R workshop

Dessie Petrova

Gerard Cardoso

Mariflor Vega

Agenda

START	FINISH	
10:15	10:30	Intro R studio ()
10:30	11:00	Objects (DP)
11:00	11:30	Functions (GC)
11:30	12:15	SQL (MV)
12:15	01:00	**Practice test 1**
01:00	01:30	Break
01:30	02:10	GGPLOT (MV)
02:10	02:40	Data Preparation (GC)
02:40	03:20	Models (DP)
03:20	04:00	**Practice test 2**

R-Objects

Introduction - How R works

- Simple and intuitive syntax e.g. *lm(x ~ y)*
- ***Objects*** are how information is stored in R. Vectors, functions, data frames, matrices, lists are all objects stored on the local machine's memory
    - Actions can be done on objects using *operators* (arithmetic, logical, comparison etc.) or *functions (*which are objects themselves)
- **Packages** consist of functions allowing a variety of manipulations and analysis of data

Why use R?

Mainly used for statistics but also has **excellent machine learning packages, data manipulation** and functionality to **build interactive web apps** in a quick and simple way.

Objects

Building Blocks

Create an **object**
All objects have 2 attributes: *mode* and *length*
4 main modes: numeric, character, logical, complex

```
x <– 15 # numeric
```

Vectors

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("cat","dog","parrot") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)# logical vector
```

Matrices - All columns in a matrix must have the same type and the same length

```
y <- matrix(1:12, nrow=3,ncol=4)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Lists - ordered collection of objects. Gather variety of objects under one name.

```
> mylist <- list(name="Rosie", mynumbers=a, mymatrix=y, age=5.3)
```

```
> mylist
```

```
$name
[1] "Rosie"
$mynumbers
[1]  1.0  2.0  5.3  6.0 -2.0  4.0
$age
[1] 5.3
```

```
$mymatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Concatenate lists

```
lists <- c(list.a, list.b, list.c)
```

Objects

Data Frames

Data frames store data tables. It is a list of vectors of equal length. Several modes possible in the same object. There are built in data frames in R for tutorial purposes - iris, titanic, mtcars; all of which will be used in this workshop.

Reading data from files

```
data("iris")
iris <- read.table(“iris”)
iris <- read.csv(“~/data/iris.csv”,
header = True)
head(iris)

# Summarize the data set
str(iris)
'data.frame':     150 obs. of  5 variables:
$ Sepal.Length: num  5.1 4.9 4.7 4.6 4.6 5 4.4 4.9 ...
$ Sepal.Width : num  3.5 3 3.2 3.1 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num  1.4 1.4 1.3 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num  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..
```

Accessing columns and rows

```
iris$Species    # accessing a column
iris[["Species"]]
iris[5]
iris[, 2]
> [1] 3.5 3.0 3.2 ...
  [30] 3.2 3.1 3.4 ...
  [59] 2.9 2.7 2.0 ...
```

Row number

```
iris[2,] # accessing a row
> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
2         4.9           3          1.4         0.2  setosa

nrow(iris)   # number of rows in data set
ncol(iris)   # number of columns in data set
```

Objects

Number indexing

```
# Access column 3
> iris[3]
    Petal.Length
1            1.4
2            1.4
```

```
# Access row 1 and column 4
iris[1,4]
> [1] 0.2
```

```
# Get rows 1 to 3 and all columns
iris[1:3, ]
> 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
```

```
# Get only rows 3 and 9 and all columns
iris[c(3,9), ]
> Sepal.Length Sepal.Width Petal.Length Petal.Width Species
3          4.7         3.2          1.3         0.2  setosa
9          4.4         2.9          1.4         0.2  setosa
```

Name indexing

```
# Get a column by name
iris["Sepal.Length"]
```

```
# Get rows by name
> iris["10",]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
10          4.9         3.1          1.5         0.1  setosa
```

```
> iris[c("10","119"),]
    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
10           4.9         3.1          1.5         0.1  setosa
119          7.7         2.6          6.9         2.3  virg..
```

Logical indexing

```
# In the setosa vector, the member value is TRUE
if value in column “Species” is equal to “setosa”
> setosa = iris$Species == "setosa"
[1]  TRUE  TRUE ..
[58] FALSE FALSE ..
```

```
> iris[setosa,]
   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
```

Objects

Subsetting data frames

```
# Subset data based on columns
myvars <-
c("Sepal.Length","Sepal.Width")
newiris <- iris[myvars]
```

Challenge: How will you subset columns 1, and 3 to 5?

```
# Excluding columns from data by name
myvars <- names(iris) %in%
  c("Sepal.Length","Sepal.Width")

newiris <- iris[!myvars]
```

```
# Exclude columns by number
newiris <- iris[c(-1,-3)]
```

```
# Subsetting data based on rows
newiris <- iris[1:10,]
```

```
# Subsetting based on conditions within
columns
newiris <- iris[
  which(iris$Sepal.Width >= 4
        & iris$Species == "setosa"),
]
```

```
# Using the subset function to subset based
on column conditions
newiris <- subset(
  iris, Sepal.Width < 2 |
    Sepal.Width > 3,
  select=c(Sepal.Length, Species))
```

Or

Objects

Missing values

```
# Testing data frame for missing value
is.na(iris)

# Replace sepal length lower than 5 with NA
iris$Sepal.Length[iris$Sepal.Length < 5]
     <- NA*

# List cases with missing values
iris[complete.cases(iris),]

# Create new data set excluding the missing values
newiris <- na.omit(iris)
```

*be careful with empty cells without *NA* being specified

Date values

Check the class of a variable

```
class(iris$Sepal.Width)
```

Use `as.Date()` to convert string to dates

```
dates <- as.Date(c("2007-06-22",
"2004-02-13"))
```

The same function can be applied to a column in the data frame.

