Data Taming R reminder sheet

Bill S. Preston Esq.

2024-08-01

1 Setup

1.1 Initialising knitr

- This first code chunk will execute, but not display in the output because of the option `include=FALSE` in the definition of the code chunk.

`knitr` general options:

- `message=FALSE` when setting the `knitr` general options to suppress messages when loading the packages later in the file.
- `results=FALSE` to suppress output. (Change this to `results=TRUE` to see the output.)

1.2 Loading packages

```
library(tidyverse)
library(inspectdf)
library(caret)
library(moments)
library(tidymodels)
library(modelr)
library(ISLR)
library(car)
```

2 Loading data

- Use `<-` to assign new datasets and variables

2.1 Data already in the library

```
data("mpg")
mpg1<-mpg
```

2.2 Read in a csv file

```
pop1<- read_csv("population.csv")
```

3 Displaying data

3.1 Displaying data frames and tibbles

```
mpg1 # Prints the first 10 lines
print(n=10, mpg) # Also prints the first 10 lines
head(mpg1, 9) # Also prints the first 9 lines
tail(mpg1, 8) # Prints the final 8 lines
View(mpg1) # This command will display data in a new window
```

3.2 To show the number of rows and columns of the data

- `dim()` gives a list of the form `[\# rows, \# columns]`

```
dim(mpg)
```

- To just find the number of rows use `nrow()`

```
nrow(mpg)
```

- To just find the number of columns use `ncol()`

```
ncol(mpg)
```

3.3 To access a particular column

```
mpg1$cyl
```

4 Extracting data

4.1 Extracting columns

```
select(mpg, cyl:fl)
```

4.2 Extracting specific row numbers

```
mpg[5,]
```

4.3 Extracting specific values

4.3.1 Just keeping the value

```
mpg[5,]$trans
class(mpg[5,]$trans)
```

4.3.2 Putting the value into its own new tibble

```
mpg[5,"trans"]
class(mpg[5,"trans"])
```

4.4 Extracting rows that match a TRUE/FALSE condition

```
filter(mpg, displ==3.1)
filter(mpg, between(mpg$displ,2.8,3.1))
```

5 Missing data

5.1 To check if there are any missing values in any columns

```
inspect_na(starwars)
```

5.2 Finding missing values in a specific column

- This returns a boolean list of TRUE/FALSE indicating the rows with missing data.
- This can be combined with the filter() command

```
is.na(starwars$species)
```

6 Manipulating data

6.1 Sorting a column

```
arrange(mpg, displ)
```

6.2 `mutate()` To add, change or remove columns

- Add column to right of dataset
  - `mutate(dataset, new_column_name = value)`

```
mutate(mpg1, IDnum=c(1:234))
mutate(mpg1, cty_hwy_avg=(cty -hwy)/2)
```

- Delete a column
  - `mutate(dataset, existing_column_name = NULL)`

```
mutate(mpg1, model=NULL)
```

- Change a column
  - `mutate(dataset, existing_column_name = value)`

```
mutate(mpg, displ=displ*10)
```

6.3 `rename()` to rename a column

- Use the syntax `rename(dataset, new_column_name=old_column_name)`

```
rename(mpg, displacement=displ)
```

6.4 `relocate()` to move a column

- Move a column “before” (to the left) of another column

```
relocate(mpg, "cyl", .before = model)
```

- Move a column “after” (to the right) of another column

```
relocate(mpg, "cyl", .after = cty)
```

6.5 To concatenate tibbles (glue them together)

- Use `bind_cols()`.

```
bind_cols(
  mpg["trans"],
  mpg["cty"]
  )
```

- Note that everything you are binding together should be a tibble, with unique variable names, otherwise the variable names in the resulting tibble will be nonsense.

```
bind_cols(
  1:50,
  1:50
  )
```

6.6 Grouping rows

- Group a set of rows together based on the values in one of the columns. Eg. this will group all the cars together by their number of cylinders.
- This can be used with the `summarise()` command to computer statistics for each group.

```
group_by(mpg,cyl)
```

6.7 Change data types

- Convert to factor (nominal categorical variable)

```
mpg1$cyl<-as.factor(mpg1$cyl)
```

- Convert to ordered factor (ordinal categorical variable)

```
mpg1$cyl<-as.ordered(mpg1$cyl)
#This next bit of code will change the order of the levels
mpg1$cyl<-factor(mpg1$cyl, levels=c("5", "8", "4", "6"), ordered=TRUE)
```

- Convert to numerical variable

```
mpg1$cyl<-as.numeric(mpg1$cyl)
```

- Convert to integer variable

```
mpg1$cyl<-as.integer(mpg1$cyl)
```

- Convert to character string

```
mpg1$cyl<-as.character(mpg1$cyl)
```

- Convert to logical/Boolean variable
  - First need a column of TRUE/FALSE or 1/0

```
mpg2<-mutate(mpg1, tf="TRUE")
mpg2$tf<-as.logical(mpg2$tf)
```

- Convert to date object

    - Using `lubridate` package commands `ymd` or `dmy`

```
mpg2<-mutate(mpg1,date="2025-10-05")
mpg2$date<-ymd(mpg2$date)
mpg2<-mutate(mpg1,date="05-10-2025")
mpg2$date<-dmy(mpg2$date)
```

6.8 Rename entries in column

- Using `fct_recode(mpg$drv, "new1"="old1", "new2"="old2", "new3"="old3")`
- Only works when column is a factor or character string
- Also converts column to `factor` type

```
fct_recode(mpg$drv, "front"="f", "4x4"="4", "rear"="r")
```

7 Control structures

7.1 Decisions

The `ifelse` command has syntax: `ifelse(condition, return if true, return if false)`

```
ifelse(10==0,1,0)
ifelse(mpg$cyl==4,mpg$cyl,-99)
```

This can be combined with `mutate()` to selectively modify tibbles:

```
mutate(mpg, take4=ifelse(mpg$cyl==4,mpg$cyl,-99))
```

7.2 Loops

- R is a vectorised progamming language, and so it is not optimised for loops. Therefore we do not use loops in this course, and you must find another way to achieve your goal.

8 A sequence of numbers

```
1:50
```

8.1 A sequence of numbers with step size

To define a sequence from a to b with steps of size s, use: `seq(a,b,s)`

```
seq(5,8,0.05)
```

9 Random sampling

9.1 Random sampling from a tibble

- Use `sample_n()`
- Uniformly at random choose 20 rows with replacement

```
sample_n(mpg, 20, replace=TRUE)
```

- Uniformly at random choose 20 rows without replacement

```
sample_n(mpg, 20, replace=FALSE)
```

9.2 Generating a list of random numbers

- Use `sample()`
- Generate 7 random integers from 1 to 50 with replacement

```
sample(1:50, 7, replace = TRUE)
```

- Generate 7 random integers from 1 to 50 without replacement

```
sample(1:50, 7, replace = FALSE)
```

9.3 Setting the seed for a random number generation

```
set.seed(1234)
```

10 Character string manipulation

10.1 Special characters

When using regular expressions you need the following commands for special characters:

- $: use `\\$`
- (: use `\\(`

10.2 Joining (concatenating) strings

- These commands `paste0` and `str_c` seem to do the same thing

```
middle<-"middle bit;"
paste0("first bit;",middle, " last bit")
str_c("first bit;", middle, " last bit")
```

