R and RMarkdown Notes

Lisa Luff

6/14/2020

Contents

R Instructions (Using RStudio)

Setting up a new R project

1. Click on project on the top right hand corner
2. Choose new project
3. Choose existing directory
4. Use the browse button
5. Select the directory you want to use
6. Select create project

Setting up a new R script

1. Click on the empty page with a green plus just under file
2. Select R Script
3. Set up the top of the document with:

- Name of the script
- Your name and the date
- Put in edit dates as you go
- The purpose of the script

4. Then you can write the script

Installing packages

- To install packages use install.packages(*packagename*)
- Need to call on packages to use them, do this with
  - library(*packagename*)

Importing data into an R project

- If you have the data in a file/document on your computer, you can look in the directory and click on the appropriate data file
  - Or you can type in load("*DocumentName*")
    - This can be used for .RData documents

- If you have text, excel, SPSS, etc documents, you can import them by clicking the import dataset button at the top of the environment window

- Txt files
- Need to manually call them in
  - *variable* = read.table("*name.txt*", header = T)

- CSV files
- Importing csv from the directory
  - You can click on the file in your directory and choose import
  - OR use the read_csv function, you will need the readr package
    - library(readr)
    - *variable* <- read_csv("*name.csv*")
- Importing csv from the web
  - download.file("*website*.csv", destfile = "*filetoputitin*.csv")
  - *variable* <- read.csv("*filetoputin*.csv")

- JSON files
  - You will need the package rjson and to call on it
  - Use fromJSON(file = "*filename*" OR "*URL*")

R basics

- Assigning variables <-, which can be done with **Alt -**
- If writing two or more commands on the same line use **;**
- If you want to reuse the previous command use the up arrow
- **Ctrl Z** to undo
- Use the little broom at the top of the environment window to erase the global environment

Standard Math Symbols

Symbol	Use
+	Plus
-	Minus
*	Multiply
/	Divide
^	To the power of
()	Brackets
%x%	Use the above on non-integers (eg. floats)

Comparisons

Symbol	Comparison
==	Is equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
any(*variable* == x)	If any element of *variable* is the same as x (or any other comparison), returns value of TRUE
all(*variable* == x)	Only if ALL elements of *variable* is the same as x (or any other comparison), returns value of TRUE

Standard functions

Function	Use
sum(x)	Sums anything in the brackets
abs(x)	Finds the absolute value of anything in x
sqrt(x)	Square root of x
exp(x)	e to the power of x
log(x)	Natural log
log10(x)	Log with base 10
sin(x), cos(x), tan(x), etc	For trig functions
pi	To represent the number pi
choose(n, k)	The statistical **choose**
combinations(N, *n1*)	The statistical *combinations*
factorial(x)	The statistical **factorial**!

Creating vectors

- Use **c(x, y, z)**
- If you assign a standard function to a variable, it will create a vector based on the specifications you enter in the function
- To create a vector of numbers within a range assign it as (x:y:z)
  - x - is the start number
  - y - is how much to go up by with each step
  - z - is the end of the range
- To view or create a subset of data:
  - To view, just use the following without assigning
  - To create you can assign
    - *variable*[x:y:z]
    - *variable*[c(x, y, z)]
- To use a non-standard function to create a vector:
  1. Set up the empty vector:
  - *variable* = as.vector(NULL)
  - Or just *variable* = NULL
  2. Use a **for** loop

Generate a set of random variables

Function	Use
set.seed(x)	x can be any number, and it will ensure the same random values everytime
sample(x = a, size = b, replace = *TRUE/FALSE*)	a - maximum value, b - number of variables to generate, replace - if sampling is with/without replacement
runif(n)	n variables from a uniform distribution
rnorm(n)	n variables from a Normal distribution
rexp(n)	n variables from an exponential distribution
rpois(n)	n variables from a poisson distribution
rsignrank(n)	n variables from a Wilcoxon Signed-Rank Statistic distribution

Analysing vector data

Function	Use
x~y	Creates a formula or equation for y as determined by x
head(x)	first 10 elements
tail(x)	last 10 elements
length(x)	Number of elements in a vector
dim(x)	Number of dimension in a dataframe
min(x)	Smallest value
max(x)	Largest value
median(x)	Median
mean(x)	Mean
weighted.mean(*datavector*, *probabilityvector*)	Discrete E[X]
sd(x)	Standard deviation
var(x)	Variance
fivenum(x)	5 number summary(min, Q1, median, Q3, max)
summary(x)	Gives all information for any x
quantile(*vector*, *nthp*) - Value of a quantile	
confint(*formula*, level = x)	Confidence interval for 0.x%
sort(x)	Put the elements of x from lowest to highest
rank(x)	Assigns ranks to variables of x from lowest to highest
sign(x)	Gives back 1 if positive, -1 if negative and 0 if 0
stby(data = x, INDICES = x\$$y$, FUN = z, a, etc.)	This gives a summary of y based on x (usually a dataframe), using functions included in z (descr for descriptive summary), with a, etc. being further arguments for FUN (such as stats = 'common' for the usual statisical descriptors

