##### **AMITY UNIVERSITY MADHYA PRADESH, GWALIOR**

##### **Practical File**

##### **OF**

**COGNITIVE SCIENCE IN ARTIFICIAL INTELLIGENCE LAB**

##### **CSA 321**

##### 

##### 



##### **SUBMITTED TO**

##### DR Madhavi Dhingra,

##### Assistant Professor

##### **SUBMITTED BY**

##### MR Rajat Gupta

##### Btech CSE ‘A’

##### Department of Computer Science & Engineering

##### Amity School of Engineering & Technology

##### Amity university madhya pradesh, gwalior

##### Jun - Dec 20201

### DEPARTMENT VISION AND MISSION

**Vision**

To emerge as a centre of excellence with global reputation by adapting latest trained and technologies in the field of Computer Science & Engineering.

### Mission

1. To provide the students the knowledge of R Programming and RStudio Operational environment.
2. To inculcate Professional behavior, strong ethical values, leadership qualities, research capabilities and life long learning.
3. To educate students to become effective problem solvers, apply knowledge with social sensitivity for the betterment of the society and humanity as a whole.

## LABORATORY OUTCOMES:

**CO [1]**To Understand Data analysis with R Environment

**CO[2]**To run R programming

**CO[3]**.To understand training and testing concepts.

**CO[4]** To Understand analytical and correlation problems.

**Do’s**

1. Come with completed observation and record
2. Wear ID card before entering into the lab.
3. Keep all your bags outside your laboratory further you are also instructed to keep the mobile phones inside your bag.
4. Read and understand how to carry out an activity thoroughly before coming to the laboratory.
5. Report any broken plugs or exposed electrical wires to your lecturer/laboratory technician immediately.
6. Write in time, out time and system details in the login register.

# Don’ts

1. Do not eat or drink in the laboratory.
2. Do not operate mobile phones in the lab. Keep mobile phones either in silent or switched off mode.
3. Do not change system settings.
4. Do not disturb your neighbouring students. They may be busy in completing tasks.
5. Do not remove anything from the computer laboratory without permission.
6. Do not use pen drives.
7. Do not misbehave.

##### **INDEX**

## List Of Experiments

## R Exporting Data to Excel, CSV, SAS, STATA, Text Filereading in data

## Correlation in R: Pearson & Spearman with Matrix

## R Aggregate Function: Summarise & Group\_by()

## R Simple, Multiple Linear and Stepwise Regression

## Decision Tree in R

## R Random Forest

## Generalized Linear Model (GLM) in R

## K-means Clustering

## Exploratory data analysis

## Feature Engineering

## Training and test split

## Model training with Random Forests, Gradient Boosting, Neural Nets, etc.

**1. R Exporting Data to Excel, CSV, SAS, STATA, Text Filereading in data**

There are numerous methods for exporting R objects into other formats .

For SPSS, SAS and Stata, It is needed to load the foreign packages.

For Excel, y the xlsReadWrite package.

To A Tab Delimited Text File

write.table(mydata, "c:/mydata.txt", sep="\t")

To an Excel Spreadsheet

library(xlsx)  
write.xlsx(mydata, "c:/mydata.xlsx")

(To work with R and Excel, try this interactive course on importing data in R.)

To SPSS

# write out text datafile and  
# an SPSS program to read it  
library(foreign)  
write.foreign(mydata, "c:/mydata.txt", "c:/mydata.sps",   package="SPSS")

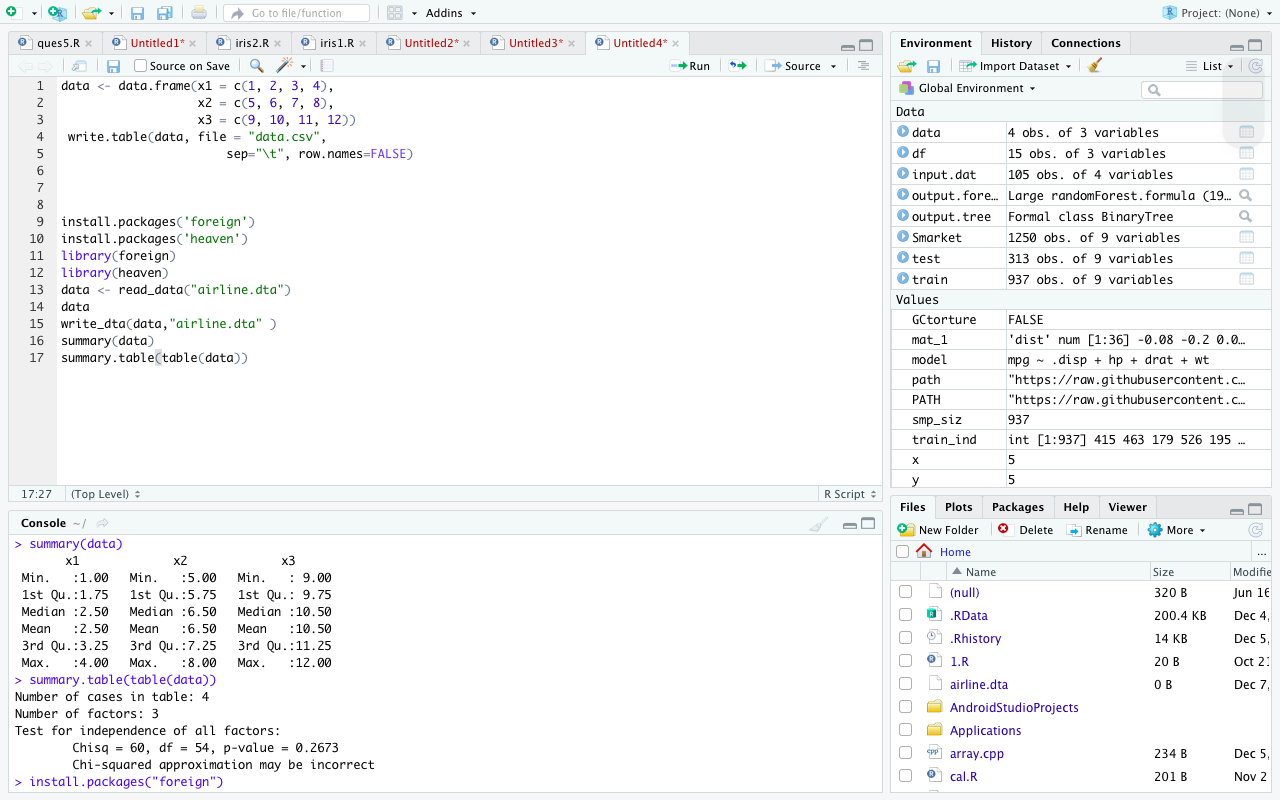
(Alternatively, to practice importing SPSS data with the foreign package, try this exercise.)