10.3 Extracting numbers from strings

- To extract numbers from strings use regular expressions. Eg. `(\\d+)`

```
df  <- tibble(
  treatment = c("A", "B", "C"),
  response = c(12, 11, 10),
  some_text1 = c("abc 7", "abc 2", "abc 5"),
  some_text2 = c("abc 7 xyz 9", "abc 1 xyz 21", "abc 0 xyz 2"),
  some_text3 = c("abc 7 xyz 9", "abc -2 xyz 21", "abc 0.5 xyz 2")
)
str_match(df$some_text1, "abc (\\d+)")
```

- Also works for extracting multiple values from a string

```
str_match(df$some_text2, "abc (\\d+) xyz (\\d+)")
```

- Only works for positive integers

```
str_match(df$some_text3, "abc (\\d+) xyz (\\d+)")
```

10.4 Extracting alphabetic characters from strings

- Works for both upper and lower case letters

```
str_match(df$some_text2, "x([:alpha:]+)")
```

10.5 Replacing parts of strings

```
mpg[6,]$trans
str_replace(mpg[6,]$trans,"a","X")
str_replace(mpg[6,]$trans,"\\(","X")
```

10.5.1 To replace all parts of the string matching the pattern

```
mpg[6,]$trans
str_replace_all(mpg[6,]$trans,"a","X")
```

11 Precision

- Rounding off to n decimal places
  - Note that n can be zero or negative. (Experiment with it to see what it does.)

```
round(15.32257,3)
round(15.32257,-1)
```

- Rounding off to n significant figures

```
signif(15.32257,3)
```

12 Statistics

- Count the number of rows that match each value of one of the columns

```
count(mpg,displ)
count(mpg,drv)
```

- Maximum and minimum values of a numeric list

```
max(mpg$displ)
min(mpg$displ)

#Be careful: any NA values will mean NA is returned
max(starwars$height)
#To ignore the NA values use na.rm=TRUE
max(starwars$height,na.rm=TRUE)
```

- Mean

```
mean(mpg$hwy)
```

- Sample standard deviation. (Note that the sample deviation uses $N-1$ in the denominator of the calculation.)

```
sd(mpg$hwy)
```

- Skewness

Use the `moments` package command, as some algorithms produce different results.

```
moments::skewness(mpg$hwy)
```

- The inter-quartile range

```
IQR(mpg$hwy)
```

12.1 Statistics on grouped data

- We can use the statistics commands via the `summarise()` command, which is especially useful for working with grouped data

```
summarise(mpg, mean_hwy = mean(hwy))
summarise(group_by(mpg,cyl), mean_hwy = mean(hwy))
```

12.2 Summary statistics

- Calculate summary statistics for all numerical variables we can use `inspect_num()`

```
inspect_num(mpg)
```

12.3 Building formulae

- To write a formula we put the response variable on the left of ∼ and the predictors on the right. Eg. `y ~ x + z`.
- To include all variables as predictors (except the response `y`) use a full-stop.
  - For example, if we have a set of predictors `x1, x2, x3` then `y ~ .` is equivalent to `y ~ x1 + x2 + x3`
- We can include interactions between our predictors with the colon ":". Eg. `y ~ x + z + x:z`
- The easy way to write a formula with all individual terms and second-order interactions as predictors is `y ~ .^2`.
  - For example, if we have a set of predictors `x1, x2, x3` then `y ~ .^2` is equivalent to `y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3`

12.4 Linear models

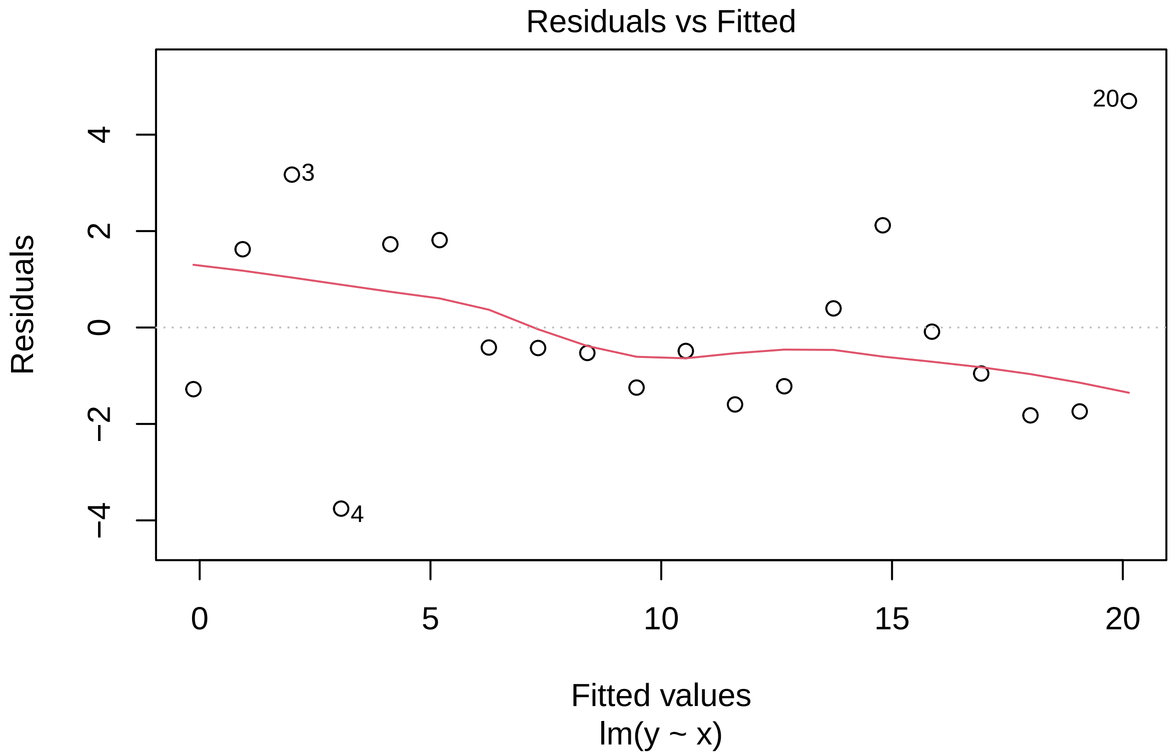
```
df<- tibble(
  x=c(1:20),
  y=x+rnorm(20,0,2)
)
df_lm<-lm(y~x,df)
summary(df_lm)
```