```
df <- data.frame(
 time=c("2014-05-10","2015-03-23",
       "2015-07-12","2016-02-17"),
 precip=c(8,9,6,7))

df$time <- as.Date(df$time,
  format = "%Y-%m-%d")
```

Specify date format

R- Functions

If/Else Statements

If/Else statements are a way of programming conditional behaviour, executing different commands if a condition is met or not

Standard If/Else

```
# One-line statement
if (condition) … else …

# Multi-line statement
if (condition) {
…
} else {
…
}
```

Nested If/Else

```
# Nested statement
if (condition) {
…
} else {
     if (condition) {
     …
     } else {
     …
     }
}
```

Works but hard to read

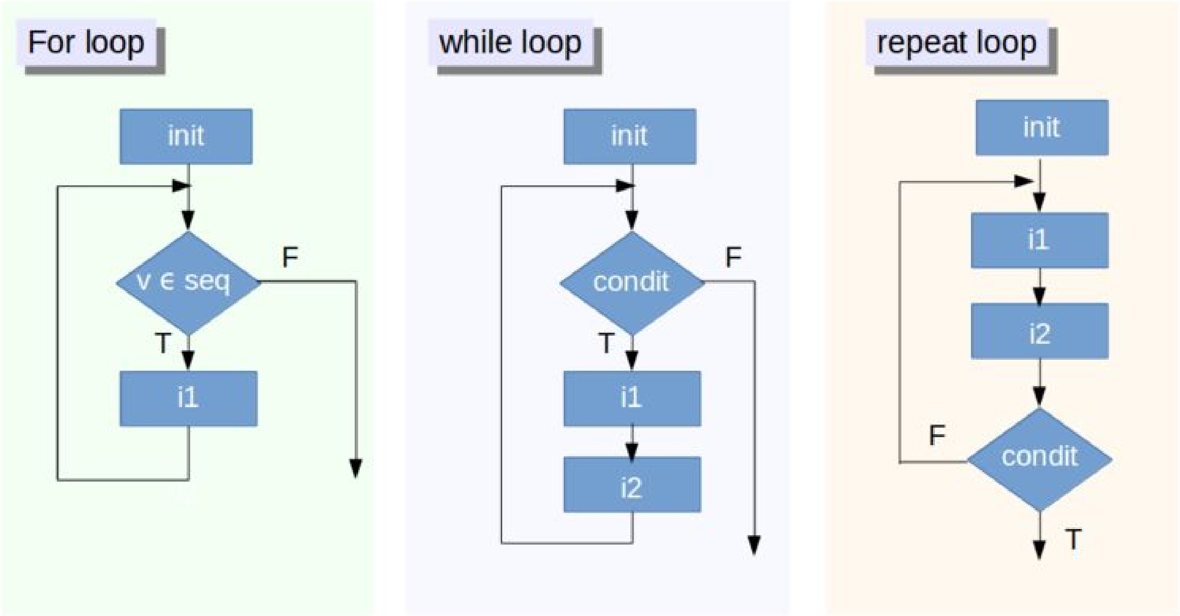
Else If Statements

```
# Else if statement
if (condition) {
…
} else if (condition)
{
…
} else {
…
}
```

Nicer to read!

Loops

*Loops allow you to repeat a piece of code for a defined number of timesteps. There are three types of loops, **for, while**, and **repeat**.*



Loops

For Loops

```
# One-line statement
for (i in 1:n) print(i)

# Multi-line statement
for (i in 1:n) {
m <- i^2 + 1
print(m)
}

# Nested For Loop
for (i in 1:n) {
  for (j in 1:m) {
    print(i*j)
  }
}
```

```
# For and If
statement
for (i in 1:n){
  if (i%%2==0){
    print("Even")
  } else {
    print("Odd")
  }
}
```

While Loops

```
# While statement
i=1
while (i<=n){
  print(i)
  i=i+1
}
```

Functions

Functions are objects in R that are called to execute a piece of code on a variable amount of inputs. There are many inbuilt functions in R which can be found [here](). Users can also define their own functions:

Functions

```
# General Form
function.name <- function(arguments)
{
  computations on the arguments
  some other code
}
```

```
WARNING: BE CAREFUL NOT TO USE A
FUNCTION NAME THAT IS ALREADY USED IN
R
```

Arguments

```
# A function can take several arguments
or none
MyFirstFun <- function(x,y,z)
{
  (x*y)^z
}

# One can define default arguments
MySecondFun <- function(x,y=2,z=3)
{
  (x*y)^z
}
```

Functions

Returns

```
# No return needed
square <- function(x){x*x}
x <- 10
m <- square(x)

# Return needed if we want specific
value
mySecFun<-function(v, M)
{
  u=c(0,0,0,0)
  for(i in 1:length(v))
  {
    u[i]=square(v[i])
  }
  return(u)
}
```

Anonymous Functions

```
# A function can be defined and used in
one line
(function(x) x*10)(10)
sapply(v, function(x) x*10)
```

Function Help

```
# Call the function without brackets to
get code
square <- function(x){x*x}

> square
function(x){x*x}
```

R- sqldf

SQL in R

Facilitates data manipulation: exploring, cleaning, crossing, building features, etc.

Very useful skill.

Even if you understand the core of SQL, you can use it on a proper DB engine or in other programming languages.

library(sqldf)

The sqldf library allows querying data frames as they were tables and saves the results in an R object.

Library: sqldf

In SQL, data is usually organized in various tables. Let's start by grabbing all of the data in one table.