Data manipulation functions

Function	Use
subset(x, subset = y, select = z)	x is the data to be subset, subset indicates which rows or elements to keep (especially vector elements), select indicates which columns or variables to keep in a dataframe
OR subset(x, c(a, b))	Where x is the dataframe, and a and b are to be included, or -c(), will exclude them
x[,-n]	Creates subsect not including nth row, or [] will exclude
x[x==y]	When entered in a variable, will create a variable with the elements from x that match y or [] will exclude
split(x, y)	Divides data in x into groups defined by y
print(x) OR return(x)	Output value for x
table(x\$$y$)	Will tell you the number of values each variable in dataframe x takes for y
x[x!=y]	Removes values of y from varible x
class(x)	The 'class' of the object
str(x)	Structure, the type of variables in x
paste()	When used with print() combines elements in a visually pleasing way
append(x, y)	Adds value of y to the end of variable x
unique (x)	Looks for unique values of elements in x
cat(x, y, etc.)	Catenates the values separated by commas
rle(x)	Run lengths of elements in *variable*
diff(x)	Give the difference between elements of x in order
duplicated(x)	Gives TRUE/FALSE depending if there are duplicated values in x
factor(x, *labels*, etc.)	This takes a vector x and allows you to replace the values with a vector or labels
rep(x, *times*, each = n)	This will repeat the values in vector x, either the number of times in vector *times* for each elements, or all of them n number of times

Function	Use
sapply(x, FUN = y, etc.)	Applies a function defined by y, with, etc. being further arguments for FUN = (can just use function(){*create function*} instead of FUN =) to each element of x

Tables, Data Frames and Matrices

- A data frame allows a mix of numerical and non-numerical data, and factors in a single matrix
  - You can call on a specific variable in a dataframe:
    - *DataFrame*$*Variable*
  - You can group one column by the values in another with:
    - *DataFrame*$*Column*[*categorycolumn*]
    - Or everything except that with *DataFrame*$*Column*[!*categorycolumn*]
  - To create a dataframe use **data.frame()**
    1. Create vectors - The vector names will be will be the names of the columns, and what you call on
    2. Use data.frame(Vector1, Vector2, Vector 3) to create the dataframe
    3. Use stringsAsFactors = TRUE/FALSE depending on if you want your non-numerical factors to be considered as factors or not (probably not)
  - You can view the columns in a dataframe with name(*df*)
- To set up a table, you can use **as.table()**
  - You can put imported data into a table by calling on the variable holding the data
  - Or you can manually create a table yourself by combining vectors with **rbind()**(To combine as rows) and **cbind()**(To combine as columns)
    - SampleTable <- as.table(rbind(c(a, b, c), c(d, e, f), c(g, h, i)))
  - You can label your dimensions using **dimnames()**
    - You can set the names as string using **list()**
    - dimnames(SampleTable) <- list(RowName = c("A", "B", "C"), ColumnName = c("D", "E", "F"))
- Matrices
  - A Matrix is essentially a dataframe of columns of grouped information
  - Create with **cbind()** or **as.matrix()** (to convert a list)
  - Matrix maths:
    - Multiplication - X %*% Y
    - Addition/subtraction - X +- Y
    - Transpose - t(X)
    - Inverse - solve(t(X) %*% X)

Structural Tools

- **If/else if/else** statements
  if (*condition is TRUE*) {
  do this
  }

  OR

- if (*condition is TRUE*) {
  do this
  } else { **when none of the other conditions are true**
  perform different action, ot nothing at all
  }
  - else if can be used in the middle, formatting like else, but argument is like if
- **For** statements:
  for (i in *range*) {
  do thing
  }
  - i can be represented by whatever you what, but it's the index
  - The range can be numeric (*x*:*y*:*z*), or otherwise set
  - To feed or pull values from an associated element of a variable, use *variable*[i]
- **While** statements

Create a Function

- To create a function we type:
  functionname <- function(*inputvariable*) {
  what the function does
  }

- To call on the function, just type in *functionname*(*inputvariable*)
- If you want to apply your functions to numerous elements, you don't have to run a for loop
  - Use **sapply(*x*, *y*)**, where *x* is a vector or list to apply the function to, *y* is the function to be applied
  - Use **apply(*x*, *y*, *z*)**, where *x* is a matrix to apply the function to, *y* is how to apply it (1 for rows, 2 for columns, and c(1, 2) for both (will double over), and *z* is the function to be applied.

Graphs

- **plot(x = a, y = b, . . .)**, OR, **plot(y~x, data = *data*)**,
  - Create a standard graph
  - a is the vector for the x axis
  - b is the vector for the y axis
    - These don't have to be given by separate vectors, but can be
  - OR y~x are the columns in a data frame to graph against each other
    - Can be used with any of the plot types
  - main = "x" is the title for the graph
  - xlab = "x", ylab = "y" is the label for the specified axis
  - type = "z" is the type of marking the plot will use
    - p for points
    - l (lower case L) is line
    - b for both
    - h for vertical lines
    - s for steps
  - col = "colourname" to decide the colour of the markings
- **barplot(x, y, . . .)**
  - Create a graph that shows the number of values that are equal to the same value
- **hist(x, y, . . .)**
  - Similar to a barplot, but shows how many are within a range of values
- **boxplot(x)**
  - Gives a boxplot with the values of the fivenum() function
  - Use split if you want to separate data based on categories
    - boxplot(split(*x*, *y*))
- **stem(x, scale = x, width = y)**
  - Create a stem and leaf plot (the first value of a decimal or large number on the left, and all the following values to the right)
  - scale = x tells it how many variables you want on the left of the line
  - width = how many to show to the right
- **3D scatter plot**
  - Need package scatterplot3d
  - with(*matrix*, {
  - scatterplot3d(x = *x*,
  - y = *y*,
  - z = *z*,
  - main = "*title*")
  - xlab/ylab/zlab = "label",
  - color = "*colour*",
  - type = "*type*", (eg. h)
  - pch = 19?
  - *coordinates* <- s3d$xyz.convert(*x*, *y*, *z*)
  - text(*coordinates*$x,
  - *coordinates*$y,
  - labels = row.names(*matrix*)
  - cex = *.size percent*
  - pos = *- to left, or + to right (number distance)*)
  - })