- Extracting the model coefficients

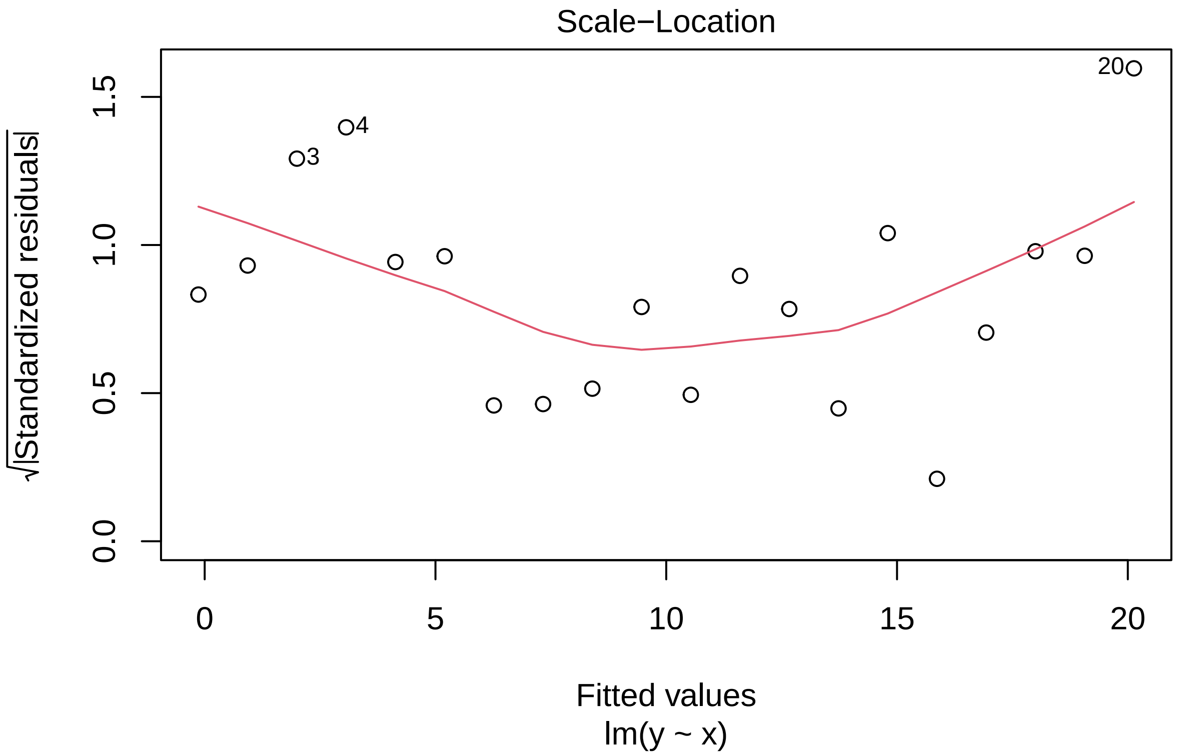
```
as.numeric(df_lm$coefficients)
```

12.4.1 Graphs for checking assumptions of linear models

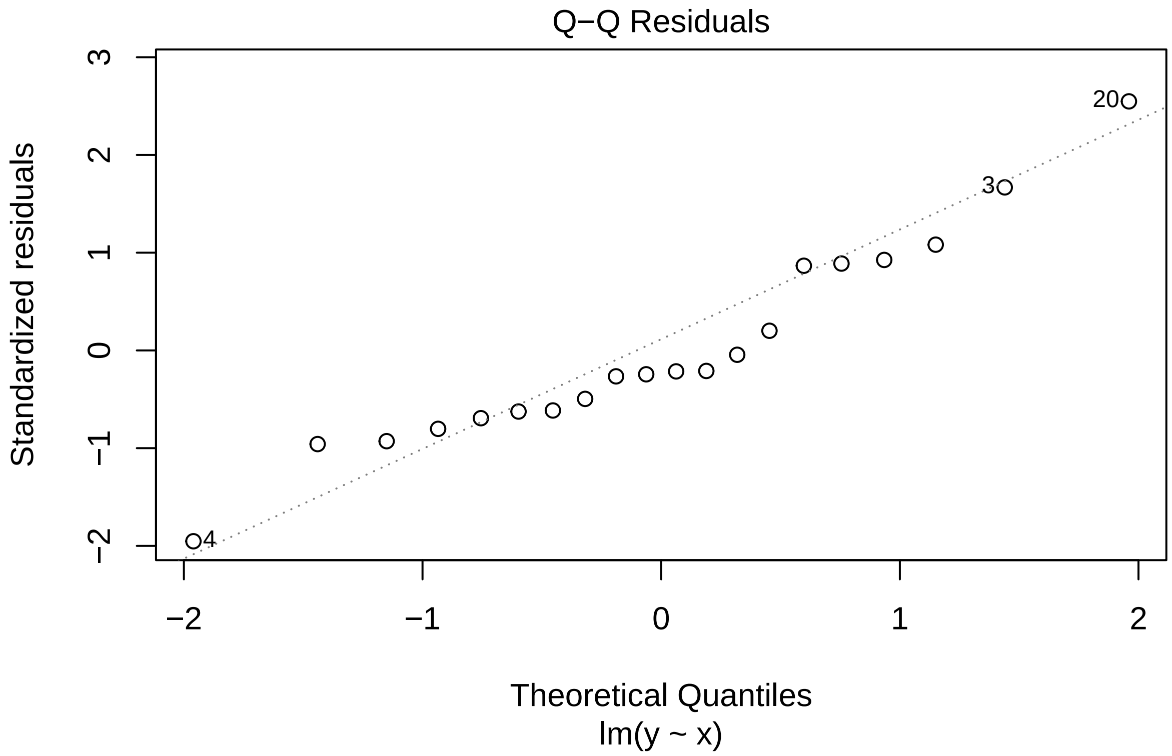
```
plot(df_lm, which=1)
```



```
plot(df_lm, which=3)
```



```
plot(df_lm, which=2)
```



12.4.2 Predicting with linear models

(Note that the explanatory values used for the prediction must be stored in a tibble/dataframe.)

- Prediction with prediction interval of level 85%

```
pred_values<-tibble(
  x=c(2.5, 7.2)
)
predict(df_lm, pred_values, interval="prediction", level = 0.85)
```

- Prediction with confidence interval of level 99%

```
predict(df_lm, pred_values, interval="confidence", level = 0.99)
```

12.5 Logistic models

- We first make a binary categorical variable in a data set

```
car_seats <- as_tibble(Carseats)
car_seats
car_seats <- car_seats %>%
  mutate("sales_high"=ifelse(Sales>8,"high","low"), .after = Sales)
car_seats$sales_high <- factor(car_seats$sales_high)
car_seats_1 <- car_seats %>%
  mutate(Sales=NULL)
```

12.5.1 Building the model

```
classification_lr <- logistic_reg() %>%
  set_engine("glm")
lrfit <- classification_lr %>%
  fit(sales_high ~ Price, data = car_seats_1)
```

12.5.2 Predicting with the logistic model

```
predict(lrfit, new_data=car_seats_1)
predict(lrfit, new_data=car_seats_1, type="prob")
```

12.6 Extracting the model data for general linear models

- Summary of the model

```
summary(lrfit$fit)
```

- Just the coefficients

```
as.numeric(lrfit$fit$coefficients)
```

12.7 Analysis of variance

```
Anova(lrfit$fit)
```

12.8 Building models with categorical predictors

- To see what new variables are introduced we can use `model_matrix()`.
- R will introduce a new binary variable for each level of the categorical predictor, except the reference level. The name of the new variable will be the concatenation of ‘’variable name” and ‘‘level name”.

```
model_matrix(mpg, ~drv)
```

- Note that the data type must not be quantitative (`<int>` of `<dbl>`), otherwise the levels won’t be assigned to new variables.

```
model_matrix(mpg, ~cyl)
```

13 Testing and training sets

13.1 Splitting the data

- You can set the proportion of rows in the training set with the option `prop = ...`

```
mpg_split<-initial_split(mpg, prop=0.7)
```

13.2 Making training and testing sets

```
mpg_train <- training(mpg_split)
mpg_test <- testing(mpg_split)
```

14 Evaluating models

14.1 Regression models

- First we build a model on the training set

```
mpg_lm1<- lm(hwy~cty, data=mpg_train)
```

- Then we predict on the testing set and put the predictions together with the true values

```
predict(mpg_lm1, mpg_test)
rpreds <- tibble(
  reg_truth = mpg_test$hwy,
  reg_preds=predict(mpg_lm1, mpg_test)
)
```

- To evaluate the model we use the `metrics()` command

```
metrics(rpreds, reg_preds, truth=reg_truth)
```