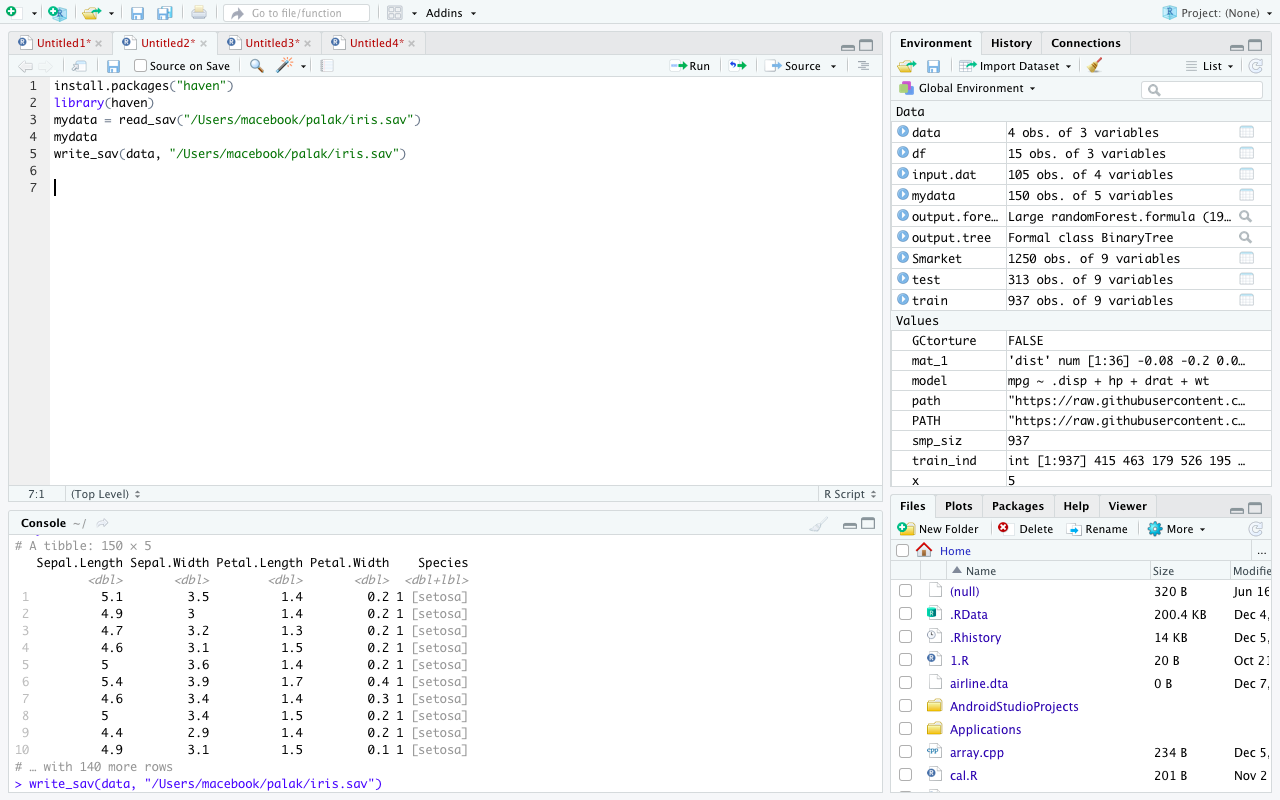
To SAS

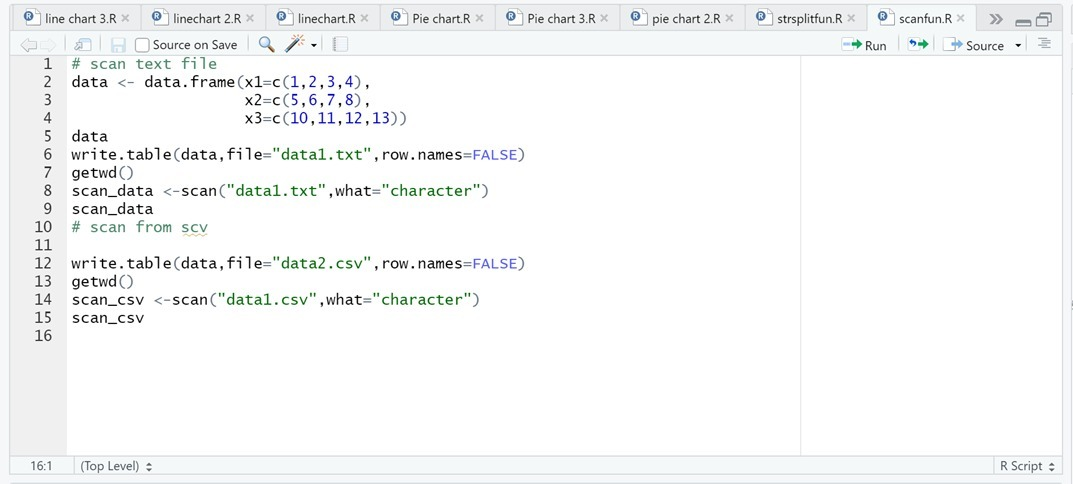
# write out text datafile and  
# an SAS program to read it  
library(foreign)  
write.foreign(mydata, "c:/mydata.txt", "c:/mydata.sas",   package="SAS")

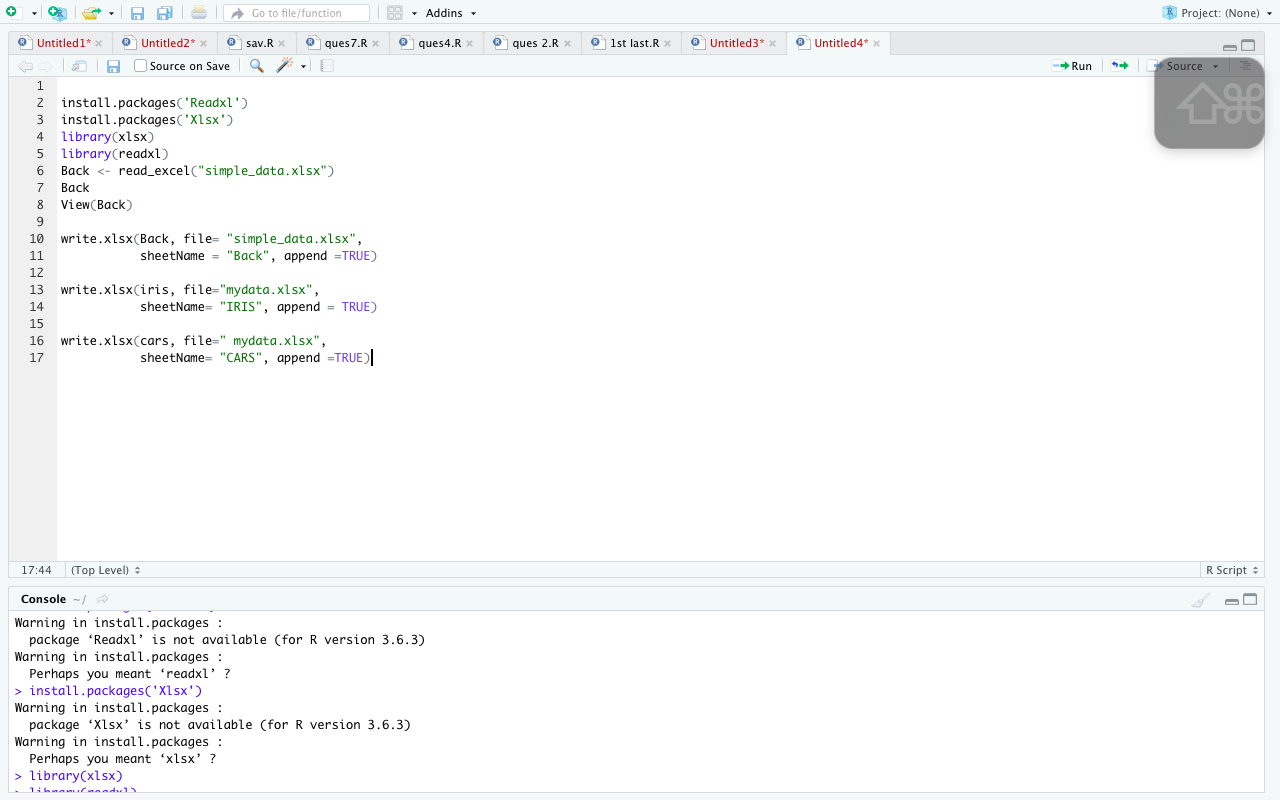
To Stata

# export data frame to Stata binary format   
library(foreign)  
write.dta(mydata, "c:/mydata.dta")









**2. Correlation in R: Pearson & Spearman with Matrix**

A bivariate relationship describes a relationship -or correlation- between two variables, and . In this tutorial, we discuss the concept of correlation and show how it can be used to measure the relationship between any two variables.

There are two primary methods to compute the correlation between two variables.

* Pearson: Parametric correlation
* Spearman: Non-parametric correlation

The Pearson correlation method is usually used as a primary check for the relationship between two variables.

The coefficient of correlation, , is a measure of the strength of the linear relationship between two variables and . It is computed as follow:

with

, i.e. standard deviation of

, i.e. standard deviation of

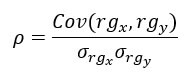
The correlation ranges between -1 and 1.

A value of near or equal to 0 implies little or no linear relationship between and .

In contrast, the closer comes to 1 or -1, the stronger the linear relationship.

Spearman Rank Correlation

A rank correlation sorts the observations by rank and computes the level of similarity between the rank. A rank correlation has the advantage of being robust to outliers and is not linked to the distribution of the data. Note that, a rank correlation is suitable for the ordinal variable.

Spearman's rank correlation, , is always between -1 and 1 with a value close to the extremity indicates strong relationship. It is computed as follow:

In R, we can use the cor() function. It takes three arguments, , and the method.

cor(x, y, method)

Arguments:

x: First vector

y: Second vector

method: The formula used to compute the correlation. Three string values:

"pearson"

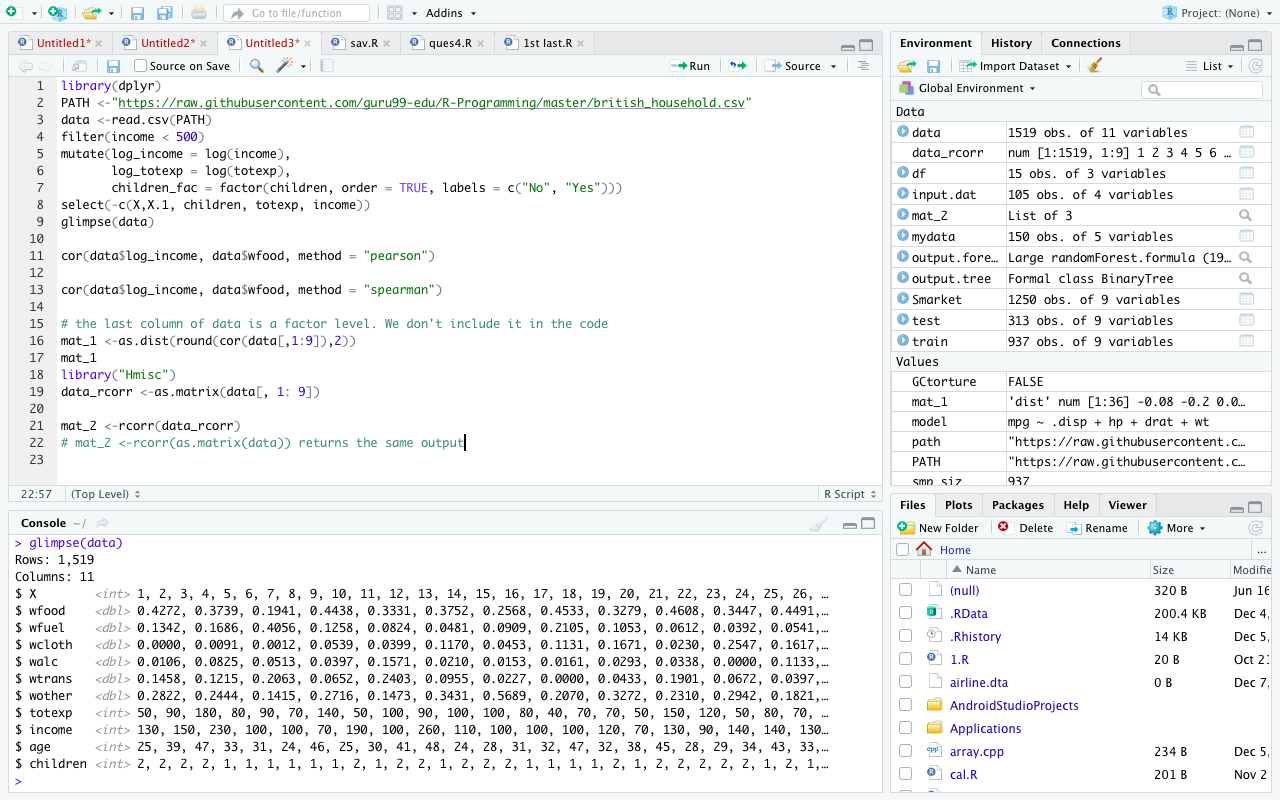
"kendall"