- To highlight certain points, use the function
  - **points(x, y, col = *colour*)**
  - pch = is point size?
- To overlay information from multiple graphs, use the function
  - **lines(a, b, col = "c")** to overlay a line over the graph specified directly above it

- To add lines to the graph, use
  - **abline(a = *intercept*, b = *slope*, v = *y-value if horizontal*, h = *x-value if vertical*, col = *colour*, lwd = *line width*)**

- To add multiple lines to the graph, use
  - **matlines(sort(*explanatory*), *variable*[order(*explanatory*), 2:3], lwd = *line width*, lty = 1)**

- To add a legend use
  - **legend("*position*, legend = c(*values*))**
- To create a group of graphs
  - **par(mfrow = c(*x*, *y*))**
    - Before doing any plotting, and then just plot the graphs to be used from left to right and then top to bottom
    - x and y is how many rows and columns of data
- Set the margins of plots with par
  - **par(mar = c(*x*, *a*, *y*, *b*) ± *n*)**, where
    - x - Bottom
    - a - Left
    - y - Top
    - b - Right
    - n - Any buffer
  - The default settings are: c(5.1, 4.1, 4.1, 5.1)
- Advanced graph labels
  - If you want a specific range:
    - **xlim(*start*, *end*) and/or ylim(*start*, *end*)**
  - **las = 2**
    - Will rotate the x labels to be perpendicular
  - To not display the standard point labels:
    - **xaxt or yaxt = "n"**
  - To set your own point labels:
    - **axis(*xy*, labels = c("*strings*"), at seq(*start*, *stop*, *step*))**, where
      - xy is which axis, x = 1 and y = 2
      - The sequence is the range and point positions for the labels
      - Can use las to specifically rotate y labels
  - Set specific graph labels:
    - **mtext(side = *xyt*, line = *p*, "*string*", cex = *n*)**, where
      - xyt - is the label axis, main title = 1, x = 2, y = 3
      - p - is the distance from the graph
      - n - is the size of the text

Analysis using distributions

Approximations

- Normal distribution
  - Probability
    - pnorm(Z)
    - OR pnorm(x, mean = a, sd = b)
  - Z score
    - qnorm(p)
  - Checking for Normality
    - qqplot(dataframe)
    - qqline(dataframe)
    - shapiro.test(dataframe)
- T distribution
  - Probability
    - pt(x, df)
  - T score
    - qt(p, df)
- Chi-squared
  - Probability (α)
    - pchisq(χ^2_{df}, df)
  - χ^2_{df}
    - qchisq(α, df)

Discrete distributions

- Binomial
  - Probability
    - pbinom(x, n, p(S))
  - Random variable X
    - qbinom(p, n, p(S))
- Negative Binomial
  - Probability
    - pnbinom(x, r, p(S))
  - Random variable X
    - qnbinom(p, r, p(S))
- Geometric
  - Probability
    - pgeom(x, p(S))
  - Random variable X
    - qgeom(p, p(S))
- Hypergeometric
  - Probability
    - phyper(x, m, n, k)
  - Random variable X
    - qhyper(p, m, n, k)
- Poisson
  - Probability
    - ppois(x, lambda)
  - Random Variable X
    - qpois(p, lambda)

Continuous distributions

- Uniform
  - Probability
    - punif(x, min = a, max = b)
  - Random Variable X
    - qunif(p, min = a, max = b)
- Log normal
  - Probability
    - plnorm(x, meanlog = a, sdlog = b)
  - Random Variable X
    - qlnorm(p, meanlog = a, sdlog = b)
- Gamma
  - Probability
    - pgamma(x, alpha, rate = beta, scale = 1/beta)
  - Random variable X
    - qgamma(p, alpha, rate = beta, scale = 1/beta)
- Exponential
  - Probability
    - pexp(x, lambda)
  - Random variable X
    - qexp(p, lambda)

Comparing Data

- T-test
  - Hypothesis testing (Single Sample)
    - t.test(*vector*, mu = *x*, alternative = *"less""greater""two.sided"*, conf.level = *%confidence*)
  - Comparing population means (Two Sample and Paired)
    - t.test(*samplevector*, *comparisonvector*, mu = 0, alternative = *"less""greater""two.sided"*, paired = *TRUE/FALSE*, conf.level = *%confidence*)
- Wilcoxon Signed-Rank Test
  - Is the non-parametric version of a t.test for when the data is not Normally distributed
  - One sample
    - wilcox.test(*vector*, mu = *x*, alternative = *"less""greater""two.sided"*, exact = *TRUE/FALSE*)
      - V - represents W statistic
    - Manual p-value if no ties and small (< 50)
      - psignrank(W, n)
  - Two sample and paired
    - wilcox.test(*vector1*, *vector2*, alternative = *"less""greater""two.sided"*, paired = *TRUE/FALSE*, exact = *TRUE/FALSE*)
    - Manual p-value if no ties and small (< 50)
      - pwilcox(W, *n_1*, *n_2*)
- Levene Test
  - Tests if samples have the same variance, giving a p-value based on the probability that the null hypothesis that they are equal is true
    - *Requires 'cars' package*
    - leveneTest(data = z, x~y)
      - Based on data frame z
      - We test y based on a formula/equation for y in relation to x

