Week 1 Summary

Elijah Luo

10/07/2020

Table of Contents

Monday

Morning

Intro to R

Basics:

- R studio requires accurate typing, word for word
- Run codes first after entering

Variables:

- Numerical: A <- 2, means "A" gets value 2, where A is the variable

```
A <-2
```

- Text: city <- "Adelaide", means "city" gets "Adelaide", where city is the variable and there has to be quotation marks around the text, such as "text".
- Important: the variables can be named as anything, therefore it is important to give it a special/specific name.

Vectors:

- No matter numerical or text, c( ) needs to be employed to create/ start the vector, such as
  - Vec.num <- c(1,3,2,7,9,15), or
  - Vec.txt <-c("wheat", "barley", "oats"). Remember the quotation mark in text!!!
- Use seq( ) to create a sequence of values.
  - Vec.seq<- seq(1,21,3), means that a sequence from 1 to 21 with a jump of 3. Sequence has to be numerical. It also cannot be a vector.
- Use rep( ) to create repeat values. Repetition works on both single number and vectors, and they can be both numerical and text.
  - Vec.rep<- rep(c(1,6), times=5) means that vector (1,6) – the vector is repeated 5 times.
  - Vec.rep<-(c(1,6), each=5) means that each element of the vector is repeated 5 times according to their position in the vector.

Referencing and subsetting:

- Vector vec.num<- c(5, 1, 8, 0, 1). 5 is in 1st, 1 is in 2nd, 8 is in 3rd, 0 is in 4th, 1 is in 5th position.
- Vector name[x] can be used to represent the element in the xth position of that vector
- Vec.num[2] represents the 2nd element in the vector, which is 1.

- Vec.num[1,3] represents the 1st and 3rd element in the vector which is (5, 8).
- Vec.num[-2] represents all the element in the vector except the 2nd, which is (5, 8, 0, 1)
- Vec.num[1:4] represent the 1st to the 4th elements, which is (5, 1, 8, 0)
- Vec.num[6] is non-exist because there is no 6th element, hence NA.
- Vec.num[2]<-4 means assigning 4 to the 2nd position of the vector to replace 1.
- The mathematical operations in R studio only works on numbers, and those are of the same type and length.

Functions:

- Mean (x) = mean of x
- Var (x) = variance
- Max (x) = max value of vector x
- Min (x) = min value of vector x
- Length (x) = length of x (how many elements)
- Sum (x) = sum of all elements of vector x
- Sort (x) = sort
- Class (x) = class of x
- Sd (x) = standard deviation of x

Some of the functions don't work with NA

However, using na.rm=TRUE to ignore NA values, such as mean(b, na.rm=True)

- When install functions, these steps must be followed: 1. Install.packages ("MASS") with quotation marks, 2. Library (MASS) without quotation marks.
- When giving out comparison commands, such as what value > 5:
  - Which(x>5) means to give the position of elements >5
  - X[x>5] means to give the values pf elements >5

Afternoon

- Practice coding from the book

Tuesday

Morning

Intro to R

Data Frame:

- A table structure containing rows can columns. Rows – observations. Columns – values of different variables.
- Columns can contain all types of data. E.g. first column numbers, second column tests, third columns logics. They MUST be the same length.
- Example:

```
a<-1:4
b<-c("Dog", "Observation 2", "Parachute", "Singapore")
c<-c(TRUE, TRUE, FALSE, NA)
```

- The data frame (mydf) can get: mydf <- data.frame(a, b, c):

```
mydf<-data.frame(a,b,c)
mydf

  a             b     c
1 1           Dog  TRUE
2 2 Observation 2  TRUE
3 3     Parachute FALSE
4 4     Singapore    NA
```

- The columns can be renamed into something meaningful by using colnames(mydf).

```
colnames(mydf)<-c("NUmbers", "Words", "Boolean")
mydf

  NUmbers         Words Boolean
1       1           Dog    TRUE
2       2 Observation 2    TRUE
3       3     Parachute   FALSE
4       4     Singapore      NA
```

- R has built in data frames and they can be accessed by type in the name of the data frame then run it.
  - head/tail(iris) will show the first/last 6 rows of the iris data frame

```
head(iris)

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
```

```
5          5.0          3.6          1.4          0.2   setosa
6          5.4          3.9          1.7          0.4   setosa
```

```
tail(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

- summary(iris) will show the summary of the data frame

```
summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
       Species  
 setosa    :50  
 versicolor:50  
 virginica :50  
```

Referencing and Factors:

- Elements can be accessed as matrices by using [x,y]
  - mydf [,1], means giving column 1
  - mydf [1,], means giving row 1
  - mydf [1,1], means giving row 1, column 1
- The column name can be accessed by using the $ function:
  - Mydf$Words, meaning giving the Words column, and also giving out the Levels.
- Levels are the categories of FACTORS only, so convert text to factors first.
  - str (iris) gives information about columns in the iris data frame

```
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

```
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1
1 1 ...
```

- iris[c(1:3, 148:150),] gives the top and the bottom of the data frame (remember the comma)

```
iris[c(1:3, 148:150),]

    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1            5.1         3.5          1.4         0.2    setosa
2            4.9         3.0          1.4         0.2    setosa
3            4.7         3.2          1.3         0.2    setosa
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

- Levels can be checked and the factor labels can be changed using levels ( ).
- levels(iris$Species) will give out the label of species in the iris data frame

```
levels(iris$Species)

[1] "setosa"     "versicolor" "virginica"
```

- levels(iris$Species) [1] <- “spuria” will replace the first label of the data frame setosa to spuria.

```
levels(iris$Species) [1] <- "spuria"
levels(iris$Species)

[1] "spuria"     "versicolor" "virginica"
```

- levels(iris*Species*)[*levels*(*iris*Species)== “spuria”] <- “setosa” means to find the labels that have been changed and to change them back.

```
levels(iris$Species)[levels(iris$Species)== "spuria"] <- "setosa"
levels(iris$Species)

[1] "setosa"     "versicolor" "virginica"
```

Reading in Data

- Give the data a name first (the data has to be saved in the same directory as the R files)
  - such as fiber <- read.table(file= “Energydigestability1.csv”, header=T, sep= ‘,’)
  - then run str(fiber)
- Can read directly from excel file:
  - require:

```
library(readxl)
```

- then use fibre <- read.xlsx("Data/Energydigestabilty1.xlsx")
- str(fibre)
- Data sets can also ben saved to computer from R
  - Tables
    - write. table (x=fiber, file = "Data/Energydigestibility1.csv")
    - write. csv (x=fiber, file = "Data/Energydigestibility1.csv")

More Advance Functions

- To see which variables are correlated, cor() can be used
  - cors<-cor(iris[,-5]) Means to exclude the column with Species

```
cors<-cor(iris[,-5])
cors

             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

- The class of the dataset indicates that type of the data
  - class(cors) can be matrix or array

```
class(cors)

[1] "matrix" "array"
```

- Linear model of the highly correlated variables can be calculated by using the lm( ) function.
- Variables can be used against each other by using the ~ symbol, y variable is on the left side of (~) and x is on the right side. It is read as "model y against x", which is "model Petal.Length against Petal.Width".
- model <- lm(formula=Petal.Length~Petal.Width, data=iris)

```
model <- lm(formula=Petal.Length~Petal.Width, data=iris)
model


Call:
lm(formula = Petal.Length ~ Petal.Width, data = iris)

Coefficients:
(Intercept)  Petal.Width
      1.084        2.230
```