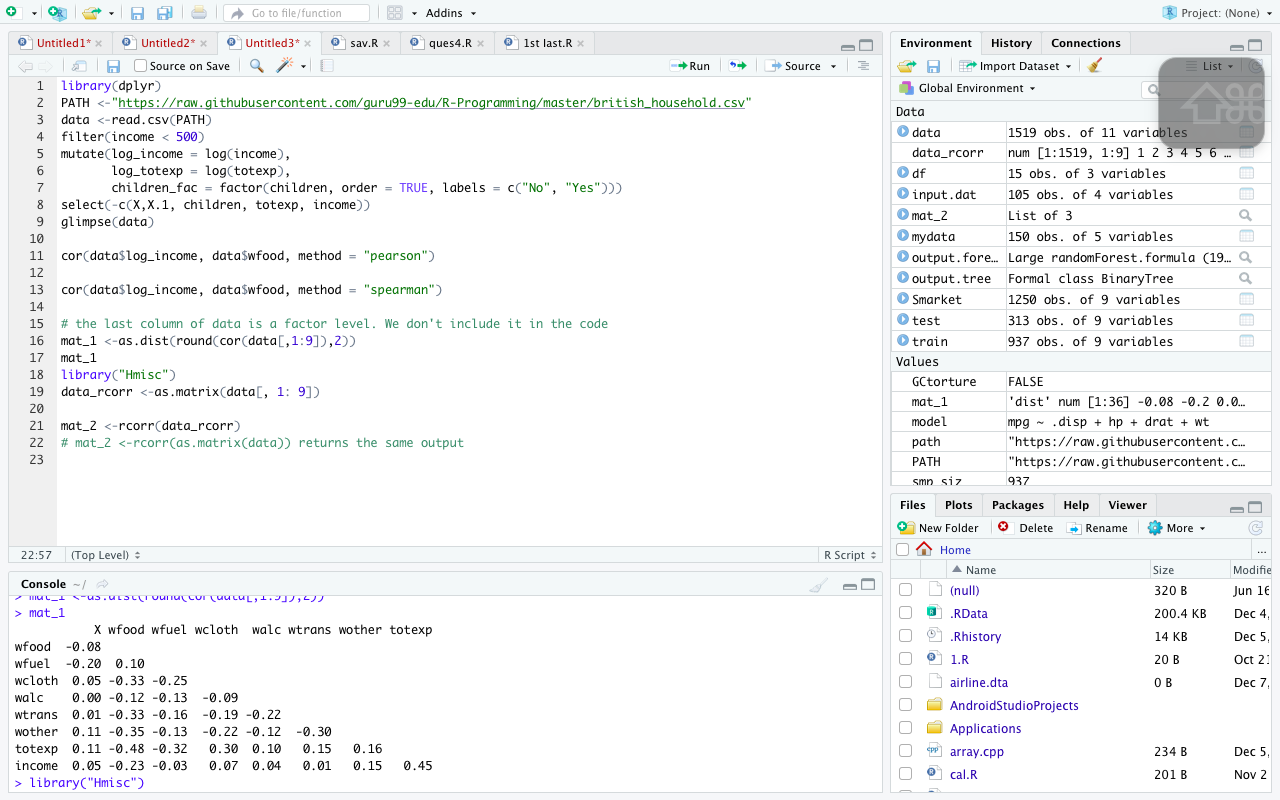
“spearman"

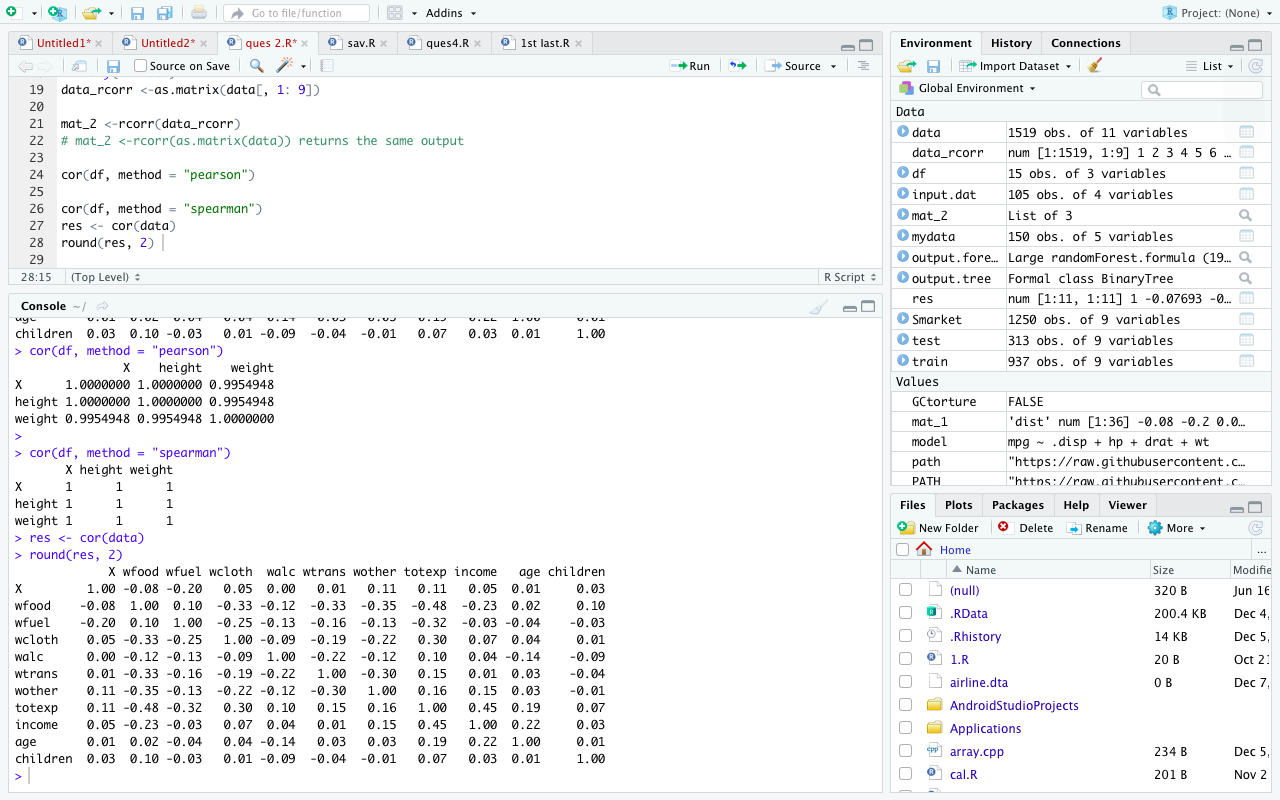
Compute correlation matrix

res <- cor(my\_data)

round(res, 2)







**3. R Aggregate Function: Summarise & Group\_by()**

**Following steps to prepare the data:**

**Step 1: Import the data**

**Step 2: Select the relevant variables**

**Step 3: Sort the data**

Summarise()

The syntax of summarise() is basic and consistent with the other verbs included in the dplyr library.

summarise(df, variable\_name=condition)

arguments:

- `df`: Dataset used to construct the summary statistics

- `variable\_name=condition`: Formula to create the new variable

Look at the code below:

summarise(data, mean\_run =mean(R))

Code Explanation

summarise(data, mean\_run = mean(R)): Creates a variable named mean\_run which is the average of the column run from the dataset data.

Group\_by vs no group\_by

The function summerise() without group\_by() does not make any sense. It creates summary statistic by group. The library dplyr applies a function automatically to the group you passed inside the verb group\_by.

Note that, group\_by works perfectly with all the other verbs (i.e. mutate(), filter(), arrange(), ...).