- ANOVA
  - Hypothesis testing for greater than two samples, or comparisons
  - The hypothesis will be whether or not the means are statistically different
    - First create the ANOVA model
      - a <- aov(data = z, x~y)
      - Based on data frame z
      - We test y based on a formula/equation for y in relation to x
    - Then we get a summary of the model
      - summary(a) **OR** (both give the same information)
      - anova(a)
- Welch's ANOVA
  - The ANOVA for is variance between variables is not equal
    - oneway.test(data = z, x~y, var.equal = *FALSE*)
      - Based on data frame z
      - We test y based on a formula/equation for y in relation to x
      - var.equal indicates which one-way ANOVA test to run
      - FALSE - specifies a Welch's ANOVA
      - TRUE - specifies a standard ANOVA, but aov is the preferred method for this
- Kruskal-Wallis Test
  - The non-parametric version of the ANOVA for when the data sets are not Normally distributed
    - kruskal.test(data = z, x~y)
      - Based on data frame z
      - We test y based on a formula/equation for y in relation to x
- Covariance
  - Joint probability distribution
    - cov(table or dataframe for XY)
  - Matched pair design
    - Pearson: cov(table or dataframe for XY, paired = TRUE, method = "Pearson")
    - Spearman: cov(table or dataframe for XY, paired = TRUE, method = "Spearman")
- Correlation
  - cor(table or dataframe, method = "Pearson""Spearman""Kendall", exact =)
  - In hypothesis testing:
    - cor.test(table or dataframe, alternative = "Pearson""Spearman""Kendall", conf.level = 0.95)
  - Graph of multiple correlations:
    - *Requires 'PerformanceAnalytics' package*
    - chart.Correlation(table or dataframe, histogram(or other type) = TRUE)
    - Provides a table of graphs and correlation coefficient values
    - Or use pairs() - Not as good
- Chi-squared test
  - Goodness-of-fit testing
    - chisq.test(Vector)
  - Independence of two-way contingency tables
    - chisq.test(table or dataframe of XY)

Comparing Data With Resampling

- Bootstrapping
  - *Requires 'boot' program*
  - boot(dataframe, theta, x)
    - Where theta is a function to find the statistic
    - And x is how many resamplings you want to do
    - Entered into a variable it will output a dataframe and $t will be the statistic for each resampling
    - Bias is the difference between the original mean and the average mean for the resamplings
- Permutations
  - *Requires 'CarletonStats' program*
  - permTest(x vector, y vector, alternative = "")

Simple Linear Regression

- Use the linear model function lm
  - lm(*explanatory~response*, data = *dataframe*)
  - Will output a dataframe
  - Information can be seen with summary()
    - β_0 - Intercept~Estimate
    - β_1 - *explanatory*~Estimate
    - Standard error for each
    - *All hypothesis tests based on null = 0*
    - T statistic for each
    - Two-sided p-value for each
    - Degrees of freedom
    - R^2 (Coefficient of Determination)
    - F statistic, degrees of freedom and two-sided p-value
- Or can use ANOVA function as before
  - *variableaov* = aov(*explanatory~response*, data = *dataframe*)
  - anova(*variableaov*)
  - Will output a dataframe
    - DFR - *explanatory* ~ df
    - DFE - Residuals ~ df
    - SSR - *explanatory* ~ Sum Sq
    - SSE - Residuals ~ Sum Sq
    - MSR - *explanatory* ~ Mean Sq
    - MSE - Residuals ~ Mean Sq
    - *All hypothesis tests based on null = 0*
    - F value
    - Pr(>F) - Two-sided p-value
- Both give the exact same p-value
- Confidence and Prediction Intervals for Linear Regression
  - Confidence Intervals
    - Use the predict function
    - *variableconfmean* = predict(*variablelm*, interval = "confidence", level = *percent as decimal*)
      - To predict as specific values add the argument
      - newdata = data.frame(*explanatory* = c(*values*))
  - Prediction Intervals
    - Use the predict function
    - *variablepi* = predict(*variablelm*, interval = "prediction", level = *percent as decimal*)
    - To predict as specific values add the argument
      - newdata = data.frame(*explanatory* = c(*values*)))

- Diagnostic Testing
  - Residuals
    - *variable*.lm$residuals
  - Standardise residuals
    - rstandard(*variable*.lm)
  - Hat matrix elements (outliers)
    - hatvalues(*variable*.lm)
  - Cooks distance (leverage)
    - cooks.distance(*variable*.lm)
  - Detailed data
    - influence.measures(*variable*.lm)
  - All in one
    - plot(*variable*.lm)
    - Creates 4 plots:
      - Residuals vs fitted
      - Standardised residuals vs fitted
      - Normal QQ
      - Residuals vs Leverage
- Linear Regression with matrices

