THE NORTHCAP UNIVERSITY™
NAAC ACCREDITED

R Programming

Lab Manual

Department of Computer Science and Engineering
The NorthCap University, Gurugram

Name: Siddharth Gulati
Roll No: 16CSU360

CONTENTS

EXPERIMENT 4

> library(dplyr)
> mydata = read.csv("https://raw.githubusercontent.com/deepanshu88/data/master/sampledata.csv")

#Selecting Random N Rows
#The sample_n function selects random rows from a data frame (or table). The second parameter of the function tells R the number of rows to select.
> sample_n(mydata,3)

#Selecting Random Fraction of Rows
 # The sample_frac function returns randomly N% of rows. In the example below, it returns randomly 10% of rows.
> sample_frac(mydata,0.1)

#Remove Duplicate Rows based on all the variables (Complete Row)
#The distinct function is used to eliminate duplicates.
> x1 = distinct(mydata)

#Remove Duplicate Rows based on a variable
#The .keep_all function is used to retain all other variables in the output data frame.
> x2 = distinct(mydata, Index, .keep_all= TRUE)

#Remove Duplicates Rows based on multiple variables
#In the example below, we are using two variables - Index, Y2010 to determine uniqueness.
> x2 = distinct(mydata, Index, Y2010, .keep_all= TRUE)

#select( ) Function
#Suppose you are asked to select only a few variables. The code below selects variables "Index", columns from "State" to "Y2008".
> mydata2 = select(mydata, Index, State:Y2008)
> head(mydata2, 3)

```
  Index   State  Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008
1     A Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945
229
```

```
2    A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 155182
6
3    A Arizona 1742027 1968140 1377583 1782199 1102568 1109382 17528
86
```

 # Dropping Variables
#The minus sign before a variable tells R to drop the variable.

```
> mydata2 = select(mydata, -Index, -State)
> head(mydata2, 2)
   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010   Y2
011   Y2012
1 1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173
1237582 1440756 1186741
2 1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541
1629616 1230866 1512804
   Y2013   Y2014   Y2015
1 1852841 1558906 1916661
2 1985302 1580394 1979143
```

#The above code can also be written like :

```
> mydata2 = select(mydata, -c(Index,State))
```

#Selecting or Dropping Variables starts with 'Y'
#The starts_with() function is used to select variables starts with an alphabet

```
> mydata3 = select(mydata, starts_with("Y"))
> head(mydata3, 2)
   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010   Y2
011   Y2012
1 1296530 1317711 1118631 1492583 1107408 1440134 1945229 1944173
1237582 1440756 1186741
2 1170302 1960378 1818085 1447852 1861639 1465841 1551826 1436541
1629616 1230866 1512804
   Y2013   Y2014   Y2015
1 1852841 1558906 1916661
2 1985302 1580394 1979143
```

#Adding a negative sign before starts_with() implies dropping the variables s tarts with 'Y'

```
> mydata33 = select(mydata, -starts_with("Y"))
```

```
> head(mydata33)
  Index      State
1     A    Alabama
2     A     Alaska
3     A    Arizona
4     A   Arkansas
5     C California
6     C   Colorado
```

#Selecting Variables contain 'I' in their names

```
> mydata4 = select(mydata, contains("I"))
```

#Reorder Variables
#The code below keeps variable 'State' in the front and the remaining variabl es follow that.

```
> mydata5 = select(mydata, State, everything())
> head(mydata5, 2)
    State Index   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
Y2010   Y2011
1 Alabama     A 1296530 1317711 1118631 1492583 1107408 1440134 1945
229 1944173 1237582 1440756
2  Alaska     A 1170302 1960378 1818085 1447852 1861639 1465841 155182
6 1436541 1629616 1230866
    Y2012   Y2013   Y2014   Y2015
1 1186741 1852841 1558906 1916661
2 1512804 1985302 1580394 1979143
```

#rename( ) Function
#It is used to change variable name.
#In the following code, we are renaming 'Index' variable to 'Index1'.

```
> mydata6 = rename(mydata, Index1=Index)
> head(mydata6, 2)
  Index1   State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y200
9   Y2010
1      A Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945
229 1944173 1237582
2      A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 15518
26 1436541 1629616
    Y2011   Y2012   Y2013   Y2014   Y2015
```

1 1440756 1186741 1852841 1558906 1916661
2 1230866 1512804 1985302 1580394 1979143

#filter( ) Function
#Suppose you need to subset data. You want to filter rows and retain only th ose values in which Index is equal to A.

```
> mydata7 = filter(mydata, Index == "A")
> head(mydata7, 2)
  Index   State  Y2002  Y2003  Y2004  Y2005  Y2006  Y2007  Y2008  Y2009
Y2010  Y2011
1     A Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945
229 1944173 1237582 1440756
2     A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 155182
6 1436541 1629616 1230866
   Y2012  Y2013  Y2014  Y2015
1 1186741 1852841 1558906 1916661
2 1512804 1985302 1580394 1979143
```

#Multiple Selection Criteria
#The %in% operator can be used to select multiple items. In the following pro gram, we are telling R to select rows against 'A' and 'C' in column 'Index'.

```
> mydata7 = filter(mydata6, Index %in% c("A", "C"))
```

#'AND' Condition in Selection Criteria
#Suppose you need to apply 'AND' condition. In this case, we are picking data for 'A' and 'C' in the column 'Index' and income greater than 1.3 million in Year 2002.

```
> mydata8 = filter(mydata6, Index %in% c("A", "C") & Y2002 >= 1300000 )
```

#'OR' Condition in Selection Criteria
#The 'I' denotes OR in the logical condition. It means any of the two condition s.

```
> mydata9 = filter(mydata6, Index %in% c("A", "C") | Y2002 >= 1300000)
```

 # NOT Condition
#The "!" sign is used to reverse the logical condition.

```
> mydata10 = filter(mydata6, !Index %in% c("A", "C"))
```

#CONTAINS Condition

#The grepl function is used to search for pattern matching. In the following code, we are looking for records wherein column state contains 'Ar' in their name.

```
> mydata10 = filter(mydata6, grepl("Ar", State))
> mydata10
  Index1   State  Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009   Y2010
1      A  Arizona 1742027 1968140 1377583 1782199 1102568 1109382 1752886 1554330 1300521
2      A Arkansas 1485531 1994927 1119299 1947979 1669191 1801213 1188104 1628980 1669295
    Y2011   Y2012   Y2013   Y2014   Y2015
1 1130709 1907284 1363279 1525866 1647724
2 1928238 1216675 1591896 1360959 1329341
```

#summarise( ) Function
#In the example below, we are calculating mean and median for the variable Y2015.

```
>summarise(mydata, Y2015_mean = mean(Y2015), Y2015_med=median(Y2015))
  Y2015_mean Y2015_med
1    1588297   1627508
```

#Summarize Multiple Variables
#In the following example, we are calculating number of records, mean and median for variables Y2005 and Y2006. The summarise_at function allows us to select multiple variables by their names.

```
> summarise_at(mydata, vars(Y2005, Y2006), funs(n(), mean, median))
  Y2005_n Y2006_n Y2005_mean Y2006_mean Y2005_median Y2006_median
1      51      51    1522064    1530969      1480280      1531641
```

#OR

```
> summarise_at(mydata, vars(Y2005, Y2006), list(n=~n(), mean=mean, median=median))
```

#OR

```
> summarise_at(mydata, vars(Y2005, Y2006), list(~n(), ~mean(.), ~median(.)))
```

#Summarize with Custom Functions
#Incase you want to add additional arguments for the functions mean and median (for example na.rm = TRUE), you can do it like the code below.

```
> summarise_at(mydata, vars(Y2011, Y2012),funs(mean, median), na.rm = TR
UE)
  Y2011_mean Y2012_mean Y2011_median Y2012_median
1    1574968    1591135      1575533      1643855
```

#We can also use custom functions in the summarise function. In this case, we are computing the number of records, number of missing values, mean and median for variables Y2011 and Y2012. The dot (.) denotes each variables specified in the second argument of the function.

```
> summarise_at(mydata, vars(Y2011, Y2012),
+           funs(n(), missing = sum(is.na(.)), mean(., na.rm = TRUE), median(.,na.
rm = TRUE)))
  Y2011_n Y2012_n Y2011_missing Y2012_missing Y2011_mean Y2012_mean
Y2011_median Y2012_median
1      51      51             0             0    1574968    1591135      1575533      1643855
```

#OR

```
> summarise_at(mydata, vars(Y2011, Y2012),
+           list(~n(), missing = ~sum(is.na(.)), ~mean(., na.rm = TRUE),
+               ~median(.,na.rm = TRUE)))
```

#How to apply Non-Standard Functions
#Suppose you want to subtract mean from its original value and then calculate variance of it.

```
> set.seed(222)
> mydata <- data.frame(X1=sample(1:100,100), X2=runif(100))
> summarise_at(mydata,vars(X1,X2), function(x) var(x - mean(x)))
        X1         X2
1 841.6667 0.07920067
```

#Summarize all Numeric Variables
#The summarise_if function allows you to summarise conditionally

```
> summarise_if(mydata, is.numeric, funs(n(),mean,median))
  X1_n X2_n X1_mean   X2_mean X1_median X2_median
1  100  100    50.5 0.4888369      50.5 0.5128759
```

#Summarize Factor Variable
#We are checking the number of levels/categories and count of missing observations in a categorical (factor) variable.

```
> summarise_all(mydata["Index"], funs(nlevels(.), nmiss=sum(is.na(.))))
```

```
  nlevels nmiss
1      19     0
```

#arrange() function
#To sort a variable in descending order, use desc(x).
#Sort Data by Multiple Variables
#The default sorting order of arrange() function is ascending. In this example , we are sorting data by multiple variables.

```
> arrange(mydata, Index, Y2011)
> head(arrange(mydata, Index, Y2011), 2)
  Index   State  Y2002  Y2003  Y2004  Y2005  Y2006  Y2007  Y2008  Y2009
Y2010  Y2011
1     A Arizona 1742027 1968140 1377583 1782199 1102568 1109382 17528
86 1554330 1300521 1130709
2     A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 155182
6 1436541 1629616 1230866
    Y2012   Y2013   Y2014   Y2015
1 1907284 1363279 1525866 1647724
2 1512804 1985302 1580394 1979143
```

#Suppose you need to sort one variable by descending order and other variable by ascending oder.

```
> arrange(mydata, desc(Index), Y2011)
> head(arrange(mydata, desc(Index), Y2011), 2)
  Index        State  Y2002  Y2003  Y2004  Y2005  Y2006  Y2007  Y2008  Y2
009  Y2010
1     W    Washington 1977749 1687136 1199490 1163092 1334864 1621989
1545621 1555554 1179331
2     W West Virginia 1677347 1380662 1176100 1888948 1922085 1740826 1
238174 1539322 1539603
    Y2011   Y2012   Y2013   Y2014   Y2015
1 1150089 1775787 1273834 1387428 1377341
2 1872519 1462137 1683127 1204344 1198791
```

#Pipe Operator %>%
#It is important to understand the pipe (%>%) operator before knowing the other functions of dplyr package. dplyr utilizes pipe operator from another package (magrittr).
#It allows you to write sub-queries like we do it in sql.

#The code below demonstrates the usage of pipe %>% operator. In this example, we are selecting 10 random observations of two variables "Index" "State" from the data frame "mydata".

```
> dt = sample_n(select(mydata, Index, State),10)
> dt
  Index      State
1     O   Oklahoma
2     M      Maine
3     M Mississippi
4     W    Wyoming
5     V   Virginia
6     I      Idaho
7     N     Nevada
8     A    Arizona
9     M   Michigan
10    K   Kentucky
```

#OR

```
> dt = mydata %>% select(Index, State) %>% sample_n(10)
```

#group_by() function
#Use : Group data by categorical variable
#Summarise Data by Categorical Variable
#We are calculating count and mean of variables Y2011 and Y2012 by variable Index.

```
> t = summarise_at(group_by(mydata, Index), vars(Y2011, Y2012), funs(n(), mean(., na.rm = TRUE)))
> head(t)
# A tibble: 6 x 5
 Index Y2011_n Y2012_n Y2011_mean Y2012_mean
 <fct>   <int>   <int>      <dbl>      <dbl>
1 A          4       4   1432642.    1455876
2 C          3       3    1750357    1547326
3 D          2       2    1336059   1981868.
4 F          1       1    1497051    1131928
5 G          1       1    1851245    1850111
6 H          1       1    1902816    1695126
```

#The above code can also be written like

```
> t = mydata %>% group_by(Index) %>%
```

```
+   summarise_at(vars(Y2011:Y2015), funs(n(), mean(., na.rm = TRUE)))
> head(t)
# A tibble: 6 x 11
  Index Y2011_n Y2012_n Y2013_n Y2014_n Y2015_n Y2011_mean Y2012_me
an Y2013_mean Y2014_mean
  <fct>   <int>   <int>   <int>   <int>   <int>      <dbl>      <dbl>      <dbl>      <dbl>
1 A           4       4       4       4       4   1432642.   1455876   1698330.   1506531.
2 C           3       3       3       3       3   1750357    1547326   1305713.   1425198.
3 D           2       2       2       2       2   1336059    1981868.  1791966.   1792669
4 F           1       1       1       1       1   1497051    1131928   1107448    1407784
5 G           1       1       1       1       1   1851245    1850111   1887157    1259353
6 H           1       1       1       1       1   1902816    1695126   1517184    1948108
# ... with 1 more variable: Y2015_mean <dbl>
```

#do() function
#Use : Compute within groups
#Filter Data within a Categorical Variable
**#Suppose you need to pull top 2 rows from 'A', 'C' and 'I' categories of variabl
e**
Index.

```
> t = mydata %>% filter(Index %in% c("A", "C","I")) %>% group_by(Index) %>%
+   do(head( . , 2))
> t
# A tibble: 6 x 16
# Groups:   Index [3]
  Index State  Y2002  Y2003  Y2004  Y2005  Y2006  Y2007  Y2008  Y2009  Y20
10  Y2011  Y2012
  <fct> <fct>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>  <int>
1 A     Alab~ 1.30e6 1.32e6 1.12e6 1.49e6 1.11e6 1.44e6 1.95e6 1.94e6 1.24e
6 1.44e6 1.19e6
2 A     Alas~ 1.17e6 1.96e6 1.82e6 1.45e6 1.86e6 1.47e6 1.55e6 1.44e6 1.63e
6 1.23e6 1.51e6
3 C     Cali~ 1.69e6 1.68e6 1.89e6 1.48e6 1.74e6 1.81e6 1.49e6 1.66e6 1.62e6
1.64e6 1.92e6
4 C     Colo~ 1.34e6 1.88e6 1.89e6 1.24e6 1.87e6 1.81e6 1.88e6 1.75e6 1.91e
6 1.67e6 1.49e6
5 I     Idaho 1.35e6 1.44e6 1.74e6 1.54e6 1.12e6 1.77e6 1.34e6 1.75e6 1.44e6
1.46e6 1.64e6
```

```
6 I    Illi~ 1.51e6 1.53e6 1.49e6 1.26e6 1.54e6 1.75e6 1.87e6 1.66e6 1.42e6 1.
75e6 1.70e6
# ... with 3 more variables: Y2013 <int>, Y2014 <int>, Y2015 <int>
```

#Selecting 3rd Maximum Value by Categorical Variable
#We are calculating third maximum value of variable Y2015 by variable Index . The following code first selects only two variables Index and Y2015. Then it filters the variable Index with 'A', 'C' and 'I' and then it groups the same variable and sorts the variable Y2015 in descending order. At last, it selects the third row.
#The slice() function is used to select rows by position.

```
> t = mydata %>% select(Index, Y2015) %>%
+   filter(Index %in% c("A", "C","I")) %>%
+   group_by(Index) %>%
+   do(arrange(.,desc(Y2015))) %>%  slice(3)
> t
# A tibble: 3 x 2
# Groups:   Index [3]
  Index   Y2015
  <fct>   <int>
1 A     1647724
2 C     1330736
3 I     1583516
```

#Using Window Functions
#Like SQL, dplyr uses window functions that are used to subset data within a group. It returns a vector of values. We could use min_rank() function that calculates rank in the preceding example,

```
> t = mydata %>% select(Index, Y2015) %>%
+   filter(Index %in% c("A", "C","I")) %>%
+   group_by(Index) %>%
+   filter(min_rank(desc(Y2015)) == 3)
> t
# A tibble: 3 x 2
# Groups:   Index [3]
  Index   Y2015
  <fct>   <int>
1 A     1647724
2 C     1330736
```

```
3 I     1583516
```

#Summarize, Group and Sort Together
#In this case, we are computing mean of variables Y2014 and Y2015 by variable Index. Then sort the result by calculated mean variable Y2015.

```
> t = mydata %>%
+   group_by(Index)%>%
+   summarise(Mean_2014 = mean(Y2014, na.rm=TRUE),
+             Mean_2015 = mean(Y2015, na.rm=TRUE)) %>%
+   arrange(desc(Mean_2015))
> head(t)
# A tibble: 6 x 3
  Index Mean_2014 Mean_2015
  <fct>     <dbl>     <dbl>
1 U       1801019   1729273
2 G       1259353   1725470
3 A       1506531.  1718217.
4 M       1596816.  1710808.
5 V       1494748.  1708159
6 P       1931500   1668232
```

#mutate() function
#Use :Creates new variables
#Create a new variable
#The following code calculates division of Y2015 by Y2014 and name it "change".
> mydata1 = mutate(mydata, change=Y2015/Y2014)

```
> head(mydata11, 2)
  Index   State  Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
Y2010   Y2011
1     A Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945
229 1944173 1237582 1440756
2     A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 155182
6 1436541 1629616 1230866
    Y2012   Y2013   Y2014   Y2015 Index_new State_new  Y2002_new  Y2003_n
ew  Y2004_new
1 1186741 1852841 1558906 1916661        NA        NA 1296530000 13177110
00 1118631000
```

2 1512804 1985302 1580394 1979143 NA NA 1170302000 19603780
00 1818085000
Y2005_new Y2006_new Y2007_new Y2008_new Y2009_new Y2010_new
Y2011_new Y2012_new
1 1492583000 1107408000 1440134000 1945229000 1944173000 12375820
00 1440756000 1186741000
2 1447852000 1861639000 1465841000 1551826000 1436541000 16296160
00 1230866000 1512804000
Y2013_new Y2014_new Y2015_new
1 1852841000 1558906000 1916661000
2 1985302000 1580394000 1979143000

#Multiply all the variables by 1000
#It creates new variables and name them with suffix "_new".

> mydata11 = mutate_all(mydata, funs("new" = .* 1000))

#Warning messages:
#It implies you are multiplying 1000 to string(character) values which are stored as factor variables. These variables are 'Index', 'State'. It does not make sense to apply multiplication operation on character variables. For these two variables, it creates newly created variables which contain only NA.

#Calculate Rank for Variables
#Suppose you need to calculate rank for variables Y2008 to Y2010.

```
> mydata12 = mutate_at(mydata, vars(Y2008:Y2010), funs(Rank=min_rank(.)))
> head(mydata12, 2)
  Index   State   Y2002   Y2003   Y2004   Y2005   Y2006   Y2007   Y2008   Y2009
Y2010   Y2011
1     A Alabama 1296530 1317711 1118631 1492583 1107408 1440134 1945
229 1944173 1237582 1440756
2     A  Alaska 1170302 1960378 1818085 1447852 1861639 1465841 155182
6 1436541 1629616 1230866
    Y2012   Y2013   Y2014   Y2015 Y2008_Rank Y2009_Rank Y2010_Rank
1 1186741 1852841 1558906 1916661         47         46          8
2 1512804 1985302 1580394 1979143         27          9         38
```

#By default, min_rank() assigns 1 to the smallest value and high number to the largest value. In case, you need to assign rank 1 to the largest value of a variable, use min_rank(desc(.))