It is convenient to use the pipeline operator when you have more than one step. You can compute the average homerun by baseball league.

data % > %

group\_by(lgID) % > %

summarise(mean\_run = mean(HR))

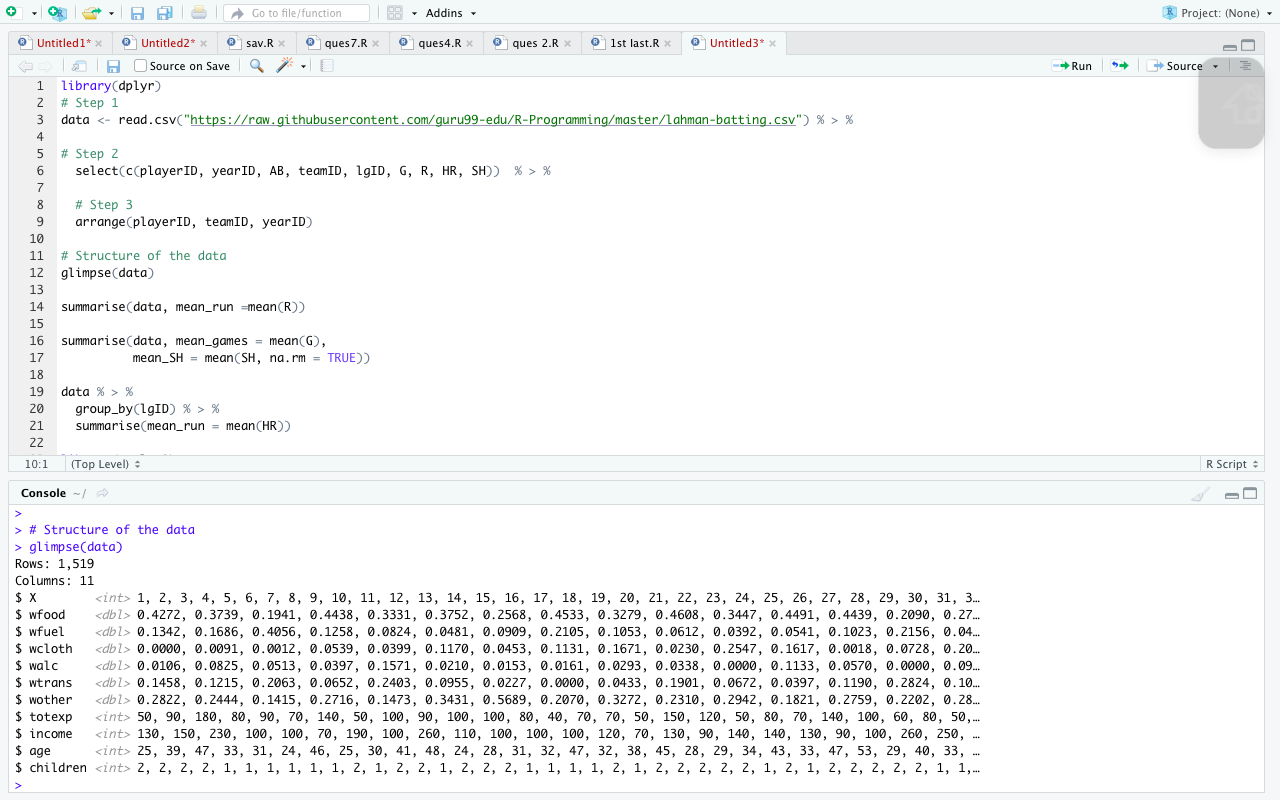
Code Explanation

data: Dataset used to construct the summary statistics

group\_by(lgID): Compute the summary by grouping the variable `lgID

summarise(mean\_run = mean(HR)): Compute the average homerun

The table below summarizes the function you learnt with summarise()



**4. R Simple, Multiple Linear and Stepwise Regression**

A very simple dataset to explain the concept of simple linear regression. We will import the Average Heights and weights for American Women. The dataset contains 15 observations. You want to measure whether Heights are positively correlated with weights.

library(ggplot2)

path <- 'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/women.csv'

df <-read.csv(path)

ggplot(df,aes(x=height, y = weight))+

geom\_point()

Multiple (Linear) Regression

R provides comprehensive support for multiple linear regression. The topics below are provided in order of increasing complexity.

Fitting the Model

# Multiple Linear Regression Example   
fit <- lm(y ~ x1 + x2 + x3, data=mydata)  
summary(fit) # show results

# Other useful functions   
coefficients(fit) # model coefficients  
confint(fit, level=0.95) # CIs for model parameters   
fitted(fit) # predicted values  
residuals(fit) # residuals  
anova(fit) # anova table   
vcov(fit) # covariance matrix for model parameters   
influence(fit) # regression diagnostics

model <- mpg~.disp + hp + drat + wt

fit <- lm(model, df)

fit

ols\_stepwise(fit, pent = 0.1, prem = 0.3, details = FALSE)

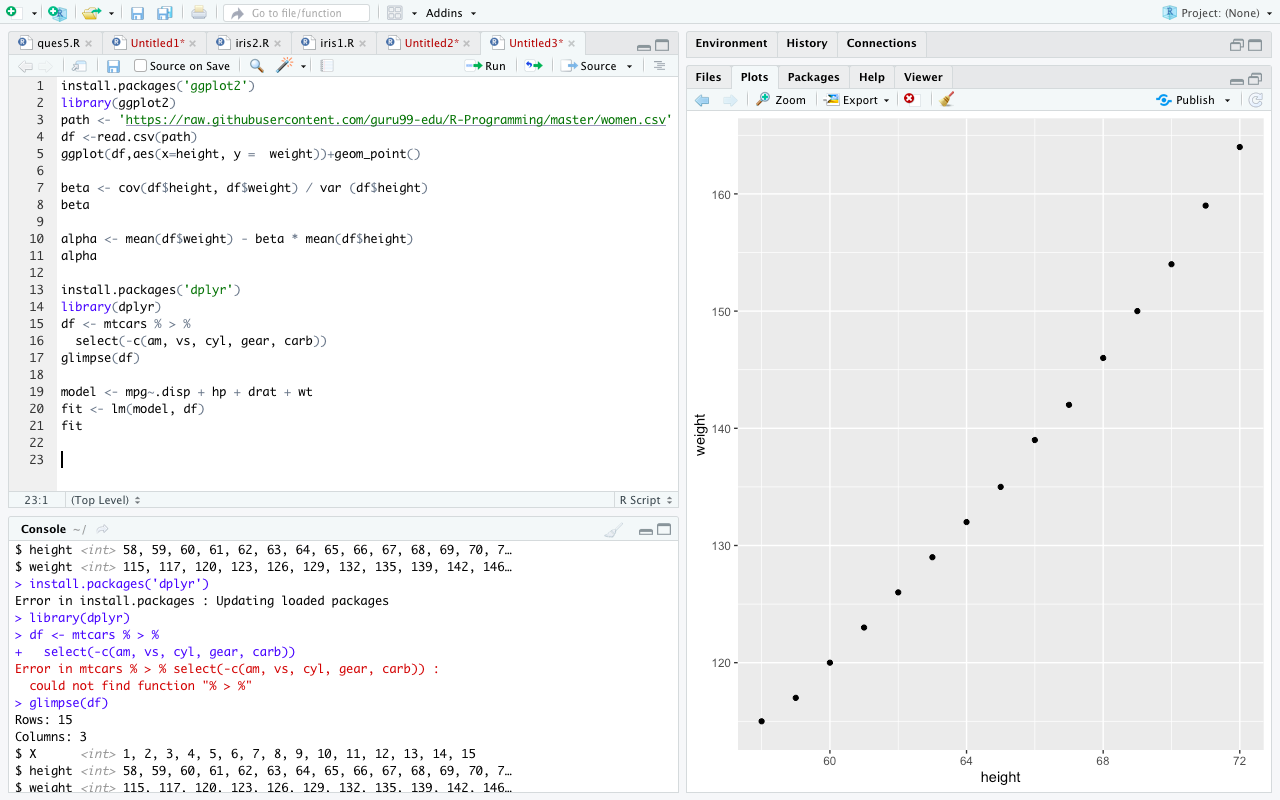
arguments:

-fit: Model to fit. Need to use `lm()`before to run `ols\_stepwise()

-pent: Threshold of the p-value used to enter a variable into the stepwise model. By default, 0.1

-prem: Threshold of the p-value used to exclude a variable into the stepwise model. By default, 0.3

-details: Print the details of each step



**5. Decision Tree in R**

# Load the party package. It will automatically load other

# dependent packages.

library(party)

# Create the input data frame.

input.dat <- readingSkills[c(1:105),]

# Give the chart file a name.

png(file = "decision\_tree.png")