- str(model) gives the structure of the model.
- names(model) gives the name of variables in the list containing information
- summary(model) gives the summary of variables in the list.
- T-test
    - t.test(1:10, y = 7:20, var.equal = TRUE) variance assumed different by default

```
t.test(1:10, y = 7:20, var.equal = TRUE)


    Two Sample t-test

data:  1:10 and 7:20
t = -5.1473, df = 22, p-value = 3.691e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.223245  -4.776755
sample estimates:
mean of x mean of y
      5.5      13.5
```

- Two Sample t-test t.test(1:10, y = c(7:20, 200))

```
t.test(1:10, y = c(7:20, 200))


    Welch Two Sample t-test

data:  1:10 and c(7:20, 200)
t = -1.6329, df = 14.165, p-value = 0.1245
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -47.242900   6.376233
sample estimates:
mean of x mean of y
  5.50000  25.93333
```

- Welch Two Sample t-test test <- t.test (extra~group, data = sleep, mu=1), mu is 0 by default

```
test <- t.test (extra~group, data = sleep, mu=1)
test


    Welch Two Sample t-test

data:  extra by group
t = -3.0385, df = 17.776, p-value = 0.007141
alternative hypothesis: true difference in means is not equal to 1
95 percent confidence interval:
```

```
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2
           0.75            2.33
```

Functions in the Apply Family

- tapply(X = iris$Petal.Length, INDEX = iris$Species, FUN = mean) gives the mean of each group

```
tapply(X = iris$Petal.Length, INDEX = iris$Species, FUN = mean)

    setosa versicolor  virginica 
     1.462      4.260      5.552 
```

Graphics

- See in book

Afternoon

- Doing exercise

Wednesday

Morning

Experimental Design

Population and Samples

- A population is a group of subjects the experimental results hope to apply

- A sample is a part of the population selected to reflect properties of the population and the statistical influence made about the population to be secure

Treatments

- A treatment is something the researchers administer to the experimental units to investigate if the treatment has an effect to the outcome or variable of interest

Experimental and Observational Units

- Experimental unit is the smallest units of a treatment to be randomised (plot of carrots)

- Observational unit is the unit which measurements are made (carrot in the plot)

- Experimental unit and observational unit may or may not be the same

Replication

- The number of times that each treatment is tested in an experiment. It allows us to estimate the variability of each treatment. More information results less variability and greater precision

Pseudo-replication

- Treatments are not replicated (though samples maybe) or the replicates are not statistically independent. Generally this involve making multiple measurements on experimental units and treating them as they reflected independent responses to treatment.

Blocking

- Experimental units are divided into blocks because they are more alike than units from other blocks

Randomisation

- Does not mean haphazard or unplanned
- Defined as all units have an equal probability of receiving any of the treatments

Confounding

- A situation where the effect of two factors cannot be separated from each other
- To avoid by identifying all confounding variables (make a list), the day of planting do not count.

Heterogeneity

- Experimental units are better to be reasonably uniform in their natural response (homogenous) as it decreases the estimate of the background variation. Therefore, if the experimental units are intrinsically different, then the experiment is more likely to be insensitive.

##$$ Factors and Levels

- Factor – qualitative
- Levels – quantitative

Main Effect and Interactions

- If not parallel on the graph, there might be minor interactions. Tests need to be done to find out, though it may not be significant.
- If crossing over, the interaction is significant

Afternoon

Experimental Design

Designs

- In order to do designs:

```r
library(BiometryTraining)
library(agricolae)
library(ggplot2)
```

- Randomising numbers:

sample()

```
sample(1:10) # means randomises the numbers 1 to 10

 [1]  2  5  6  9  7  8 10  1  4  3

sample(1:10, 2) # means randomly selects two numbers between 1 to 10

[1] 2 9
```

- When creating a design, is it important to know the: Aim: what are you investigating/determining Observation: the sample units and size Arrangements: rows and columns, blocks, and the structures within the block Replicates: how many times are each treatment repeated Design: what type of design suits best

- Residue df greater than 12 is enough to test for variation.

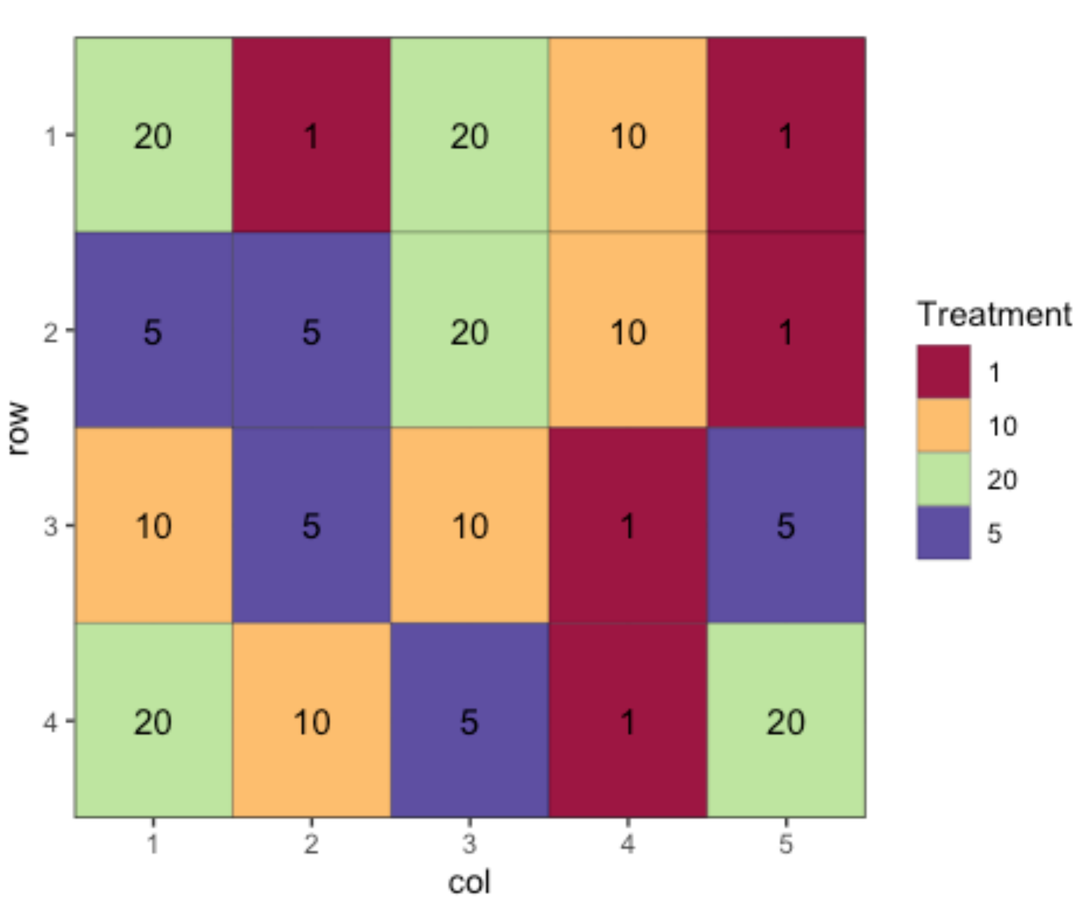
Completely Randomised Design

- The simplest form of statistical design

- Best used when the experimental units are unstructured and homogeneous

- Example:

```
library(BiometryTraining)
library(agricolae)
trt<-c(1,5,10,20) # the treatment levels
rep<-5 # replications
outdesign<-design.crd(trt, r=rep) # create design
design.out<-des.info(design.obj = outdesign, nrows = 4, ncols =5)

Source of Variation                         df
 =============================================
 trt                                         3
 Residual                                    16
 =============================================
 Total                                       19
```