> mydata13 = mutate_at(mydata, vars(Y2008:Y2010), funs(Rank=min_rank(desc(.))))

#Select State that generated highest income among the variable 'Index'

```
> out = mydata %>% group_by(Index) %>% filter(min_rank(desc(Y2015)) == 1)
%>% select(Index, State, Y2015)
> head(out)
# A tibble: 6 x 3
# Groups:   Index [6]
  Index State       Y2015
  <fct> <fct>       <int>
1 A     Alaska      1979143
2 C     Connecticut 1718072
3 D     Delaware    1627508
4 F     Florida     1170389
5 G     Georgia     1725470
6 H     Hawaii      1150882
```

#Cumulative Income of 'Index' variable
#The cumsum function calculates cumulative sum of a variable. With mutate function, we insert a new variable called 'Total' which contains values of cum ulative income of variable Index.

```
> out2 = mydata %>% group_by(Index) %>% mutate(Total=cumsum(Y2015)) %
>%
+   select(Index, Y2015, Total)
> out2
# A tibble: 51 x 3
# Groups:   Index [19]
   Index   Y2015   Total
   <fct>   <int>   <int>
 1 A     1916661 1916661
 2 A     1979143 3895804
 3 A     1647724 5543528
 4 A     1329341 6872869
 5 C     1644607 1644607
 6 C     1330736 2975343
 7 C     1718072 4693415
 8 D     1627508 1627508
 9 D     1410183 3037691
10 F     1170389 1170389
# ... with 41 more rows
```

#join() function
#Use : Join two datasets

```
> df1 = data.frame(ID = c(1, 2, 3, 4, 5),
+               w = c('a', 'b', 'c', 'd', 'e'),
+               x = c(1, 1, 0, 0, 1),
+               y=rnorm(5),
+               z=letters[1:5])
> df2 = data.frame(ID = c(1, 7, 3, 6, 8),
+               a = c('z', 'b', 'k', 'd', 'l'),
+               b = c(1, 2, 3, 0, 4),
+               c =rnorm(5),
+               d =letters[2:6])
```

#INNER JOIN returns rows when there is a match in both tables. In this example, we are merging df1 and df2 with ID as common variable (primary key).

```
> df3 = inner_join(df1, df2, by = "ID")
> df3
  ID w x          y z a b          c d
1  1 a 1 -0.6867824 a z 1 -1.1669585 b
2  3 c 0  0.1151648 c k 3 -0.3271407 d
```

#If the primary key does not have same name in both the tables, try the following way:

```
> inner_join(df1, df2, by = c("ID"="ID1"))
```

#Applying LEFT JOIN
#LEFT JOIN : It returns all rows from the left table, even if there are no matches in the right table.

```
> left_join(df1, df2, by = "ID")
  ID w x           y z    a  b          c    d
1  1 a 1  0.06096335 a    z  1  1.6174028    b
2  2 b 1 -1.00450073 b <NA> NA         NA <NA>
3  3 c 0  0.54243043 c    k  3 -0.3567927    d
4  4 d 0  0.67425149 d <NA> NA         NA <NA>
5  5 e 1  0.20212179 e <NA> NA         NA <NA>
```

#Combine Data Vertically
#Rows that appear in both x and y.

```
> intersect(x, y)
```

#Rows that appear in either or both x and y.
```
> union(x, y)
```

#Rows that appear in x but not y.
```
> setdiff(x, y)
```

#Applying INTERSECT
#Prepare Sample Data
```
> mtcars$model <- rownames(mtcars)
> first <- mtcars[1:20, ]
> second <- mtcars[10:32, ]
```
#INTERSECT selects unique rows that are common to both the data frames.
```
> head(intersect(first, second))
   mpg cyl  disp  hp drat    wt  qsec vs am gear carb              model
1 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4           Merc 280
2 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4          Merc 280C
3 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3         Merc 450SE
4 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3         Merc 450SL
5 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3        Merc 450SLC
6 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4 Cadillac Fleetwood
```

#Applying UNION
#UNION displays all rows from both the tables and removes duplicate record s from the combined dataset. By using union_all function, it allows duplicate rows in the combined dataset.
```
> x=data.frame(ID = 1:6, ID1= 1:6)
> y=data.frame(ID = 1:6,  ID1 = 1:6)
> union(x,y)
  ID ID1
1  1   1
2  2   2
3  3   3
4  4   4
5  5   5
6  6   6

> union_all(x,y)
   ID ID1
```

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

#Rows appear in one table but not in other table

```
> setdiff(first, second)
   mpg cyl  disp  hp drat    wt  qsec vs am gear carb             model
1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4         Mazda RX4
2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4     Mazda RX4 Wag
3 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1        Datsun 710
4 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1    Hornet 4 Drive
5 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2 Hornet Sportabout
6 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1           Valiant
7 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4        Duster 360
8 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2         Merc 240D
9 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2          Merc 230
```

#IF ELSE Statement
#true : Value if condition meets
#false : Value if condition does not meet
#missing : Value if missing cases.It will be used to replace missing values (Default : NULL)

```
> df <- c(-10,2, NA)
> if_else(df < 0, "negative", "positive", missing = "missing value")
[1] "negative"     "positive"     "missing value"
```

#Create a new variable with IF_ELSE
#If a value is less than 5, add it to 1 and if it is greater than or equal to 5, add it to 2. Otherwise 0.

```
> df =data.frame(x = c(1,5,6,NA))
```

```
> df %>% mutate(newvar=if_else(x<5, x+1, x+2,0))
   x newvar
1  1      2
2  5      7
3  6      8
4 NA      0
```

#Nested IF ELSE
#Multiple IF ELSE statement can be written using if_else() function. See the e xample below -

```
> mydf =data.frame(x = c(1:5,NA))
> mydf %>% mutate(newvar= if_else(is.na(x),"I am missing",
+                         if_else(x==1,"I am one",
+                                 if_else(x==2,"I am two",
+                                         if_else(x==3,"I am three","Others")))))
   x      newvar
1  1    I am one
2  2    I am two
3  3  I am three
4  4      Others
5  5      Others
6 NA I am missing
```

#SQL-Style CASE WHEN Statement
#We can use case_when() function to write nested if-else queries. In case_w hen(), you can use variables directly within case_when() wrapper. TRUE refer s to ELSE statement.

```
> mydf %>% mutate(flag = case_when(is.na(x) ~ "I am missing",
+                          x == 1 ~ "I am one",
+                          x == 2 ~ "I am two",
+                          x == 3 ~ "I am three",
+                          TRUE ~ "Others"))
   x        flag
1  1    I am one
2  2    I am two
3  3  I am three
4  4      Others
5  5      Others
6 NA I am missing
```

#Make sure you set is.na() condition at the beginning in nested ifelse. Other wise, it would not be executed.

#Apply ROW WISE Operation
#Suppose you want to find maximum value in each row of variables 2012, 20 13, 2014, 2015. The rowwise() function allows you to apply functions to rows .

```
> df = mydata %>%
+   rowwise() %>% mutate(Max= max(Y2012,Y2013,Y2014,Y2015)) %>%
+   select(Y2012:Y2015,Max)
> head(df)
Source: local data frame [6 x 5]
Groups: <by row>

# A tibble: 6 x 5
   Y2012   Y2013   Y2014   Y2015     Max
   <int>   <int>   <int>   <int>   <int>
1 1186741 1852841 1558906 1916661 1916661
2 1512804 1985302 1580394 1979143 1985302
3 1907284 1363279 1525866 1647724 1907284
4 1216675 1591896 1360959 1329341 1591896
5 1921845 1156536 1388461 1644607 1921845
6 1491604 1178355 1383978 1330736 1491604
```

#Combine Data Frames
#Suppose you are asked to combine two data frames. Let's first create two s ample datasets.

```
> df1=data.frame(ID = 1:6,  x=letters[1:6])
> df2=data.frame(ID = 7:12, x=letters[7:12])
```

#The bind_rows() function combine two datasets with rows. So combined dat aset would contain 12 rows (6+6) and 2 columns.

```
> xy = bind_rows(df1,df2)
> xy
  ID x
1   1 a
2   2 b
3   3 c
4   4 d
```

```
5   5 e
6   6 f
7   7 g
8   8 h
9   9 i
10 10 j
11 11 k
12 12 l
```

#It is equivalent to base R function rbind.

```
> xy = rbind(df1,df2)
```

#The bind_cols() function combine two datasets with columns. So combined dataset would contain 4 columns and 6 rows.

```
> xy = bind_cols(df1,df2)
> xy
  ID x ID1 x1
1  1 a   7  g
2  2 b   8  h
3  3 c   9  i
4  4 d  10  j
5  5 e  11  k
6  6 f  12  l
```

#OR

```
> xy = cbind(x,y)
```

#Calculate Percentile Values
#The quantile() function is used to determine Nth percentile value. In this ex ample, we are computing percentile values by variable Index.

```
> mydata %>% group_by(Index) %>%
+   summarise(Pecentile_25=quantile(Y2015, probs=0.25),
+           Pecentile_50=quantile(Y2015, probs=0.5),
+           Pecentile_75=quantile(Y2015, probs=0.75),
+           Pecentile_99=quantile(Y2015, probs=0.99))
# A tibble: 19 x 5
   Index Pecentile_25 Pecentile_50 Pecentile_75 Pecentile_99
   <fct>        <dbl>        <dbl>        <dbl>        <dbl>
 1 A         1568128.     1782192.     1932282.     1977269.
```

```
 2 C       1487672.   1644607    1681340.   1716603.
 3 D       1464514.   1518846.   1573177.   1625335.
 4 F       1170389    1170389    1170389    1170389
 5 G       1725470    1725470    1725470    1725470
 6 H       1150882    1150882    1150882    1150882
 7 I       1554540.   1612691    1670692.   1753712.
 8 K       1517484.   1649439    1781394.   1908072.
 9 L       1403857    1403857    1403857    1403857
10 M       1559483.   1755594.   1970311.   1995671.
11 N       1333050    1703164.   1877410    1959147.
12 O       1440398.   1573117    1733316    1887107.
13 P       1668232    1668232    1668232    1668232
14 R       1611730    1611730    1611730    1611730
15 S       1117102    1123549    1129996    1136185.
16 T       1297954.   1433743    1569532.   1699890.
17 U       1729273    1729273    1729273    1729273
18 V       1637042.   1708159    1779276.   1847549.
19 W       1332704.   1611790.   1848143    1853629.
```

#The ntile() function is used to divide the data into N bins.

```
> x= data.frame(N= 1:10)
> x = mutate(x, pos = ntile(x$N,5))
> x
    N pos
1   1   1
2   2   1
3   3   2
4   4   2
5   5   3
6   6   3
7   7   4
8   8   4
9   9   5
10 10   5
```

#Automate Model Building
#This example explains the advanced usage of do() function. In this example , we are building linear regression model for each level of a categorical varia ble. There are 3 levels in variable cyl of dataset mtcars.

```
> length(unique(mtcars$cyl))
[1] 3

> by_cyl <- group_by(mtcars, cyl)
> models <- by_cyl %>% do(mod = lm(mpg ~ disp, data = .))
> summarise(models, rsq = summary(mod)$r.squared)
# A tibble: 3 x 1
    rsq
  <dbl>
1 0.648
2 0.0106
3 0.270

> models %>% do(data.frame(
+   var = names(coef(.$mod)),
+   coef(summary(.$mod)))
+ )
Source: local data frame [6 x 5]
Groups: <by row>

# A tibble: 6 x 5
  var         Estimate Std..Error t.value   Pr...t..
* <fct>          <dbl>      <dbl>   <dbl>      <dbl>
1 (Intercept) 40.9       3.59     11.4   0.0000000120
2 disp        -0.135     0.0332   -4.07  0.00278
3 (Intercept) 19.1       2.91      6.55  0.00124
4 disp         0.00361   0.0156    0.232 0.826
5 (Intercept) 22.0       3.35      6.59  0.0000259
6 disp        -0.0196    0.00932  -2.11  0.0568
```

#if() Family of Functions
#It includes functions like select_if, mutate_if, summarise_if. They come into action only when logical condition meets.
#Select only numeric columns
#The select_if() function returns only those columns where logical condition is TRUE. The is.numeric refers to retain only numeric variables.

```
> mydata2 = select_if(mydata, is.numeric)
```

#Similarly, you can use the following code for selecting factor columns

> mydata3 = select_if(mydata, is.factor)

#Number of levels in factor variables
#Like select_if() function, summarise_if() function lets you to summarise onl y for variables where logical condition holds.

```
> summarise_if(mydata, is.factor, funs(nlevels(.)))
  Index State
1    19    51
```

#Multiply by 1000 to numeric variables

```
> mydata11 = mutate_if(mydata, is.numeric, funs("new" = .* 1000))
```

#Convert value to NA
#In this example, we are converting "" to NA using na_if() function.

```
> k <- c("a", "b", "", "d")
> na_if(k, "")
[1] "a" "b" NA  "d"
```

#Use of pull( ) function
#iris %>% pull(Sepal.Length) is equivalent to writing iris$Sepal.Length or iris[["Sepal.Length"]] If you want output to be in vector rather than data frame (de fault method), you can use pull( ) function.

```
> iris %>% filter(Sepal.Length > 5.5) %>% pull(Species)
 [1] setosa     setosa     setosa     versicolor versicolor versicolor versicolor ver
sicolor
 [9] versicolor versicolor versicolor versicolor versicolor versicolor versicolor v
ersicolor
[17] versicolor versicolor versicolor versicolor versicolor versicolor versicolor v
ersicolor
[25] versicolor versicolor versicolor versicolor versicolor versicolor versicolor v
ersicolor
[33] versicolor versicolor versicolor versicolor versicolor versicolor versicolor v
ersicolor
[41] versicolor versicolor virginica  virginica  virginica  virginica  virginica  virgin
ica
[49] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginic
a
[57] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginic
a
```

```
[65] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginic
a
[73] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginic
a
[81] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virginic
a
[89] virginica  virginica  virginica
Levels: setosa versicolor virginica
```

EXPERIMENT – 5 (STRING+REGEX)

> states = rownames(USArrests)

 # substr
 # suppose we want to abbreviate the names using the first four characters of each name. One way to do that is by
#using the function substr() which substrings a character vector
> substr(x = states, start = 1, stop = 4)
 [1] "Alab" "Alas" "Ariz" "Arka" "Cali" "Colo" "Conn" "Dela" "Flor" "Geor" "Hawa" "Id
ah" "Illi"
[14] "Indi" "Iowa" "Kans" "Kent" "Loui" "Main" "Mary" "Mass" "Mich" "Minn" "Miss
" "Miss" "Mont"
[27] "Nebr" "Neva" "New " "New " "New " "New " "Nort" "Nort" "Ohio" "Okla" "Oreg
" "Penn" "Rhod"
[40] "Sout" "Sout" "Tenn" "Texa" "Utah" "Verm" "Virg" "Wash" "West" "Wisc" "Wyo
m"

 # there are four states with the same abbreviation
#"New " (New Hampshire, New Jersey, New Mexico, New York)
 # The soultion is to use abbreviate() function
 # abbreviate state names
> states2 = abbreviate(states)

 # remove vector names (for convenience)
> names(states2) = NULL
> states2
 [1] "Albm" "Alsk" "Arzn" "Arkn" "Clfr" "Clrd" "Cnnc" "Dlwr" "Flrd" "Gerg" "Hawa" "I
dah" "Illn"
[14] "Indn" "Iowa" "Knss" "Kntc" "Losn" "Main" "Mryl" "Mssc" "Mchg" "Mnns" "Ms
ss" "Mssr" "Mntn"
[27] "Nbrs" "Nevd" "NwHm" "NwJr" "NwMx" "NwYr" "NrtC" "NrtD" "Ohio" "Oklh" "
Orgn" "Pnns" "RhdI"
[40] "SthC" "SthD" "Tnns" "Texs" "Utah" "Vrmn" "Vrgn" "Wshn" "WstV" "Wscn" "W
ymn"

**#If we decide to try an abbreviation with five letters we just simply change th
e argument**

#min.ength = 5
 # abbreviate state names with 5 letters
```
> abbreviate(states, minlength = 5)
```

#Getting the longest name
#we need to count the number of letters in each name. The function nchar() comes handy for that purpose.
 # size (in characters) of each name
```
> state_chars = nchar(states)
```

 # longest name
```
> states[which(state_chars == max(state_chars))]
[1] "North Carolina" "South Carolina"
```

#Selecting States
 # we wish to select those states containing the letter "k".
 # get states names with 'k'
```
> grep(pattern = "k", x = states, value = TRUE)
[1] "Alaska"     "Arkansas"    "Kentucky"    "Nebraska"    "New York"     "North
Dakota"
[7] "Oklahoma"     "South Dakota"
```

 # get states names with 'w'
```
> grep(pattern = "w", x = states, value = TRUE)
[1] "Delaware"      "Hawaii"        "Iowa"          "New Hampshire" "New Jersey"
[6] "New Mexico"    "New York"
```

#we only selected those states with lowercase "w". But what about those stat es with uppercase "W"?
 # one option is
 # get states names with 'w' or 'W'
```
> grep(pattern = "[wW]", x = states, value = TRUE)
 [1] "Delaware"      "Hawaii"        "Iowa"          "New Hampshire" "New Jersey"
 [6] "New Mexico"    "New York"      "Washington"    "West Virginia" "Wisconsin"
[11] "Wyoming"
```

 # second option is to use tolower() function
 # get states names with 'w'
```
> grep(pattern = "w", x = tolower(states), value = TRUE)
```

```
 [1] "delaware"      "hawaii"        "iowa"          "new hampshire" "new jersey"
 [6] "new mexico"    "new york"      "washington"    "west virginia" "wisconsin"
[11] "wyoming"
```

 # OR
 # get states names with 'W'

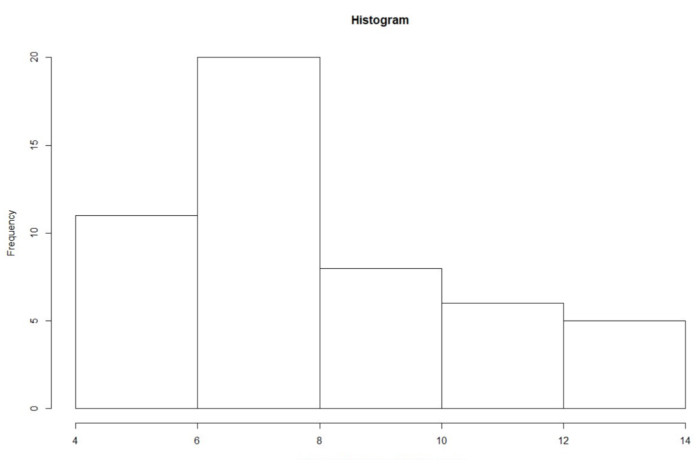
```
> grep(pattern = "W", x = toupper(states), value = TRUE)
```

 # A third solution
 # get states names with 'w'

```
> grep(pattern = "w", x = states, value = TRUE, ignore.case = TRUE)
 [1] "Delaware"      "Hawaii"        "Iowa"          "New Hampshire" "New Jersey"
 [6] "New Mexico"    "New York"      "Washington"    "West Virginia" "Wisconsin"
[11] "Wyoming"
```