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{pmatrix} = \boldsymbol{X} = \begin{pmatrix} 1 & X_1 \\ 1 & X_2 \\ . & . \\ . & . \\ 1 & X_n \end{pmatrix} x\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{pmatrix}$$

  - $(\boldsymbol{X'X})$ is t(**X**) %*% **X**
  - $(\boldsymbol{X'X})^{-1}$ is solve(t(**X**) %*% **X**)
  - $(\boldsymbol{X'Y})$ is t(**X**) %*% **Y**
  - $\hat{\beta}$ is solve(t(**X**) %*% **X**) %*% t(**X**) %*% **Y**
  - $\boldsymbol{\hat{Y}} = \boldsymbol{X}\boldsymbol{\hat{\beta}}$
  - $\boldsymbol{\hat{\varepsilon}} = \boldsymbol{Y} - \boldsymbol{\hat{Y}}$
  - $SSE = \boldsymbol{\varepsilon'\varepsilon}$
  - $s^2 = \frac{SSE}{n-2}$
  - lm = lm($\boldsymbol{Y} \sim \boldsymbol{X}$, data = *matrix*)
    - Will give the usual info

Multiple Linear Regression

- Uses matrices: $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{pmatrix} = \boldsymbol{X} = \begin{pmatrix} 1 & X_{11} & X_{12} & ... & X_{1(p+1)} \\ 1 & X_{21} & X_{22} & ... & X_{2(p+1)} \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & X_{n1} & X_{n2} & ... & X_{n(p+1)} \end{pmatrix} x\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_{p+1} \end{pmatrix} + \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{pmatrix};$$

  - Where $X = X_{ij}$, where i = 1, 2, ..., n, and j = 1, 2, ..., p, p + 1
- Exploratory analysis done with
  - Multivariate correlation plots
  - Multiple boxplots
  - 3-D scatterplot if few enough variables
- lm = lm($\mathbf{Y} \sim \boldsymbol{X_1} + \boldsymbol{X_2} + ... + \boldsymbol{X_n}$, data = *matrix*)
  - OR lm = lm($\mathbf{Y} \sim .$, data = *matrix*) - if all variables
  - Can use x = TRUE to keep the variables in the order provided
  - Gives the usual info, but for each β_j
  - Each β_j needs to be interpretted as the increase/decrease **when all other variables being held constant**
  - Need to use Adjusted R-squared for R^2
- Most equations the same as for simple linear regression
- Variance is $s^2 = \frac{SSE}{n-p-1}$
  - summary(lm)$sigma gives estimation s
- Variance/Correlation matrix

		x1	x2
	$\boxed{Var}$	Cov	Cov
x1	Cov	$\boxed{Var}$	Cov
x2	Cov	Cov	$\boxed{Var}$

  - With solve(t($\mathbf{X}$) %*% $\mathbf{X}$) %*% $\mathbf{X}$
- ANOVA -
  - $H_0 : \beta_1 = \beta_2 = ... = \beta_p = 0$
  - H_A : at least one of the $\beta_i \neq 0$
    - Create lm of each combo
      - This might just be lm($y \sim 1$, data = *data*)
    - anova(*lm1*, *lm2*)
    - OR use package rms: anova(rms::ols($y \sim x_1, x_2, ..., x_i$, data = *data*)
- F-test for comparing models
  - **Model 1** - $H_0 : \boldsymbol{Y} = \boldsymbol{X_1\beta_1} + \boldsymbol{\varepsilon}$
  - **Model 2** - $H_A : \boldsymbol{Y} = \boldsymbol{X_1\beta_1} + \boldsymbol{X_2\beta_2} + \boldsymbol{\varepsilon}$
    - Use ANOVA or summary
- Partial F-test to compare models with variables removed
  - $H_0 : \beta_c1 = \beta_c2 = ... = 0$
  - $H_A : \beta_c1, \beta_c2, ...$ are not all 0
    - Use ANOVA or summary

- Leverage -
  - Plotting:
    - For plotting with actual data:
      - plot(1:n, hatvalues(lm), xlab = x_i, ylab = leverage)
      - abline(h = $\frac{2(p+1)}{n}$)
    - For plotting with standarised variables:
      - plot(hatvalues(lm), stdres(lm), xlab = leverage, ylab = standardised residuals)
      - abline(v = $\frac{2(p+1)}{n}$)
      - abline(h = 0)
  - Get hat values with:
    - For actual data:
      - identify(1:n, hatvalues(lm), labels = rownames($data$))
    - For standardised data
      - identify(hatvalues(lm), rstandard(lm), labels = rownames($data$))
- Influential Observations (Cooks Distance) -
  - Plotting:
    - Cooks distance against actual data:
      - plot(cooks.distance(lm), xlab = x_i, ylab = cooks distance)
    - Cooks distance against leverage:
      - plot(hatvalues(lm), cooks.distance(lm), xlab = leverage, ylab = cooks distance)
      - abline(v = $\frac{2(p+1)}{n}$, h = $2(p+1)/(n-2)$)
  - Get Cooks distance values:
    - identify(1:n, cooks.distance(lm), label = rownames(lm))
      - **Note**: Must be run in the console
- Categorical/indicator variable -
  - Linear model:
    - lm(formula = $y \sim x$ + *category* + x*category, *data* = data*) OR
    - lm($y \sim x$ + *category* + x:*category*, data = *data*), where
  - To only get intercept:
    - lm($y \sim x$ + *category*, data = *data*)
  - Summary:
    - summary(lm), where
      - Intercept = β_0 - Category 1 intercept
      - $x = \beta_1$ - Category 1 slope
      - *categoryn* = $\beta_{n+1} z_{ni}$ - Category n intercept
      - x:*categoryn* = $\beta_m x_i z_{ni}$ = category n slope
  - T-test of the linear model
    - t.test($y \sim$ *category*, var.equal = TRUE, data = *data*), where
      - p-value is if there is significant difference between observations between variables
  - T-test of the linear regression
    - lm($y \sim$ *category*, data = *data*)
    - summary(lm), where
      - p-value will be exact same as t-test of the linear model