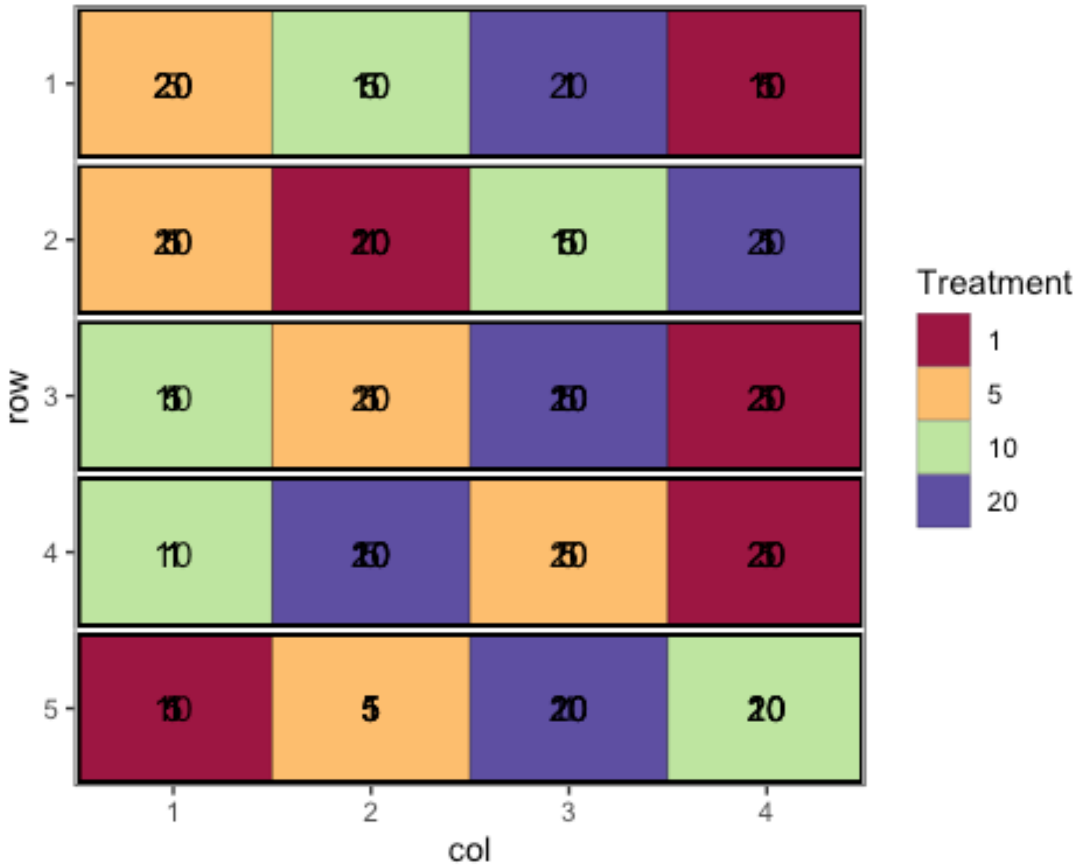
write.csv(des.out$design, “design file name.csv”, row.names=FALSE) – save in csv

Randomised Complete Block Design

- Number of each experimental units in each block must equal to the number of treatments
- Treatments are randomly allocated within each block
- Only to be used when the number of treatment is equal to block size. E.g. there are 11 treatments, the in the experimental design, there must be 11 rows in 1 column.
- Example:

```
library(BiometryTraining)
library(agricolae)
trt<-c(1,5,10,20) # the treatment levels
rep<-20  # replications
outdesign<-design.rcbd(trt, r=rep)  # create design
design.out<-des.info(design.obj = outdesign, nrows = 5, ncols =4,
brows = 1, bcol = 4)

Source of Variation                        df
 =============================================
 Block stratum                              19
```

```
---------------------------------------------
trt                                        3
Residual                                   57
=============================================
Total                                      79
```