```
SELECT * FROM titanic
```

Select only some columns:

```
SELECT pclass,survived,name
FROM titanic
```

WHERE : Filtering rows

In order to select particular rows from a data frame, you need to use the `WHERE` keyword. Plus, a filter with =,>,>=,<=,<,<> if the attribute is numerical.

```
SELECT pclass,survived,name, sex, age
FROM titanic
WHERE age<20 AND embarked IN (“Southampton”,“Cherbourg”)
```

You can aldo search for rows:

- that match with multiple attributes by using the `AND` keyword;
- that match any of multiple attributes by using the `OR` keyword.
- where the attribute is in a list of several possible values by using the `IN` keyword; Also `NOT IN`.
- where the attribute is (not) NULL by using `IS (NOT) NULL`.
- where the attribute is like a piece of string (but not necessarily the exact value) by using the `LIKE` keyword.

GROUP BY: aggregating data!

You can use aggregate functions such as `COUNT`, `SUM`, `AVG`, `MAX`, and `MIN` with the `GROUP BY` clause. When you `GROUP BY` something, you split the table into different piles based on the value of each row.

```
SELECT pclass,sex, COUNT(1) cnt, MAX(age) max_age, MIN(age)
min_age, AVG(age) mean_age
FROM titanic
WHERE age<20 AND embarked IN (“Southampton”,“Cherbourg”)
GROUP BY 1,2
```

HAVING: filtering GROUP BY

You can filter the results of aggregating data with the `HAVING` clause.

Also, you can aggregate attributes column-wise by using a `CASE WHEN THEN ELSE END` statement and return certain values when the scenario is true.

```
SELECT pclass,SUM(CASE WHEN sex="female" THEN 1 ELSE 0 END)
female_cnt,SUM(CASE WHEN sex="male" THEN 1 ELSE 0 END) male_cnt
FROM titanic
GROUP BY 1
HAVING SUM(CASE WHEN sex="male" THEN 1 ELSE 0 END)>120
```

LEFT JOIN/INNER JOIN: connecting Data!

With `JOIN`, you are merging two data frames using some key attributes. Then a row from one data frame is joined to the row from the second data frame when the key attributes are matched.

`INNER JOIN` only returns rows whose key attributes matched.

`LEFT JOIN` returns all the rows from the left data frame, with the attribute values from the right data frame whose key attributes matched, and NULL values for the rows from the right data frame that didn't match.