#we could ask for the distribution of the State namesâ ™ length. To find the answer we can use nchar()
 # histogram

```
> hist(nchar(states), main = "Histogram", xlab = "number of characters in US State names")
```



#regexpr()
 # regexpr() to get the number of times that a searched pattern is found in a character vector. When there is no match, we get a -1.
#What is the distribution of the vowels in the names of the States?
 # letâ ™s start with the number of aâ ™s in each name.
 # position of a's

```
> positions_a = gregexpr(pattern = "a", text = states, ignore.case = TRUE)
```

 # how many a's?

```
> num_a = sapply(positions_a, function(x) ifelse(x[1] > 0, length(x), 0))
> num_a
 [1] 4 3 2 3 2 1 0 2 1 1 2 1 0 2 1 2 0 2 1 2 2 1 1 0 0 2 2 2 1 0 0 0 2 2 0 2 0 2 1 2 2
0 1 1 0 1
[47] 1 1 0 0
```

#The same operation can be performed by using the function str_count() fro m the package "stringr".

```
> library(stringr)
```

 # total number of a's

```
> str_count(states, "a")
 [1] 3 2 1 2 2 1 0 2 1 1 2 1 0 2 1 2 0 2 1 2 2 1 1 0 0 2 2 2 1 0 0 0 2 2 0 2 0 2 1 2 2
0 1 1 0 1
[47] 1 1 0 0
```

#Notice that we are only getting the number of a's in lower case
#we need to transform all letters to lower case, and then count the number of a's
 # total number of a's

```
> str_count(tolower(states), "a")
 [1] 4 3 2 3 2 1 0 2 1 1 2 1 0 2 1 2 0 2 1 2 2 1 1 0 0 2 2 2 1 0 0 0 2 2 0 2 0 2 1 2 2
0 1 1 0 1
[47] 1 1 0 0
```

 # how to find the count for all the vowels
 # vector of vowels

```
> vowels = c("a", "e", "i", "o", "u")
```

 # vector for storing results

```
> num_vowels = vector(mode = "integer", length = 5)
```

 # calculate number of vowels in each name

```
> for (j in seq_along(vowels)) {
+   num_aux = str_count(tolower(states), vowels[j])
+   num_vowels[j] = sum(num_aux)
```

```
+ }
```

 # add vowel names

```
> names(num_vowels) = vowels
```

 # total number of vowels

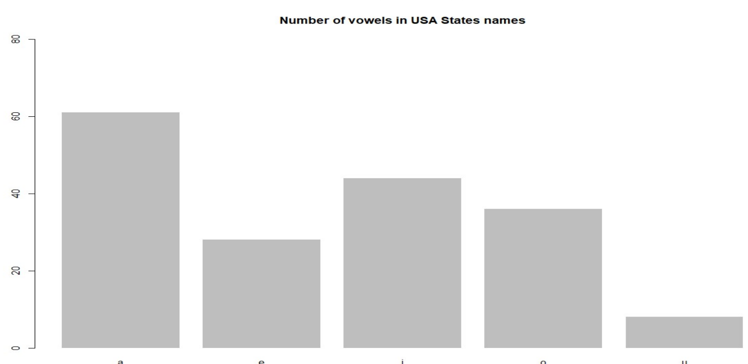
```
> num_vowels
 a  e  i  o  u 
61 28 44 36  8 
```

 # sort them in decreasing order

```
> sort(num_vowels, decreasing = TRUE)
 a  i  o  e  u 
61 44 36 28  8 
```

#And finally, we can visualize the distribution with a barplot:
 # barplot

```
> barplot(num_vowels, main = "Number of vowels in USA States names",
+         border = NA, ylim = c(0, 80))
```



 # Creating Character Strings
#R provides the function character() to create character strings.
 # empty string

```
> empty_str = ""
> empty_str
[1] ""
> class(empty_str)
[1] "character"
> empty_chr = character(0)
> empty_chr
```

```
character(0)
> class(empty_chr)
[1] "character"
```

 # Both emptry string and empty charater vectors are differemt have different length
 # length of empty string

```
> length(empty_str)
[1] 1
> length(empty_chr)
[1] 0
> char_vector = character(5)
> char_vector
[1] "" "" "" "" ""
```

#Once an empty character object has been created, new components may be added to it simply
#by giving it an index value outside its previous range
 # another example

```
> example = character(0)
> example
character(0)
> length(example)
[1] 0
> example[1] = "first"
> example
[1] "first"
> length(example)
[1] 1
> example[4] = "fourth"
> example
[1] "first"  NA       NA       "fourth"
> length(example)
[1] 4
```

#R allows you to convert non-character objects into character strings with
#the function as.character()

```
> b=8+7
> b = as.character(b)
```

```
> b
[1] "15"
> class(b)
[1] "character"

> PI = paste("The life of", pi)
> PI
[1] "The life of 3.14159265358979"
```

#the default separator is a blank space (sep = " ").

```
> PI = paste("The life of", pi, sep="  =  ")
> PI
[1] "The life of  =  3.14159265358979"
```

 # paste a single character "X" with the sequence 1:5, and separator sep = "."
 # paste with objects of different lengths

```
> paste("X", 1:5, sep = ".")
[1] "X.1" "X.2" "X.3" "X.4" "X.5"
```

 # paste with collapsing

```
> paste(1:3, c("!", "?", "+"), sep = "", collapse = "")
[1] "1!2?3+"
```

 # paste without collapsing

```
> paste(1:3, c("!", "?", "+"), sep = "")
[1] "1!" "2?" "3+"
```

#Printing characters
#Printing values with print()
 # text string

```
> my_string = "programming with data is fun"
> print(my_string)
[1] "programming with data is fun"
> print(my_string, quote = FALSE)
[1] programming with data is fun
```

#Unquoted characters with noquote()
 # noquote

```
> noquote(my_string)
```

[1] programming with data is fun

#Concatenate and print with cat()
#the strings are concatenated with a space character as separator
 # concatenate and print

```
> cat(my_string, "with R")
programming with data is fun with R
> cat(1:10, sep = "-")
1-2-3-4-5-6-7-8-9-10> # first four months
> cat(month.name[1:4], sep = " ")
January February March April> # first four months
> cat(month.name[1:4], sep = " ")
January February March April>
```

#The argument fill allows us to break long strings # fill = 30

```
> cat("Looooooooooong strings", "can be displayed", "in a nice format",
+     "by using the 'fill' argument", fill = 30)
Looooooooooong strings
can be displayed
in a nice format
by using the 'fill' argument
```

#Encoding strings with format()
 # use of 'nsmall'

```
> format(13.7, nsmall = 3)
[1] "13.700"
```

 # use of 'digits'

```
> format(c(6, 13.1), digits = 2)
[1] " 6" "13"
```

 # justify options

```
> format(c("A", "BB", "CCC"), width = 5, justify = "centre")
[1] "  A  " " BB  " " CCC "
```

#C-style string formatting with sprintf()
#returns a formatted string combining text and variable values.
 # '%f' indicates 'fixed point' decimal notation

```
> sprintf("%f", pi)
```

```
[1] "3.141593"
```

 # print with sign (positive)

```
> sprintf("%+f", pi)
[1] "+3.141593"
```

#Converting objects to strings with toString()
#allows us to convert an R object to a character string
 # combining two objects

```
> toString(c(17.04, 1978))
[1] "17.04, 1978"
```

 # combining several objects

```
> toString(c("Bonjour", 123, TRUE, NA, log(exp(1))))
[1] "Bonjour, 123, TRUE, NA, 1"
```

#Count number of characters with nchar()
 # how many characters?

```
> nchar(c("How", "many", "characters?"))
[1]  3  4 11
```

#Character translation with chartr()
#chartr() takes
#three arguments: an old string, a new string, and a character vector x
 # replace 'a' by 'A'

```
> chartr("a", "A", "This is a boring string")
[1] "This is A boring string"
```

#Replace substrings with substr()
 # extract 'bcd'

```
> substr("abcdef", 2, 4)
[1] "bcd"
```

 # replace 2nd letter with hash symbol

```
> x = c("may", "the", "force", "be", "with", "you")
> substr(x, 2, 2) <- "#"
> x
[1] "m#y"   "t#e"   "f#rce" "b#"    "w#th"  "y#u"
```

 # replace 2nd and 3rd letters with happy face
```
> y = c("may", "the", "force", "be", "with", "you")
> substr(y, 2, 3) <- ":)"
> y
[1] "m:)"   "t:)"   "f:)ce" "b:"    "w:)h"  "y:)"
```

#Set union with union()
 # two character vectors
```
> set1 = c("some", "random", "words", "some")
> set2 = c("some", "many", "none", "few")
```

 # union of set1 and set2
```
> union(set1, set2)
[1] "some"   "random" "words"  "many"   "none"   "few"
```

#Set intersection with intersect()
 # two character vectors
```
> set3 = c("some", "random", "few", "words")
> set4 = c("some", "many", "none", "few")
```

 # intersect of set3 and set4
```
> intersect(set3, set4)
[1] "some" "few"
```

#Set difference with setdiff()
```
# two character vectors
> set5 = c("some", "random", "few", "words")
> set6 = c("some", "many", "none", "few")
> # difference between set5 and set6
> setdiff(set5, set6)
[1] "random" "words"
```

#Sorting with sort()
```
> set11 = c("today", "produced", "example", "beautiful", "a", "nicely")
> sort(set11)
[1] "a"         "beautiful" "example"   "nicely"    "produced"  "today"
> sort(set11, decreasing = TRUE)
[1] "today"     "produced"  "nicely"    "example"   "beautiful" "a"
```

```
#String manipulations with stringr
#1) basic manipulations
#2) regular expression operations

# Concatenating with str_c()//diff than paste add "" b/w strings
# default usage
> str_c("May", "The", "Force", "Be", "With", "You")
[1] "MayTheForceBeWithYou"

#Number of characters with str_length()
# some text (NA included)
> some_text = c("one", "two", "three", NA, "five")

# compare 'str_length' with 'nchar'
> nchar(some_text)
[1]  3  3  5 NA  4
> str_length(some_text)
[1]  3  3  5 NA  4

# Substring with str_sub()
# apply 'str_sub'
> lorem = "Lorem Ipsum"
> str_sub(lorem, start = 1, end = 5)
[1] "Lorem"

> lorem = "Lorem Ipsum"
> str_sub(lorem, 1, 5) <- "Nullam"
> lorem
[1] "Nullam Ipsum"

# replacing with negative positions
> lorem = "Lorem Ipsum"
> str_sub(lorem, -1) <- "Nullam"
> lorem
[1] "Lorem IpsuNullam"
> str_sub(lorem, -2) <- "Nullam"
> lorem
[1] "Lorem IpsuNullNullam"
```

#Duplication with str dup()
#duplicates and concatenates strings within a character vector
 # default usage
#str_dup(string, times)

```
> str_dup("hola", 3)
[1] "holaholahola"
> str_dup("adios", 1:3)
[1] "adios"         "adiosadios"      "adiosadiosadios"
```

 # Padding with str pad()
#str_pad(string, width, side = "left", pad = " ")
 # default usage

```
> str_pad("hola", width = 7)
[1] "   hola"
```

 # pad both sides

```
> str_pad("adios", width = 7, side = "both")
[1] " adios "
```

 # left padding with '#'

```
> str_pad("hashtag", width = 8, pad = "#")
[1] "#hashtag"
```

 # pad both sides with '-'

```
> str_pad("hashtag", width = 9, side = "both", pad = "-")
[1] "-hashtag-"
```

 # Regular Expression
 # string

```
> money = "$money"
```

#replace the dollar sign $ with an empty string ""

```
> sub(pattern = "\\$", replacement = "", x = money)
[1] "money"
> sub("\\+", "", "Peace+Love")
[1] "PeaceLove"
> sub("\\+", "  ", "Peace+Love")
[1] "Peace  Love"
> sub("\\\\", "", "Peace\\Love")
[1] "PeaceLove"
```

#Sequences

 # replace digit with '_'
> sub("\\d", "_", "the dandelion war 2010") **# for one occurence**
[1] "the dandelion war _010"
> gsub("\\d", "_", "the dandelion war 2010") **# for all occurences**
[1] "the dandelion war ____"
> sub("\\D", "_", "the dandelion war 2010")
[1] "_he dandelion war 2010"
> sub("\\s", "_", "the dandelion war 2010")
[1] "the_dandelion war 2010"

#Character Classes
 # some string
> transport = c("car", "bike", "plane", "boat")

 # look for 'e' or 'i'
> grep(pattern = "[ei]", transport, value = TRUE)
[1] "bike" "plane"
> grep(pattern = "[ei]", transport)
[1] 2 3

 # some numeric strings
> numerics = c("123", "17-April", "I-II-III", "R 3.0.1")

 # match strings with 0 or 1
> grep(pattern = "[01]", numerics, value = TRUE)
[1] "123"   "17-April" "R 3.0.1"

 # match any digit
> grep(pattern = "[0-9]", numerics, value = TRUE)
[1] "123"   "17-April" "R 3.0.1"

#Quantifiers
 # people names
> people = c("rori", "emilia", "matteo", "mehmet", "filipe", "anna", "tyler", "rasmus"
, "jacob", "youna", "flora", "adi")

 # match 'm' at most once
> grep(pattern = "m?", people, value = TRUE)

```
 [1] "rori"   "emilia" "matteo" "mehmet" "filipe" "anna"   "tyler"  "rasmus" "jacob"  "
youna"
[11] "flora"  "adi"
```

 # match 'm' exactly once

```
> grep(pattern = "m{1}", people, value = TRUE, perl = FALSE)
[1] "emilia" "matteo" "mehmet" "rasmus"
>
```

 # match 'm' zero or more times, and 't'

```
> grep(pattern = "m*t", people, value = TRUE)
[1] "matteo" "mehmet" "tyler"
```

 # match 't' zero or more times, and 'm'

```
> grep(pattern = "t*m", people, value = TRUE)
[1] "emilia" "matteo" "mehmet" "rasmus"
```

 # match 'm' one or more times

```
> grep(pattern = "m+", people, value = TRUE)
[1] "emilia" "matteo" "mehmet" "rasmus"

> strings <- c("a", "ab", "acb", "accb", "acccb", "accccb")
> grep("ac*b", strings, value = TRUE) # 0 or more time
[1] "ab"     "acb"    "accb"   "acccb"  "accccb"
> grep("ac+b", strings, value = TRUE) # atleast one time
[1] "acb"    "accb"   "acccb"  "accccb"
> grep("ac?b", strings, value = TRUE) # atmost 1 time
[1] "ab"  "acb"
> grep("ac{2}b", strings, value = TRUE) # exactly n times
[1] "accb"
> grep("ac{2,}b", strings, value = TRUE) # atleast n times
[1] "accb"   "acccb"  "accccb"
> grep("ac{2,3}b", strings, value = TRUE)
[1] "accb"  "acccb"
> stringr::str_extract_all(strings, "ac{2,3}b", simplify = TRUE)
    [,1]
[1,] ""
[2,] ""
[3,] ""
[4,] "accb"
```

[5,] "acccb"
[6,] ""

#Position of pattern within the string
#^: matches the start of the string.
#$: matches the end of the string.
#\b: matches the empty string at either edge of a word. Donâ ™t confuse it wi th ^ $ which marks the edge of a string.
#\B: matches the empty string provided it is not at an edge of a word.
#\b is not a recognized escape character, so we need to double slash it \\b.

```
> strings <- c("abcd", "cdab", "cabd", "c abd")
> strings
[1] "abcd"  "cdab"  "cabd"  "c abd"
> grep("ab", strings, value = TRUE)
[1] "abcd"  "cdab"  "cabd"  "c abd"
> grep("^ab", strings, value = TRUE)
[1] "abcd"
> grep("ab$", strings, value = TRUE)
[1] "cdab"
```

#Character classes
#[:digit:] is equal to [0-9]
#[:lower:]: equivalent to [a-z]

```
> strings <- c("^ab", "abbbab", "abcab", "acbd", "acbe", "ab 12")
> strings
[1] "^ab"    "abbbab" "abcab"  "acbd"   "acbe"   "ab 12"
> grep("ab.", strings, value = TRUE) # starting with ab
[1] "abbbab" "abcab"  "ab 12"
> strings <- c("^ab", "ab", "abc", "abd", "abe", "ab 12")
> strings
[1] "^ab"   "ab"    "abc"   "abd"   "abe"   "ab 12"
> grep("ab[c-e]", strings, value = TRUE)
[1] "abc" "abd" "abe"
> grep("\\^ab", strings, value = TRUE)
[1] "^ab"
> grep("abc|abd", strings, value = TRUE)
[1] "abc" "abd"
```

#detect the string "percent%."

```
> dt <- c("percent%","percent")
> grep(pattern = "percent\\%",x = dt,value = T)
[1] "percent%"
```

#detect all strings

```
> dt <- c("may?","money$","and&")
> grep(pattern = "[a-z][\\?-\\$-\\&]",x = dt,value = T)
[1] "may?"   "money$" "and&"
```

 # replace all

```
> gsub(pattern = "[\\?-\\$-\\&]",replacement = "",x = dt)
[1] "may"   "money" "and"
> gsub(pattern = "\\\\",replacement = "-",x = "Barcelona\\Spain")
[1] "Barcelona-Spain"
```

#symbol .* is known as a greedy quantifier.
#try to match the pattern as many times as its repetition are available.
#The symbol .? is known as a non-greedy quantifier
#it will stop at the first match.

```
> number <- "101000000000100"
```

#greedy

```
> regmatches(number, gregexpr(pattern = "1.*1",text = number))
[[1]]
[1] "1010000000001"
```

#non greedy

```
> regmatches(number, gregexpr(pattern = "1.?1",text = number))
[[1]]
[1] "101"
> names <- c("anna","crissy","puerto","cristian","garcia","steven","alex","rudy")
```

#doesn't matter if e is a match

```
> grep(pattern = "e*",x = names,value = T)
[1] "anna"     "crissy"   "puerto"   "cristian" "garcia"   "steven"   "alex"     "rudy"
```

#must match t one or more times

```
> grep(pattern = "t+",x = names,value = T)
```

[1] "puerto"   "cristian" "steven"

#must match n two times
```
> grep(pattern = "n{2}",x = names,value = T)
[1] "anna"
> string <- "I have been to Paris 20 times"
```

#match a digit
```
> gsub(pattern = "\\d+",replacement = "_",x = string)
[1] "I have been to Paris _ times"
> regmatches(string,regexpr(pattern = "\\d+",text = string))
[1] "20"
```

#match a non-digit
```
> gsub(pattern = "\\D+",replacement = "_",x = string)
[1] "_20_"
> sub(pattern = "\\D+",replacement = "_",x = string)
[1] "_20 times"
> regmatches(string,regexpr(pattern = "\\D+",text = string))
[1] "I have been to Paris "
```

#match a space - returns positions
```
> gregexpr(pattern = "\\s+",text = string)
[[1]]
[1]  2  7 12 15 21 24
attr(,"match.length")
[1] 1 1 1 1 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE

> string <- "20 people got killed in the mob attack. 14 got severely injured"
```
#extract numbers
```
> regmatches(x = string,gregexpr("[0-9]+",text = string))
[[1]]
[1] "20" "14"
```

```
> string <- c("I sleep 16 hours\n, a day","I sleep 8 hours\n a day.","You sleep ho
w many\t hours ?")
```
#get digits
```
> unlist(regmatches(string,gregexpr("[[:digit:]]+",text = string)))
[1] "16" "8"
```