write.csv(des.out$design, “design file name.csv”, row.names=FALSE) – save in csv

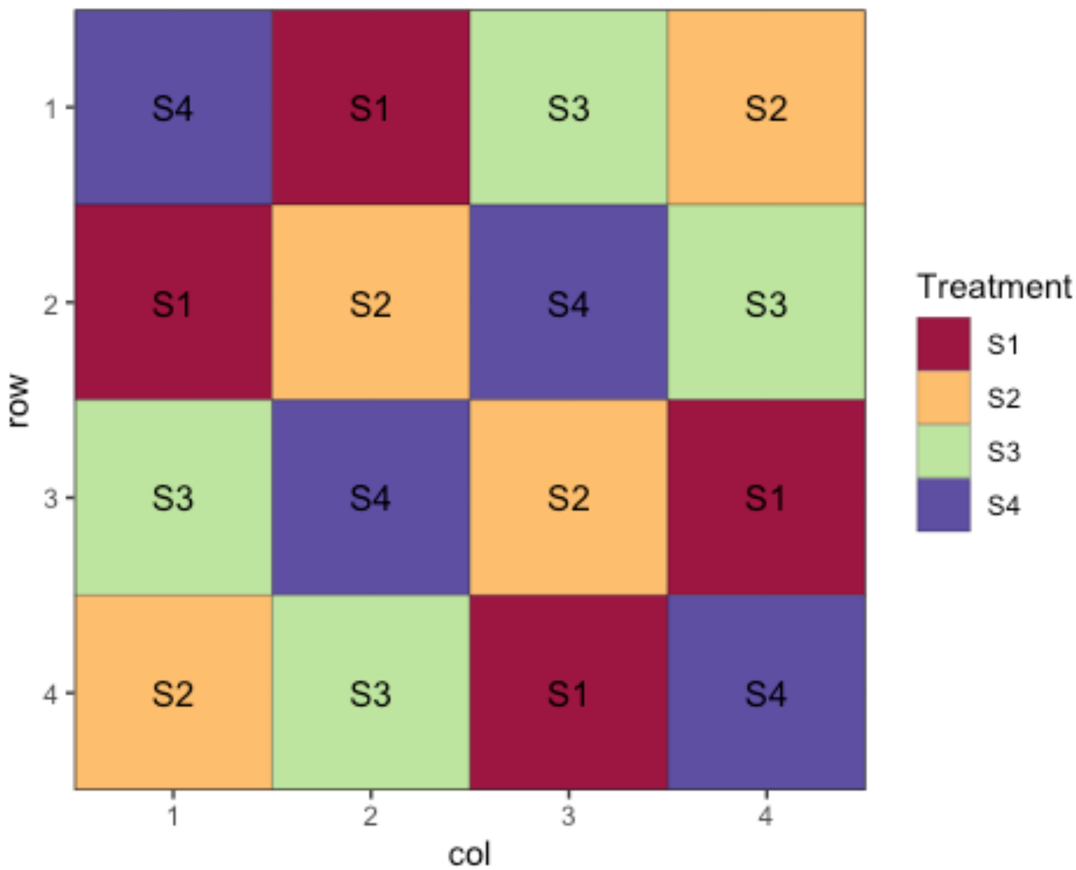
Latin Square

- The numbers of rows and columns have to respond to the treatment levels. E.g. 4 levels, 4 rows and 4 columns
- Include two blocking factors to reduce unexplained variations in response variable
- Example:

```
trt<- c("S1", "S2", "S3", "S4")
outdesign<- design.lsd(trt)
des.out<-des.info(design.obj = outdesign, nrows = 4, ncols = 4)
```

```
Source of Variation                        df
 =============================================
 Row                                        3
 Column                                     3
```

```
trt                                        3
Residual                                   6
===============================================
Total                                      15
```



Thursday

Morning

Experimental Design

Treatment Structure

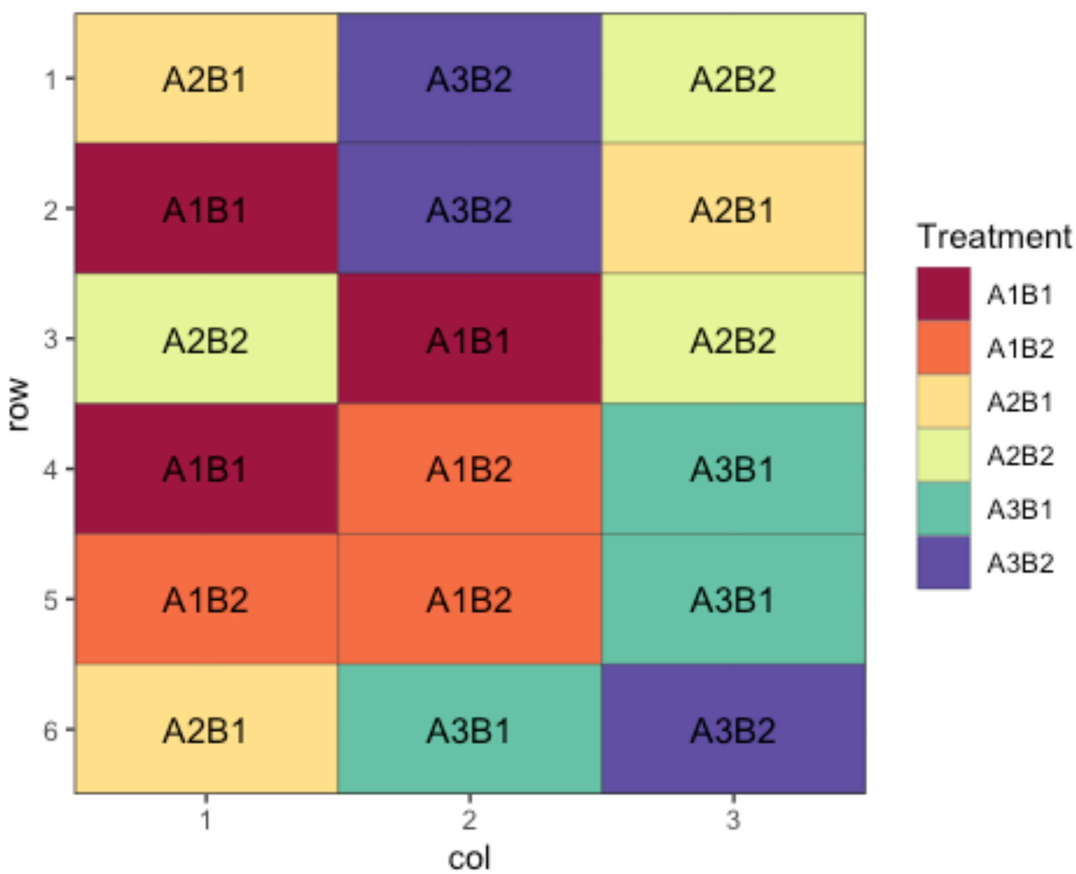
- Crossed (experimental units receiving all treatments)
- Nested (experimental units receiving a single treatment)

Creating the Design

- Write down these first: Aim (what to investigate) Observations (sample size) Arrangements (rows ⍰ columns) Treatments (numbers of factors and their levels, total number of treatments is calculated by number of factors ⍰ number of levels) Replicates (observations ⍰ total number of treatments) Design: what type of design

- It is good to draw the design out on paper before programming in R
- Example for randomised design:

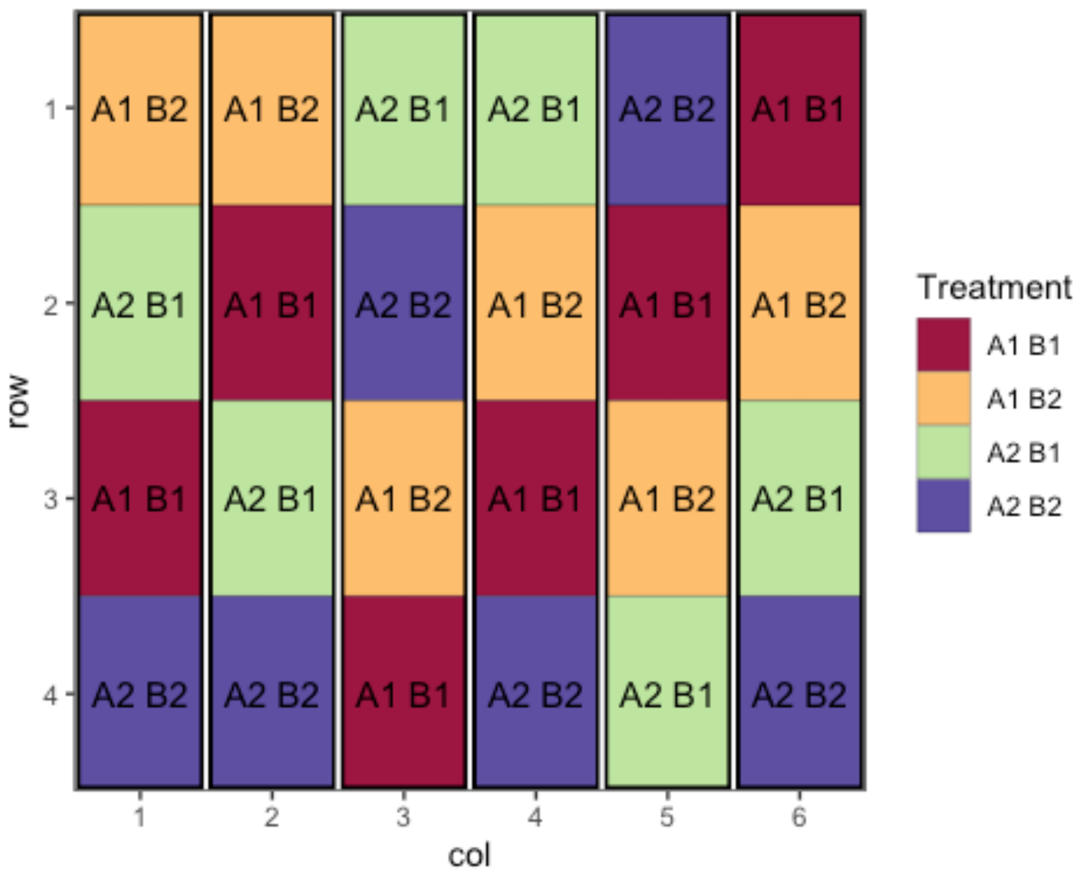
```
trt <- c(3,2) # factorial 3*2
rep <- 3 # replication 3
outdesign <-design.ab(trt, r=rep, design ="crd")
des.out <- des.info(design.obj = outdesign, nrows = 6, ncols =3)

Source of Variation                     df
 =============================================
 A                                       2
 B                                       1
 AB                                      2
 Residual                                12
 =============================================
 Total                                   17
```