```
SELECT a.*, female_teens_cnt, male_teens_cnt
FROM titanic_kids a LEFT JOIN titanic_teens_2 b ON
a.pclass=b.pclass
```

PRACTICE TEST 1

1.- Upload Boston Data

2.- What variables are numerical or categorical

3.- Describe your variables (hint: look to slide 9)

4.- Identify missing values in the column CRIM and replace them with the mean

5.- Create a function that takes as input the column TAX and generates a new column called TAXBIN where the TAX values are binned into the following ranges:

- <200: 0, 200-299: 1, 300-399: 2, 400-499: 3, >=500: 4

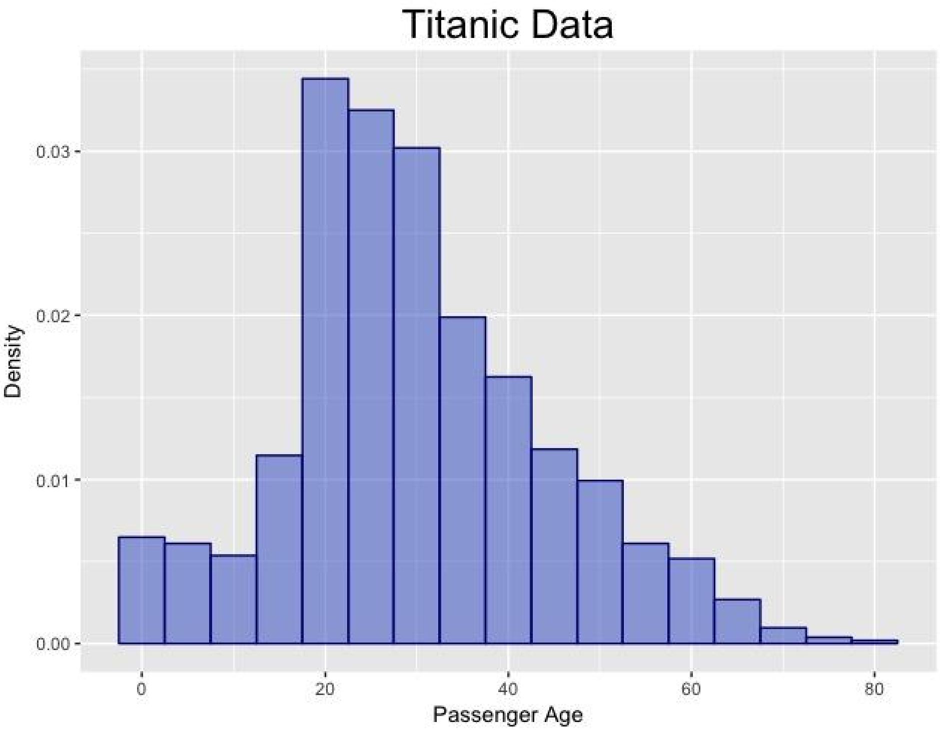
6.- Add to every row in Boston, the ratio between the CRIME Rate and the RAD average CRIME Rate.**

*Hint - if you encounter an error, google it and you will mostly likely find its solution.

** Hint - Using SQL, aggregate the RAD feature to count the number of rows, the average/maximum/minimum of the Crime Rate. Disregard null values

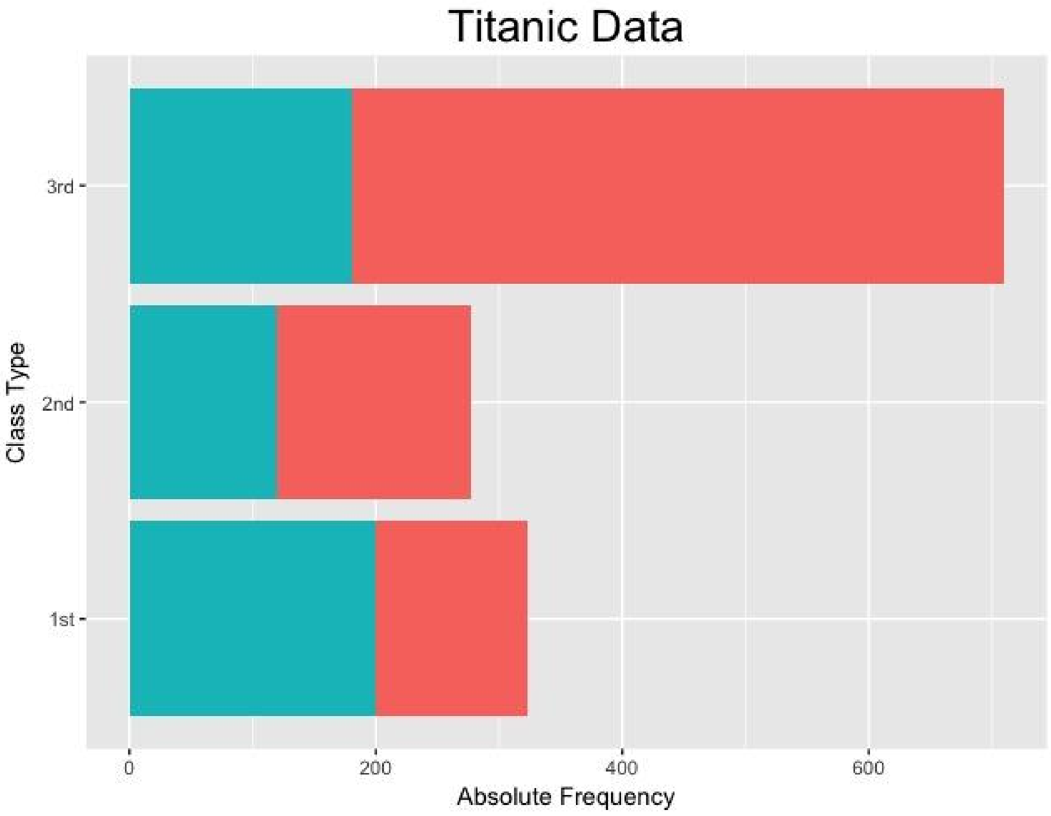
R- GGPLOT

Histograms



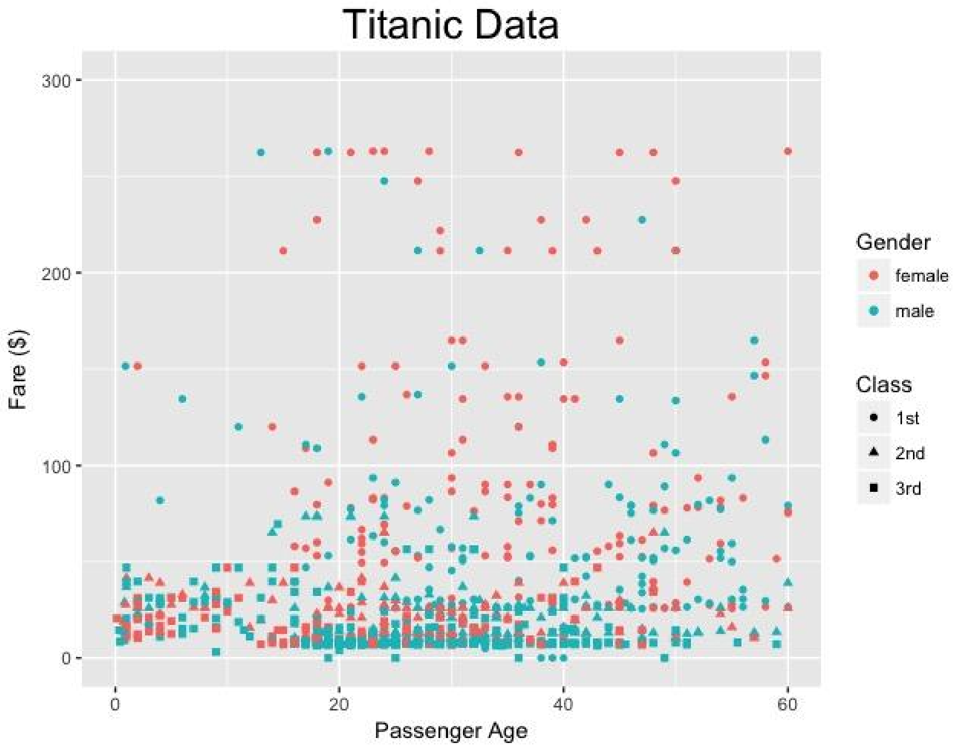