#remove punctuations
```
> gsub(pattern = "[[:punct:]]+",replacement = "",x = string)
[1] "I sleep 16 hours\n a day"   "I sleep 8 hours\n a day"    "You sleep how many
\t hours "
```

#remove spaces
```
> gsub(pattern = "[[:blank:]]",replacement = "-",x = string)
[1] "I-sleep-16-hours\n,-a-day"   "I-sleep-8-hours\n-a-day."    "You-sleep-how-ma
ny--hours-?"
```

EXPERIMENT – 6 (DATE TIME)

```
> Sys.time()
[1] "2020-04-25 22:56:35 IST"
> Sys.Date()
[1] "2020-04-25"
> Sys.timezone()
[1] "Asia/Calcutta"
> Sys.getlocale()
[1] "LC_COLLATE=English_United States.1252;LC_CTYPE=English_United Stat
es.1252;LC_MONETARY=English_United States.1252;LC_NUMERIC=C;LC_TIM
E=English_United States.1252"
> Sys.getlocale("LC_TIME")
[1] "English_United States.1252"
```

 # as.Date() handles dates (without time)
#to convert strings to dates

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

 # number of days between 6/22/07 and 2/13/04

```
> days <- mydates[1] - mydates[2]
```

 # print today's date

```
> today <- Sys.Date()
> format(today, format="%B %d %Y")
[1] "April 25 2020"
> "June 20 2007"
[1] "June 20 2007"
> x<-as.Date('1915-6-16')
> class(x)
[1] "Date"
> y<-Sys.time()
> class(y)
[1] "POSIXct" "POSIXt"
> p <- as.POSIXlt(y)
```

#The POSIXlt object contains some useful metadata.

```
> names(unclass(p))
```

```
 [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"   "isdst"  "zon
e"
[11] "gmtoff"
> #day of week
> p$wday
[1] 6
```

#if used with POSIXct()

```
> y<-Sys.time()
> unclass(y)
[1] 1587835596
> y$sec # error as it is not possible with POSIXct
Error in y$sec : $ operator is invalid for atomic vectors
```

#correct it with

```
> p <- as.POSIXlt(y)
> p$sec
[1] 35.52451
```

#format()
#The default format is yyyy-mm-dd
 # %d-day a number
#%a %A-abbreviated/unabbreviated weekday
#%m-month (0-12)
#%b %B-abbreviated/unabbreviated month name
#%y %Y- abbreviated/unabbreviated year (2 digit/4 digit)
 # print today's date

```
> today <- Sys.Date()
> format(today, format="%B %d %Y")
[1] "April 25 2020"
> weekdays(Sys.Date())
[1] "Saturday"
> months(Sys.Date())
[1] "April"
> quarters(Sys.Date())
[1] "Q2"
```

#strptime() to convert date/time into POSIXlt

```
> datestring <- c("January 10, 2012 10:40", "December 9, 2011 9:10")
```

```
> datestring
[1] "January 10, 2012 10:40" "December 9, 2011 9:10"
> x <- strptime(datestring, "%B %d, %Y %H:%M")
> x
[1] "2012-01-10 10:40:00 IST" "2011-12-09 09:10:00 IST"
> class(x)
[1] "POSIXlt" "POSIXt"
```

#Operations on Dates and Times

```
> x <- as.Date("2012-01-01")
> y <- strptime("9 Jan 2011 11:34:21", "%d %b %Y %H:%M:%S")
> x-y # need to convert x to POSIXlt
Error in x - y : non-numeric argument to binary operator
In addition: Warning message:
Incompatible methods ("-.Date", "-.POSIXt") for "-"
> x <- as.POSIXlt(x)
> x-y
Time difference of 356.747 days

> x <- as.Date("2012-03-01")
> y <- as.Date("2012-02-28")
> x-y
Time difference of 2 days
```

 # measuring time

#POSIXct and POSIXlt classes allow for dates and times
#with control for time zones
#In POSIXct Times are stored internally as the number of seconds since 197 0-01-#01
#In POSIXlt times are stored as list of seconds, minutes, hours etc

```
> start = as.POSIXct(Sys.time())
> timeDate::Nairobi()
+ for (i in 1:1000) {
+ print(i)
+ }
> stop = as.POSIXct(Sys.time())
> timetaken = stop - start
> timetaken
```

```
Time difference of 0.1289949 secs
```

 # sleeping of time

```
> Sys.time()
[1] "2020-04-25 23:07:53 IST"
> Sys.sleep(10)
> Sys.time()
[1] "2020-04-25 23:08:03 IST"
> testit <- function(x)
+ {
+     p1 <- as.POSIXct(Sys.time())
+     Sys.sleep(x)
+     Sys.time() - p1 # The cpu usage should be negligible
+ }
> testit(3.7)
Time difference of 3.704667 secs
```

 # converting time to numeric
 # calculating time taken in seconds

```
> start_num = Sys.time()
> Sys.sleep(10)
> stop_num = Sys.time()

> timetaken_num = as.numeric(stop_num) - as.numeric(start_num)
> timetaken_num
[1] 10.04055
```

 # obbtaining time zones and locations

```
> Sys.timezone()
[1] "Asia/Calcutta"

> x <- "2018-01-01 12:00:00"
> as.POSIXct(x) #IST is default Zone
[1] "2018-01-01 12:00:00 IST"
> as.POSIXct(x, tz = "America/Chicago")
[1] "2018-01-01 12:00:00 CST"
```

#Notice the only thing that changed is the timezone;
#the clock time is the same

```
> as.POSIXct(x, tz = "America/Chicago") - as.POSIXct(x)
Time difference of 11.5 hours

> OlsonNames()  #set of all zone

> Sys.timezone()
[1] "Asia/Calcutta"
> as.POSIXct(Sys.time(), tz='Africa/Nairobi')
[1] "2020-04-25 23:08:17 IST"
```

 # FORMATTING DATES
 # weekday abbreviation

```
> format.Date(Sys.Date(), '%a')
[1] "Sat"
```

 # weekday full

```
> format.Date(Sys.Date(), '%A')
[1] "Saturday"
```

 # month abbreviation

```
> format.Date(Sys.Date(), '%b')
[1] "Apr"
```

 # month full

```
> format.Date(Sys.Date(), '%B')
[1] "April"
```

 # month

```
> format.Date(Sys.Date(), '%m')
[1] "04"
```

 # day

```
> format.Date(Sys.Date(), '%d')
[1] "25"
```

 # year with century

```
> format.Date(Sys.Date(), '%Y')
[1] "2020"
```

```
# full date formating and abbreviation
> format(Sys.Date(),'%a %b %Y %m')
[1] "Sat Apr 2020 04"
> format(Sys.Date(),'%A %B %Y')
[1] "Saturday April 2020"

# Parsing strings into Date objects
> as.Date('09/30/2019', format = '%m/%d/%Y')
[1] "2019-09-30"
> as.Date('09-30-2019', format = '%m-%d-%Y')
[1] "2019-09-30"
> as.Date('September 30th, 2019', '%B %dth, %Y')
[1] "2019-09-30"

# coercing into a date
> d = as.Date('2019-09-30')
> d
[1] "2019-09-30"
> class(d)
[1] "Date"

> as.Date('09-30-2019', format = '%m-%d-%Y')
[1] "2019-09-30"

# Date and time arithmetic
# POSIXct and Sys.time assumes time in seconds
> as.POSIXct(Sys.Date()) + 120
[1] "2020-04-25 05:32:00 IST"
> Sys.time()
[1] "2020-04-25 23:08:18 IST"
> Sys.time() + 60
[1] "2020-04-25 23:09:18 IST"
> Sys.time()
[1] "2020-04-25 23:08:18 IST"

# using diff
# system time + 2hrs 30min 10secs
> Sys.time() + as.difftime(2, units = 'hours') +
+   as.difftime(30, units = 'mins') +
```

```
+   as.difftime(10, units = 'secs')
[1] "2020-04-26 01:38:28 IST"
```

 # calculating time differences using difftime
 # diff = time1 - time2
#a simple function to calculate time difference

```
> time_dif = function(time1, time2){
+   return(difftime(time1,time2))
+ }
> time_dif(Sys.time() + 5, Sys.time())
Time difference of 5 secs
```

 # using POSIXct

```
> time_dif = function(time1, time2){
+   time1 = as.POSIXct(time1)
+   time2 = as.POSIXct(time2)
+   return(difftime(time1,time2))
+ }
> time_dif(Sys.time() + 5, Sys.time())
Time difference of 5 secs
```

EXPERIMENT – 6 (DATE TIME USING LUBRIDATE)

```
> library(lubridate)
> ymd("20110604")
[1] "2011-06-04"
> mdy("06-04-2011")
[1] "2011-06-04"
> dmy("04/06/2011")
[1] "2011-06-04"
```

#basic date-time manipulation

```
> date <- now()
> year(date)
[1] 2020
> minute(date)
[1] 15
> month(date, label = TRUE)
[1] Apr
Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < Oct < Nov < Dec
> wday(date, label = TRUE, abbr = FALSE)
[1] Saturday
Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < Friday < Satur
day
```

 # change date to assigned value

```
> day(date) <- 5
> date
[1] "2020-04-05 23:15:11 IST"
```

#update function to modify multiple attributes same time

```
> date
[1] "2020-04-05 23:15:11 IST"
> update(date, year = 2010, month = 1, day = 1)
[1] "2010-01-01 23:15:11 IST"
```

 # instants, intervals, durations, and periods (helper classes)

 # instant

```
> start_2012 <- ymd_hms("2012-01-01 12:00:00")
```

```
> is.instant(start_2012)
[1] TRUE
> is.instant(today) #current day
[1] FALSE
```

#intervals
#An interval is a span of time that occurs between two specific instants

```
> start_2011 <- ymd_hms("2011-01-01 12:00:00")
> start_2010 <- ymd_hms("2010-01-01 12:00:00")
> span <- start_2011 - start_2010
```

#access the start and end dates of an interval object
#with int_start() and int_end()

```
> int_start(span)
Error in int_start(span) :
  trying to get slot "start" from an object (class "difftime") that is not an S4 obje
ct
> int_end(span)
Error in int_end(span) :
  trying to get slot "start" from an object (class "difftime") that is not an S4 obje
ct
```

 # Duration
#Durations measure the exact amount of time that occurs between two insta nts

```
> duration(60)
[1] "60s (~1 minutes)"
> 1:3 * dhours(1)
[1] "3600s (~1 hours)"  "7200s (~2 hours)"  "10800s (~3 hours)"
```

#Durations can be added and subtracted to any instant object

```
> start_2011 + dyears(1)
[1] "2012-01-01 12:00:00 UTC"
```

#Periods
#Periods measure the change in clock time that occurs between two instants
#Periods record a time span in units larger than seconds

```
> months(3)
[1] "3m 0d 0H 0M 0S"
```

```
> months(3) + days(2)
[1] "3m 2d 0H 0M 0S"
```

#Division with timespans
#"how many weeks are there between Halloween and Christmas?"

```
> halloween <- ymd("2010-10-31")
> christmas <- ymd("2010-12-25")
> interval <- new_interval(halloween, christmas)
> interval / dweeks(1)
[1] 7.857143
```

#TimeZones

```
> date
[1] "2020-04-05 23:15:11 IST"
> with_tz(date, "UTC")
[1] "2020-04-05 17:45:11 UTC"
> force_tz(date, "UTC")
[1] "2020-04-05 23:15:11 UTC"
```

#Parsing

```
> arrive <- ymd_hms("2011-06-04 12:00:00", tz = "Pacific/Auckland")
> arrive
[1] "2011-06-04 12:00:00 NZST"
> leave <- ymd_hms("2011-08-10 14:00:00", tz = "Pacific/Auckland")
> leave
[1] "2011-08-10 14:00:00 NZST"
```

#Setting and Extracting Info

```
> second(arrive)
[1] 0
> second(arrive) <- 25
> arrive
[1] "2011-06-04 12:00:25 NZST"
> second(arrive) <- 0
> wday(arrive)
[1] 7
> wday(arrive, label = TRUE)
[1] Sat
Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

#Time Zones

```
> meeting <- ymd_hms("2011-07-01 09:00:00", tz = "Pacific/Auckland")
> with_tz(meeting, "America/Chicago")
[1] "2011-06-30 16:00:00 CDT"
> mistake <- force_tz(meeting, "America/Chicago")
> with_tz(mistake, "Pacific/Auckland")
[1] "2011-07-02 02:00:00 NZST"
```

#Time Intervals

```
> auckland <- interval(arrive, leave)
> auckland
[1] 2011-06-04 12:00:00 NZST--2011-08-10 14:00:00 NZST
> auckland <- arrive %--% leave
> auckland
[1] 2011-06-04 12:00:00 NZST--2011-08-10 14:00:00 NZST
> jsm <- interval(ymd(20110720, tz = "Pacific/Auckland"), ymd(20110831, tz =
"Pacific/Auckland"))
> jsm
[1] 2011-07-20 NZST--2011-08-31 NZST
> int_overlaps(jsm, auckland)
[1] TRUE
> setdiff(auckland, jsm)
[1] 2011-06-04 12:00:00 NZST--2011-07-20 NZST
```

> #Arithmetic with Date time

```
> minutes(2) ## period
[1] "2M 0S"
> dminutes(2) ## duration
[1] "120s (~2 minutes)"
> leap_year(2011) ## regular year
[1] FALSE
> ymd(20110101) + dyears(1)
[1] "2012-01-01"
> ymd(20110101) + years(1)
[1] "2012-01-01"
> leap_year(2012) ## leap year
[1] TRUE
> ymd(20120101) + dyears(1)
```

```
[1] "2012-12-31"
> ymd(20120101) + years(1)
[1] "2013-01-01"
```

#person YY set up a reoccuring weekly skype meeting with person XXX

```
> meetings <- meeting + weeks(0:5) #person YY meeting timing
#Which of these meetings would be affecting XXX
> jsm <- interval(ymd(20110720, tz = "Pacific/Auckland"), ymd(20110831, tz =
"Pacific/Auckland"))
> meetings %within% jsm
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE

> auckland / ddays(1)
[1] 67.08333
> auckland / ddays(2)
[1] 33.54167
> auckland / dminutes(1)
[1] 96600
> auckland %/% months(1)
Note: method with signature 'Timespan#Timespan' chosen for function '%/%',
target signature 'Interval#Period'.
"Interval#ANY", "ANY#Period" would also be valid
[1] 2
> auckland %% months(1)
[1] 2011-08-04 12:00:00 NZST--2011-08-10 14:00:00 NZST
> as.period(auckland %% months(1))
[1] "6d 2H 0M 0S"
> as.period(auckland)
[1] "2m 6d 2H 0M 0S"

> jan31 <- ymd("2013-01-31")
> jan31 + months(0:11)
 [1] "2013-01-31" NA           "2013-03-31" NA           "2013-05-31" NA           "2013-
07-31"
 [8] "2013-08-31" NA           "2013-10-31" NA           "2013-12-31"

> floor_date(jan31, "month") + months(0:11) + days(31)
 [1] "2013-02-01" "2013-03-04" "2013-04-01" "2013-05-02" "2013-06-01" "2013-0
7-02" "2013-08-01"
```

```
 [8] "2013-09-01" "2013-10-02" "2013-11-01" "2013-12-02" "2014-01-01"

> jan31 %m+% months(0:11)
 [1] "2013-01-31" "2013-02-28" "2013-03-31" "2013-04-30" "2013-05-31" "2013-0
6-30" "2013-07-31"
 [8] "2013-08-31" "2013-09-30" "2013-10-31" "2013-11-30" "2013-12-31"
```

EXPERIMENT – 7 (TIDYR)

 # tidyr for data manipulation. tidyr is a package by Hadley Wickham that mak es it easy to tidy your data.
 # It is often used in conjunction with dplyr.
 # Data is said to be tidy when each column represents a variable, and each ro w represents an observation.

 # the following four functions from the tidyr package:
 # gather - converts wide data to longer format. It is analogous to the melt fun ction from reshape2.
 # spread - converts long data to wider format. It is analogous to the cast func tion from reshape2.
 # unite - combines two or more columns into a single column.
 # separate - splits one column into two or more columns.

```
> library(tidyr)
> library(dplyr)
```

#The Dataframe

```
> n=10
> wide <- data.frame(
+   ID = c(1:n),
+   Face.1 = c(411,723,325,456,579,612,709,513,527,379),
+   Face.2 = c(123,300,400,500,600,654,789,906,413,567),
+   Face.3 = c(1457,1000,569,896,956,2345,780,599,1023,678)
+ )
> wide
   ID Face.1 Face.2 Face.3
1   1    411    123   1457
2   2    723    300   1000
3   3    325    400    569
4   4    456    500    896
5   5    579    600    956
6   6    612    654   2345
7   7    709    789    780
8   8    513    906    599
9   9    527    413   1023
10 10    379    567    678
```

#Gather()
#transform the data from wide to long
#gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

```
> long <- wide %>% gather(Face, ResponseTime, Face.1:Face.3)
```

#Separate()

```
> long_separate <- long %>% separate(Face, c("Target", "Number"))
> long_separate
   ID Target Number ResponseTime
1   1   Face      1          411
2   2   Face      1          723
3   3   Face      1          325
4   4   Face      1          456
5   5   Face      1          579
6   6   Face      1          612
7   7   Face      1          709
8   8   Face      1          513
9   9   Face      1          527
10 10   Face      1          379
11  1   Face      2          123
12  2   Face      2          300
13  3   Face      2          400
14  4   Face      2          500
15  5   Face      2          600
16  6   Face      2          654
17  7   Face      2          789
18  8   Face      2          906
19  9   Face      2          413
20 10   Face      2          567
21  1   Face      3         1457
22  2   Face      3         1000
23  3   Face      3          569
24  4   Face      3          896
25  5   Face      3          956
26  6   Face      3         2345
27  7   Face      3          780
28  8   Face      3          599
29  9   Face      3         1023
```

```
30 10   Face     3         678
```

#Unite()

```
> long_unite <- long_separate %>% unite(Face, Target, Number, sep = ".")
> long_unite
   ID   Face ResponseTime
1   1 Face.1          411
2   2 Face.1          723
3   3 Face.1          325
4   4 Face.1          456
5   5 Face.1          579
6   6 Face.1          612
7   7 Face.1          709
8   8 Face.1          513
9   9 Face.1          527
10 10 Face.1          379
11  1 Face.2          123
12  2 Face.2          300
13  3 Face.2          400
14  4 Face.2          500
15  5 Face.2          600
16  6 Face.2          654
17  7 Face.2          789
18  8 Face.2          906
19  9 Face.2          413
20 10 Face.2          567
21  1 Face.3         1457
22  2 Face.3         1000
23  3 Face.3          569
24  4 Face.3          896
25  5 Face.3          956
26  6 Face.3         2345
27  7 Face.3          780
28  8 Face.3          599
29  9 Face.3         1023
30 10 Face.3          678
```

#Spread()
#transform the data from long back to wide

```
> back_to_wide <- long_unite %>% spread(Face, ResponseTime)
> back_to_wide
   ID Face.1 Face.2 Face.3
1   1    411    123   1457
2   2    723    300   1000
3   3    325    400    569
4   4    456    500    896
5   5    579    600    956
6   6    612    654   2345
7   7    709    789    780
8   8    513    906    599
9   9    527    413   1023
10 10    379    567    678

> mtcars$car <- rownames(mtcars)
```

 # in tibble (check dplyr)

```
> mtcars_df <- tbl_df(mtcars)
> mtcars_df
# A tibble: 32 x 12
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb car
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4 Mazda RX4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4 Mazda RX4 Wag
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1 Datsun 710
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1 Hornet 4 Drive
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2 Hornet Sportabout
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1 Valiant
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4 Duster 360
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2 Merc 240D
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2 Merc 230
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4 Merc 280
# ... with 22 more rows