14.2 Classification models

- We train the model on the training set

```
car_split<-initial_split(car_seats_1)
car_train<-training(car_split)
car_test<-testing(car_split)

lrfit1 <- classification_lr %>%
  fit(sales_high ~ Price, data = car_train)
```

- Then we predict on the testing set, and collect the predictions together

```
logpreds <- bind_cols(
  car_test["sales_high"],
  predict(lrfit1, new_data=car_test, type="class"),
  predict(lrfit, new_data=car_test, type="prob")
)
```

14.2.1 Confusion matrix

```
cm <- logpreds %>%
  conf_mat(
    .pred_class,
    truth = sales_high
  )
```

14.2.2 Accuracy

```
logpreds %>% accuracy(
  .pred_class,
  truth = sales_high
)
```

14.2.3 Sensitivity

- Note that you may have to change `event_level` between "first" and "second" depending on what counts as "success" in your model.

```
logpreds %>% sens(
  .pred_class,
  truth = sales_high,
  event_level="first"
)
```

14.2.4 Specificity

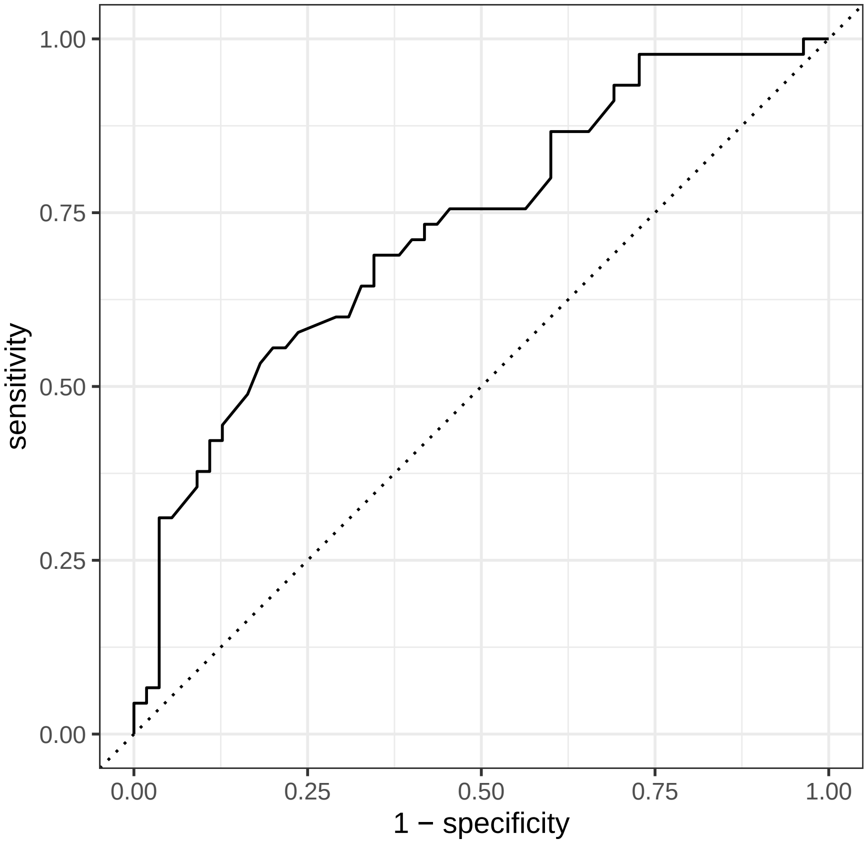
- Note that you may have to change `event_level` between "first" and "second" depending on what counts as "success" in your model.

```
logpreds %>% spec(
  .pred_class,
  truth = sales_high,
  event_level="first"
)
```

14.2.5 ROC and AUC

- Note that there are 4 possible ROC curves given by the options
  - `.pred_high` and `.pred_low` (These names will depend on your data.)
  - `event_level="first"` and `event_level="second"`
- You will need to experiment to make sure you choose the right combination for your data.

```
logpreds %>%
  roc_curve(
    .pred_high,
    truth = sales_high,
    event_level = "first"
  ) %>%
  autoplot()
```



```
logpreds %>%
  roc_auc(
    .pred_high,
    truth = sales_high,
    event_level = "first"
  )
```

15 Data transforms

15.1 Standardising the variables

To apply standardisation to our variables (centring and dividing by the standard deviation) we can use the command `preProcess()` in conjunction with `predict()`

```
mpg_preprocess  <- preProcess(mpg)
predict(mpg_preprocess, mpg)
```

15.2 Box-Cox transform

15.2.1 Finding λ value

```
df_bc<-BoxCoxTrans(y=df$y, x=df$x)
df_bc$lambda
```

The default range for λ is $[-2, 2]$. If you want to search over a bigger range, then you can use the `seq()` command with `lambda` option.

```
df_bc<-BoxCoxTrans(y=df$y, x=df$x, lambda=seq(-5,5,0.05))
df_bc$lambda
```

15.2.2 Transforming the data

```
predict(df_bc,df$y)
```

16 Manipulating time

- Calculate the duration between two date objects in days (as a `difftime` data type)

```
dmy("05-11-2028")- dmy("05-10-2025")
```

- Calculate the duration between two date objects in seconds (as a `duration` data type)

```
as.duration(dmy("05-11-2028")- dmy("05-10-2025"))
```

- If you want these values as integers then wrap the commands in `as.integer()`

```
as.integer(dmy("05-11-2028")- dmy("05-10-2025"))
as.integer(as.duration(dmy("05-11-2028")- dmy("05-10-2025")))
```

17 Cleaning data

17.1 Finding duplicates

```
df<- tibble(
  x=c(1:7,5,2),
  y=x^2
)
duplicated(df)
```

(To use this to extract the duplicated values, see Section "Extracting rows that match a TRUE/FALSE condition".)

18 Tidying data

- Convert to "long" form ("melting" your data)
  - `gather(dataset, key= "first_new_column", value = "second_new_column", start:end)`

```
data("table4a")
table4a
TB_cases<- gather(table4a, key = "year", value = "cases", `1999`:`2000`)
```