- Variable selection -
  - Brute-force -
    - View p-value for all variables -
      print(load(".RData"))
      lm <- lm($y \sim .$, data = *data*)
      summary(*lm*)$coefficients
    - Evaluate n best models with up to x variables
      require(leaps)
      subsets <- regsubsets($y \sim .$, nbest = n, nvmax = x, data = *data*)
      subsetsummary <- summary(*subsets*)
      subsetsummaryoutmat$*$*Betterversionofthegraphlibrary*(*kable*)*kable*($*$*subsetsummary*$*$outmat)
      - Will show a list of models, far left number is number of variables, second number is if it's the 1st, 2nd, etc best option for that number of variables
      - Compare the graphs of models
        par(mfrow = c(1, 3))
        # Plot R^2
        plot(1:10, *subsetssummary*$adjr2, **log = "*y*"**) (If using log of values)
        # Plot BIC
        plot(1:10, *subsetssummary*$bic, **log = "*y*"**) (If using log of values)
        # Plot C_p
        plot(1:10, *subsetssummary*$cp, **log = "*y*"**) (If using log of values)
    - Look at p-value for each model with x number of variables
      subsetmatrix <- *subsetssummary*$outmat *lmp* <- lm(formula(paste("y", paste(names(which(*subsetmatrix*[,*x*]=="*")), collapse="+"))), data = *data*)
      summary(*lmp*)
    - Extract AIC
      extractAIC(*lm*, k = *df weight*)
  - Forward selection -
    - Minimal model
      lm -> lm($y \sim 1$, data = *df*)
    - See the list of AIC values
      lm <- lm($y \sim$ *current model variables*, data = *data*)
      lmforward <- step(*lm*, scope = $\sim x_1 + x_2 + ... + x_n$ (not including variables in current model), direction = "forward")
      - is the current model
    - OR
      step(*lm*, scope = formula(*df*), direction = "forward")
    - Can use trace = 0 to hide the steps
    - OR
      regsubsets($y \sim .$, nbest = n, data = *data*, method = "forward")
  - Backward selection -
    - Same as forward, but with direction/method = "backward", and don't need scope
  - Stepwise selection/both direction selection
    - Same as forward and backward, except direction/method = "both"
      - Don't actually need direction, as "both" is default

- Interaction
  - PRESS
    - Can use leaps package, but doesn't output fitted models
      press(*modellm*)
    - Otherwise there is a package DAAG with a press function
      press(*modellm*)
  - Creating a training and test set
    samplevalues -> sample(nrow(*df*), floor(0.2, * nrow(*df*)))
    test -> *df*[*samplevalues*,]
    training -> *df*[-*samplevalues*,]
  - Compare model to prediction
    prediction -> predict(*modellm*, newdata = *test*)
    actual -> *df*$*y*
    plot(*actual*, *prediction*)
    abline(0, 1)

  - RMSEP
    RMSEP <- sqrt(sum((*actual* - *prediction*)^2)/length(*actual*))
  - Multi-collinearity
    - Coloured graph showing levels of correlation
      require(corrplot)
      corrplot(cor(*data*[,*columnstoremove*]))
      - To see a single row
        corrplot(cor(*data*[,*columnscinluded*])[*columnsincluded*, *columntoview*, drop = FALSE], cl.pos = 'n', method = 'number')
    - A different, uglier version
      require(lattice)
      splom(~*data*[,*columnstoremove*], groups = *category*, data = *data*, pscales = 0, varname.cex = 0.5)
    - VIF (variance inflation score) - **want to be less than 5**, else possible multicollinearity
      library(car)
      vif(*lm*)

Polynomial Regression

- Polynomial Regression -
  - Linear model:
    - lm($y \sim x$ + Ix^2, data = *data*)
  - Fitted model:
    - *lm*$fitted
  - Summary:
    - summary(*lm*), where
      - Intercept = β_0
      - $x = \beta_1$
      - I(x^2) = β_2

RMarkdown Instructions

Setting up a new RMarkdown

1. Click on the empty page with a green plus just under file
2. Select RMarkdown...
3. Enter the name and choose to output as html, you will still be able to output as pdf or word by clicking the arrow to the right of the knit button at the top of the RMarkdown window
4. You can erase everything up to the r setup box
5. Now you can write the document

Output options

- You can manipulate the way your document outputs using the output: section at the very top

For a Contents Table

- More complex
  output:
  > pdf_document:
  » toc: yes
  > html_notebook:
  » highlight: haddock
  » number_sections: yes
  » theme: flatly
  > html_document:
  » highlight: haddock
  » number_sections: yes
  » theme: flatly
  » toc: yes
  > word_document:
  » highlight: tango
  » toc: yes