> mtcars <- mtcars[, c(12, 1:11)]
> mtcarsNew <- mtcars %>% gather(attribute, value, -car)
> head(mtcarsNew)
                car attribute value
1         Mazda RX4       mpg  21.0
2     Mazda RX4 Wag       mpg  21.0
```

```
3       Datsun 710      mpg  22.8
4    Hornet 4 Drive      mpg  21.4
5 Hornet Sportabout      mpg  18.7
6          Valiant      mpg  18.1
> tail(mtcarsNew)
             car attribute value
347  Porsche 914-2     carb     2
348   Lotus Europa     carb     2
349 Ford Pantera L     carb     4
350   Ferrari Dino     carb     6
351  Maserati Bora     carb     8
352     Volvo 142E     carb     2
```

 # # The great thing about tidyr is that you can gather only certain columns an d leave the others alone.
#If we want to gather all the columns from mpg to gear and leave the carb an d car columns as they are,
#we can do it as follows:

```
> mtcarsNew <- mtcars %>% gather(attribute, value, mpg:gear)
> head(mtcarsNew)
              car carb attribute value
1        Mazda RX4    4      mpg  21.0
2    Mazda RX4 Wag    4      mpg  21.0
3       Datsun 710    1      mpg  22.8
4    Hornet 4 Drive    1      mpg  21.4
5 Hornet Sportabout    2      mpg  18.7
6          Valiant    1      mpg  18.1
```

#spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE)
#We can replicate what cast does as follows:

```
> mtcarsSpread <- mtcarsNew %>% spread(attribute, value)
> head(mtcarsSpread)
               car carb am cyl disp drat gear  hp  mpg  qsec vs    wt
1       AMC Javelin    2  0   8  304 3.15    3 150 15.2 17.30  0 3.435
2 Cadillac Fleetwood    4  0   8  472 2.93    3 205 10.4 17.98  0 5.250
3        Camaro Z28    4  0   8  350 3.73    3 245 13.3 15.41  0 3.840
4  Chrysler Imperial    4  0   8  440 3.23    3 230 14.7 17.42  0 5.345
5        Datsun 710    1  1   4  108 3.85    4  93 22.8 18.61  1 2.320
6   Dodge Challenger    2  0   8  318 2.76    3 150 15.5 16.87  0 3.520
```

#unite
#unite(data, col, ..., sep = "_", remove = TRUE)

```
> set.seed(1)
> date <- as.Date('2016-01-01') + 0:14
> hour <- sample(1:24, 15)
> min <- sample(1:60, 15)
> second <- sample(1:60, 15)
> event <- sample(letters, 15)
> data <- data.frame(date, hour, min, second, event)
> data
         date hour min second event
1  2016-01-01    4  15     35     w
2  2016-01-02    7  21      6     x
3  2016-01-03    1  37     10     f
4  2016-01-04    2  41     42     g
5  2016-01-05   11  25     38     s
6  2016-01-06   14  46     47     j
7  2016-01-07   18  58     20     y
8  2016-01-08   22  54     28     n
9  2016-01-09    5  34     54     b
10 2016-01-10   16  42     44     m
11 2016-01-11   10  56     23     r
12 2016-01-12    6  44     59     t
13 2016-01-13   19  60     40     v
14 2016-01-14   23  33     51     o
15 2016-01-15    9  20     25     a
```

 # Now, let us combine the date, hour, min, and second columns into a new column called datetime.
 # Usually, datetime in R is of the form Year-Month-Day Hour:Min:Second.

```
> dataNew <- data %>%
+   unite(datehour, date, hour, sep = ' ') %>%
+   unite(datetime, datehour, min, second, sep = ':')
> dataNew
             datetime event
1   2016-01-01 4:15:35     w
2    2016-01-02 7:21:6     x
3   2016-01-03 1:37:10     f
```

```
4   2016-01-04 2:41:42    g
5  2016-01-05 11:25:38    s
6  2016-01-06 14:46:47    j
7  2016-01-07 18:58:20    y
8  2016-01-08 22:54:28    n
9   2016-01-09 5:34:54    b
10 2016-01-10 16:42:44    m
11 2016-01-11 10:56:23    r
12  2016-01-12 6:44:59    t
13 2016-01-13 19:60:40    v
14 2016-01-14 23:33:51    o
15  2016-01-15 9:20:25    a
```

#separate
 # separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE,
 #   convert = FALSE, extra = "warn", fill = "warn", ...)

```
> data1 <- dataNew %>%
+   separate(datetime, c('date', 'time'), sep = ' ') %>%
+   separate(time, c('hour', 'min', 'second'), sep = ':')
> data1
         date hour min second event
1  2016-01-01    4  15     35     w
2  2016-01-02    7  21      6     x
3  2016-01-03    1  37     10     f
4  2016-01-04    2  41     42     g
5  2016-01-05   11  25     38     s
6  2016-01-06   14  46     47     j
7  2016-01-07   18  58     20     y
8  2016-01-08   22  54     28     n
9  2016-01-09    5  34     54     b
10 2016-01-10   16  42     44     m
11 2016-01-11   10  56     23     r
12 2016-01-12    6  44     59     t
13 2016-01-13   19  60     40     v
14 2016-01-14   23  33     51     o
15 2016-01-15    9  20     25     a
```

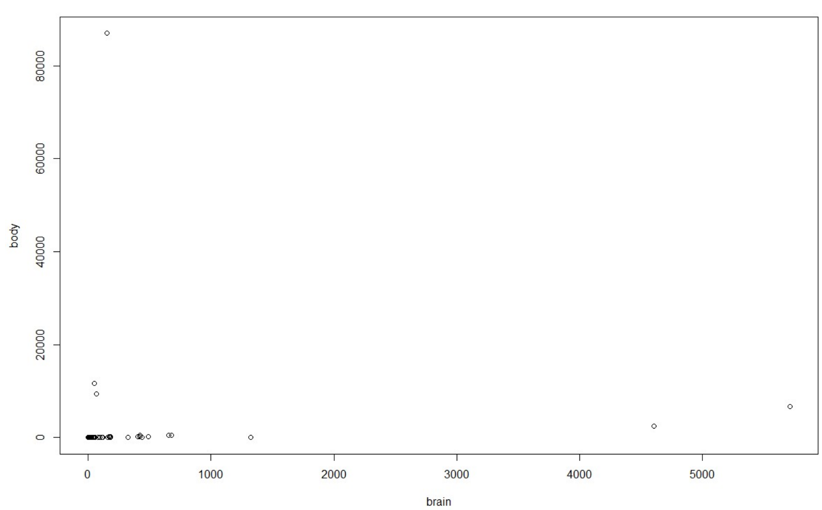
#It first splits the datetime column into date and time, and then splits time int o hour, min, and second.

EXPERIMENT – 7 (REGRESSION)

```
> animal <- read.csv("C:/Users/Tushar/Downloads/Animals2 (1).csv")
> head(animal)
                         X  body brain
1 Lesser short-tailed shrew 0.005  0.14
2         Little brown bat 0.010  0.25
3           Big brown bat 0.023  0.30
4                   Mouse 0.023  0.40
5             Musk shrew 0.048  0.33
6         Star-nosed mole 0.060  1.00
```

First step - inspect the data

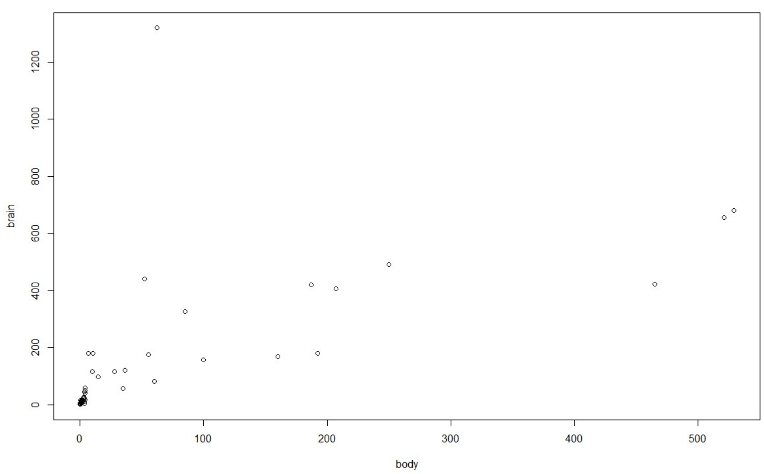
```
> require(dplyr) # the 'select' function is from dplyr
> plot(select(animal,brain,body))
```



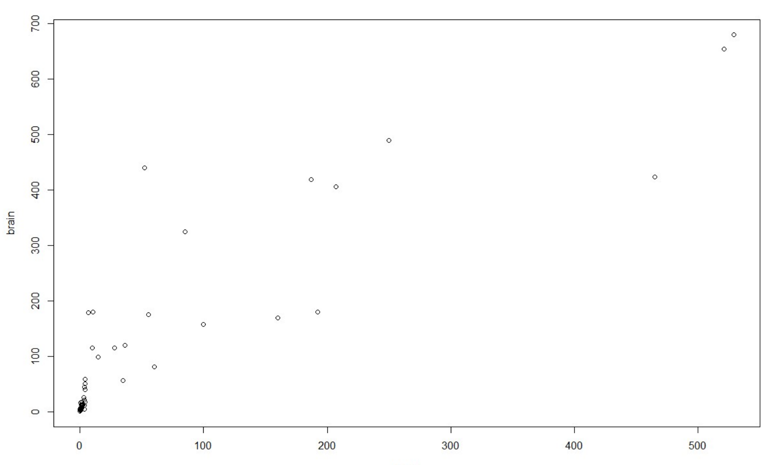
##Remove outliers

```
> filter(animal, brain > 3000)
               X body brain
1   Asian elephant 2547  4603
2 African elephant 6654  5712
> filter(animal, body > 8000)
            X  body brain
```

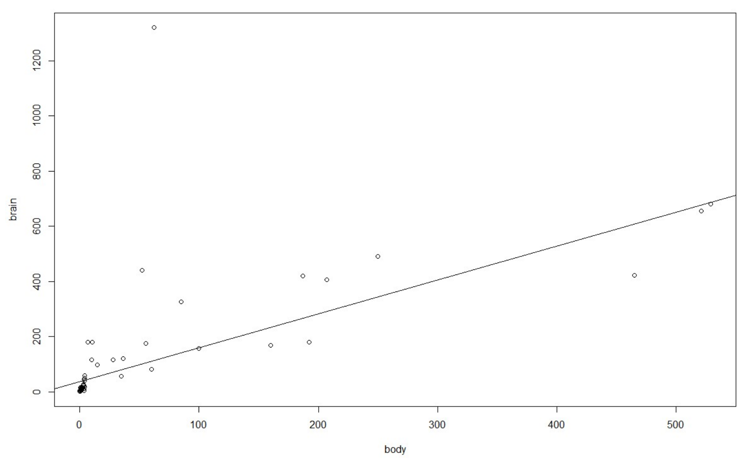
```
1   Triceratops  9400  70.0
2   Dipliodocus 11700  50.0
3 Brachiosaurus 87000 154.5

> animal.1 <- filter(animal, brain <= 3000, body <=8000)
> plot(select(animal.1,body,brain))
```




```
> filter(animal.1, brain >= 1200)
      X body brain
1 Human   62  1320
> animal.2 <- filter(animal.1, X != "Human")
> plot(select(animal.2,body,brain))
```



```
## Modelling - Linear Regression
#- The **lm** function is used for linear regression
#- lm stands for linear models
#- $$ y = a + bx + e $$
#- e is the error
> m <- lm(brain ~ body, data = animal.2)
> abline(lm(brain ~ body, data = animal.2))
```

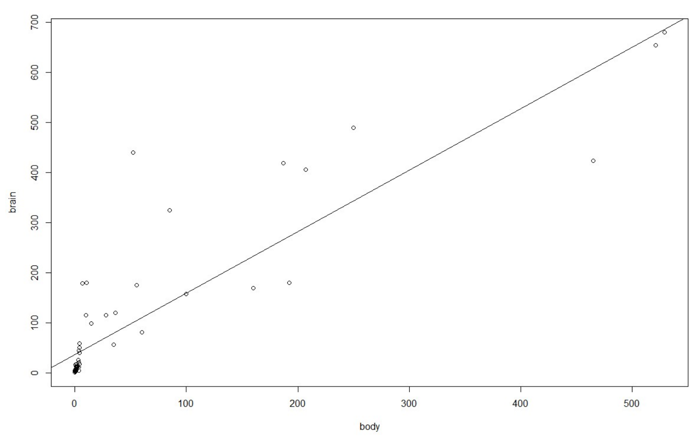



```
> summary(m)
Call:
lm(formula = brain ~ body, data = animal.2)

Residuals:
    Min      1Q  Median      3Q     Max 
-184.81  -34.52  -27.16    0.67  339.35 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 36.57231   10.95089    3.34  0.00148 ** 
body         1.22847    0.08408   14.61  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 77.15 on 57 degrees of freedom
Multiple R-squared:  0.7893,	Adjusted R-squared:  0.7856
```

F-statistic: 213.5 on 1 and 57 DF, p-value: < 2.2e-16

 # What to look for?
#- The coefficients
#- The p-value of coefficients
 # - a test was performed to make sure that the coefficients is statistically si gnificant
 # - the lower the p-value the more statistically signficiant it is
 # - a 5% threshold is usually used
#- R-squared - a measure of the proportion of variation in the data explained by the model

```
> plot(select(animal.2,body, brain))
> abline(m)
```



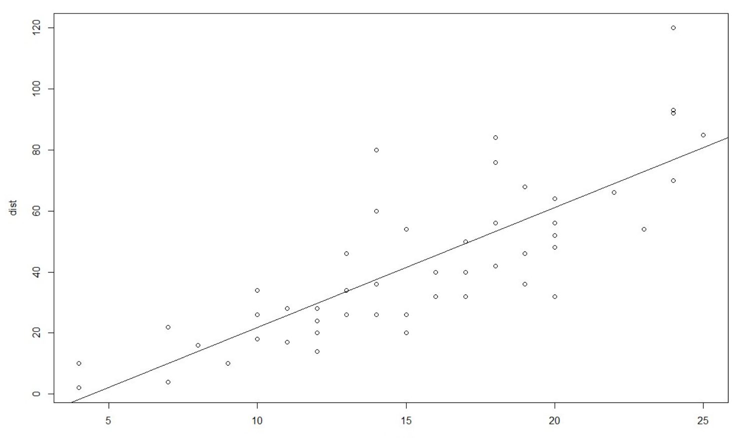
Modelling - Linear Regression {.smaller}

```
> m <- lm(log(brain) ~ log(body), data = animal.2)
> summary(m)
Call:
lm(formula = log(brain) ~ log(body), data = animal.2)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.67407 -0.48992 -0.03502  0.47545  1.66400 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)  2.11392    0.09138   23.13   <2e-16 ***
log(body)    0.73528    0.02993   24.57   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6588 on 57 degrees of freedom
Multiple R-squared:  0.9137,       Adjusted R-squared:  0.9122
F-statistic: 603.4 on 1 and 57 DF,  p-value: < 2.2e-16
```

How to assess linear regression models?
 # - Check if the coefficients are significant (p-val < 0.05)
 # - Check the R-square
 #   - It represent the proportion of variance in the data explained by the model . It is a number from 0 to 1.00. The higher the better
 # - Assess the data visually
 # - Try a few transformations of the raw data and assess

```
> m <- lm(dist ~ speed , data=cars)
> plot(select(cars,speed,dist))
> abline(m)
```



#predict() Function
 # The predictor vector.
```
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
```

 # The resposne vector.

```
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

 # Apply the lm() function.

```
> relation <- lm(y~x)
```

 # Find weight of a person with height 170.(x= height and y=weight)

```
> a <- data.frame(x = 170)
> result <-  predict(relation,a)
> print(result)
      1
76.22869
```

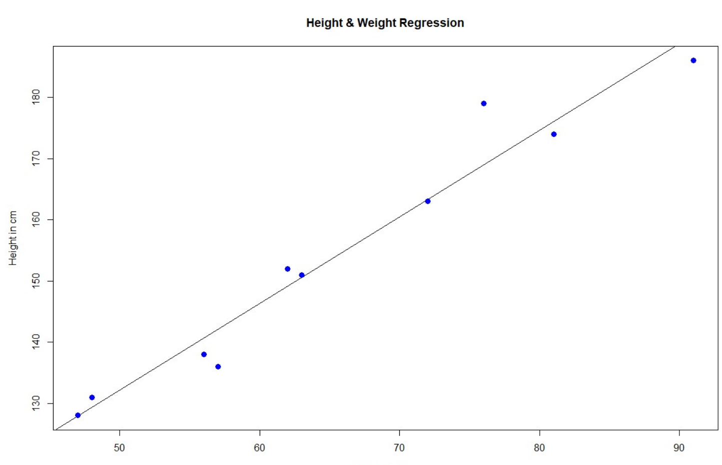
```
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> relation <- lm(y~x)
```

 # Give the chart file a name.

```
> png(file = "linearregression.png")
```

 # Plot the chart.

```
> plot(y,x,col = "blue",main = "Height & Weight Regression",
+     abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in c
m")
```



 # Save the file.

```
> dev.off()
null device
```

```
   1
```

#==== Multiple Regression

```
>
> input <- mtcars[,c("mpg","disp","hp","wt")]
> print(head(input))
                   mpg disp  hp    wt
Mazda RX4         21.0  160 110 2.620
Mazda RX4 Wag     21.0  160 110 2.875
Datsun 710        22.8  108  93 2.320
Hornet 4 Drive    21.4  258 110 3.215
Hornet Sportabout 18.7  360 175 3.440
Valiant           18.1  225 105 3.460
```

#Create Relationship Model & get the Coefficients

```
> input <- mtcars[,c("mpg","disp","hp","wt")]
```

 # Create the relationship model.

```
> model <- lm(mpg~disp+hp+wt, data = input)
```

 # Show the model.

```
> print(model)
Call:
lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:
(Intercept)        disp          hp          wt
  37.105505   -0.000937   -0.031157   -3.800891
```

 # Get the Intercept and coefficients as vector elements.

```
> cat("# # # # The Coefficient Values # # # ","\n")
# # # # The Coefficient Values # # #
> a <- coef(model)[1]
> print(a)
(Intercept)
   37.10551
> Xdisp <- coef(model)[2]
> Xhp <- coef(model)[3]
> Xwt <- coef(model)[4]
> print(Xdisp)
         disp
```

```
-0.0009370091
> print(Xhp)
        hp
-0.03115655
> print(Xwt)
      wt
-3.800891
```

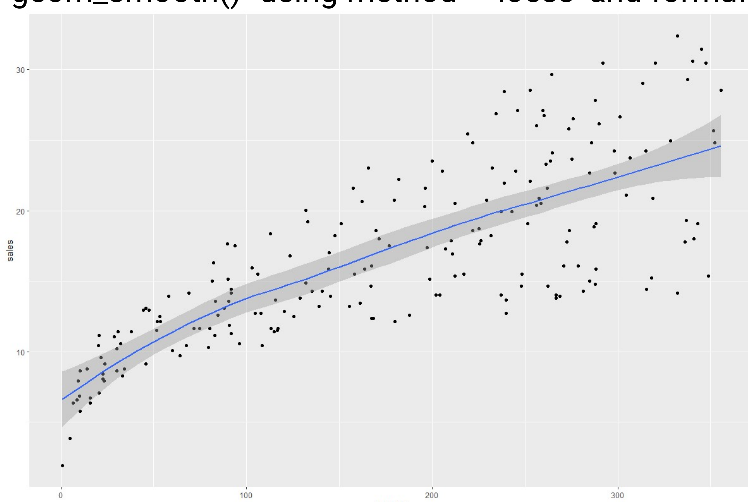
EXPERIMENT – 7 (LINEAR REGRESSION)

```
> data("marketing", package = "datarium")
> head(marketing, 4)
  youtube facebook newspaper sales
1  276.12    45.36     83.04 26.52
2   53.40    47.16     54.12 12.48
3   20.64    55.08     83.16 11.16
4  181.80    49.56     70.20 22.20
```

#It contains the impact of three advertising medias (youtube, facebook and n ewspaper)
#on sales.
#We want to predict future sales on the basis of advertising budget spent on youtube.