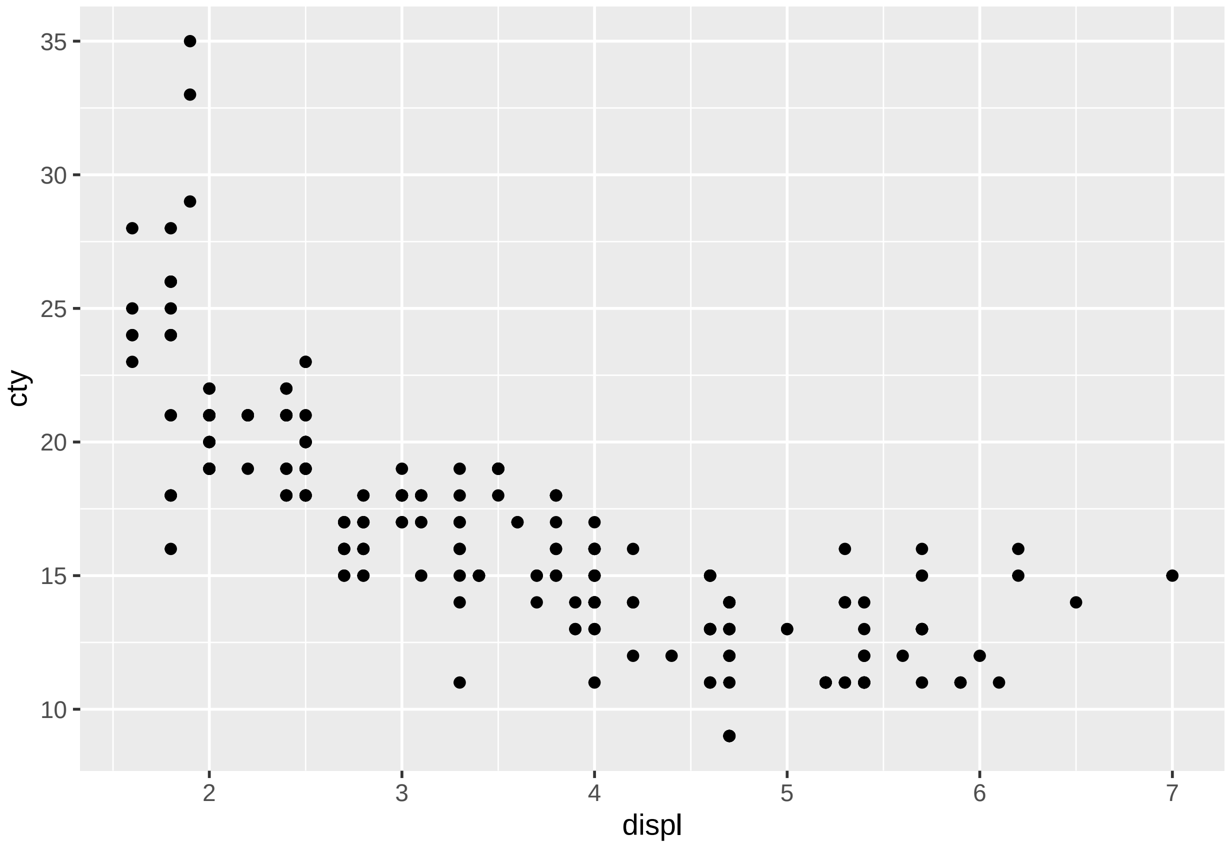
- Convert back to "wide" form, where
  - `key` is the variable that you want to put as new column headings
  - `value` is the the variable that you want to put in these new columns
  - `spread(dataset, key= "key_column", value = "data_column")`

```
spread(TB_cases, key = "year", value = "cases")
```

19 Plotting

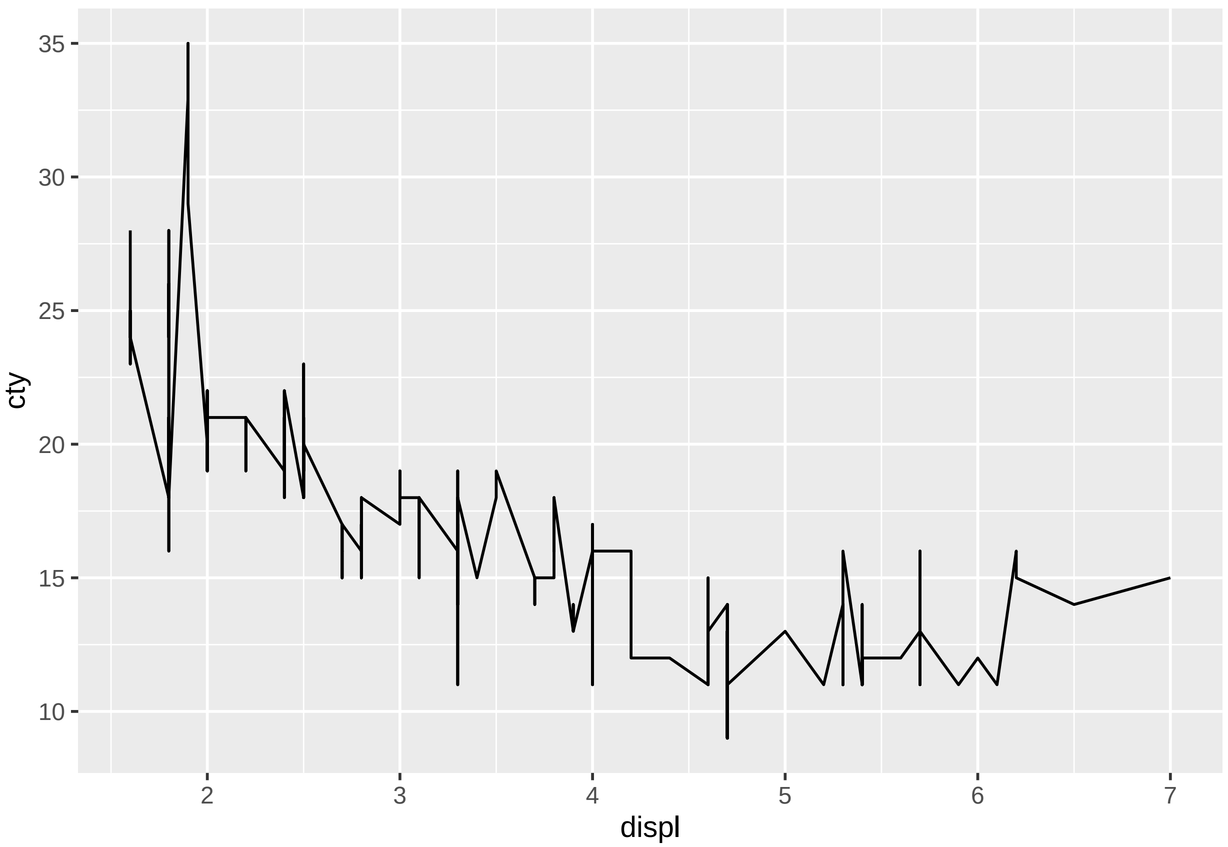
- Scatter plot

```
ggplot(mpg, aes(x=displ, y=cty))+
  geom_point()
```



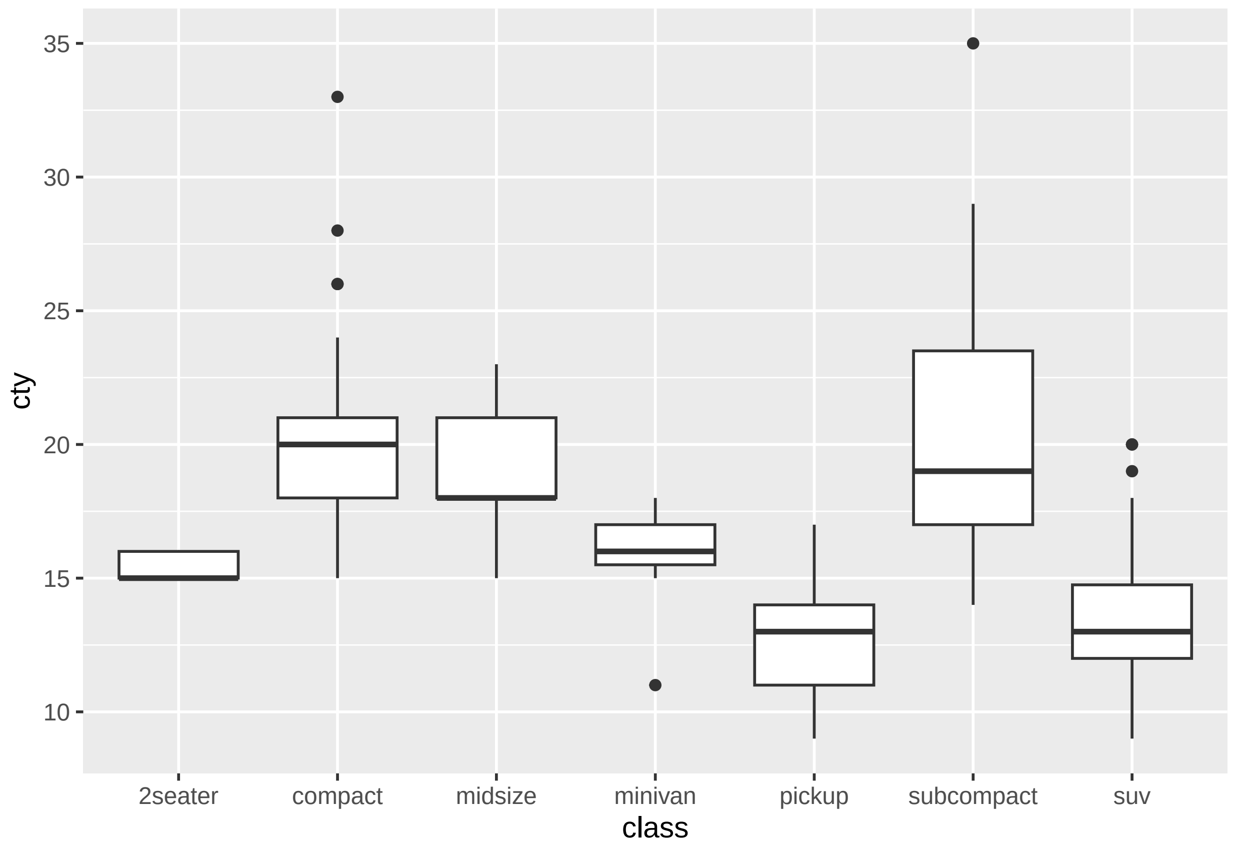
- Line plot

```
ggplot(mpg, aes(x=displ, y=cty))+
  geom_line()
```