- Example for completely randomised blocked design:

```
trt <- c(2,2) # factorial 2�2
rep <- 6 # replication 6
outdesign <-design.ab(trt, r=rep, design = "rcbd")
des.out <- des.info(design.obj = outdesign, nrows = 4, ncols =6, brows = 4,
bcols = 1)
```

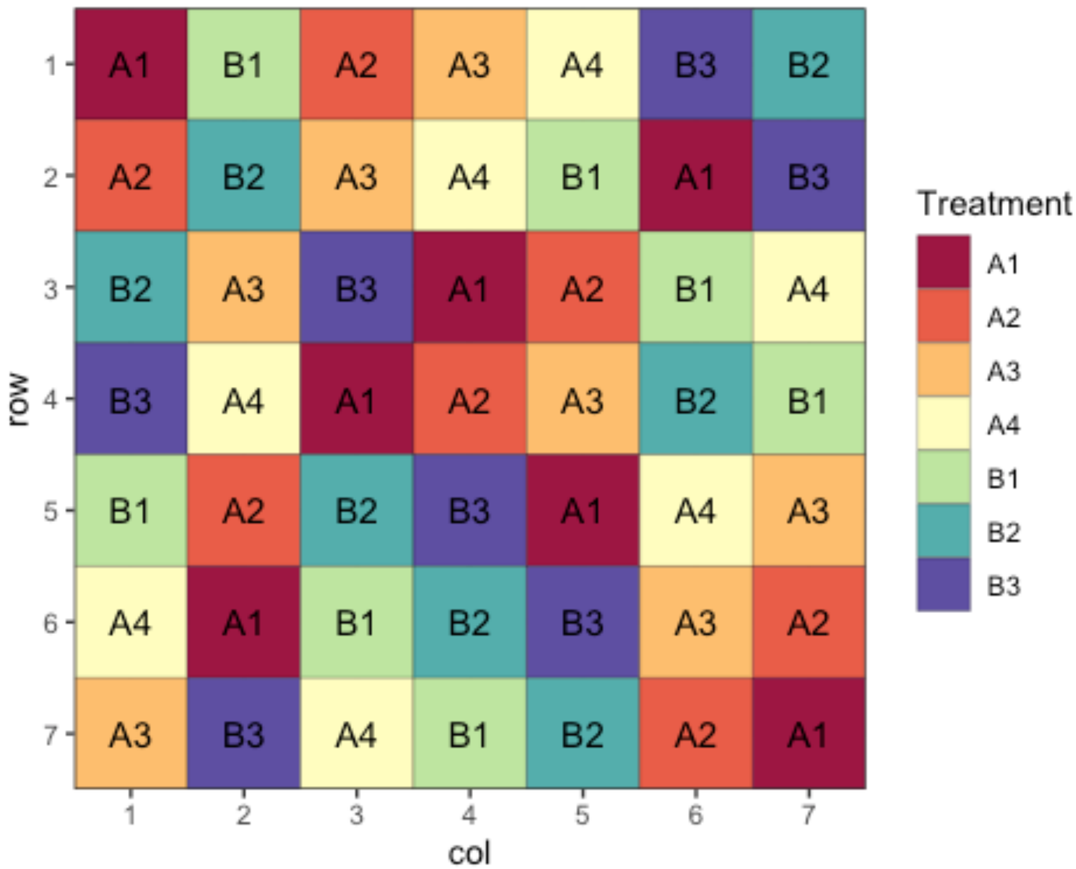
```
Source of Variation                          df
 =============================================
 Block stratum                                5
 ---------------------------------------------
 A                                            1
 B                                            1
 AB                                           1
 Residual                                     15
 =============================================
 Total                                        23
```



- Example for Latin square design:

```
trt <- c("A1", "A2", "A3", "A4", "B1", "B2", "B3")
outdesign <- design.lsd(trt)
des.out<-des.info(design.obj = outdesign, nrows = 7, ncols = 7)

Source of Variation                          df
 =============================================
 Row                                          6
 Column                                       6
 trt                                          6
 Residual                                     30
```

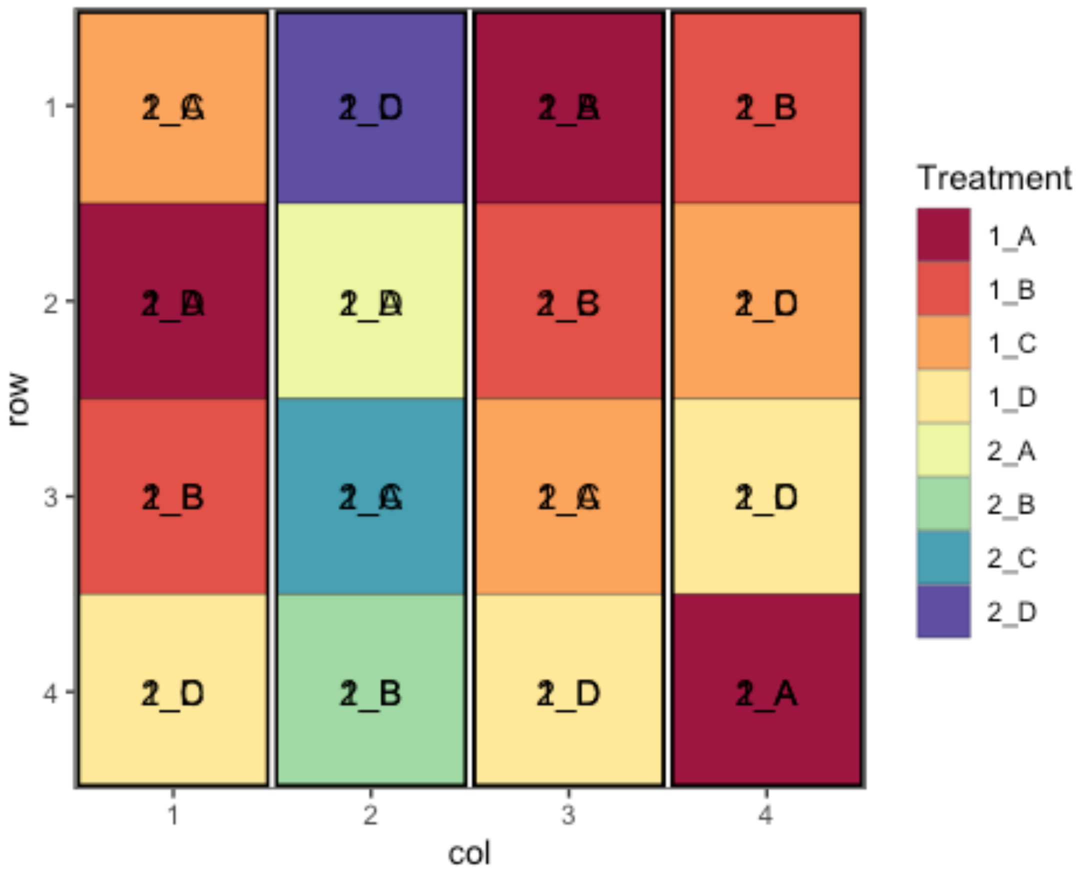
```
=============================================
Total                                       48
```