#Visualization

```
> ggplot(marketing, aes(x = youtube, y = sales)) + geom_point() + stat_smooth
()
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



#linearly increasing relationship between the sales and the youtube variables

#correlation coefficient (ranges from -1 to +1)(A low correlation (-0.2 < x < 0. 2))

```
> cor(marketing$sales, marketing$youtube)
```

[1] 0.7822244

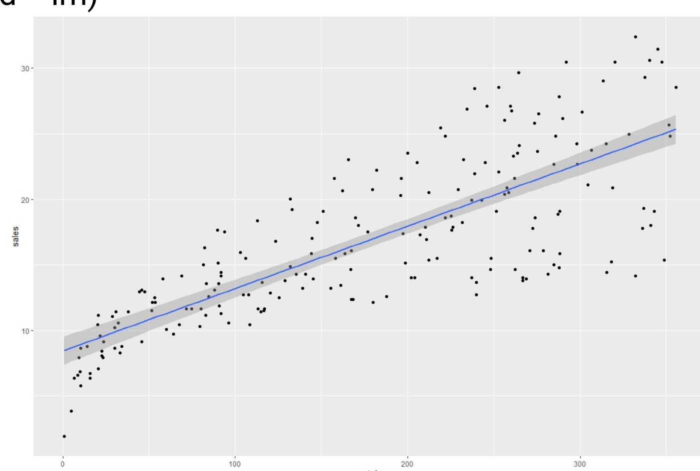
#The linear model equation can be written as follow: sales = b0 + b1 * youtub e

```
> model <- lm(sales ~ youtube, data = marketing)
> model
Call:
lm(formula = sales ~ youtube, data = marketing)

Coefficients:
(Intercept)     youtube
   8.43911      0.04754
```

 # interpretation: sales = 8.44 + 0.048*youtube
#This means that, for a youtube advertising budget equal to 1000 dollars,
#we can expect an increase of 48 units (0.048*1000) in sales.
#As we are operating in units of thousand dollars, this represents a sale of 5 6440 dollars.
#sales = 8.44 + 0.048*1000 = 56.44 units

#regression line
#The confidence bands reflect the uncertainty about the line.

```
> ggplot(marketing, aes(youtube, sales)) + geom_point() + stat_smooth(metho d = lm)
```



#Model summary (The summary outputs shows 6 components)

```
> summary(model)
Call:
```

```
lm(formula = sales ~ youtube, data = marketing)

Residuals:
    Min      1Q  Median      3Q     Max 
-10.0632  -2.3454  -0.2295   2.4805   8.6548 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 8.439112   0.549412   15.36   <2e-16 ***
youtube     0.047537   0.002691   17.67   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.91 on 198 degrees of freedom
Multiple R-squared:  0.6119,	Adjusted R-squared:  0.6099 
F-statistic: 312.1 on 1 and 198 DF,  p-value: < 2.2e-16
```

#RSE = 3.91, meaning that the observed sales values deviate from the true re gression line
#by approximately 3.9 units in average

#Standard errors and confidence intervals:
#Check at least, one predictor variable is significantly associated the outcom e

```
> confint(model)
                 2.5 %     97.5 %
(Intercept) 7.35566312 9.52256140
youtube     0.04223072 0.05284256
```

#Model accuracy
#checking how well the model fits the data
#Residual standard error (RSE):representing the average variation of the obs ervations points around the fitted regression line

```
> sigma(model)*100/mean(marketing$sales)
[1] 23.23877
```

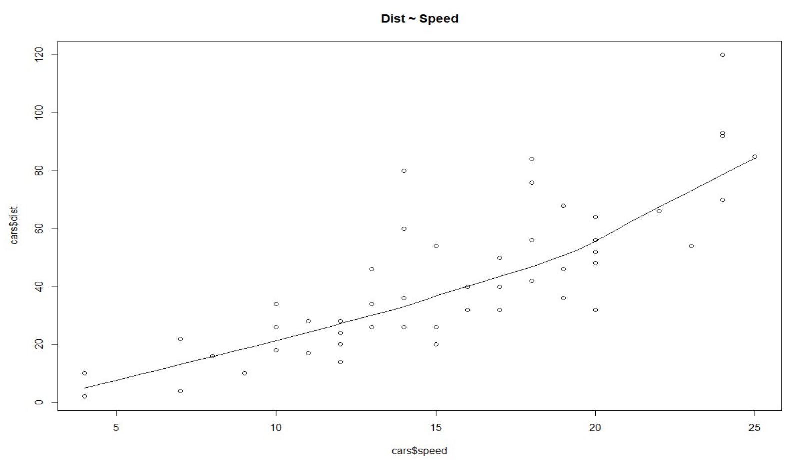
#In data set, the mean value of sales is 16.827,
#and so the percentage error is 3.9/16.827 = 23%.

 # cars dataset (50 obs and 2 variables)

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

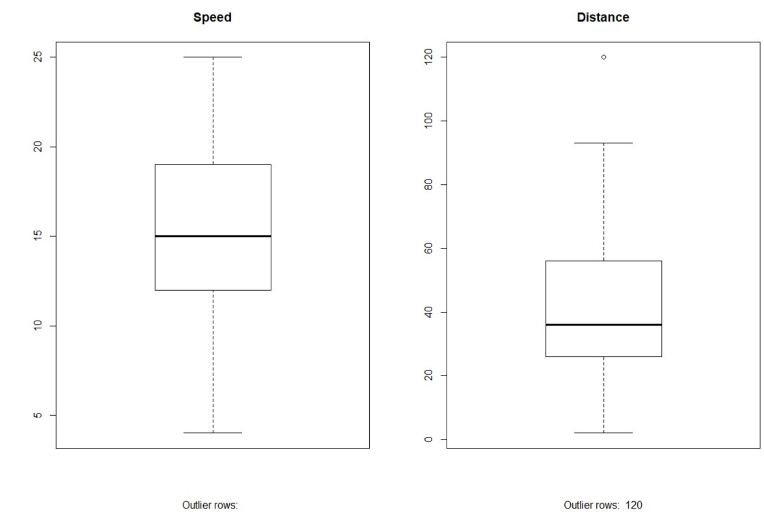
#Scatter Plot To Visualise The Relationship

> scatter.smooth(x=cars$speed, y=cars$dist, main="Dist ~ Speed") **# scatterplot**



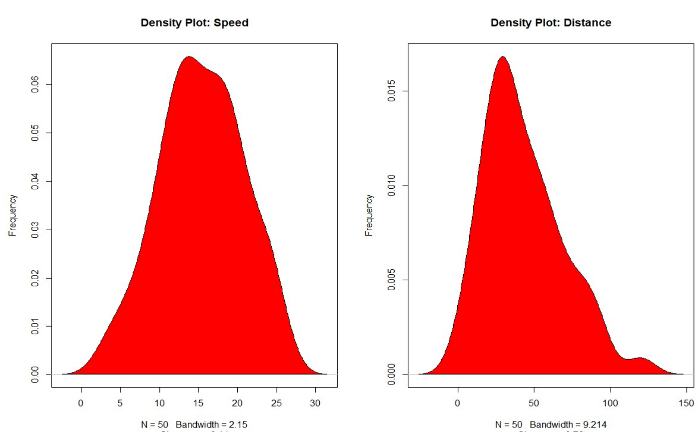
#Using BoxPlot To Check For Outliers
#an outlier is any datapoint that lies outside the 1.5 * inter quartile range (IQR).
#IQR is calculated as the distance between the 25th percentile and 75th percentile values for that variable.

> par(mfrow=c(1, 2)) **# divide graph area in 2 columns**
> boxplot(cars$speed, main="Speed", sub=paste("Outlier rows: ", boxplot.stats(cars$speed)$out)) **# box plot for 'speed'**
> boxplot(cars$dist, main="Distance", sub=paste("Outlier rows: ", boxplot.stats(cars$dist)$out)) **# box plot for 'distance'**



#Using Density Plot To Check If Response Variable Is Close To Normal

```
> library(e1071)  # for skewness function
> par(mfrow=c(1, 2))  # divide graph area in 2 columns

> plot(density(cars$speed), main="Density Plot: Speed", ylab="Frequency", sub
=paste("Skewness:", round(e1071::skewness(cars$speed), 2)))  # density plot
for 'speed'
> polygon(density(cars$speed), col="red")
> plot(density(cars$dist), main="Density Plot: Distance", ylab="Frequency", sub
=paste("Skewness:", round(e1071::skewness(cars$dist), 2)))  # density plot fo
r 'dist'
> polygon(density(cars$dist), col="red")
```



 # build linear regression model on full data
```
> linearMod <- lm(dist ~ speed, data=cars)
> print(linearMod)
Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)       speed
   -17.579       3.932

> summary(linearMod)
Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,      Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

 # capture model summary as an object
```
> modelSummary <- summary(linearMod)
```

 # model coefficients
```
> modelCoeffs <- modelSummary$coefficients
> modelCoeffs
              Estimate Std. Error   t value     Pr(>|t|)
(Intercept) -17.579095  6.7584402 -2.601058 1.231882e-02
speed         3.932409  0.4155128  9.463990 1.489836e-12
```

```
# get beta estimate for speed
> beta.estimate <- modelCoeffs["speed", "Estimate"]
> beta.estimate
[1] 3.932409

# get std.error for speed
> std.error <- modelCoeffs["speed", "Std. Error"]
> std.error
[1] 0.4155128

# calc t statistic
> t_value <- beta.estimate/std.error
> t_value
[1] 9.46399

# calc p Value
> p_value <- 2*pt(-abs(t_value), df=nrow(cars)-ncol(cars))
> p_value
[1] 1.489836e-12

# fstatistic
> f_statistic <- linearMod$fstatistic[1]
> f_statistic
NULL

# parameters for model p-value calc
> f <- summary(linearMod)$fstatistic
> model_p <- pf(f[1], f[2], f[3], lower=FALSE)

#Step 1: Create the training and test data
# Create Training and Test data -
> set.seed(100)  # setting seed to reproduce results of random sampling
> trainingRowIndex <- sample(1:nrow(cars), 0.8*nrow(cars))  # row indices for training data
> trainingData <- cars[trainingRowIndex, ]  # model training data
> testData  <- cars[-trainingRowIndex, ]   # test data

#Step 2: Fit the model on training data and predict dist on test data
# Build the model on training data
```

```
> lmMod <- lm(dist ~ speed, data=trainingData)  # build the model
> distPred <- predict(lmMod, testData)  # predict distance
```

#Step 3: Review diagnostic measures.

```
> summary (lmMod)
Call:
lm(formula = dist ~ speed, data = trainingData)

Residuals:
    Min      1Q  Median      3Q     Max
-24.726 -11.242  -2.564  10.436  40.565

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -20.1796     7.8254  -2.579   0.0139 *
speed         4.2582     0.4947   8.608 1.85e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.49 on 38 degrees of freedom
Multiple R-squared:  0.661,	Adjusted R-squared:  0.6521
F-statistic: 74.11 on 1 and 38 DF,  p-value: 1.848e-10
```

#Step 4: Calculate prediction accuracy and error rates

```
> actuals_preds <- data.frame(cbind(actuals=testData$dist, predicteds=distPred))  # make actuals_predicteds dataframe.
> correlation_accuracy <- cor(actuals_preds)  # 82.7%
> head(actuals_preds)
   actuals predicteds
3        4   9.627845
5       16  13.886057
17      34  35.177120
24      20  43.693545
28      40  47.951757
32      42  56.468182
```

 # Min-Max Accuracy Calculation

```
> min_max_accuracy <- mean(apply(actuals_preds, 1, min) / apply(actuals_preds, 1, max))
```

 # => 38.00%, min_max accuracy

MAPE Calculation

> mape <- mean(abs((actuals_preds$predicteds - actuals_preds$actuals))/actuals_preds$actuals)

 # => 69.95%, mean absolute percentage deviation

EXPERIMENT – 7 (MULTIPLE LINEAR REGRESSION)

```
> library(tidyverse)
> data("marketing", package = "datarium")
> head(marketing, 4)
  youtube facebook newspaper sales
1  276.12    45.36     83.04 26.52
2   53.40    47.16     54.12 12.48
3   20.64    55.08     83.16 11.16
4  181.80    49.56     70.20 22.20
```

#Building model
 # We want to build a model for estimating sales based on the advertising bud get invested in youtube, facebook and newspaper, as follow:

```
> model <- lm(sales ~ youtube + facebook + newspaper, data = marketing)
> summary(model)
Call:
lm(formula = sales ~ youtube + facebook + newspaper, data = marketing)

Residuals:
     Min       1Q   Median       3Q      Max
-10.5932  -1.0690   0.2902   1.4272   3.3951

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.526667   0.374290   9.422   <2e-16 ***
youtube      0.045765   0.001395  32.809   <2e-16 ***
facebook     0.188530   0.008611  21.893   <2e-16 ***
newspaper   -0.001037   0.005871  -0.177     0.86
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.023 on 196 degrees of freedom
Multiple R-squared:  0.8972,	Adjusted R-squared:  0.8956
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16
```

 # Interpretation
 # The first step in interpreting the multiple regression analysis is

 # to examine the F-statistic and the associated p-value, at the bottom of model summary.
 # In our example, it can be seen that p-value of the F-statistic is < 2.2e-16, which is highly significant.
 # This means that, at least, one of the predictor variables is significantly related to the outcome variable.
 # To see which predictor variables are significant, you can examine the coefficients table, which shows the estimate of regression beta coefficients and the associated t-statitic p-values:

```
> summary(model)$coefficient
              Estimate  Std. Error    t value     Pr(>|t|)
(Intercept)  3.526667243 0.374289884  9.4222884 1.267295e-17
youtube      0.045764645 0.001394897 32.8086244 1.509960e-81
facebook     0.188530017 0.008611234 21.8934961 1.505339e-54
newspaper   -0.001037493 0.005871010 -0.1767146 8.599151e-01
```

 # For a given the predictor, the t-statistic evaluates whether or not there is significant association between the predictor and the outcome variable, that is whether the beta coefficient of the predictor is significantly different from zero.

 # It can be seen that, changing in youtube and facebook advertising budget are significantly associated to changes in sales while changes in newspaper budget is not significantly associated with sales.

 # For a given predictor variable, the coefficient (b) can be interpreted as the average effect on y of a one unit increase in predictor, holding all other predictors fixed.

 # For example, for a fixed amount of youtube and newspaper advertising budget, spending an additional 1 000 dollars on facebook advertising leads to an increase in sales by approximately 0.1885*1000 = 189 sale units, on average.

 # The youtube coefficient suggests that for every 1 000 dollars increase in youtube advertising budget, holding all other predictors constant, we can expect an increase of 0.045*1000 = 45 sales units, on average.

 # We found that newspaper is not significant in the multiple regression mode l. This means that, for a fixed amount of youtube and newspaper advertising budget, changes in the newspaper advertising budget will not significantly af fect sales units.

 # As the newspaper variable is not significant, it is possible to remove it from the model:

 # spending one unit (1000 $)in facebook advertisingleads to total sale increa se
#to 0.1885*1000 = 189 sale units, on average.(without varying youtube and n ewspaper budget)

```
> model  <- lm(sales ~ youtube + facebook, data = marketing)
> summary(model)
Call:
lm(formula = sales ~ youtube + facebook, data = marketing)

Residuals:
    Min      1Q   Median      3Q     Max
-10.5572  -1.0502   0.2906   1.4049   3.3994

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.50532    0.35339   9.919   <2e-16 ***
youtube      0.04575    0.00139  32.909   <2e-16 ***
facebook     0.18799    0.00804  23.382   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.018 on 197 degrees of freedom
Multiple R-squared:  0.8972,	Adjusted R-squared:  0.8962
F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16
```

 # Finally, our model equation can be written as follow:
#sales = 3.5 + 0.045*youtube + 0.187*facebook.
#the adjusted R2 = 0.89, meaning that "89% of the variance in the measure of sales
#can be predicted by youtube and facebook advertising budgets.

 # The confidence interval of the model coefficient can be extracted as follow :

```
> confint(model)
                 2.5 %     97.5 %
(Intercept) 2.80841159 4.20222820
youtube     0.04301292 0.04849671
facebook    0.17213877 0.20384969
```

#Residual Standard Error (RSE), or sigma:The RSE estimate gives a measure of error of prediction.
#The lower the RSE, the more accurate the model.

```
> sigma(model)/mean(marketing$sales)
[1] 0.1199045
```

#the RSE is 2.023 corresponding to 12% error rate.

```
> library(dplyr)
> df <- mtcars %>% select(-c(am, vs, cyl, gear, carb))
> glimpse(df)
Observations: 32
Variables: 6
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8, 16.4, 17.3, ...
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 167.6, 167.6, 2...
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180, 205, 215, 2...
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92, 3.07, 3.07, ...
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.440, 3.440, 4...
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18.30, 18.90, 1...
```

#Your objective is to estimate the mile per gallon based on a set of variables.
#The equation to estimate is:
#mpg=b0+b1*disp+b2*hp+b3*drat+b4*wt+e

```
> model <- mpg~disp + hp + drat + wt
> fit <- lm(model, df)
```

```
> fit
Call:
lm(formula = model, data = df)

Coefficients:
(Intercept)         disp           hp         drat           wt
  29.148738     0.003815    -0.034784     1.768049    -3.479668
```

#mpg=29.14+0.003*disp+(-0.034)*hp+1.76*drat+(-3.47)*wt

 # model <- mpg ~disp + hp + drat+ wt: Store the model to estimate
 # lm(model, df): Estimate the model with the data frame df
 # The output does not provide enough information about the quality of the fit . You can access more details such as the significance of the coefficients,
 # the degree of freedom and the shape of the residuals with the summary() f unction.

```
> summary(fit)
Call:
lm(formula = model, data = df)

Residuals:
    Min      1Q  Median      3Q     Max
-3.5077 -1.9052 -0.5057  0.9821  5.6883

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 29.148738   6.293588   4.631  8.2e-05 ***
disp         0.003815   0.010805   0.353  0.72675
hp          -0.034784   0.011597  -2.999  0.00576 **
drat         1.768049   1.319779   1.340  0.19153
wt          -3.479668   1.078371  -3.227  0.00327 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.602 on 27 degrees of freedom
Multiple R-squared:  0.8376,	Adjusted R-squared:  0.8136
F-statistic: 34.82 on 4 and 27 DF,  p-value: 2.704e-10
```

 # The above table proves that there is a strong negative relationship between wt and mileage and positive relationship with drat.