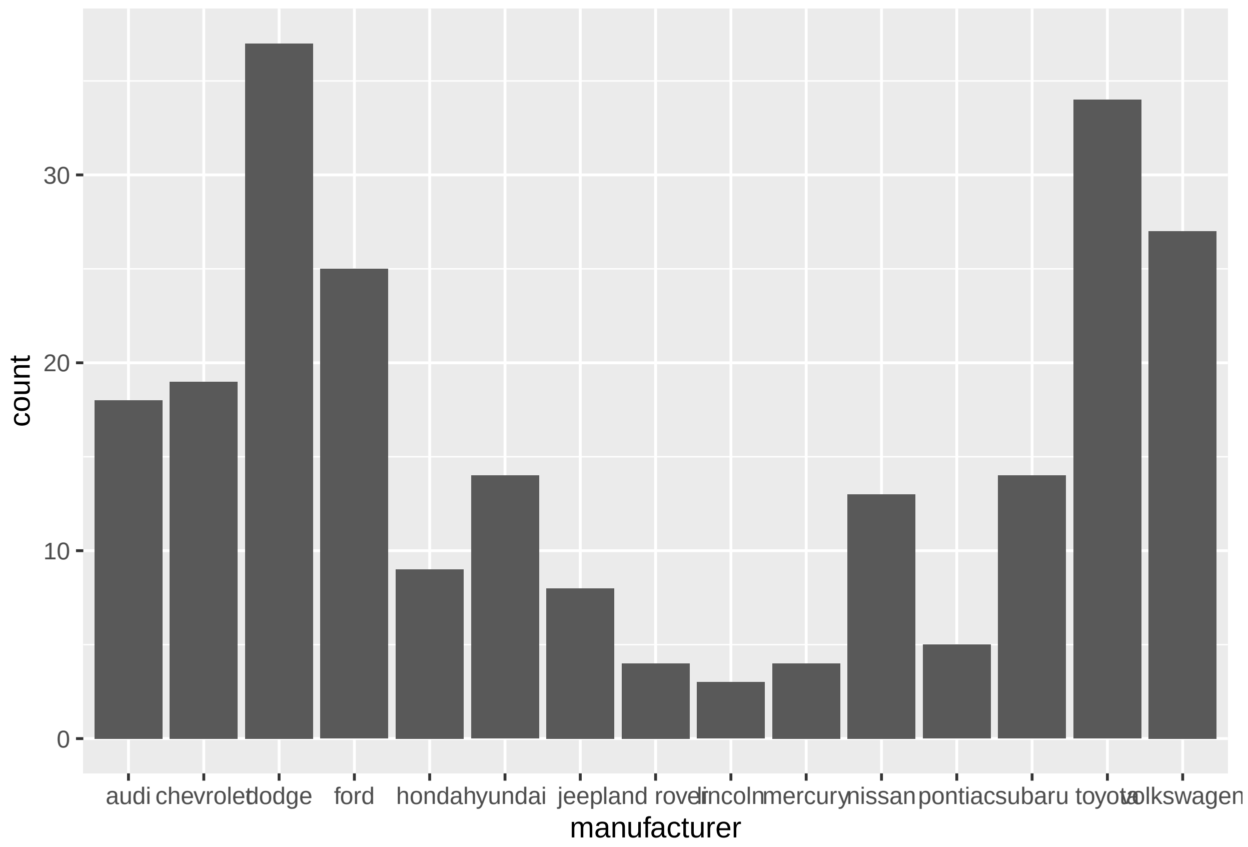
- Box plot

```
ggplot(mpg, aes(class, cty))+
  geom_boxplot()
```



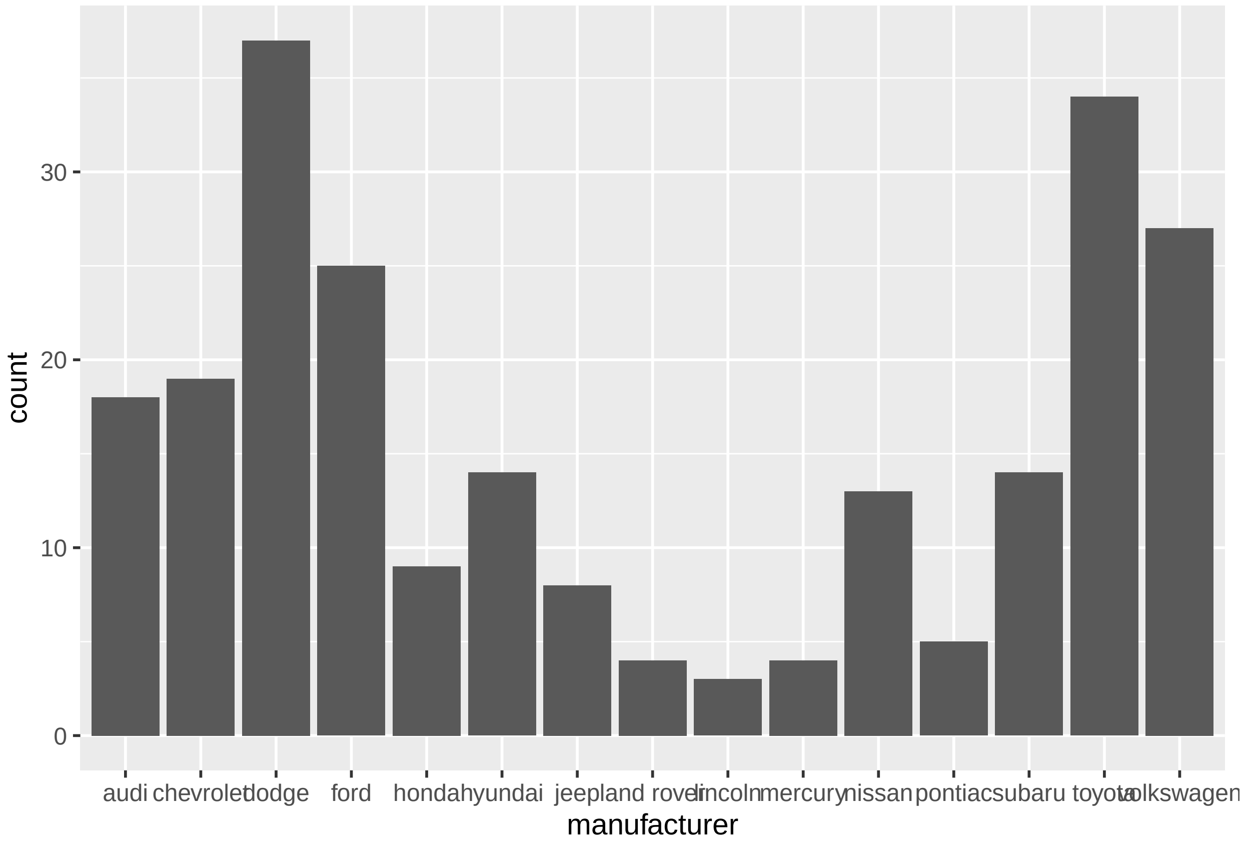
- Bar graph

```
ggplot(mpg, aes(x = manufacturer)) + geom_bar()
```