- Example for split-plot design:

```
wholeplot<- c(1,2)
subplot<- c("A", "B", "C", "D")
outdesign <- design.split(wholeplot, subplot, r=4)
des.out <- des.info(design.obj = outdesign, nrows = 8, ncols = 4, brows = 4,
bcols =1)

Source of Variation                                df
 ==================================================
 Block stratum                                      3
 --------------------------------------------------
 Whole plot stratum
          wholeplot                                1
 Whole plot Residual                                3
 ==================================================
 Subplot stratum
          subplot                                  3
          wholeplot:subplot                        3
          Subplot Residual                         18
```

```
=====================================================
Total                                         31
```



Residual Degrees of Freedom

- Be careful when choosing an experimental design, as more complexed design will decrease the residual degrees of freedom, which will then decrease the power of the experiment to detect variations.

Code Functions to Systematically Solve Programming Problems

- Don't do these when coding:
  - Writing line by line
  - Don't try to keep track and remember everything
- Manage complexity is the most important technical topic in software development.
- Having logical steps is the key
- The quickest way to do it is to code your functions, which is included in every programming language
- Using syntax in R to code functions

- Get the skeleton of coding then fill in the blanks
- Order:

  1. Read first
  2. Sort
  3. Compute the score

  a. The alphabetical value
  b. Sum up all the alphabet order numbers to get alphabetical value

  4. Multiply
  5. Sum up total

Friday

Morning

Pete’s Talk

Tibble Table

- Install the required packages first

```
require(tidyverse)

Loading required package: tidyverse

── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

✓ tibble  3.0.1     ✓ dplyr   1.0.0
✓ tidyr   1.1.0     ✓ stringr 1.4.0
✓ readr   1.3.1     ✓ forcats 0.5.0
✓ purrr   0.3.4

── Conflicts ──────────────────────────────────────────────── tidyverse_conflicts() ──
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()

require(xtable)

Loading required package: xtable
```

- Exaample:

```
soils<-tibble(x=rnorm(5), y=rnorm(5))
weather<-tibble(x=rnorm(5))
```

```
field<-cbind(soils, weather)
colnames(field)<-c("Rye","Barley","Rain_mm")
field

         Rye    Barley    Rain_mm
1 -0.8961443  2.3518245 -1.2965044
2  1.4314173  0.8125635 -0.3589246
3 -0.5869627 -0.4200624  0.8507829
4 -0.6202802 -1.4888224 -0.6415131
5  1.1329231 -0.4203724  0.2982968

xtable(field)

% latex table generated in R 4.0.2 by xtable 1.8-4 package
% Fri Jul 24 08:54:53 2020
\begin{table}[ht]
\centering
\begin{tabular}{rrrr}
  \hline
 & Rye & Barley & Rain\_mm \\
  \hline
1 & -0.90 & 2.35 & -1.30 \\
  2 & 1.43 & 0.81 & -0.36 \\
  3 & -0.59 & -0.42 & 0.85 \\
  4 & -0.62 & -1.49 & -0.64 \\
  5 & 1.13 & -0.42 & 0.30 \\
   \hline
\end{tabular}
\end{table}

field<-as.data.frame(field)
```

Subsection (Functions)

- Define it once, then call it multiple times throughout the code
- They all follow the same pattern
- Example

```
require(tidyverse)
require(xtable)

x<-4
y<-6

SN<- function(x,y){
  add_small<-x+2
  add_big<-y+5
  value<-add_small+add_big

return(value)
```

```
}

SN(x,y)

[1] 17
```

Subsection (Tidy Data)

- 3 rules have to be followed for data to be tidy
    1. Each variable must have its own column
    2. Each observation must have its own row
    3. Each value must have its own cell
- Look into the help files, it will let you know if it will take a tibble object or it requires a data frame, which is usefull when tranforming untidy data into tidy data
- Different plots require data in different formats

Subsection (Pipe)

- Can be read as "then" when reading code
- String multiple functions together, THEN forwards the value or results of the expression to the next function

Subsection (Verb: Gathering)

- Gather data that is unstacked and spread out across columns
    - gather_table <- table2 %>% gather(1999, 2000, key = "year", value = "cases")
    - xtable(gather_table,caption = "The table has been re-arranged")

Subsection (Verb: Spreading)

- The opposite of gathering

```
spread_table <- spread(table2, key = type, value = count)

spread_table

# A tibble: 6 x 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

Subsection (Separating)

- Table 3 has 2 variables - cases and population. Using separate functions, these two columns can be split
- Will split the string at first avalible place
- Use "conver = TRUE" to change the format of the column from character to number
- Use "sep" to separate long lists of digits

```
table3 %>%  separate(rate, into = c("cases", "population"))

# A tibble: 6 x 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583

table3 %>%  separate(rate, into = c("cases", "population"),sep = "/")

# A tibble: 6 x 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583

table3 %>% separate(rate, into = c("cases", "population"), convert = TRUE)

# A tibble: 6 x 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583

table3 %>%separate(year, into = c("century", "year"), sep = 2)

# A tibble: 6 x 4
  country     century year  rate
  <chr>       <chr>   <chr> <chr>
1 Afghanistan 19      99    745/19987071
```

```
2 Afghanistan 20      00    2666/20595360
3 Brazil      19      99    37737/172006362
4 Brazil      20      00    80488/174504898
5 China       19      99    212258/1272915272
6 China       20      00    213766/1280428583
```

Subsection (unite)

- The opposite of separate

```
table5 %>% unite(new,century,year) #give it a underscore

# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <chr>
1 Afghanistan 19_99 745/19987071
2 Afghanistan 20_00 2666/20595360
3 Brazil      19_99 37737/172006362
4 Brazil      20_00 80488/174504898
5 China       19_99 212258/1272915272
6 China       20_00 213766/1280428583

table5 %>% unite(new,century,year,sep = "") #""means separate things..

# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <chr>
1 Afghanistan 1999  745/19987071
2 Afghanistan 2000  2666/20595360
3 Brazil      1999  37737/172006362
4 Brazil      2000  80488/174504898
5 China       1999  212258/1272915272
6 China       2000  213766/1280428583

table5 %>% unite(new,century,year,sep = "%") #separate things with %

# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <chr>
1 Afghanistan 19%99 745/19987071
2 Afghanistan 20%00 2666/20595360
3 Brazil      19%99 37737/172006362
4 Brazil      20%00 80488/174504898
5 China       19%99 212258/1272915272
6 China       20%00 213766/1280428583
```

Subsection (Missing Values)

```
stocks <- tibble(
  year   = c(2015, 2015, 2015, 2015, 2016, 2016, 2016),
  qtr    = c(   1,    2,    3,    4,    2,    3,    4),
  return = c(1.88, 0.59, 0.35,   NA, 0.92, 0.17, 2.66))
```

```
stocks %>% spread(year,return) %>%
  gather(year,return,`2015`:`2016`,na.rm=TRUE) #use na.rn =T to stop the
effect of na

# A tibble: 6 x 3
    qtr year  return
  <dbl> <chr>  <dbl>
1     1 2015    1.88
2     2 2015    0.59
3     3 2015    0.35
4     2 2016    0.92
5     3 2016    0.17
6     4 2016    2.66

stocks %>% complete(year, qtr)

# A tibble: 8 x 3
   year   qtr return
  <dbl> <dbl>  <dbl>
1  2015     1   1.88
2  2015     2   0.59
3  2015     3   0.35
4  2015     4  NA
5  2016     1  NA
6  2016     2   0.92
7  2016     3   0.17
8  2016     4   2.66
```

- Function complete () takes a set of columns and get rid of all non complete rows. Be careful, do not throw data away

```
df <- tibble(
  group = c(1:2, 1),
  item_id = c(1:2, 2),
  item_name = c("a", "b", "b"),
  value1 = 1:3,
  value2 = 4:6
)
df %>% complete(group, nesting(item_id, item_name))

# A tibble: 4 x 5
  group item_id item_name value1 value2
  <dbl>   <dbl> <chr>      <int>  <int>
1     1       1 a              1      4
2     1       2 b              3      6
3     2       1 a             NA     NA
4     2       2 b              2      5

# You can also choose to fill in missing values
df %>% complete(group, nesting(item_id, item_name), fill = list(value1 = 0))
```

```
# A tibble: 4 x 5
  group item_id item_name value1 value2
  <dbl>   <dbl> <chr>      <dbl>  <int>
1     1       1 a              1      4
2     1       2 b              3      6
3     2       1 a              0     NA
4     2       2 b              2      5
```

- can find all missing values and replace them with NAs.
- To remove NAs, use na.rm=TRUE inside the function

Subsection (Filter)

- Allow you to pick observations by their values
  - First argument: data frame
  - Second argument: expressions that filters the data frame

```
library(nycflights13)


Attaching package: 'nycflights13'

The following object is masked _by_ '.GlobalEnv':

    weather

flights

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1      517            515         2      830
819
 2  2013     1     1      533            529         4      850
830
 3  2013     1     1      542            540         2      923
850
 4  2013     1     1      544            545        -1     1004
1022
 5  2013     1     1      554            600        -6      812
837
 6  2013     1     1      554            558        -4      740
728
 7  2013     1     1      555            600        -5      913
854
 8  2013     1     1      557            600        -3      709
723
```

```
 9  2013     1     1      557            600        -3      838
846
10  2013     1     1      558            600        -2      753
745
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
<dttm>

filter(flights, month == 1, day == 1) #give everything where month = 1 and
day = 1, use ==

# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1      517            515         2      830
819
 2  2013     1     1      533            529         4      850
830
 3  2013     1     1      542            540         2      923
850
 4  2013     1     1      544            545        -1     1004
1022
 5  2013     1     1      554            600        -6      812
837
 6  2013     1     1      554            558        -4      740
728
 7  2013     1     1      555            600        -5      913
854
 8  2013     1     1      557            600        -3      709
723
 9  2013     1     1      557            600        -3      838
846
10  2013     1     1      558            600        -2      753
745
# … with 832 more rows, and 11 more variables: arr_delay <dbl>, carrier
<chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>

filter(flights, arr_time < 100)

# A tibble: 6,906 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1     1929           1920         9        3
7
```

```
 2  2013     1     1     1939           1840        59       29
2151
 3  2013     1     1     2058           2100        -2        8
2359
 4  2013     1     1     2108           2057        11       25
39
 5  2013     1     1     2120           2130       -10       16
18
 6  2013     1     1     2121           2040        41        6
2323
 7  2013     1     1     2128           2135        -7       26
50
 8  2013     1     1     2134           2045        49       20
2352
 9  2013     1     1     2136           2145        -9       25
39
10  2013     1     1     2157           2155         2       43
41
# … with 6,896 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
<dttm>
```

```
filter(flights, month == 1 & day == 1 & dep_delay == -1)
```

```
# A tibble: 57 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1      544            545        -1     1004
1022
 2  2013     1     1      559            600        -1      941
910
 3  2013     1     1      559            600        -1      854
902
 4  2013     1     1      629            630        -1      824
810
 5  2013     1     1      629            630        -1      721
740
 6  2013     1     1      629            630        -1      824
833
 7  2013     1     1      639            640        -1      739
749
 8  2013     1     1      659            700        -1     1008
1015
 9  2013     1     1      659            700        -1     1008
1007
10  2013     1     1      659            700        -1      959
1008
```

```
# … with 47 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Subsection (Arrange)

- Similar to filter, but instead of filtering, it reaarange them by columns in descending order

```
arrange(flights, year, month, day)

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1      517            515         2      830
819
 2  2013     1     1      533            529         4      850
830
 3  2013     1     1      542            540         2      923
850
 4  2013     1     1      544            545        -1     1004
1022
 5  2013     1     1      554            600        -6      812
837
 6  2013     1     1      554            558        -4      740
728
 7  2013     1     1      555            600        -5      913
854
 8  2013     1     1      557            600        -3      709
723
 9  2013     1     1      557            600        -3      838
846
10  2013     1     1      558            600        -2      753
745
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
<dttm>

arrange(flights, desc(arr_delay), month)

# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     9      641            900      1301     1242
1530
 2  2013     6    15     1432           1935      1137     1607
2120
```

```
 3  2013     1    10     1121            1635      1126     1239
1810
 4  2013     9    20     1139            1845      1014     1457
2210
 5  2013     7    22      845            1600      1005     1044
1815
 6  2013     4    10     1100            1900       960     1342
2211
 7  2013     3    17     2321             810       911      135
1020
 8  2013     7    22     2257             759       898      121
1026
 9  2013    12     5      756            1700       896     1058
2020
10  2013     5     3     1133            2055       878     1250
2215
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
<dttm>
```

Subsection (Select)

- Allows you to select specific column variables in a data set AND create a new subset
- Other functions that can be used in select() are ends_with("xyz"), contains("ijk"), which matches variables that contain specified characters.

```
select(flights, year, month, day)

# A tibble: 336,776 x 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# … with 336,766 more rows

select(flights, -(year:day))

# A tibble: 336,776 x 16
   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
carrier
      <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
 1      517            515         2      830            819        11 UA
```

```
 2      533            529         4      850            830         20 UA
 3      542            540         2      923            850         33 AA
 4      544            545        -1     1004           1022        -18 B6
 5      554            600        -6      812            837        -25 DL
 6      554            558        -4      740            728         12 UA
 7      555            600        -5      913            854         19 B6
 8      557            600        -3      709            723        -14 EV
 9      557            600        -3      838            846         -8 B6
10      558            600        -2      753            745          8 AA
# … with 336,766 more rows, and 9 more variables: flight <int>, tailnum
<chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