 # Only the variable wt has a statistical impact on mpg. Remember, to test a hypothesis in statistic, we use:
 #   H0: No statistical impact
 # H3: The predictor has a meaningful impact on y
 # If the p value is lower than 0.05, it indicates the variable is statistically significant
 # Adjusted R-squared: Variance explained by the model.
 # the model explained 82 percent of the variance of y. R squared is always between 0 and 1. The higher the better

#You can run the ANOVA test to estimate the effect of each feature on the variances with the anova() function.

```
> anova(fit)
Analysis of Variance Table

Response: mpg
        Df Sum Sq Mean Sq  F value    Pr(>F)
disp      1 808.89  808.89 119.4502 2.041e-11 ***
hp        1  33.67   33.67   4.9714  0.034281 *
drat      1  30.15   30.15   4.4519  0.044270 *
wt        1  70.51   70.51  10.4121  0.003272 **
Residuals 27 182.84    6.77
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

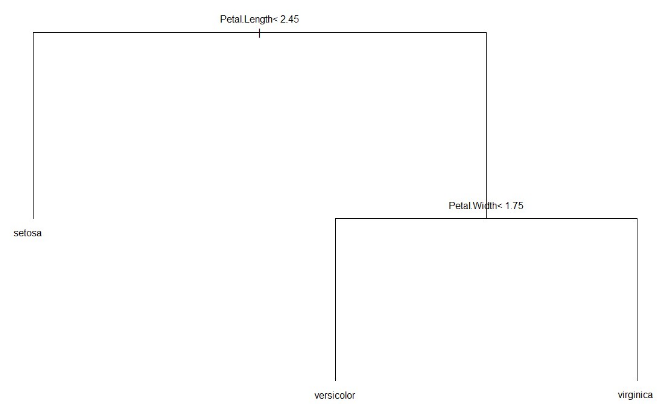
EXPERIMENT – 8 (DECISION TREES)

#ID3

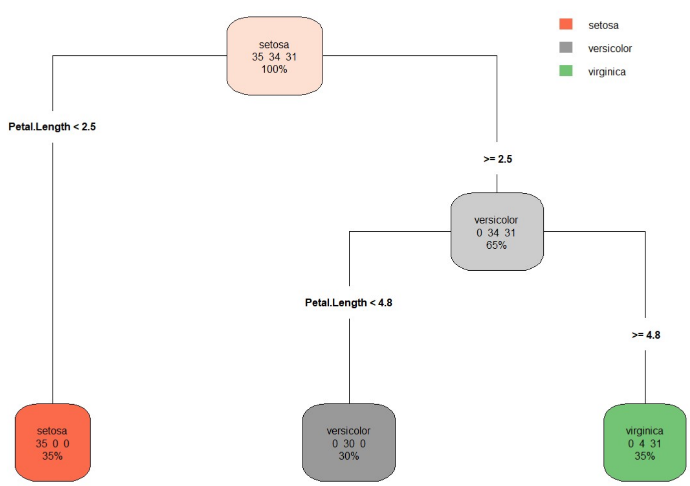
```
> str(iris)
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> dim(iris)
[1] 150   5

> s<-sample(150,100)
> s
  [1]  33 131 106  98  60  97  68 102 126  66  14 134   6  87  38  86 110 120   5
69 150  42
 [23]  93 136  57  46  58  48  39 139  11 109  50 132 138  83 133   3  20 114 12
4  76   9  54
 [45]  41  65  77   4  78 117 103 107  44  91  74   8  61 111  75  23  63  34  53  2
6 104  12
 [67]  81  79 116  19  73  28 146  32  56 141 115 101  45 147  15 142  64  85  2
4  30  59 140
 [89]  89  90  37 105  70  94  49 148  51  21  99  80

> iris_train<- iris[s,]
> iris_test<- iris[-s,]
> dim(iris_train)
[1] 100   5
> dtm<-rpart(Species~., iris_train, method="class")
> dtm
n= 100 node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 100 63 versicolor (0.3200000 0.3700000 0.3100000)
  2) Petal.Length< 2.45 32  0 setosa (1.0000000 0.0000000 0.0000000) *
  3) Petal.Length>=2.45 68 31 versicolor (0.0000000 0.5441176 0.4558824)
```

```
   6) Petal.Width< 1.75 40  3 versicolor (0.0000000 0.9250000 0.0750000) *
   7) Petal.Width>=1.75 28  0 virginica (0.0000000 0.0000000 1.0000000) *
```

```
> par(xpd =NA)
> plot(dtm)
> text(dtm)
```



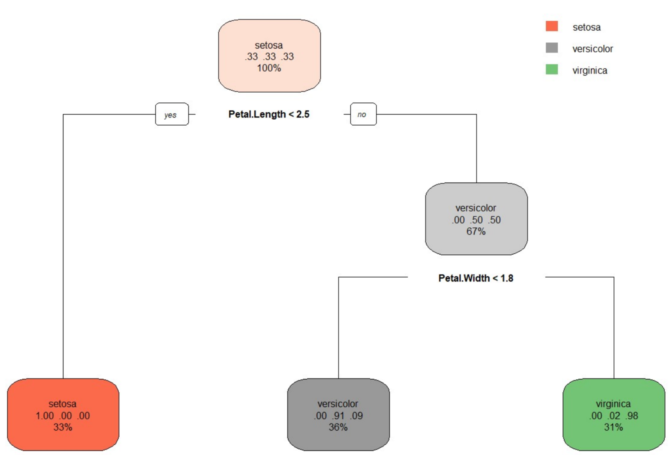

```
> rpart.plot(dtm, type=4, extra=101)
```




```
> p<-predict(dtm, iris_test, type="class")
> table(iris_test[,5], p)
           p
            setosa versicolor virginica
```

```
 setosa       18      0      0
 versicolor    0     12      1
 virginica     0      2     17
```

```
> library(rpart)
> model <- rpart(Species ~., data = iris)
> par(xpd = T) # otherwise on some devices the text is clipped
> plot(model)
> text(model)
> print(model)
n= 150

node), split, n, loss, yval, (yprob)
     * denotes terminal node

1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
  2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.500000
00)
    6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259)
*
    7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087)
*
> rpart.plot(model)
```



#Making predictions

```
> newdata <- data.frame(Sepal.Length = 6.5, Sepal.Width = 3.0, Petal.Length =
5.2, Petal.Width = 2.0)
> model %>% predict(newdata, "class")
        1
virginica
Levels: setosa versicolor virginica
```

#Classification trees
#Data set: PimaIndiansDiabetes2 [in mlbench package],
 # Load the data and remove NAs

```
> data("PimaIndiansDiabetes2", package = "mlbench")
> PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
```

 # Inspect the data

```
> sample_n(PimaIndiansDiabetes2, 3)
  pregnant glucose pressure triceps insulin mass pedigree age diabetes
1        3     106       54      21     158 30.9    0.292  24      neg
2        1     125       50      40     167 33.3    0.962  28      pos
3        1     109       56      21     135 25.2    0.833  23      neg
```

 # Split the data into training and test set

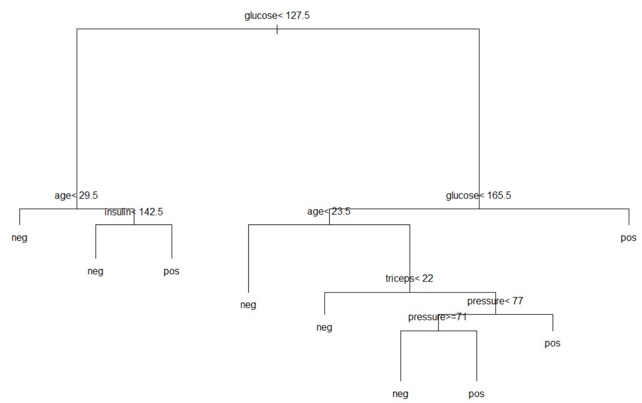
```
> set.seed(123)
> training.samples <- PimaIndiansDiabetes2$diabetes %>% createDataPartitio
n(p = 0.8, list = FALSE)
> train.data  <- PimaIndiansDiabetes2[training.samples, ]
> test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

#Fully grown trees
 # Build the model

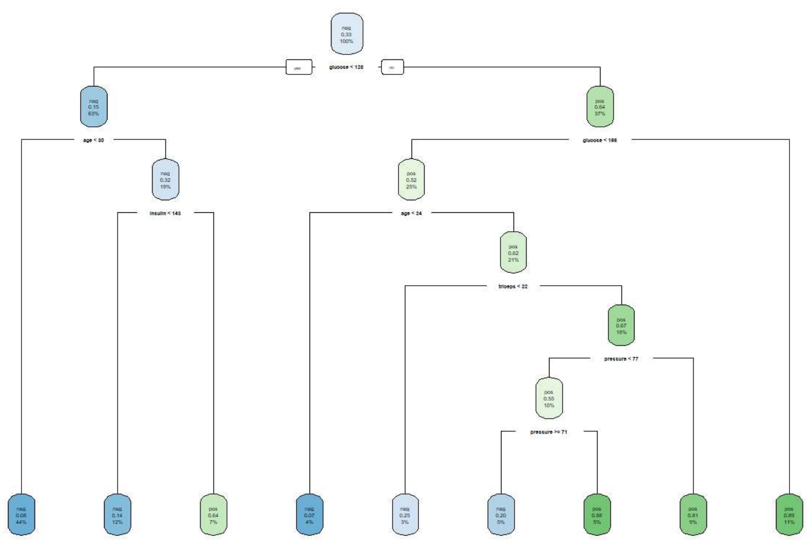
```
> set.seed(123)
> model1 <- rpart(diabetes ~., data = train.data, method = "class")
```

 # Plot the trees

```
> par(xpd = NA)
> plot(model1)
> text(model1)
```



> rpart.plot(model1)



 # Make predictions on the test data
> predicted.classes <- model1 %>% predict(test.data, type = "class")
> head(predicted.classes)
 19 21 32 55 64 71
neg neg neg pos pos neg
Levels: neg pos

#confusion matrix

```
> p<-predict(model1, test.data, type="class")
> table(test.data[,9], p)
     p
      neg pos
  neg  41  11
  pos   8  18
```

 # Compute model accuracy rate on test data

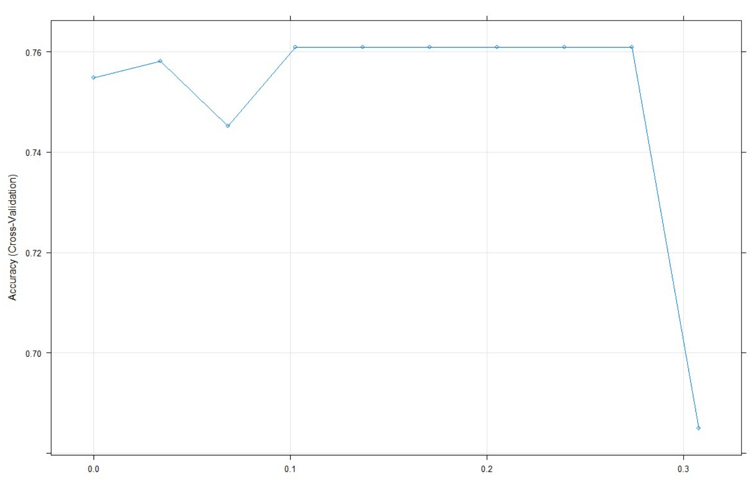
```
> mean(predicted.classes == test.data$diabetes)
[1] 0.7564103
```

#Pruning the tree
#You can use the following arguments in the function train() [from caret package]:
#trControl, to set up 10-fold cross validation
#tuneLength, to specify the number of possible cp values to evaluate. Default value is 3, here we will use 10.
 # Fit the model on the training set

```
> set.seed(123)
> model2 <- train(diabetes ~., data = train.data, method = "rpart", trControl = trainControl("cv", number = 10), tuneLength = 10)
```

 # Plot model accuracy vs different values of
 # cp (complexity parameter):default is 0.01

```
> plot(model2)
```



Print the best tuning parameter cp that
 # maximizes the model accuracy

```
> model2$bestTune
         cp
9 0.2735043
```

Plot the final tree model

```
> par(xpd = T)
> plot(model2$finalModel)
> text(model2$finalModel)
```

 # Decision rules in the model

```
> model2$finalModel
n= 314 node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 314 104 neg (0.6687898 0.3312102)
  2) glucose< 127.5 198  30 neg (0.8484848 0.1515152) *
  3) glucose>=127.5 116  42 pos (0.3620690 0.6379310) *
```

 # Make predictions on the test data

```
> predicted.classes <- model2 %>% predict(test.data)
```

 # Compute model accuracy rate on test data

```
> mean(predicted.classes == test.data$diabetes)
[1] 0.7307692
```

#CART

```
> library(rpart)
> data(iris)
> fit <- rpart(Species~., data=iris)
> summary(fit)
Call:
rpart(formula = Species ~ ., data = iris)
  n= 150

    CP nsplit rel error xerror       xstd
1 0.50      0      1.00   1.19 0.04959167
2 0.44      1      0.50   0.72 0.06118823
3 0.01      2      0.06   0.09 0.02908608

Variable importance
 Petal.Width Petal.Length Sepal.Length  Sepal.Width
          34           31           21           14

Node number 1: 150 observations,    complexity param=0.5
  predicted class=setosa      expected loss=0.6666667  P(node) =1
    class counts:    50    50    50
   probabilities: 0.333 0.333 0.333
  left son=2 (50 obs) right son=3 (100 obs)
  Primary splits:
      Petal.Length < 2.45 to the left,  improve=50.00000, (0 missing)
      Petal.Width  < 0.8  to the left,  improve=50.00000, (0 missing)
      Sepal.Length < 5.45 to the left,  improve=34.16405, (0 missing)
      Sepal.Width  < 3.35 to the right, improve=19.03851, (0 missing)
  Surrogate splits:
      Petal.Width  < 0.8  to the left,  agree=1.000, adj=1.00, (0 split)
      Sepal.Length < 5.45 to the left,  agree=0.920, adj=0.76, (0 split)
      Sepal.Width  < 3.35 to the right, agree=0.833, adj=0.50, (0 split)

Node number 2: 50 observations
  predicted class=setosa      expected loss=0  P(node) =0.3333333
    class counts:    50     0     0
```

```
  probabilities: 1.000 0.000 0.000

Node number 3: 100 observations,    complexity param=0.44
  predicted class=versicolor  expected loss=0.5  P(node) =0.6666667
    class counts:     0    50    50
   probabilities: 0.000 0.500 0.500
  left son=6 (54 obs) right son=7 (46 obs)
  Primary splits:
      Petal.Width  < 1.75 to the left,  improve=38.969400, (0 missing)
      Petal.Length < 4.75 to the left,  improve=37.353540, (0 missing)
      Sepal.Length < 6.15 to the left,  improve=10.686870, (0 missing)
      Sepal.Width  < 2.45 to the left,  improve= 3.555556, (0 missing)
  Surrogate splits:
      Petal.Length < 4.75 to the left,  agree=0.91, adj=0.804, (0 split)
      Sepal.Length < 6.15 to the left,  agree=0.73, adj=0.413, (0 split)
      Sepal.Width  < 2.95 to the left,  agree=0.67, adj=0.283, (0 split)

Node number 6: 54 observations
  predicted class=versicolor  expected loss=0.09259259  P(node) =0.36
    class counts:     0    49     5
   probabilities: 0.000 0.907 0.093

Node number 7: 46 observations
  predicted class=virginica   expected loss=0.02173913  P(node) =0.3066667
    class counts:     0     1    45
   probabilities: 0.000 0.022 0.978
```

 # make predictions

```
> predictions <- predict(fit, iris[,1:4], type="class")
```

 # summarize accuracy

```
> table(predictions, iris$Species)

predictions  setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         5
  virginica       0          1        45
```

C4.5

 # The C4.5 algorithm
 # is an extension of the ID3 algorithm and constructs a
 # decision tree to maximize information gain (difference in entropy).
 # load the package

```
> library(RWeka)
> data(iris)
> fit <- J48(Species~., data=iris)
> summary(fit)

=== Summary ===

Correctly Classified Instances        147               98      %
Incorrectly Classified Instances        3                2      %
Kappa statistic                       0.97
Mean absolute error                    0.0233
Root mean squared error                 0.108
Relative absolute error                5.2482 %
Root relative squared error           22.9089 %
Total Number of Instances              150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 49  1 |  b = versicolor
  0  2 48 |  c = virginica
```

 # make predictions

```
> predictions <- predict(fit, iris[,1:4])
> predictions
  [1] setosa    setosa    setosa    setosa    setosa    setosa    setosa    setosa
  [9] setosa    setosa    setosa    setosa    setosa    setosa    setosa    setosa
 [17] setosa    setosa    setosa    setosa    setosa    setosa    setosa    setos
a
 [25] setosa    setosa    setosa    setosa    setosa    setosa    setosa    setos
a
```

```
 [33] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setos
a
 [41] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setos
a
 [49] setosa     setosa     versicolor versicolor versicolor versicolor versicolor ve
rsicolor
 [57] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
versicolor
 [65] versicolor versicolor versicolor versicolor versicolor versicolor virginica  v
ersicolor
 [73] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
versicolor
 [81] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
versicolor
 [89] versicolor versicolor versicolor versicolor versicolor versicolor versicolor
versicolor
 [97] versicolor versicolor versicolor versicolor virginica  virginica  virginica  vir
ginica
[105] virginica  virginica  versicolor virginica  virginica  virginica  virginica  virgi
nica
[113] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgini
ca
[121] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgini
ca
[129] virginica  versicolor virginica  virginica  virginica  virginica  virginica  virgi
nica
[137] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgini
ca
[145] virginica  virginica  virginica  virginica  virginica  virginica
Levels: setosa versicolor virginica
```

 # summarize accuracy

```
> table(predictions, iris$Species)

predictions  setosa versicolor virginica
  setosa       50          0         0
  versicolor    0         49         2
  virginica     0          1        48
```

PART

 # PART is a rule system that creates pruned C4.5 decision trees for the data set
 # and extracts rules and those instances that are covered by the rules are re moved from the training data.
 # The process is repeated until all instances are covered by extracted rules.

```
> library(RWeka)
> data(iris)
> fit <- PART(Species~., data=iris)
> summary(fit)

=== Summary ===

Correctly Classified Instances        146              97.3333 %
Incorrectly Classified Instances        4               2.6667 %
Kappa statistic                       0.96
Mean absolute error                     0.0338
Root mean squared error                 0.1301
Relative absolute error                 7.6122 %
Root relative squared error            27.5902 %
Total Number of Instances             150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = setosa
  0 47  3 |  b = versicolor
  0  1 49 |  c = virginica
```

 # make predictions

```
> predictions <- predict(fit, iris[,1:4])
```

 # summarize accuracy

```
> table(predictions, iris$Species)

predictions  setosa versicolor virginica
  setosa        50          0         0
```

```
  versicolor     0        47        1
  virginica      0         3       49


> set.seed(678)
> titanic <-read.csv("C:/Users/Tushar/Downloads/titanic.csv")
> head(titanic)
  x pclass survived                                    name    sex    age sibsp parch
1 1      1        1                Allen, Miss. Elisabeth Walton female     29     0     0
2 2      1        1               Allison, Master. Hudson Trevor   male 0.9167     1     2
3 3      1        0                 Allison, Miss. Helen Loraine female      2     1     2
4 4      1        0         Allison, Mr. Hudson Joshua Creighton   male     30     1     2
5 5      1        0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female     25
1     2
6 6      1        1                        Anderson, Mr. Harry   male     48     0     0
  ticket     fare   cabin embarked                    home.dest
1  24160 211.3375      B5        S                 St Louis, MO
2 113781   151.55 C22 C26        S Montreal, PQ / Chesterville, ON
3 113781   151.55 C22 C26        S Montreal, PQ / Chesterville, ON
4 113781   151.55 C22 C26        S Montreal, PQ / Chesterville, ON
5 113781   151.55 C22 C26        S Montreal, PQ / Chesterville, ON
6  19952    26.55     E12        S                 New York, NY
```