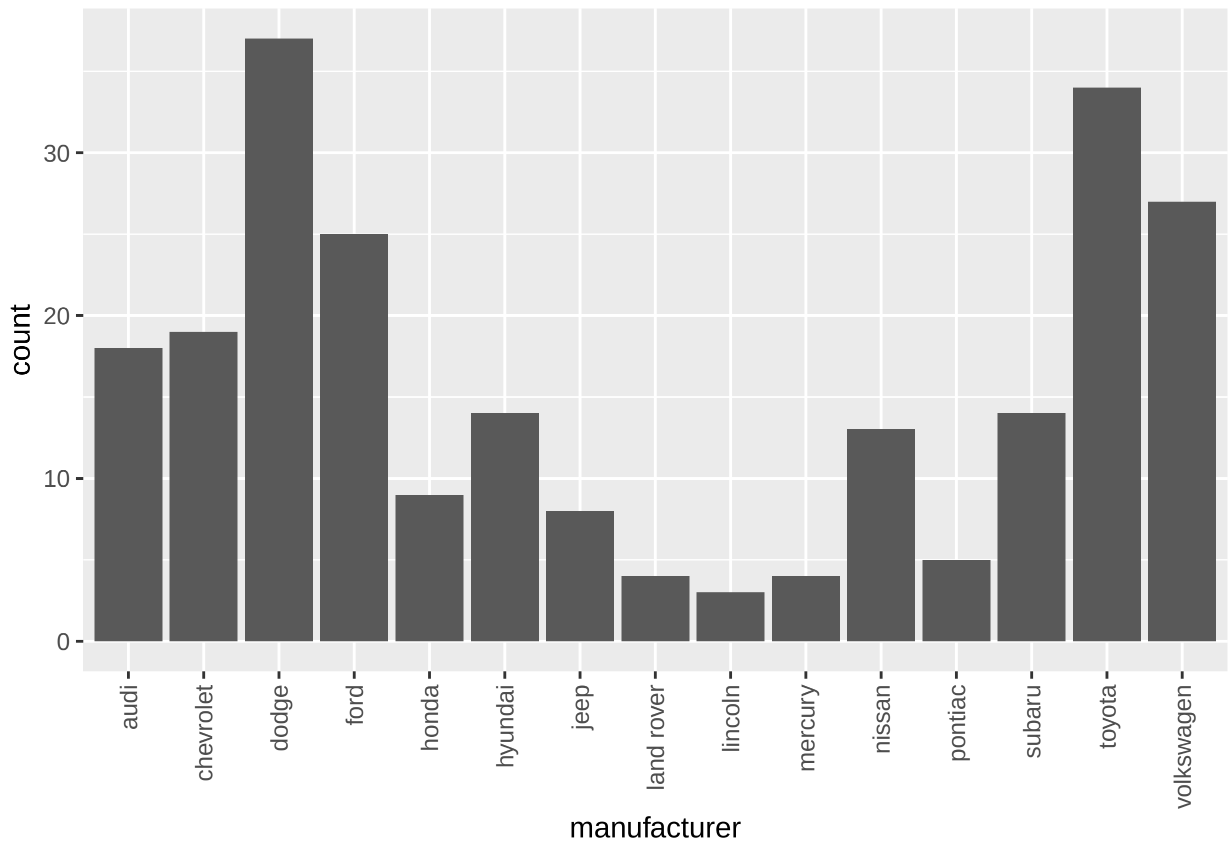
This is similar to a histogram (but we should use bar charts for categorical variables, and histograms for quantitative variables)

```
ggplot(mpg, aes(manufacturer))+
  geom_histogram(stat="count")
```