- Easier and more detailed
  - In the body of the document from where you want the contents to be curated:
    \tableofcontents

Change font and font size

- To change font, in the YAML header above output, put:
  mainfont: *fontname* (Will start with a capital, not all available)
  fontsize: *number*pt
- To make italic or bold
  - *word* - *word*
  - **word**- **word**

Set page margins

- Use geometry in the YAML header about output, where:
  - left, is left margin
  - right, is right margin
  - top, is top margin,
  - bottom, is bottom margin
    geometry: "left=*n*cm, right=*n*cm, top=*n*cm, bottom=*n*cm"

Page layout

- \centering - will center everything below
- \raggedright or \raggedleft - will right or left align everything below
- \clearpage will leave a clearpage for a front page

To have columns

- Simple:
  \twocolumns
  - Can get specific with:
    \twocolumns \onecolumn
- Complex, but specific:
  output:
  pdf_document:
  includes:
  in_header: preamble.tex
  html_document:
  css: preamble.css
  - And be able to access the text documents it references
  - You will also need to put the following in your r setup insert:
    knitr::opts_chunk$set(echo = TRUE, cache = TRUE)
- To knit to html - /* See: https://bookdown.org/yihui/rmarkdown-cookbook/multi-column-layout.html
  */ .columns {display: flex; }
- To knit to pdf -
  %% See: https://bookdown.org/yihui/rmarkdown-cookbook/multi-column-layout.html
  \newenvironment{columns}[1][]{}{}
  %%
  \newenvironment{column}[1]{\begin{minipage}[t]{#1}}{%
  \end{minipage}
  > \ifhmode\unskip\fi
  \aftergroup\useignorespacesandallpars}
  %%
  \def\useignorespacesandallpars#1\ignorespaces\fi{%
  #1\fi\ignorespacesandallpars}
  %%
  \makeatletter
  \def\ignorespacesandallpars{%
  > \@ifnextchar\par
  » {\expandafterbackslash ignorespacesandallpars\@gobble}%
  » {}%
  }
  \makeatother

- Then use the following to create the columns, obviously you can change the percentages to add more columns, and the middle one is just there to leave some space between the two columns:

:::::: {.columns}

::: {.column width="48%" data-latex="{0.48\textwidth}"}

text

:::

::: {.column width="4%" data-latex="{0.04\textwidth}"}

\

:::

::: {.column width="48%" data-latex="{0.48\textwidth}"}

text

:::

::::::

\newline

RMarkdown basics

- You can use # to set headings. The more #'s the smaller the heading
  - Headings are used by RMarkdown and pdf's as a way to jump straight to that sectoin of the document, so handy to use them a fair bit even if they're getting really small
- Using a * at the start of a line create a bullet point, and if you use tab it will indent and use a different type of bullet point
  - If you don't use it for every line in a paragraph it messes up though
- You can create a numbered list just using 1. and so on, however you cannot put bullet points inside numbers, if you want to indent you have to use a), i), etc.
- If you want to use any mathemtical symbols you can use LaTeX to write them by putting a dollar sign either side
  - See LaTex commands for more details
- If you aren't using lists or some kind, or you are using LaTeX insert, you need to put a double space to indicate when you want it to be a new line
- You can access a number of commands using backslash
  - Eg. \newline or \newpage

Entering code into a document

- You can write code directly into the document by clicking on the green square with a C and a plus, you can choose from several languages
- When coding with R, you can enter a number of commands in the header of the insert, the main ones I use:
  - echo = TRUE/FALSE depends if you want to print the script in the document, if you put FALSE, but want to display the output, just use the print() function
  - comment = NA will hide the line numbering in the document
  - eval = FALSE will just show the code without trying to run it
- For other types of code you might need to run a different engine.
  - eg, {bash, eval=FALSE, engine="sh"}

Displaying graphs

- You can change the size of a graph, and even had them side by side
  - This is done by putting the following in the header of your inserted R code
    - fig.hold='hold', out.width="n%", out.height="m%"
  - If you want side by side, you will need to make sure you don't display your code (echo = FALSE), or it will separate them, so if you want to show code, put the graphs in a separate chunk of code
- You can set the graph position with:
  - {fig.align="center"}

Flow Chart

- You can create a flow graph using R code
  - To use in pdf first enter the following into the console
    install.packages('webshot')
    webshot::install_phantomjs()

  - To have it align nicely in a pdf, need to export it as a png file
    - In the YAML header, put keep_tex: yes under pdf_document in output
    - In the R code instructions put fig.cap = "*FigName*", dev = "png"
- Need DiagrammeR package

```
library(DiagrammeR)

# Create graph with grViz function
grViz("
      # Used to make graph
      digraph flow {

      # Gives graph instructions, dot is flow chart
      graph [layout = dot, shape = rectangle]

      # Gives instructions for the actual information in the chart
      node [shape = box,
      fontname = Arial]

      # Set the nodes and What will show up in their place
      A [label = A]
      B [label = B]
      C [label = C]
      D [label = D]
      E [label = E]
      F [Label = F]

      # Markers to create grid shape
      node [shape = none, width = 0, height = 0, label = '']
      mark13
      mark31
      mark33

      # Creates columns of data
      A -> C -> mark13 [style = invis]
      B -> D -> F [style = invis]
      mark13 -> E -> mark33 [style = invis]

      # Create rows of data
      {rank = same ; A -> B -> mark13 [style = invis]}
      {rank = same ; C -> D -> E [style = invis]}
      {rank = same ; mark31 -> F -> mark33 [style = invis]}

      # Actual connections where ever you want
      edge [constraint = false]
      A -> C [color = red]
      B -> {D E} [color = blue]
      C -> F [color = green]

      }
      ")
```