# Create the tree.

output.tree <- ctree(

nativeSpeaker ~ age + shoeSize + score,

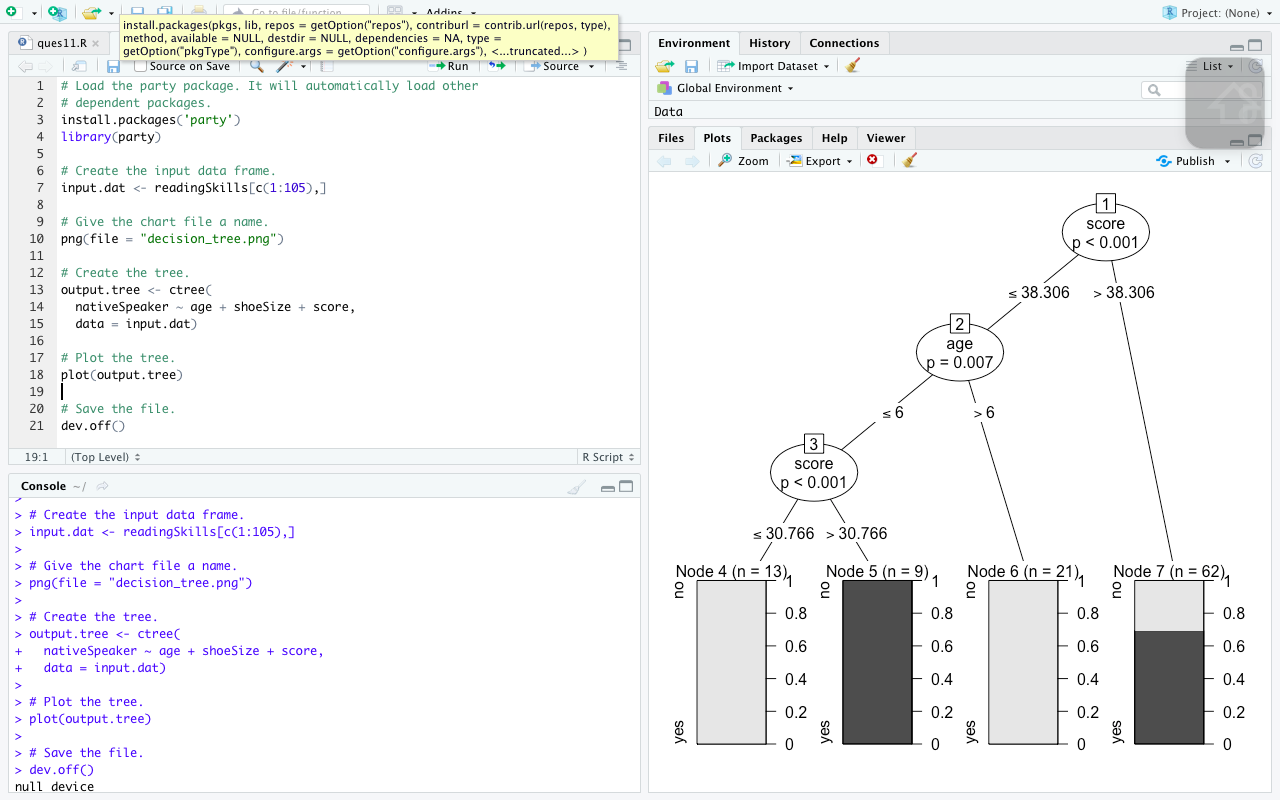
data = input.dat)

# Plot the tree.

plot(output.tree)

# Save the file.

dev.off()



**6. R Random Forest**

# Load the party package. It will automatically load other

# required packages.

library(party)

library(randomForest)

# Create the forest.

output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,

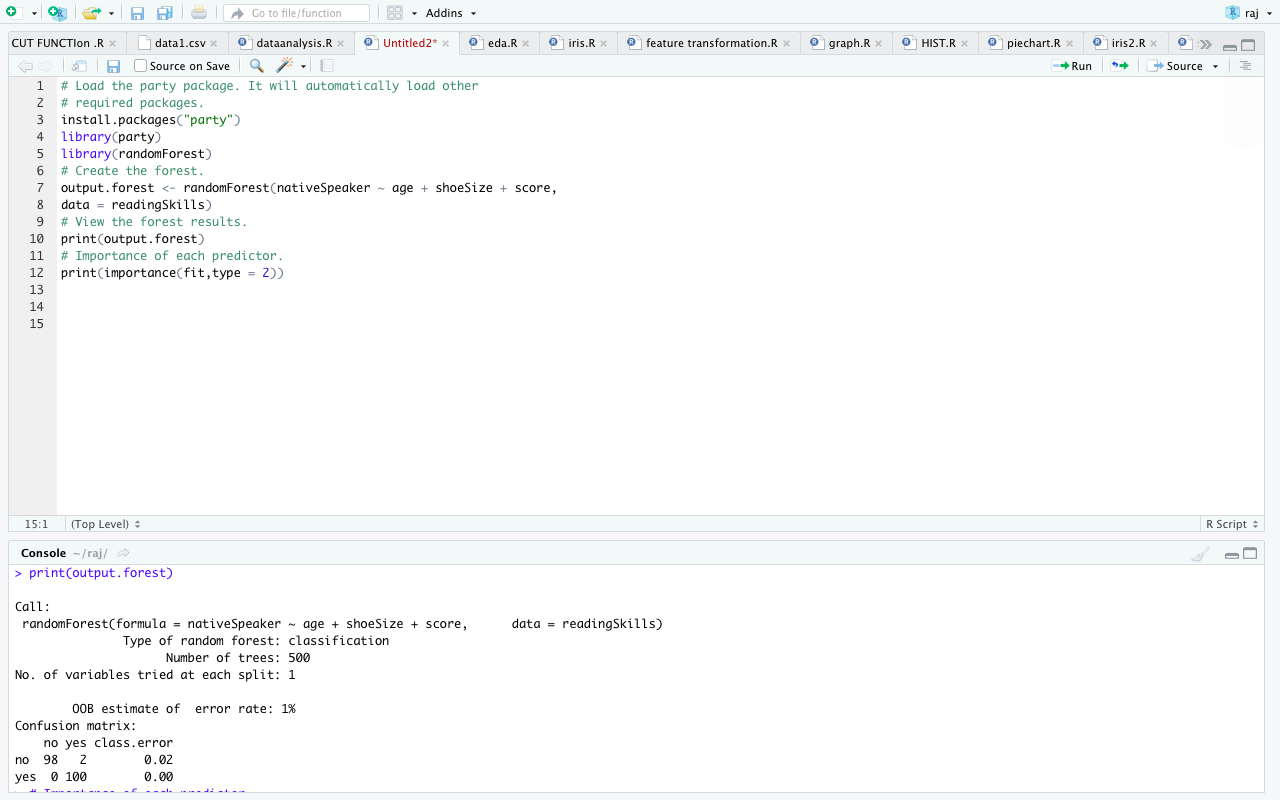
data = readingSkills)

# View the forest results.

print(output.forest)

# Importance of each predictor.

print(importance(fit,type = 2))