19.1 Rotating x-axis label

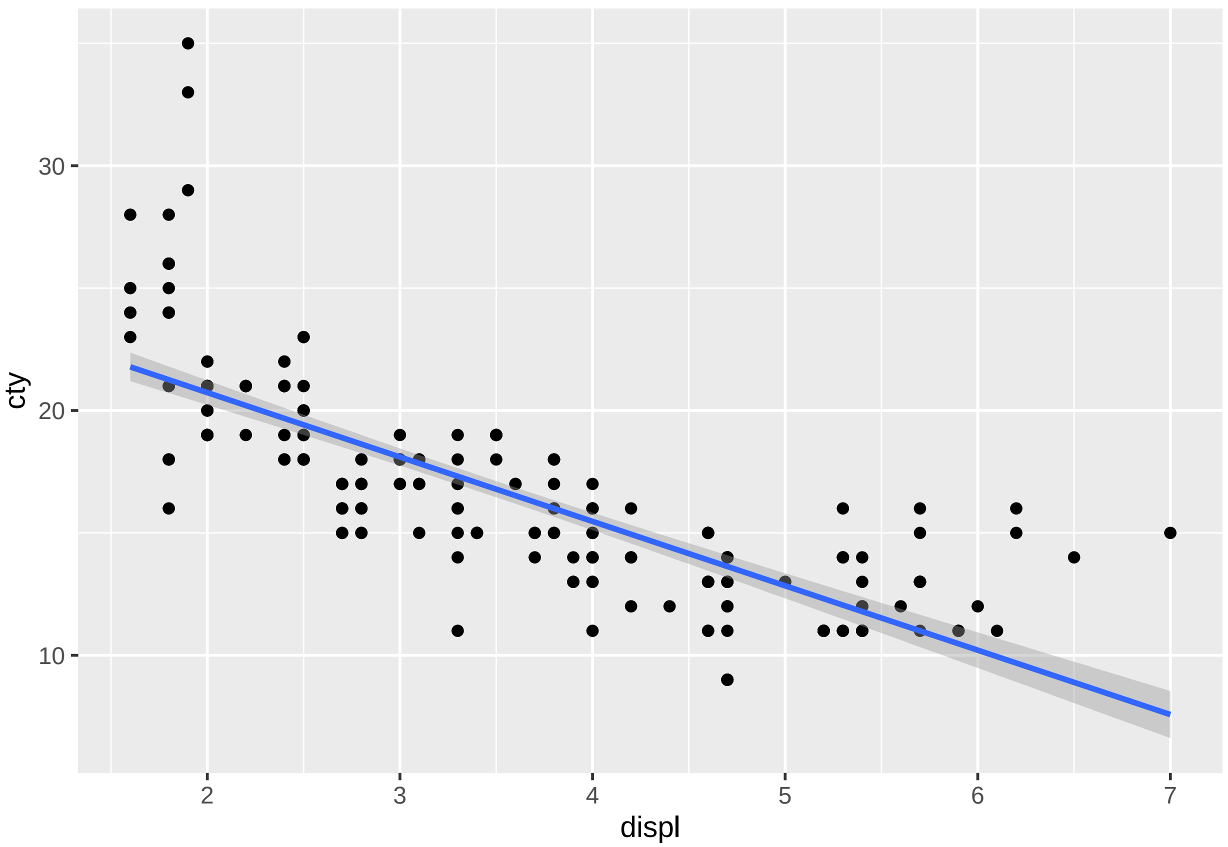
```
ggplot(mpg, aes(x = manufacturer))+
  geom_bar()+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



19.2 Fitting lines and curves to the data

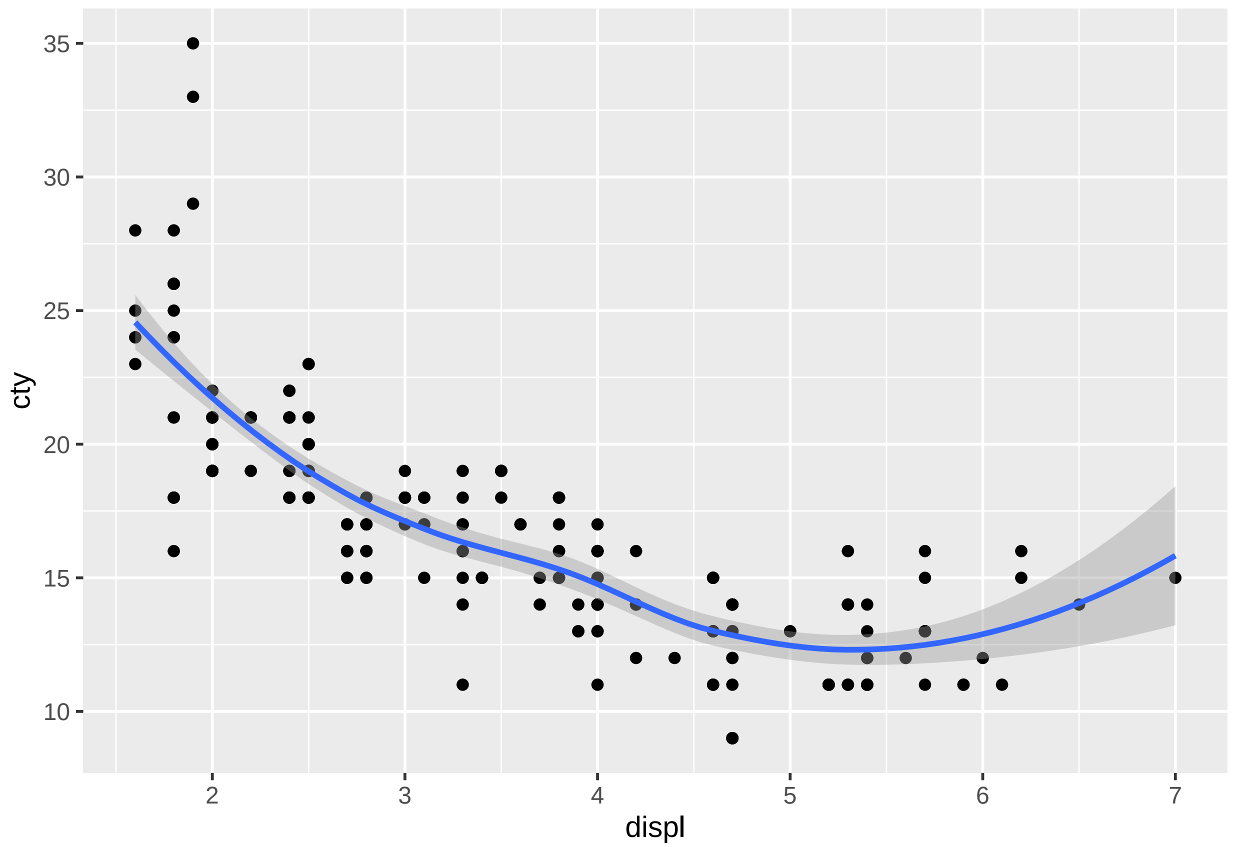
- A straight trend line. Need the method “lm” (as in “linear model”)

```
ggplot(mpg, aes(x=displ, y=cty))+
  geom_point()+
  geom_smooth(method="lm")
```



- A possibly curved smoothed average line

```
ggplot(mpg, aes(x=displ, y=cty))+
  geom_point()+
  geom_smooth()
```



20 R Markdown

20.1 Writing equations

- R Markdown uses Latex conventions for equations (so if you know Latex, you can just type it directly into R Markdown)
- To write in-line maths (expressions in the middle of the text) use a single dollar \$ followed by your maths then a second single dollar \$. Eg. this equation $y = mx + c$ is an in-line maths expression.
  - Note that `\(...\)` is equivalent to `$...$` and you can use either.
  - Note that this means you can't just type a dollar symbol in R Markdown. To type a dollar symbol use `\$`. Eg. a \$10 note is blue.
- To write "displayed" maths (expressions separated from the text) use a double dollar \$\$ followed by your maths, then another double dollar \$\$. Eg. this equation

$$y = x^3 + 4x + 1$$

  is a "displayed" expression.

  - Note that `\[...\]` is equivalent to `$$...$$` and you can use either.

20.2 Writing long equations

- We can use the `align` environment.
- You end a row in the equation with \\\\
- Use an ampersand (&) to align the rows

- This also puts an equation number on every row of our equation

$$\begin{align} y &= \alpha x^3 + 4\beta x + 3\gamma \tag{1} \\ &+ 321 - \omega \tag{2} \\ &= 789 \tag{3} \end{align}$$

- If you don't want the equation number, use the `align*` environment

$$\begin{align*} y &= \alpha x^3 + 4\beta x + 3\gamma \\ &+ 321 - \omega \\ &= 789 \end{align*}$$

20.3 Writing maths symbols (including Greek letters)

- The easiest way to write maths symbols, including Greek letters, is to write them as in-line maths. Eg. here is an α, here is a β, here is a γ and here is an ϵ.
  - To type a Greek letter, type a backslash followed by the name of the letter, eg. `\alpha`, `\beta`, `\gamma`, `\epsilon`.
- To put a "hat" on a maths symbol write `\hat{<symbol>}`. Eg. here is a regular α and here is alpha-hat $\hat{\alpha}$ using `\hat{\alpha}`.
- To write subscripts use an underscore{} after the symbol `<symbol>_{subscript}`. Eg. here is a regular α and here is $\alpha_{1,2}$ using `\alpha_{1,2}`
  - If you've only got a single character in the subscript, then you don't need the curly brackets `{}`, but it's good practice to use them anyway. Eg. here is a regular α and here is α_1 (no curly brackets) and here is α_{1} (with curly brackets).

20.3.1 Some other symbols

- Approximation: `\approx`, ie. $\approx$

20.4 Including R output in your text

- To include some numerical output from R directly into your knitted text use `` `r <variable name>` ``

```
v_1 <- 10.5324
```

Then we can print the value of the variable 10.5324.

- To include the numerical output in an an equation, just do the same thing inside $, or $$ or the `align` environments

Our variable is $v_1 = 10.5324$, or we could write

$$\begin{align*} y &= v_1 \alpha \\ &= 10.5324\alpha \end{align*}$$