```
ggplot(titanic, aes(age))+
  geom_histogram(aes(y=..density..), binwidth = 5,
          color ="navyblue", fill ="royalblue3", alpha = 0.5)+
  labs(title = "Titanic Data", y = "Density", x = "Passenger Age")+
  theme(plot.title = element_text(hjust = 0.5, size=20))
```

Bar Plot



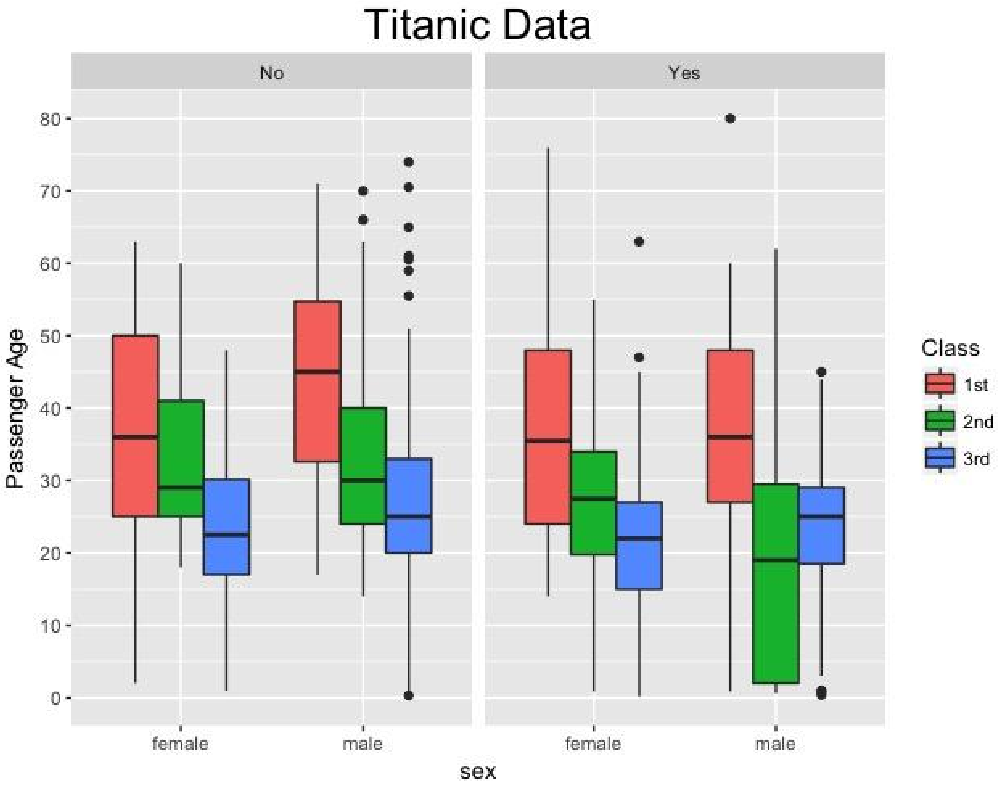