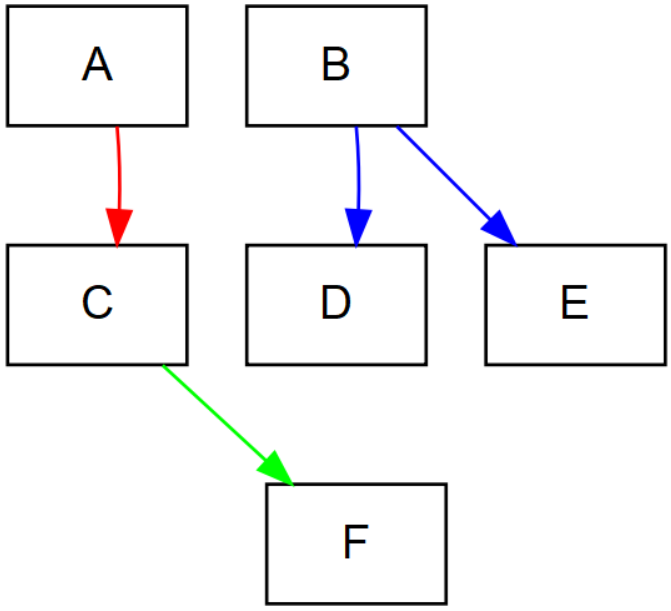

Figure 1: Flow Chart

Inserting images

- If you want to include images in your document just put:
  -
  - Where your title goes between the []
  - You need to have it saved in your current working file, or enter the full path
- You can change the size using:
  - {width=*n*%, height=*n*%} - after the brackets holding the filename, you can also use specific measurements instead of a scale percent

Creating Tables with R

- You can use:
Title1	*Title2*
Thing	*Description*
- You do need to leave a space on top and bottom to do this
- Example:

Title1	*Title2*
Thing	*Description*

LaTeX Commands

- Formatting
  - Putting a \ next to a command will stop it from doing the command, and will instead display it, *escapes a special character*
  - \to - an arrow pointing right
  - \text{} - will display any text normal-ish and will allow for spaces which otherwise don't show
    - Or \textit{} - for italised text
  - \mathit{} will italicise math
  - \boldsymbol{} will bold math
  - \begin{aligned} \end{aligned} - will align multiple lines with each other where an *&* represents the point you want to be centred in the line and \ to show a break between the end of one line and the beginning of the next
  - \left(/{/[and \right)/}/] - will show that bracket at the front (works the same for other bracket types)

Creating Tables with LaTex

- Need to include tables: true above output in the YAML header
- If using LaTeX maths symbols, etc inside the table, need to use $$x$$
- Use:
  $$\begin{tabular}{| *a b* |} (Where *a b* are how you want the column text aligned, and | create a border where wanted)
  \toprule (creates line at top of table)
  Title1 & Title2 \\
  \hline (creates line between title and info)
  Thing & Description \\
  \toprule (Use for bottom line too) \end{tabular}$$
- Example:

Title1	*Title2*
Thing	*Description*

Mathematical symbols

Type	Symbol
\infty	∞
\pm	$\pm$
\ne	$\neq$
\le	$\leq$
\ge	$\geq$
\sim	$\sim$
a^{b}	a^{b}
a_{b}	a_{b}
\bar{a}	$\bar{a}$
\overline{a}	$\overline{a}$
\hat{a}	$\hat{a}$
\tilde{a}	$\tilde{a}$
\sqrt{a}	$\sqrt{a}$
\frac{a}{b}	$\frac{a}{b}$
\sum^a_b	$\sum^{a}_{b}$
\int^a_b	$\int^{a}_{b}$
{a \choose b} OR \binom{a}{b}	$\binom{a}{b}$ OR $\binom{a}{b}$
\cap	$\cap$
\cup	$\cup$
\subset	$\subset$
\subseteq	$\subseteq$

Greek symbols

Type	Symbol
\alpha	α
\beta	β
\gamma / \Gamma	γ/Γ
\delta / \Delta	δ/Δ
\epsilon / \varepsilon	ϵ/ε
\zeta	ζ
\eta	η
\theta / \Theta / \vartheta	$\theta/\Theta/\vartheta$
\iota	ι
\kappa / \varkappa	$\kappa/\varkappa$
\lambda / \Lambda	λ/Λ
\mu	μ
\nu	ν
\xi / \Xi	ξ/Ξ
\pi / \Pi / \varpi	$\pi/\Pi/\varpi$
\rho / \varrho	ρ/ϱ
\sigma / \Sigma / \varsigma	$\sigma/\Sigma/\varsigma$
\tau	τ
\upsilon / \Upsilon	υ/Υ
\phi / \Phi / \varphi	$\phi/\Phi/\varphi$
\chi	χ
\psi / \Psi	ψ/Ψ
\omega / \Omega	ω/Ω

Other symbols

Type	Symbol
\checkmark	✓
\boxed{a}	$\boxed{a}$