE) # do this to leave off the color legend



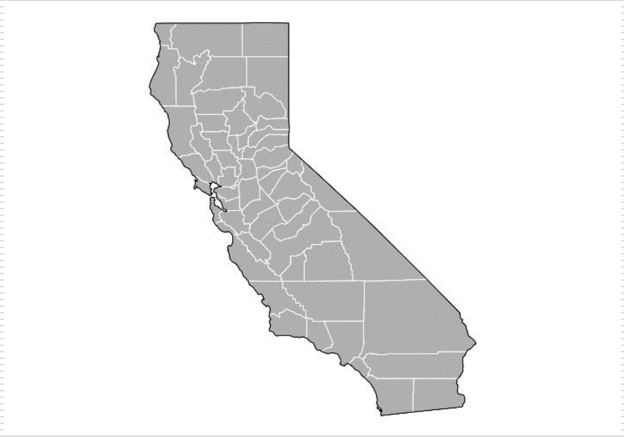
8. Spatial

west_coast <- subset(states, region %in% c("california", "oregon", "washington")) ggplot(data = west_coast) + geom_polygon(aes(x = long, y = lat, group = group), fill = "palegreen", color = "black") +
coord_fixed(1.3)



8. Spatial

```
ca_df <- subset(states, region == "california")
counties <- map_data("county")
ca_county <- subset(counties, region == "california")
ca_base <- ggplot(data = ca_df, mapping = aes(x = long, y = lat, group =
group)) + coord_fixed(1.3) + geom_polygon(color = "black", fill = "gray")
ca_base + theme_nothing() + geom_polygon(data = ca_county, fill = NA,
color = "white") + geom_polygon(color = "black", fill = NA)
```



8. Spatial

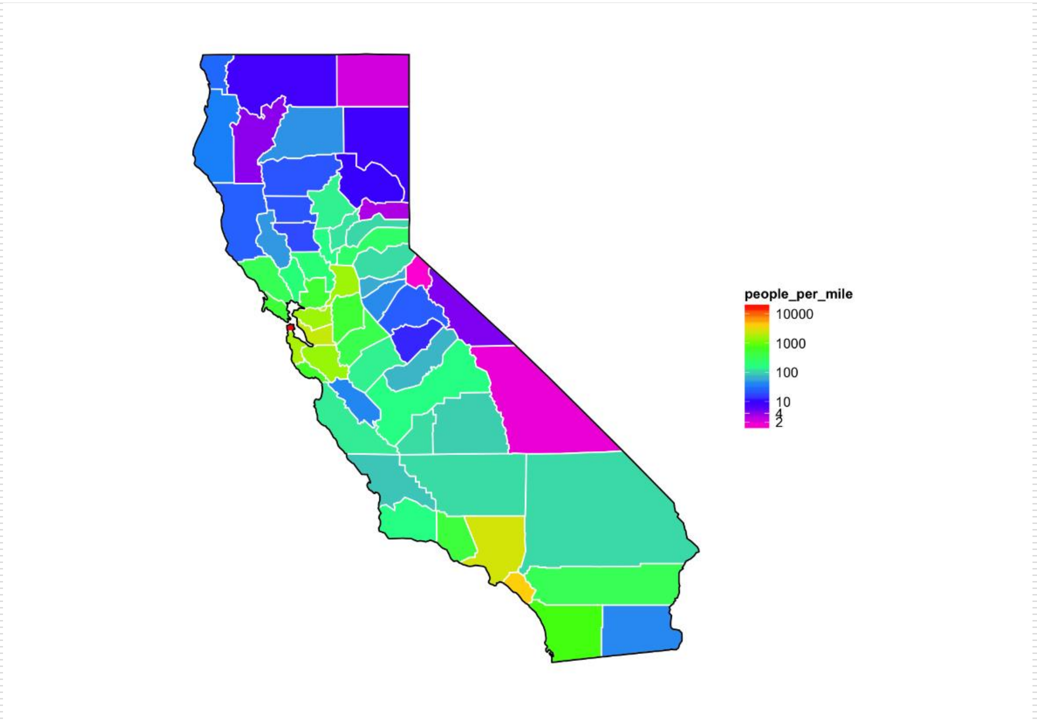
- **Get some facts about the counties**
  - The above is pretty cool, but it seems like it would be a lot cooler if we could plot some information about those counties.
  - Now I can go to http://www.california-demographics.com/counties_by_population and copy the table into area_pop_ca.csv. In the class web site you will find the data.
  - pop_and_area<-fread("area_pop_ca.csv")
  - pop_and_area$subregion<-tolower(pop_and_area$subregion)
  - cacopa <- inner_join(ca_county, pop_and_area, by = "subregion")
  - cacopa$area<-as.numeric(cacopa$area)
  - cacopa$population<-as.numeric(cacopa$population)
  - cacopa$people_per_mile <- cacopa$population / cacopa$area

8. Spatial

```
# prepare to drop the axes and ticks but leave the guides and legends
# We can't just throw down a theme_nothing()!
ditch_the_axes <- theme(
  axis.text = element_blank(),
  axis.line = element_blank(),
  axis.ticks = element_blank(),
  panel.border = element_blank(),
  panel.grid = element_blank(),
  axis.title = element_blank()
  )
elbow_room1 <- ca_base +
      geom_polygon(data = cacopa, aes(fill = people_per_mile), color =
"white") +
      geom_polygon(color = "black", fill = NA) +
      theme_bw() + ditch_the_axes
```

8. Spatial

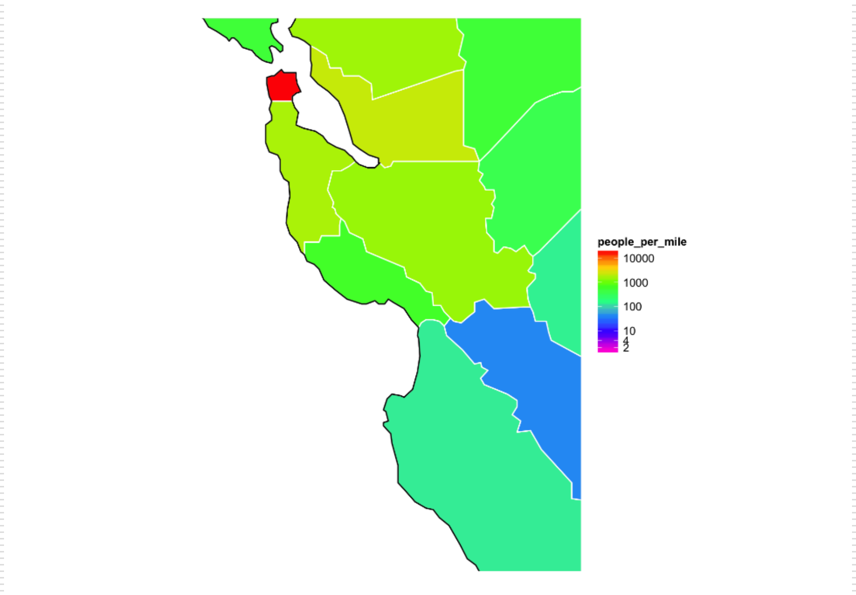
```
eb2 <- elbow_room1 + scale_fill_gradientn(colours = rev(rainbow(7)),
                       breaks = c(2, 4, 10, 100, 1000, 10000),
                       trans = "log10")
eb2
```



8. Spatial

Zoom in

```
eb2 + coord_fixed(xlim = c(-123, -121.0),  ylim = c(36, 38), ratio = 1.3)
```


people_per_mile
10000
1000
100
10
4
2