```
ggplot(titanic, aes(pclass))+
  geom_bar( aes(fill = survived))+
  scale_fill_discrete(breaks=c("0","1"),
          labels=c("No","Yes"))+
  coord_flip() +
  labs(title = "Titanic Data", y = "Absolute Frequency",
          x ="Class Type", fill = "Survived Flag")+
  theme(plot.title = element_text(hjust = 0.5,
          size=20), legend.position = "bottom")
```

Scatter Plot



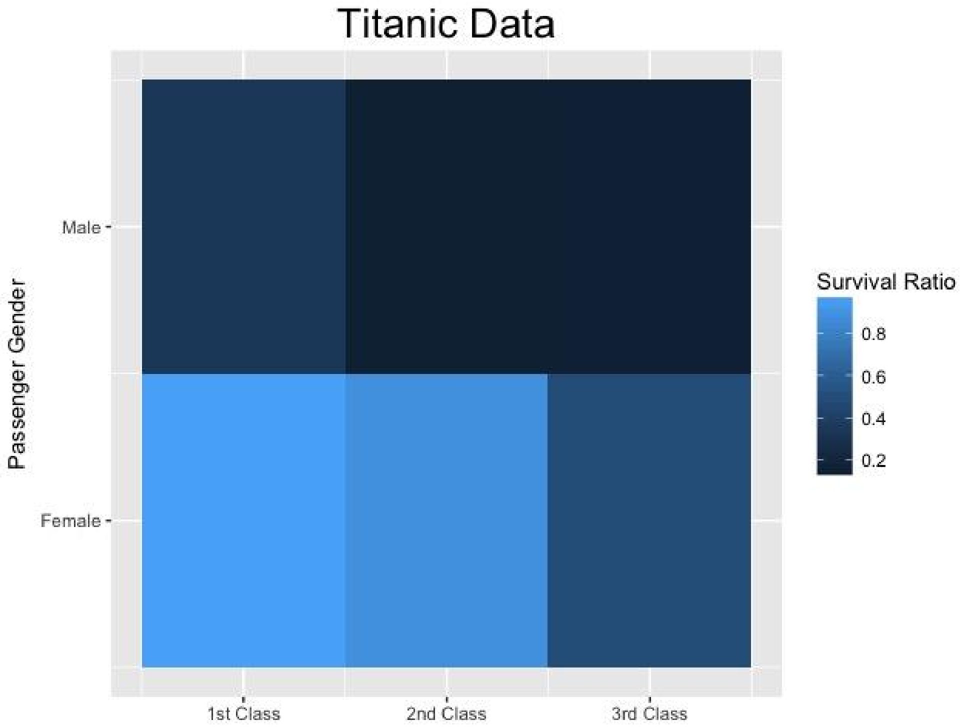

```
ggplot(titanic, aes(age,fare, sex,pclass))+
  geom_point(aes(color=sex, shape=pclass))+
  labs(title="Titanic Data", y="Fare ($)", x="Passenger
        Age", shape="Class", color="Gender")+
  theme(plot.title = element_text(hjust = 0.5,
        size=20))+
  ylim(0,300)+xlim(0,60)
```

Box Plot




```
ggplot(titanic, aes(y = age, x = sex, fill = pclass))+
  geom_boxplot()+
  labs(title = "Titanic Data", y = "Passenger Age",
       fill = "Class")+
  theme(plot.title = element_text(hjust = 0.5,
       size=20))+
  scale_y_continuous(breaks = seq(0, 80, 10)) +
  facet_grid(. ~ survived, labeller = label_value)
```

Heat Map




```
ggplot(titanic_agg1, aes(as.numeric(pclass),
        as.numeric(sex) ))+
  geom_tile(aes(fill = survival_ratio))+
  scale_y_continuous(breaks = c(1,2),
        labels = c("Female","Male"))+
  scale_x_continuous(breaks = c(1,2,3),
        labels = c("1st Class","2nd Class","3rd Class"))+
  labs(title ="Titanic Data", y = "Passenger Gender", x = "",
        fill = "Survival Ratio")+
  theme(plot.title = element_text(hjust = 0.5, size = 20))
```

R-Data Preparation

Missing Values

Dealing with missing values

- It is very common in the real world to get messy data, often with a lot of missing values
- Before working with the data and modelling, one must understand any underlying patterns of missing data and deal with the data

MCAR: Missing Completely at Random. This is what we want to deal with, there is no structure to the missing data.
MNAR: Missing Not at Random. This is a serious issue and often requires going back to investigate the data gathering process. Filling in the missing values here could have a serious impact on models.

Packages to visualise missing data

`mice``: A library for obtaining statistics on missing data and imputing data`
`VIM:` `A library for visualising missing data`

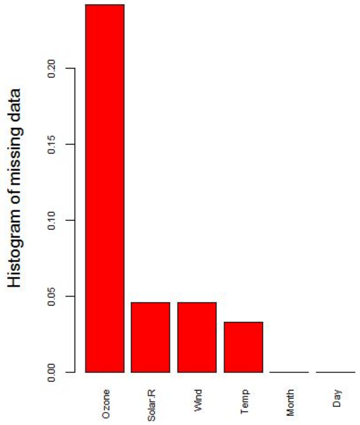
Missing Values

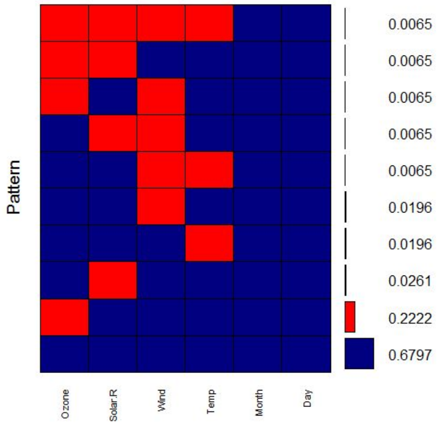
Visualising Missing Values

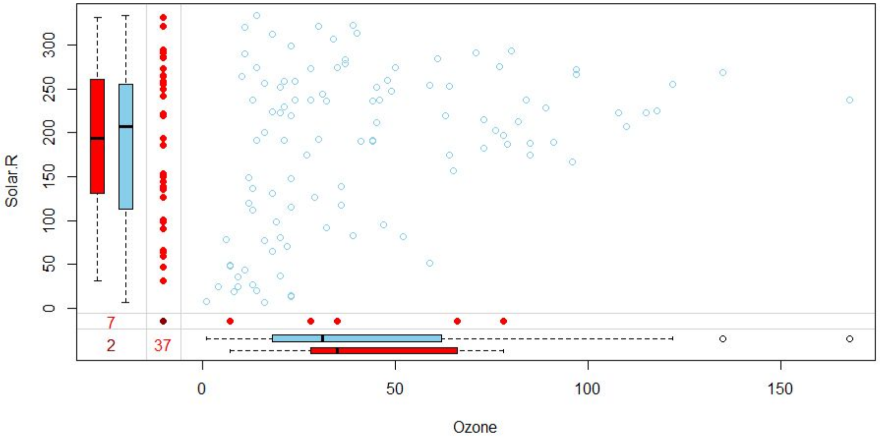
- Mice package allows you to see the patterns of missing data in rows