```
select(flights,starts_with("Ar")) #can be case sensitive
```

```
# A tibble: 336,776 x 2
   arr_time arr_delay
      <int>     <dbl>
 1      830        11
 2      850        20
 3      923        33
 4     1004       -18
 5      812       -25
 6      740        12
 7      913        19
 8      709       -14
 9      838        -8
10      753         8
# … with 336,766 more rows
```

```
select(flights, time_hour, air_time, everything())
```

```
# A tibble: 336,776 x 19
   time_hour           air_time  year month   day dep_time sched_dep_time
   <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
 1 2013-01-01 05:00:00      227  2013     1     1      517            515
 2 2013-01-01 05:00:00      227  2013     1     1      533            529
 3 2013-01-01 05:00:00      160  2013     1     1      542            540
 4 2013-01-01 05:00:00      183  2013     1     1      544            545
 5 2013-01-01 06:00:00      116  2013     1     1      554            600
 6 2013-01-01 05:00:00      150  2013     1     1      554            558
 7 2013-01-01 06:00:00      158  2013     1     1      555            600
 8 2013-01-01 06:00:00       53  2013     1     1      557            600
 9 2013-01-01 06:00:00      140  2013     1     1      557            600
10 2013-01-01 06:00:00      138  2013     1     1      558            600
# … with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
#   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
#   hour <dbl>, minute <dbl>
```

```
select(flights, year:day, ends_with("delay"), distance, air_time)

# A tibble: 336,776 x 7
    year month   day dep_delay arr_delay distance air_time
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
 1  2013     1     1         2        11     1400      227
 2  2013     1     1         4        20     1416      227
 3  2013     1     1         2        33     1089      160
 4  2013     1     1        -1       -18     1576      183
 5  2013     1     1        -6       -25      762      116
 6  2013     1     1        -4        12      719      150
 7  2013     1     1        -5        19     1065      158
 8  2013     1     1        -3       -14      229       53
 9  2013     1     1        -3        -8      944      140
10  2013     1     1        -2         8      733      138
# … with 336,766 more rows
```

Subsection (Mutate)

- Create new variables

```
#keeps the original columns
  select(flights, ends_with("delay"), distance, air_time) %>%
  mutate( gain = arr_delay - dep_delay,
          speed = distance / air_time * 60,
          new = gain + speed)

# A tibble: 336,776 x 7
   dep_delay arr_delay distance air_time  gain speed   new
       <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>
 1         2        11     1400      227     9  370.  379.
 2         4        20     1416      227    16  374.  390.
 3         2        33     1089      160    31  408.  439.
 4        -1       -18     1576      183   -17  517.  500.
 5        -6       -25      762      116   -19  394.  375.
 6        -4        12      719      150    16  288.  304.
 7        -5        19     1065      158    24  404.  428.
 8        -3       -14      229       53   -11  259.  248.
 9        -3        -8      944      140    -5  405.  400.
10        -2         8      733      138    10  319.  329.
# … with 336,766 more rows

#create new column
```

- Create new variable but not new columns

```
#opposite of mutate #doesnt keep original columns
select(flights, ends_with("delay"), distance, air_time) %>%
  transmute( gain = arr_delay - dep_delay,
          speed = distance / air_time * 60,
          new = gain + speed)
```

```
# A tibble: 336,776 x 3
    gain speed   new
   <dbl> <dbl> <dbl>
 1     9  370.  379.
 2    16  374.  390.
 3    31  408.  439.
 4   -17  517.  500.
 5   -19  394.  375.
 6    16  288.  304.
 7    24  404.  428.
 8   -11  259.  248.
 9    -5  405.  400.
10    10  319.  329.
# … with 336,766 more rows
```

Subsection (Group By)

```
flights %>% group_by(origin) %>%
  summarise(mean = mean(air_time,
                        na.rm=TRUE), median=median(air_time,na.rm=TRUE),
            variance = var(air_time,na.rm=TRUE)) %>%
  arrange(mean)

`summarise()` ungrouping output (override with `.groups` argument)

# A tibble: 3 x 4
  origin  mean median variance
  <chr>  <dbl>  <dbl>    <dbl>
1 LGA     118.    115    2440.
2 EWR     153.    130    8713.
3 JFK     178.    149   12949.
```

Subsection (Summarise)

- Provides initial summary statistic

```
flights %>% group_by(origin) %>%
  summarise(mean = mean(air_time,
                        na.rm=TRUE), median=median(air_time,na.rm=TRUE),
            variance = var(air_time,na.rm=TRUE)) %>%
  arrange(mean)

`summarise()` ungrouping output (override with `.groups` argument)

# A tibble: 3 x 4
  origin  mean median variance
  <chr>  <dbl>  <dbl>    <dbl>
1 LGA     118.    115    2440.
2 EWR     153.    130    8713.
3 JFK     178.    149   12949.
```

Subsection (Counts)

```
group_by(flights, carrier) %>%
summarise(n())
```

```
`summarise()` ungrouping output (override with `.groups` argument)

# A tibble: 16 x 2
   carrier `n()`
   <chr>   <int>
 1 9E      18460
 2 AA      32729
 3 AS        714
 4 B6      54635
 5 DL      48110
 6 EV      54173
 7 F9        685
 8 FL       3260
 9 HA        342
10 MQ      26397
11 OO         32
12 UA      58665
13 US      20536
14 VX       5162
15 WN      12275
16 YV        601
```

```
group_by(flights, carrier) %>%
mutate(n = n())
```

```
# A tibble: 336,776 x 20
# Groups:   carrier [16]
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1     1      517            515         2      830
819
 2  2013     1     1      533            529         4      850
830
 3  2013     1     1      542            540         2      923
850
 4  2013     1     1      544            545        -1     1004
1022
 5  2013     1     1      554            600        -6      812
837
 6  2013     1     1      554            558        -4      740
728
 7  2013     1     1      555            600        -5      913
854
 8  2013     1     1      557            600        -3      709
723
 9  2013     1     1      557            600        -3      838
846
10  2013     1     1      558            600        -2      753
```

```
745
# … with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour
<dttm>,
#   n <int>

group_by(flights, carrier) %>%
filter(n() < 100)

# A tibble: 32 x 19
# Groups:   carrier [1]
    year month   day dep_time sched_dep_time dep_delay arr_time
sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
<int>
 1  2013     1    30     1222           1115        67     1402
1215
 2  2013    11     3     1424           1430        -6     1629
1634
 3  2013    11    10     1443           1430        13     1701
1634
 4  2013    11    17     1422           1430        -8     1610
1634
 5  2013    11    25     1803           1759         4     2011
2017
 6  2013    11    30     1648           1647         1     1814
1811
 7  2013     6    15     1626           1635        -9     1810
1830
 8  2013     6    22     1846           1635       131     2107
1830
 9  2013     8    27     1755           1805       -10     1956
1953
10  2013     8    28     2039           1805       154     2213
1953
# … with 22 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Afternoon

Sam’s Talk

RMarkDown

- Seen in the “Reporting Research with Rmarkdown”