**7. Generalized Linear Model (GLM) in R**

*Step 1: Check continuous variables*

*Step 2: Check factor variables*

*Step 3: Feature engineering*

*Step 4: Summary statistic*

*Step 5: Train/test set*

*Step 6: Build the model*

*Step 7: Assess the performance of the model*

*Step 8: Improve the model*

*The other GLM type of models are:*

*- binomial: (link = "logit")*

*- gaussian: (link = "identity")*

*- Gamma: (link = "inverse")*

*- inverse.gaussian: (link = "1/mu^2")*

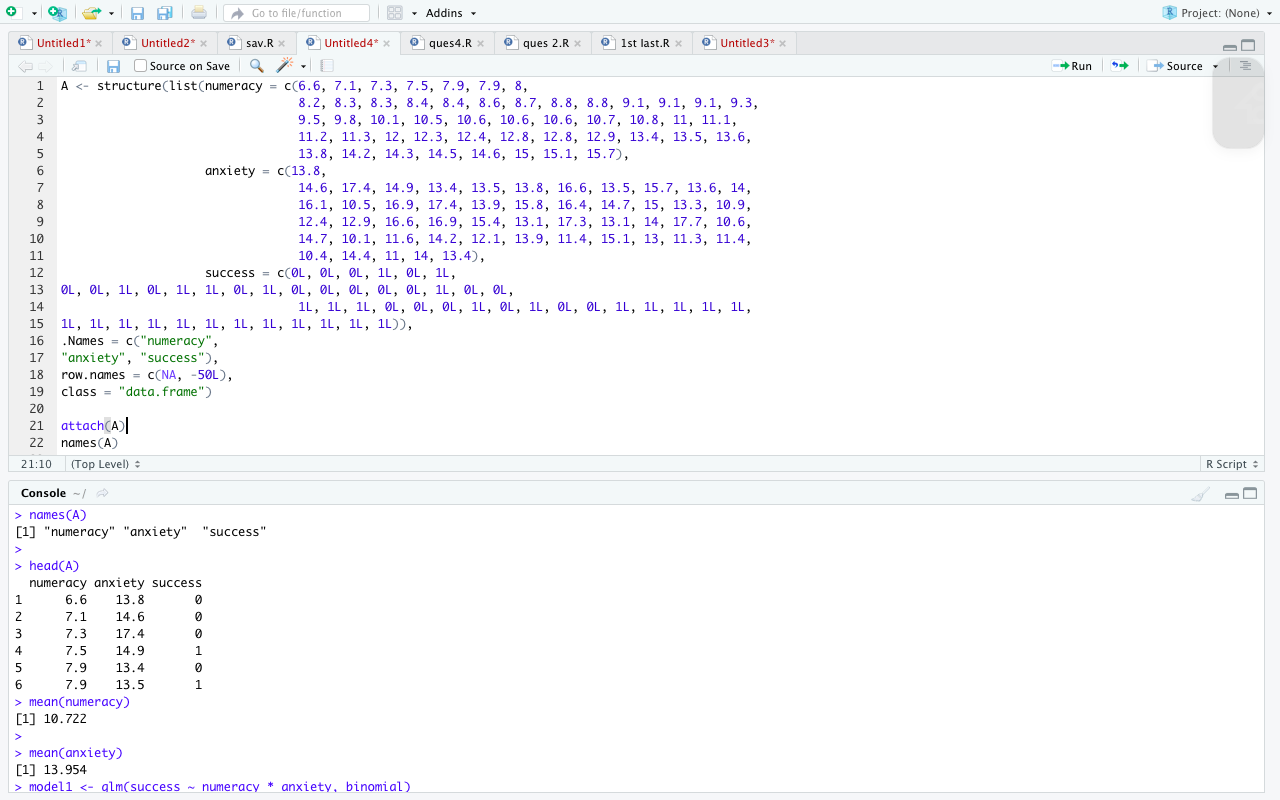
*- poisson: (link = "log")*

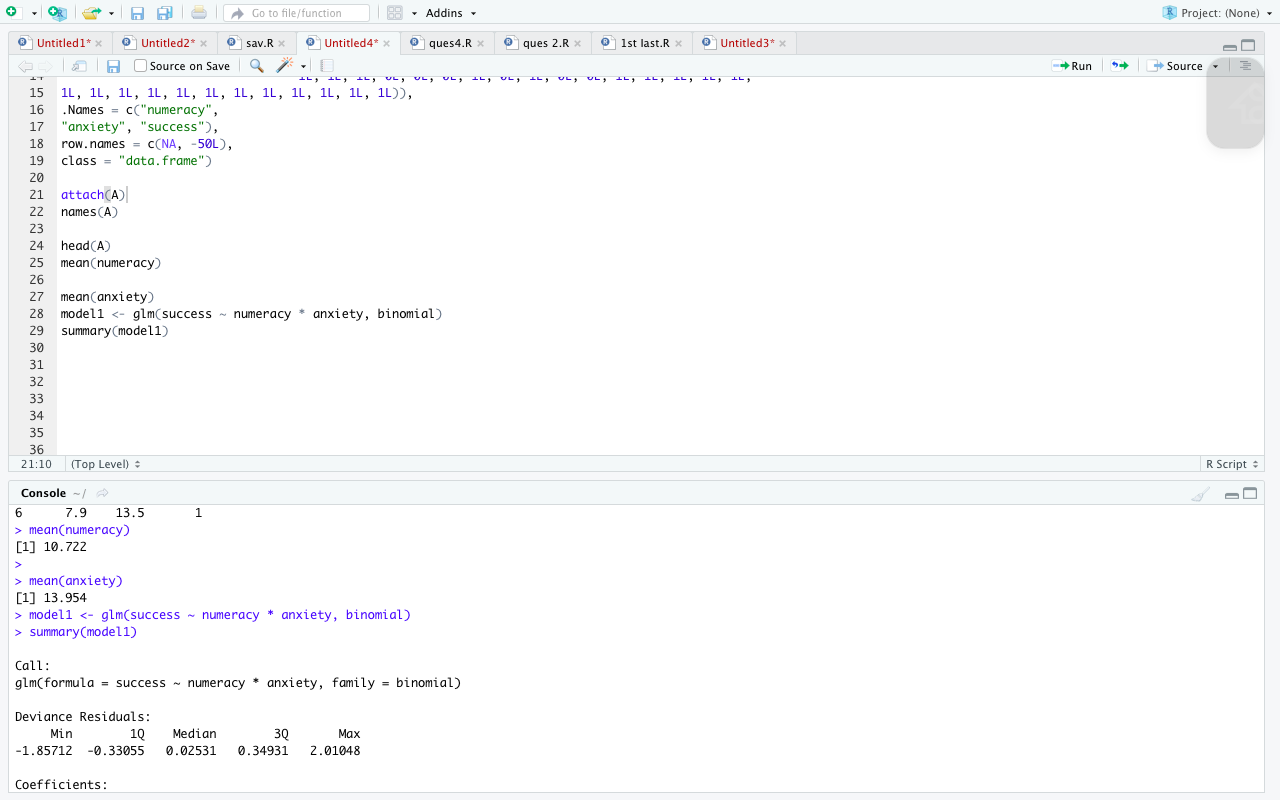
*- quasi: (link = "identity", variance = "constant")*

*- quasibinomial: (link = "logit")*

*- quasipoisson: (link = "log")*

|  |  |  |  |
| --- | --- | --- | --- |
| Create train/test dataset | create\_train\_set() | data, size, train | |
| glm | Train a Generalized Linear Model | glm() | formula, data, family\* |
| glm | Summarize the model | summary() | fitted model |
| base | Make prediction | predict() | fitted model, dataset, type = 'response' |
| base | Create a confusion matrix | table() | y, predict() |
| base | Create accuracy score | sum(diag(table())/sum(table() | |
| ROCR | Create ROC : Step 1 Create prediction | prediction() | predict(), y |
| ROCR | Create ROC : Step 2 Create performance | performance() | prediction(), 'tpr', 'fpr' |
| ROCR | Create ROC : Step 3 Plot graph | plot() | performance() |





**8. K-means Clustering**

R base has a function to run the k mean algorithm. The basic function of k mean is:

kmeans(df, k)

arguments:

-df: dataset used to run the algorithm

-k: Number of clusters

Step 1: R randomly chooses three points

Step 2: Compute the Euclidean distance and draw the clusters. You have one cluster in green at the bottom left, one large cluster colored in black at the right and a red one between them.

Step 3: Compute the centroid, i.e. the mean of the clusters

Repeat until no data changes cluster

R has an amazing variety of functions for cluster analysis. In this section, I will describe three of the many approaches: hierarchical agglomerative, partitioning, and model based. While there are no best solutions for the problem of determining the number of clusters to extract, several approaches are given below.

Data Preparation

Prior to clustering data, you may want to remove or estimate missing data and rescale variables for comparability.

# Prepare Data  
mydata <- na.omit(mydata) # listwise deletion of missing  
mydata <- scale(mydata) # standardize variables

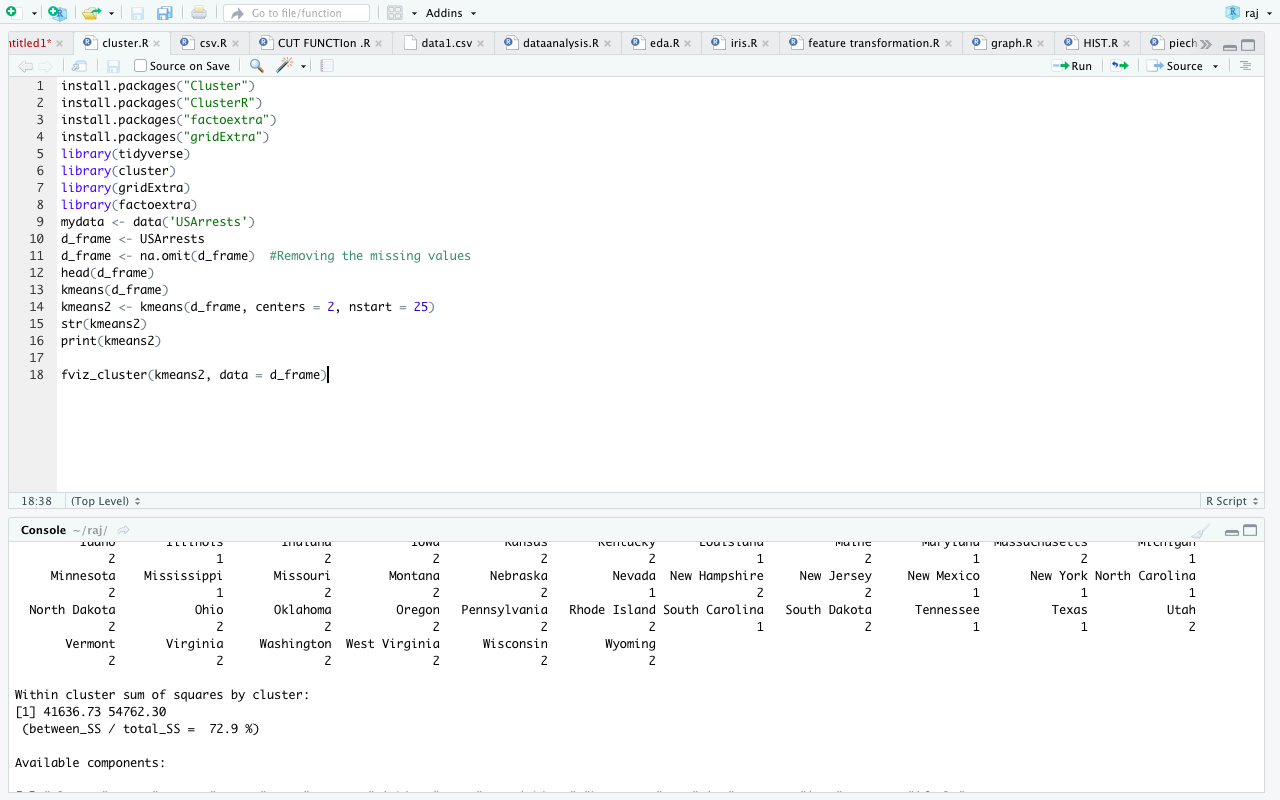
Partitioning

K-means clustering is the most popular partitioning method. It requires the analyst to specify the number of clusters to extract. A plot of the within groups sum of squares by number of clusters extracted can help determine the appropriate number of clusters. The analyst looks for a bend in the plot similar to a scree test in factor analysis.

# Determine number of clusters  
wss <- (nrow(mydata)-1)\*sum(apply(mydata,2,var))  
for (i in 2:15) wss[i] <- sum(kmeans(mydata,   
   centers=i)$withinss)  
plot(1:15, wss, type="b", xlab="Number of Clusters",  
  ylab="Within groups sum of squares")

# K-Means Cluster Analysis  
fit <- kmeans(mydata, 5) # 5 cluster solution  
# get cluster means   
aggregate(mydata,by=list(fit$cluster),FUN=mean)  
# append cluster assignment  
mydata <- data.frame(mydata, fit$cluster)

A robust version of K-means based on mediods can be invoked by using pam( ) instead of kmeans( ). The function pamk( ) in the fpc package is a wrapper for pam that also prints the suggested number of clusters based on optimum average silhouette width.