```
> md.pattern(data)
    Month Day Temp Solar.R Wind Ozone
104     1   1    1       1    1     1  0
 34     1   1    1       1    1     0  1
  4     1   1    1       0    1     1  1
  3     1   1    1       1    0     1  1
  3     1   1    0       1    1     1  1
  1     1   1    1       0    1     0  2
  1     1   1    1       1    0     0  2
  1     1   1    1       0    0     1  2
  1     1   1    0       1    0     1  2
  1     1   1    0       0    0     0  4
        0   0    5       7    7    37 56
```

- VIM presents histograms of missing data to examine each column or you can use `marginplot` to examine the distribution of missing data



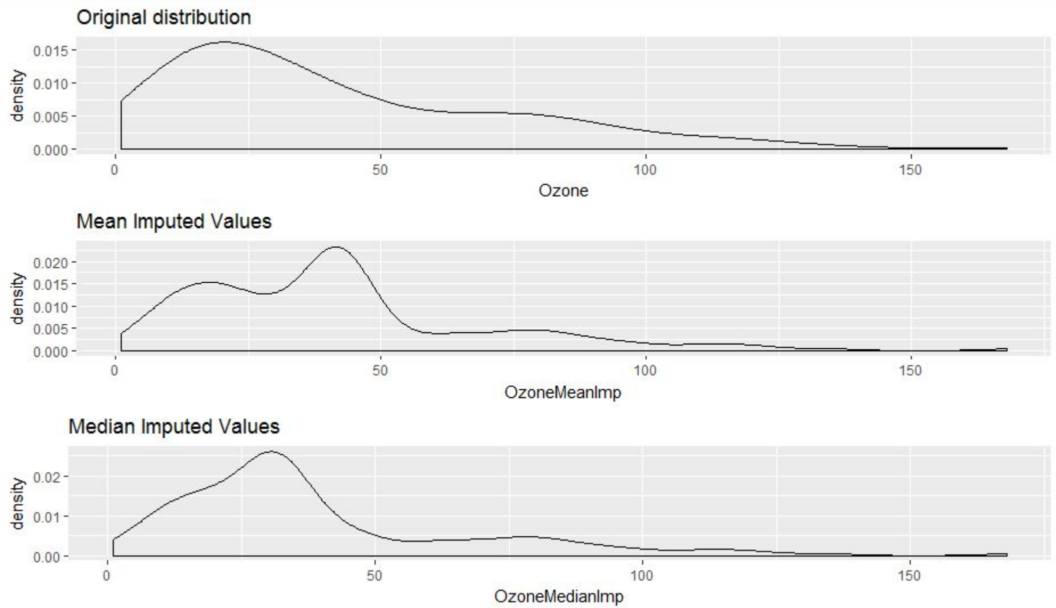




Missing Values

Imputing Missing Values

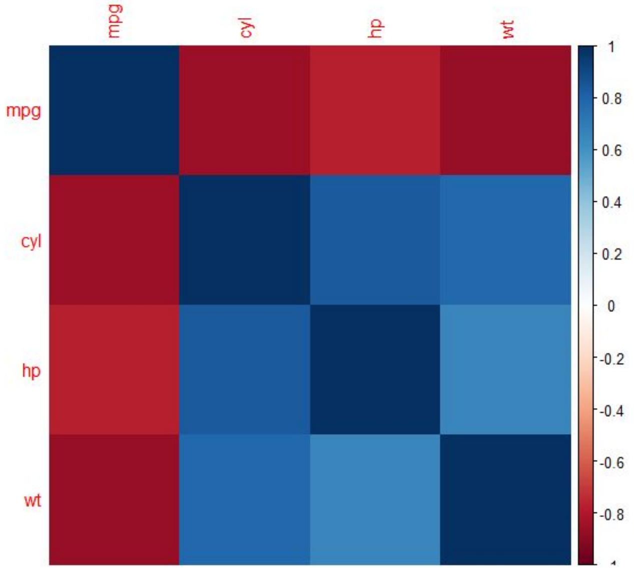
- Once we're more confident that values are MCAR we can perform imputation. This is when missing values are filled using one of a variety of methods
- Simplest form of imputation is mean/median imputation
- It is important to plot the distribution of values to make sure that the distribution is most unchanged when values are imputed



Correlation Plots

Understanding Correlation

- As a data preparation step, it is helpful to understand the correlations between variables in the data. The correlation can be found using the inbuilt `corr` function. This function returns a large matrix, but for a large number of variables it can be hard to read.
- Plotting the correlation is a very nice way to represent the relationships between variables. This is done through the `corrplot` library.



R-Models

Modeling in R

Structure of models in R

- The key to modeling in R is the **formula object** - a shorthand method to describe the exact model to fit to the data.
- **Modeling functions** require a formula object as an argument.
- The modeling functions return a model object that contains all the **information about the fit.**

*Good data science practice is to partition the data into **training** and **testing** set. This is done so that the model can "learn" from past observations. Then the predictions can be compared with the testing data set and **measure the accuracy** of the results against **real observations**.*

Modeling packages

`e1701`: `A library for various statistical and machine learning algorithms.`

`caret`: `An incredibly useful library containing a set of functions for streamlining the process of creating predictive models.` *`caret`* `inherits from a number of other R libraries including` *`kernlab`*`,` *`e1701`* `and klaR.`

`kernlab`: `A comprehensive library of kernel algorithms, including SVMs for classification and regression, kernel principal components analysis and Gaussian processes.`

Linear Regression

Definition: Linear regression modeling is used to describe the relationship between a *continuous* dependent variable ***Y*** and one or more explanatory variables ***X***.

lm(y ~ x)