**9. Exploratory data analysis**

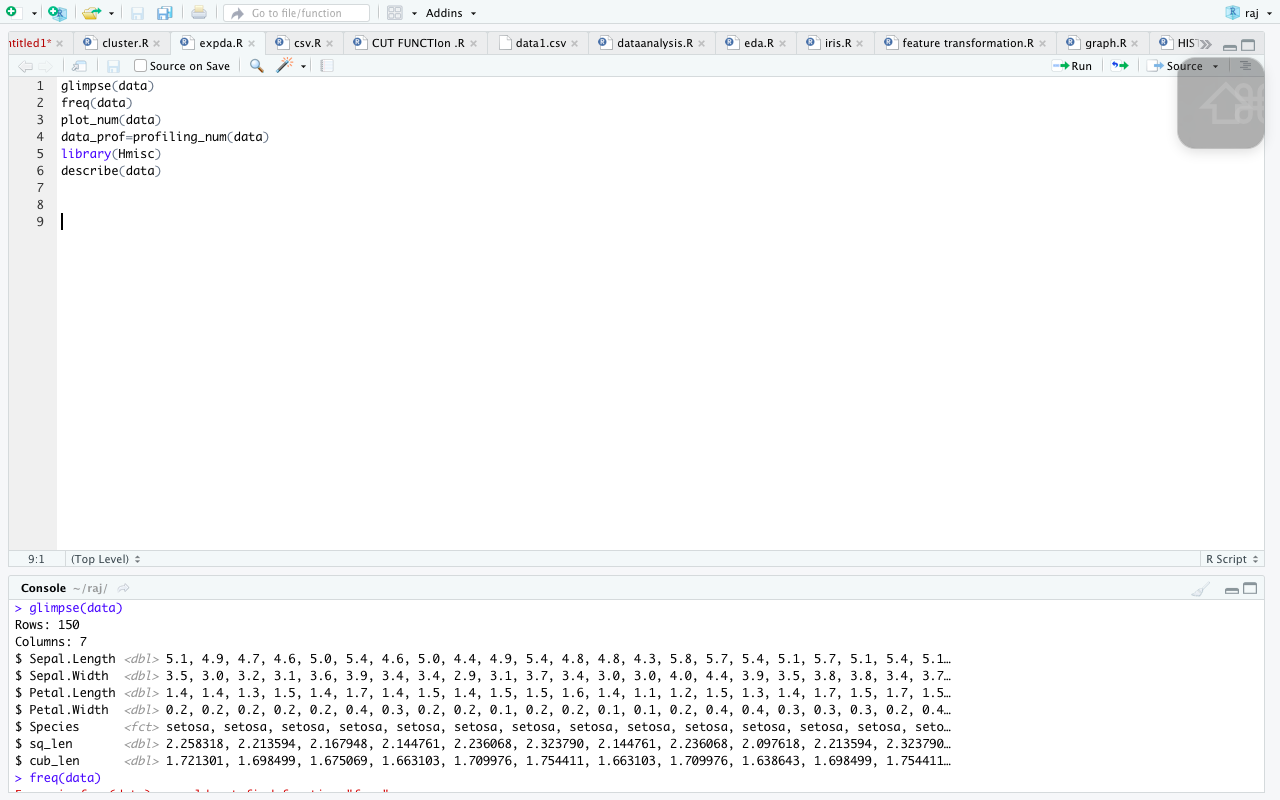
EDA consists of univariate (1-variable) and bivariate (2-variables) analysis.

In this post we will review some functions that lead us to the analysis of the first case.

Step 1 - First approach to data

Number of observations (rows) and variables, and a head of the first cases.

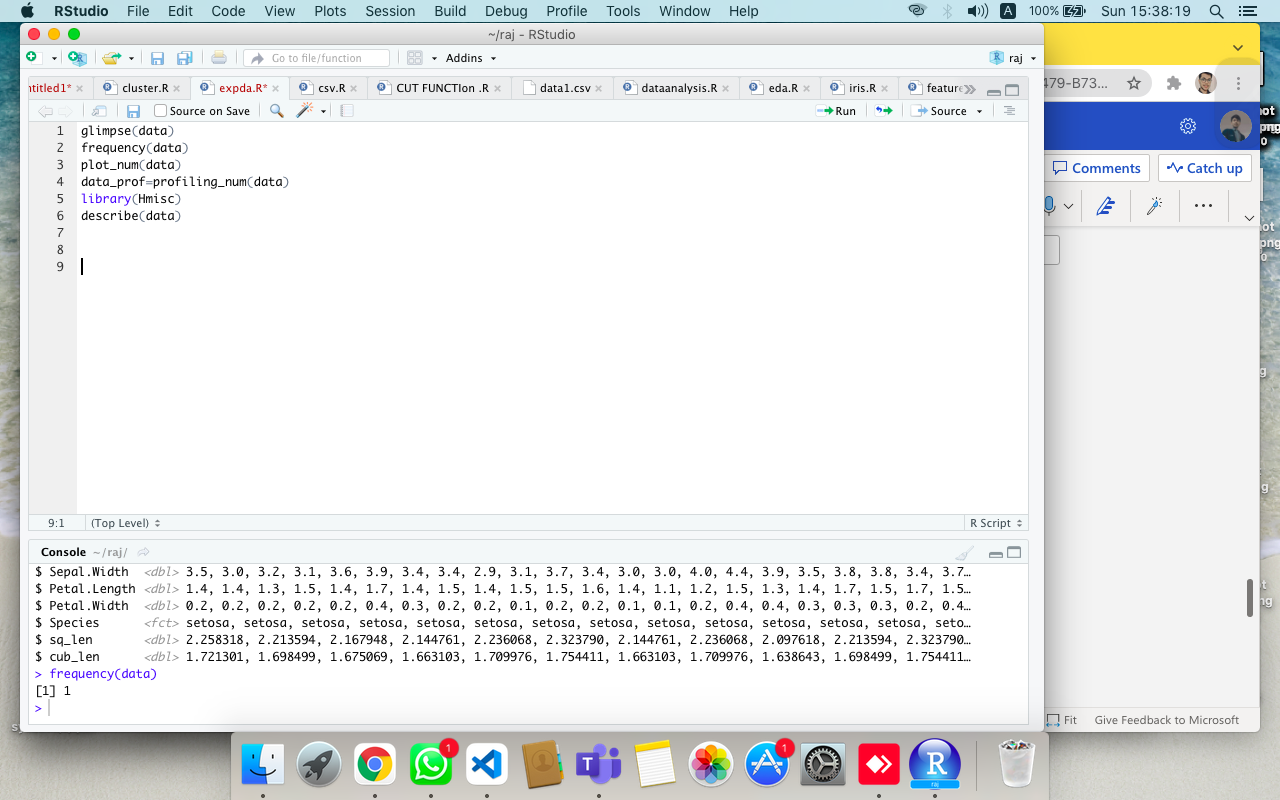
glimpse(data)



Step 2 - Analyzing categorical variables

freq function runs for all factor or character variables automatically:

freq(data)



Step 3 - Analyzing numerical variables

plot\_num and profiling\_num. Both run automatically for all numerical/integer variables:

Graphically

plot\_num(data)

Quantitatively

profiling\_num runs for all numerical/integer variables automatically:

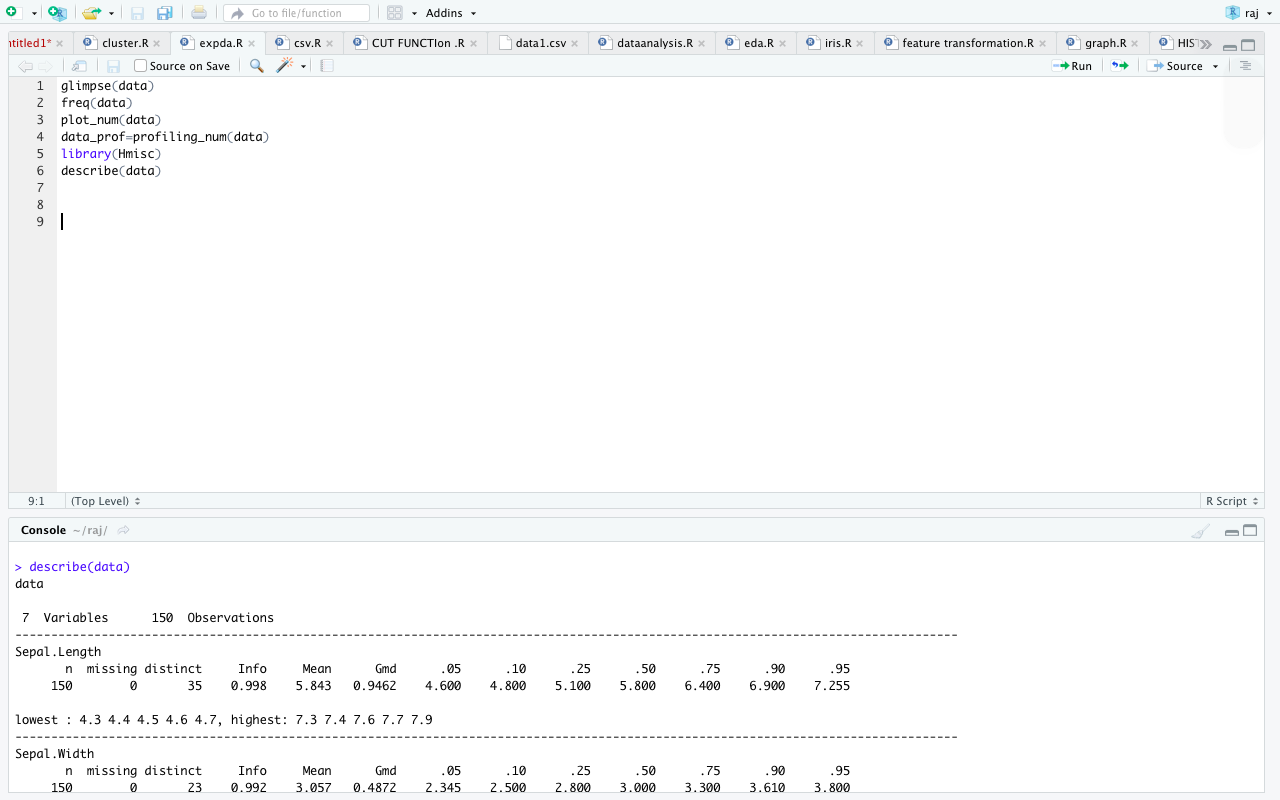
data\_prof=profiling\_num(data)

Step 4 - Analyzing numerical and categorical at the same time

describe from Hmisc package.

library(Hmisc)

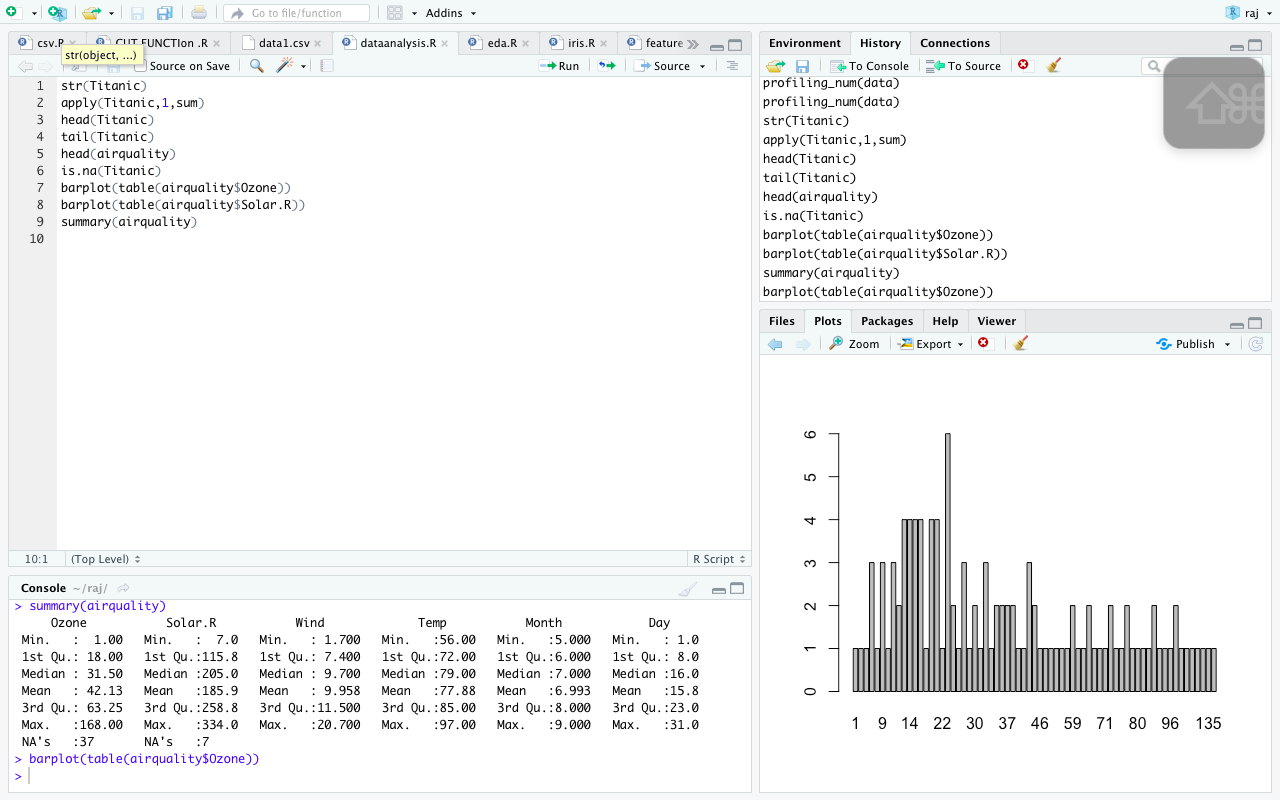
describe(data)



**10. Feature Engineering**

Data Scientists/Analysts often encounter three common problems when presented with new datasets:

* Missing observations: Too many NA values can compromise data and resultant insights
* Poorly formatted data: Data without proper headers can leave the scientist wondering what he’s looking at
* Analysis unfriendly data: Data that has several variables that need to be modified for analysis



**11. Training and test split**

library(ISLR)

attach(Smarket)

smp\_siz = floor(0.75\*nrow(Smarket)) # creates a value for dividing the data into train and test. In this case the value is defined as 75% of the number of rows in the dataset

smp\_siz # shows the value of the sample size

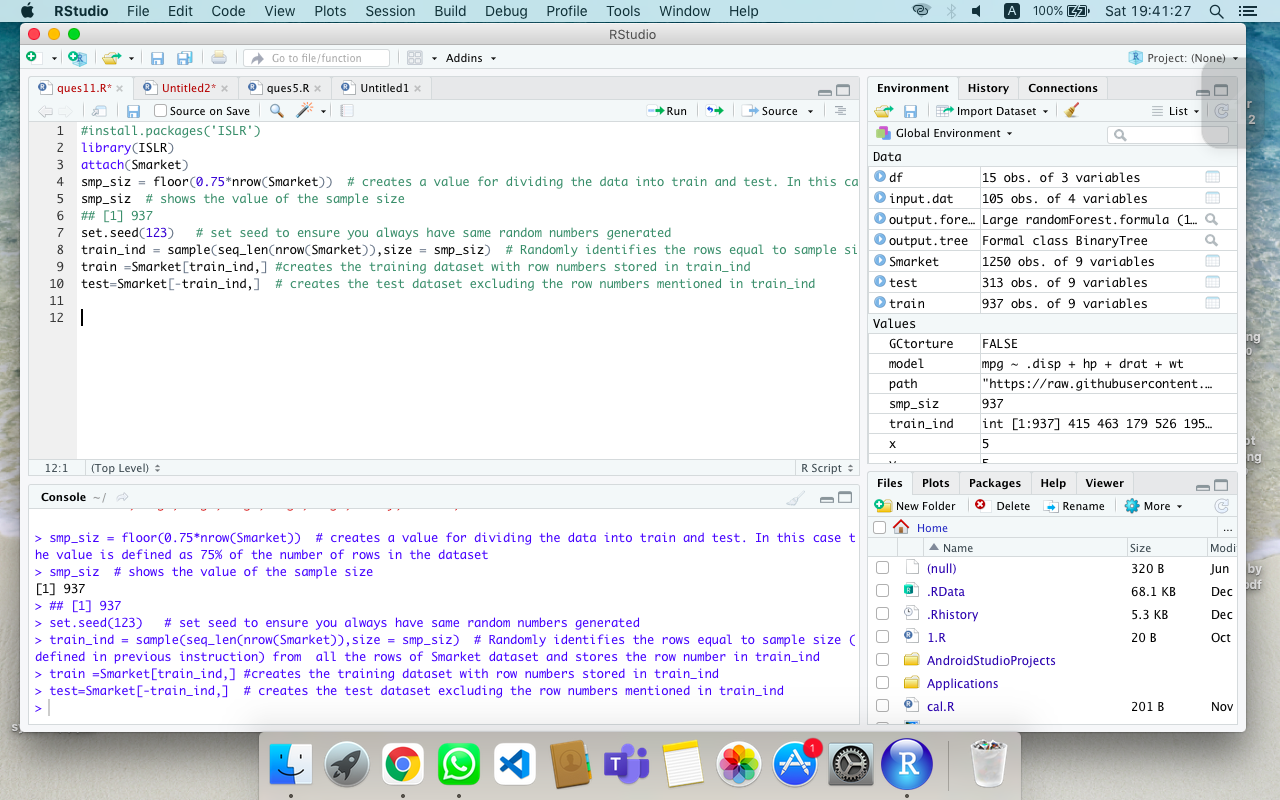
## [1] 937

set.seed(123) # set seed to ensure you always have same random numbers generated

train\_ind = sample(seq\_len(nrow(Smarket)),size = smp\_siz) # Randomly identifies the rows equal to sample size ( defined in previous instruction) from all the rows of Smarket dataset and stores the row number in train\_ind

train =Smarket[train\_ind,] #creates the training dataset with row numbers stored in train\_ind

test=Smarket[-train\_ind,] # creates the test dataset excluding the row numbers mentioned in train\_ind



**12. Model training with Random Forests, Gradient Boosting, Neural Nets, etc.**

1. Loading the dataset.
2. Summarizing the dataset.
3. Visualizing the dataset.
4. Evaluating some algorithms.
5. Making some predictions.

# a) linear algorithms

set.seed(7)

fit.lda <- train(Species~., data=dataset, method="lda", metric=metric, trControl=control)

# b) nonlinear algorithms

# CART

set.seed(7)

fit.cart <- train(Species~., data=dataset, method="rpart", metric=metric, trControl=control)

# kNN

set.seed(7)

fit.knn <- train(Species~., data=dataset, method="knn", metric=metric, trControl=control)

# c) advanced algorithms

# SVM

set.seed(7)

fit.svm <- train(Species~., data=dataset, method="svmRadial", metric=metric, trControl=control)

# Random Forest

set.seed(7)

fit.rf <- train(Species~., data=dataset, method="rf", metric=metric, trControl=control)