```
# Subset only the numerical variables of the data
iris.num <- subset(iris, select =
c("Sepal.Length", "Sepal.Width", "Petal.Length",
"Petal.Width"))

# correlation between variables
cor(iris.num)

# Simple plot of the data
plot(iris.num)

# Fit regression model with one explanatory
variable
iris.lr <- lm(Sepal.Length ~ Petal.Length,
data=iris.num)

# Summary of the model
summary(iris.lr)
```

lm(y ~ x1 + x2 + x3)

```
# Linear regression multiple explan. variables
iris.lr2 <- lm(Sepal.Length ~ Petal.Length +
Sepal.Width + Petal.Width, data=iris.num)
```

Data frame used

```
# Print regression coefficients
summary(iris.lr2)
plot(iris.lr2) # plot results

# Model accuracy evaluation
# Residual sum of squares
rss <- c(crossprod(iris.lr$residuals))

# Mean squared error
mse <- rss / length(iris.lr$residuals)

# Root Mean Squared Error
rmse <- sqrt(mse)
```

Logistic Regression

Regression model where the dependent variable is *categorical*. Therefore, it used to solve classification problems where we seek a 0 or 1 answer - pass / fail, hungry / full etc.

glm(y ~ x1 + x2 + x3) **or** **caret package**

```
# Load data set
data(GermanCredit)

# Load data set
log_model <- glm(Class ~ Age +
ForeignWorker
    + Property.RealEstate +
    Housing.Own +
    CreditHistory.Critical,
data=GermanCredit,
    family="binomial")

# Test diagnostics
anova(log_model, test = “Chisq”)
```

```
library(caret)
# Partition data so that 60% is used for training
train <- createDataPartition(GermanCredit$Class,
p=0.6, list=FALSE)

# Separate data into training and testing
training <- GermanCredit[ train, ]
testing <- GermanCredit[ -train, ]

# Fit the model using the training data set
car_model <- train(Class ~ Age + ForeignWorker +
Property.RealEstate + Housing.Own +
CreditHistory.Critical,  data=training,
method="glm", family="binomial")

# Predict Good / Bad
predict(car_model, newdata=testing)
# Predict probability of Good / Bad to occur
predict(car_model, newdata=testing, type="prob")
```

K-means clustering

Unsupervised learning technique. It is used on **unlabeled data** with the goal to find groups in the data, where the number of groups are represented by the variable K. The algorithm works iteratively to assign each data point to one of ***K* groups based on the features** that are provided. Data points are clustered based on their feature similarity.

```
# Select the petal length and width as the features to base the clustering on
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
```

```
# Comparison between the clusters and the actual species
table(irisCluster$cluster, iris$Species)
```

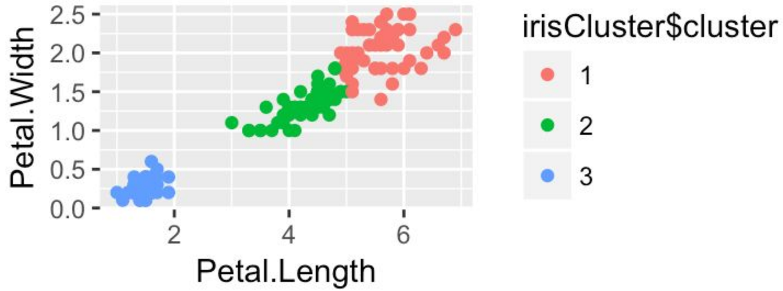
```
>   setosa versicolor virginica
 1       0          2        46
 2       0         48         4
 3      50          0         0
```

Cluster numbers

Only 2 observations were incorrect

```
# Change resulting clusters to factors for easier plotting
irisCluster$cluster <-
    as.factor(irisCluster$cluster)
```

```
# Plot the clustering results using ggplot
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
```



PRACTICE TEST 2

10.- Plot some the variables (the most interesting ones)

11.- From 10, can you identify any outliers?

12.- Calculate the correlation matrix (only numerical), what are the variable that are most correlated with Crime?

13.- Split your data in training and test.

14.- Create a predictive model for Crime rates

15.- Create a ggplot graph to visualise your results