#shuffle the data for splitting itno training and test data further
#Generate a random list of index from 1 to 1309 (i.e. the maximum number o f rows).

```
> shuffled <- sample(1:nrow(titanic))
> head(shuffled)
[1]   57  774  796 1044  681  920
> titanic <- titanic[shuffled, ]
> head(titanic)
       x pclass survived                                    name    sex  age sibsp
57     57      1        1                    Carter, Mr. William Ernest   male   36     1
774   774      3        0                            Dimic, Mr. Jovan   male   42     0
796   796      3        0                      Emir, Mr. Farred Chehab   male    ?     0
1044 1044      3        1                Murphy, Miss. Margaret Jane female    ?     1
681   681      3        0                           Boulos, Mr. Hanna   male    ?     0
920   920      3        0 Katavelas, Mr. Vassilios ('Catavelas Vassilios')   male 18.5
0
```

```
    parch ticket   fare   cabin embarked     home.dest
57       2 113760    120 B96 B98       S Bryn Mawr, PA
774      0 315088 8.6625      ?       S            ?
796      0   2631  7.225      ?       C            ?
1044     0 367230   15.5      ?       Q            ?
681      0   2664  7.225      ?       C        Syria
920      0   2682 7.2292      ?       C            ?
```

#Clean the dataset
 # Drop variables home.dest,cabin, name, X and ticket
 # Create factor variables for pclass and survived
 # Drop the NA

```
> library(dplyr)
```

 # preprocessing

```
> clean_titanic <- titanic %>%   select(survived, embarked, sex,sibsp, parch, far
e) %>% mutate(embarked = factor(embarked),sex = factor(sex))
```

##Create train/test set
 # The common practice is to split the data 80/20, 80 percent of the data serves to train the model,
 # and 20 percent to make predictions. You need to create two separate data frames.
 # You don't want to touch the test set until you finish building your model.

```
> create_train_test <- function(data, size = 0.8, train = TRUE) {
+   n_row = nrow(data)
+   total_row = size * n_row
+   train_sample <- 1: total_row
+   if (train == TRUE) {
+     return (data[train_sample, ])
+   } else {
+     return (data[-train_sample, ])
+   }
+ }

> data_train <- create_train_test(clean_titanic, 0.8, train = TRUE)
> data_test <- create_train_test(clean_titanic, 0.8, train = FALSE)
> dim(data_train)
[1] 1047    6
```

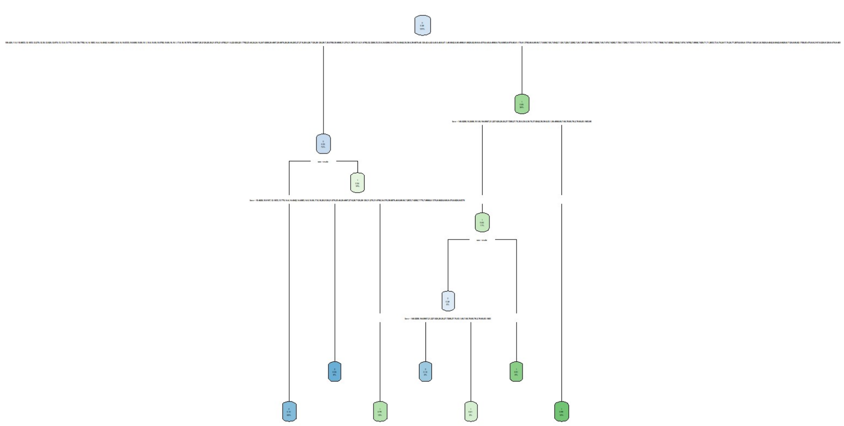
#To check the percentage portions after randomisation

```
> prop.table(table(data_train$survived))

        0         1 
0.6189112 0.3810888 
> prop.table(table(data_test$survived))

        0         1 
0.6145038 0.3854962 
```

#In both dataset, the amount of survivors is the same, about 40 percent.

#Build the decision tree model

```
> library(rpart)
> library(rpart.plot)
> fit <- rpart(survived~., data = data_train,method='class')
> rpart.plot(fit)
```



```
> predict_unseen <-predict(fit, data_test, type = 'class')
> predict_unseen
1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061
1062 1063 1064 1065 1066
   1    0    0    0    1    0    1    1    0    1    1    1    1    1    1    1    1    0    0
1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080
1081 1082 1083 1084 1085
   0    0    1    0    0    0    0    0    1    1    1    1    0    0    1    1    1    0    1
```

```
1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099
1100 1101 1102 1103 1104
   0    0    0    0    1    0    0    0    1    0    1    1    0    1    0    0    0    1    0
1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118
1119 1120 1121 1122 1123
   0    0    0    0    1    0    0    1    1    0    1    0    0    0    0    1    1    0    1
1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137
1138 1139 1140 1141 1142
   0    1    1    0    0    0    1    1    0    0    1    0    0    0    0    1    1    0    0
1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156
1157 1158 1159 1160 1161
   0    1    1    1    1    0    1    1    0    1    1    1    1    1    0    0    1    0    0
1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175
1176 1177 1178 1179 1180
   0    0    0    1    0    1    0    1    0    0    1    0    1    0    0    0    1    0    0
1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194
1195 1196 1197 1198 1199
   1    0    0    1    0    0    0    0    1    1    1    0    0    0    1    0    0    1    1
1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213
1214 1215 1216 1217 1218
   0    1    0    0    1    1    0    0    0    0    0    0    0    0    0    1    1    1    0
1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232
1233 1234 1235 1236 1237
   0    0    1    1    1    0    1    0    0    1    0    0    0    1    0    1    1    0    1
1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251
1252 1253 1254 1255 1256
   0    1    0    0    1    0    1    1    0    0    0    0    0    1    0    1    1    0    0
1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270
1271 1272 1273 1274 1275
   0    0    0    0    1    1    1    0    0    0    0    1    1    1    0    0    1    0    0
1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289
1290 1291 1292 1293 1294
   1    1    0    1    1    0    1    0    0    0    0    0    0    1    0    1    0    1    0
1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308
1309
   1    1    1    1    1    0    0    0    0    1    1    1    1    1    1
Levels: 0 1
```

#predict(fit, data_test, type = 'class'): Predict the class (0/1) of the test set

#Testing the passenger who didn't make it and those who did.

```
> table_mat <- table(data_test$survived, predict_unseen)
> table_mat
   predict_unseen
     0   1
  0 127  34
  1  20  81
```

 # table(data_test$survived, predict_unseen): Create a table to count
 # how many passengers are classified as survivors and passed away compare to the correct classification

###Measure performance
#Accuracy :It is the proportion of true positive and true negative over the sum of the matrix.
#With R, you can code as follow:

```
> accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
```

 # sum(diag(table_mat)): Sum of the diagonal
 # sum(table_mat): Sum of the matrix.

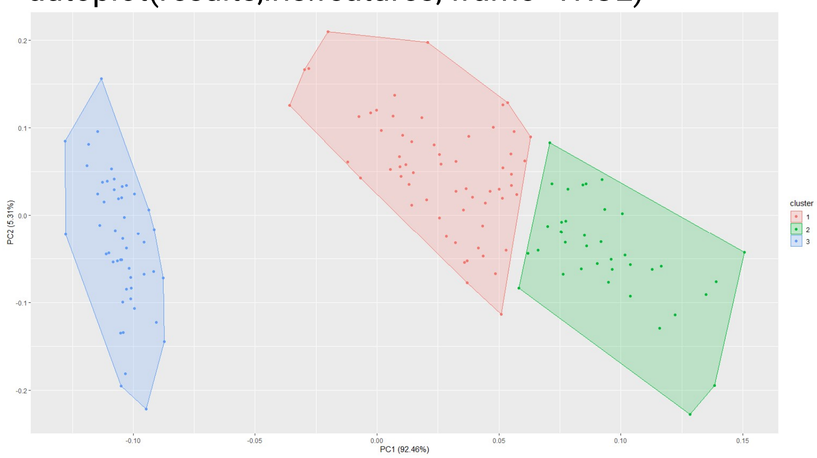
#You can print the accuracy of the test set:

```
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.793893129770992"
```

EXPERIMENT – 9 (KMEANS)

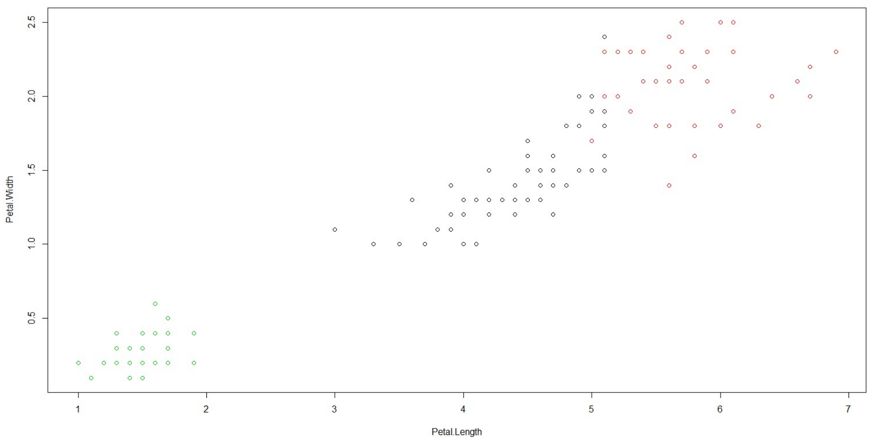
```
> library("stats")
> library("dplyr")
> library("ggplot2")
> library("ggfortify")

> 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.features=iris
> iris.features$Species<-NULL
> View(iris.features)
> results<-kmeans(iris.features, 3)
> results
K-means clustering with 3 clusters of sizes 62, 38, 50

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.901613    2.748387     4.393548    1.433871
2     6.850000    3.073684     5.742105    2.071053
3     5.006000    3.428000     1.462000    0.246000

Clustering vector:
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3
 [46] 3 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 1 1 1 1
 [91] 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2
2 2 2 1 2
[136] 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1

Within cluster sum of squares by cluster:
```

```
[1] 39.82097 23.87947 15.15100
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"

> autoplot(results,iris.features, frame=TRUE)
```

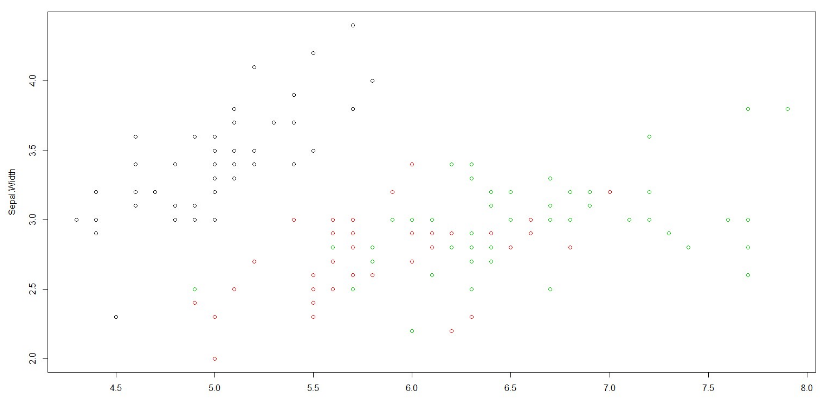



```
> results$cluster
  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3
 [46] 3 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 1 1 1 1
 [91] 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2
2 2 2 1 2
[136] 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1
> results$size
[1] 62 38 50
> table(iris$Species, results$cluster)

             1  2  3
  setosa      0  0 50
  versicolor 48  2  0
  virginica  14 36  0
```

```
> plot(iris[c("Petal.Length", "Petal.Width")], col= results$cluster)
```




```
> plot(iris[c("Petal.Length", "Petal.Width")], col= iris$Species)
> plot(iris[c("Sepal.Length", "Sepal.Width")], col= results$cluster)
> plot(iris[c("Sepal.Length", "Sepal.Width")], col= iris$Species)
```




```
> data("USArrests")
> dim(USArrests)
[1] 50  4
> str(USArrests)
'data.frame':	50 obs. of  4 variables:
 $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
 $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
 $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
 $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
> head(USArrests)
```

```
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
> df <- scale(USArrests) # Scaling the data
> head(df)
               Murder   Assault   UrbanPop         Rape
Alabama    1.24256408 0.7828393 -0.5209066 -0.003416473
Alaska     0.50786248 1.1068225 -1.2117642  2.484202941
Arizona    0.07163341 1.4788032  0.9989801  1.042878388
Arkansas   0.23234938 0.2308680 -1.0735927 -0.184916602
California 0.27826823 1.2628144  1.7589234  2.067820292
Colorado   0.02571456 0.3988593  0.8608085  1.864967207
```

#Required R packages and functions
#The standard R function for k-means clustering is kmeans() [stats package]
,
#which simplified format is as follow:
#kmeans(x, centers, iter.max = 10, nstart = 1)
#x: numeric matrix, numeric data frame or a numeric vector
#centers: Possible values are the number of clusters (k) or a set of initial (dis tinct) cluster centers. If a number, a random set of (distinct) rows in x is chos en as the initial centers.
#iter.max: The maximum number of iterations allowed. Default value is 10.
#nstart: The number of random starting partitions when centers is a number. Trying nstart > 1 is often recommended.

#To create a beautiful graph of the clusters generated with the kmeans()
#function, will use the factoextra package.

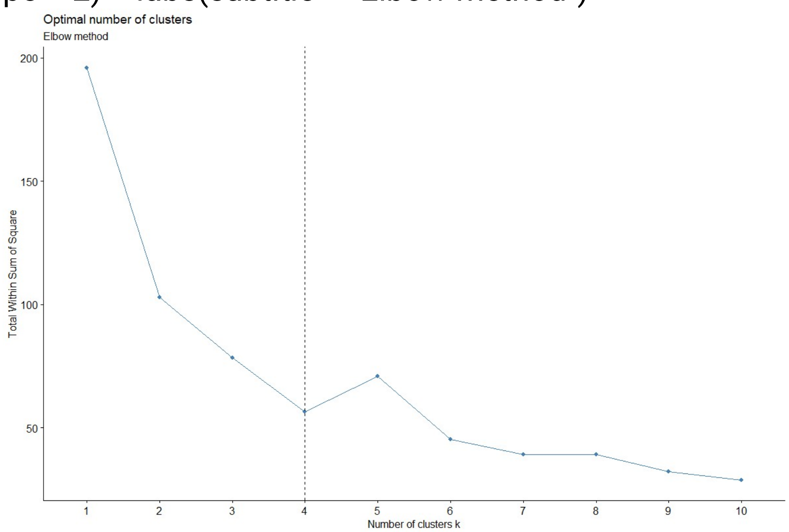
```
> library(factoextra)
```

 # optimal number of clusters
#fviz_nbclust() [in factoextra package] provides a convenient solution to esti mate the optimal number of clusters.
#The simplified format is as follow:
#fviz_nbclust(x, FUNcluster, method = c("silhouette", "wss", "gap_stat"))

#x: numeric matrix or data frame
#FUNcluster: a partitioning function. Allowed values include kmeans, pam, clara and hcut (for hierarchical clustering).
#method: the method to be used for determining the optimal number of clusters.

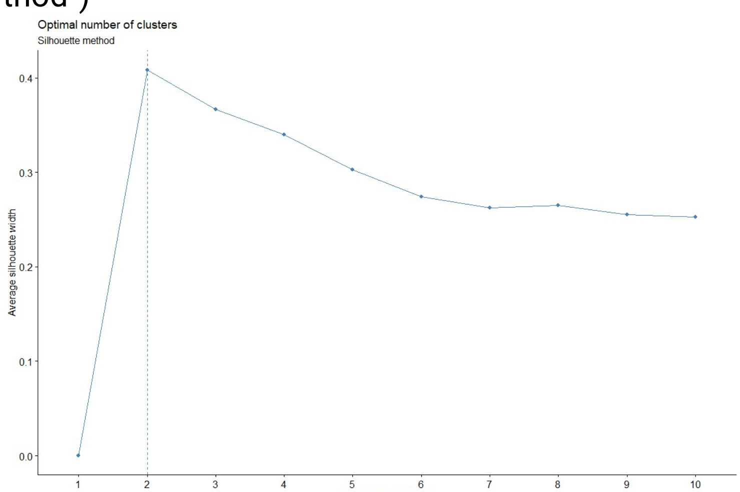
 # Elbow method

> fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")



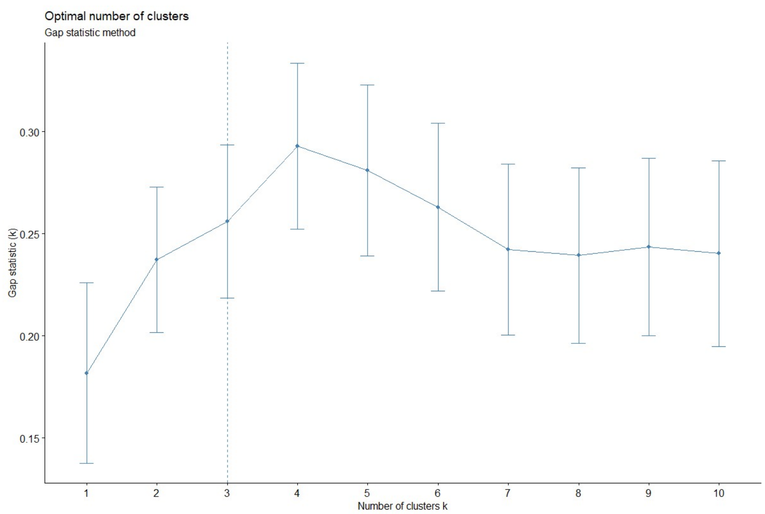
 # Silhouette method

> fviz_nbclust(df, kmeans, method="silhouette")+labs(subtitle = "Silhouette method")



 # Gap statistic
 # nboot = 50 to keep the function speedy.
 # recommended value: nboot= 500 for your analysis.
 # Use verbose = FALSE to hide computing progression.

```
> set.seed(123)
> fviz_nbclust(df, kmeans, nstart = 25,  method = "gap_stat", nboot = 50)+
+   labs(subtitle = "Gap statistic method")
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 50)  [one "." per sample]:
.................................................. 50
```



#Elbow method: 4 clusters solution suggested
#Silhouette method: 2 clusters solution suggested
#Gap statistic method: 4 clusters solution suggested
#define k = 4 as the optimal number of clusters in the data.

#Computing k-means clustering
 # the R code below performs k-means clustering with k = 4:
 # Compute k-means with k = 4

```
> set.seed(123)
> km.res <- kmeans(df, 4, nstart = 25)
```

 # Print the results

```
> print(km.res)
K-means clustering with 4 clusters of sizes 8, 13, 16, 13
```

```
Cluster means:
      Murder    Assault   UrbanPop        Rape
1  1.4118898  0.8743346 -0.8145211  0.01927104
2 -0.9615407 -1.1066010 -0.9301069 -0.96676331
3 -0.4894375 -0.3826001  0.5758298 -0.26165379
4  0.6950701  1.0394414  0.7226370  1.27693964

Clustering vector:
       Alabama         Alaska        Arizona       Arkansas     California       Colorado
             1              4              4              1              4              4
   Connecticut       Delaware        Florida        Georgia         Hawaii          Idaho
             3              3              4              1              3              2
      Illinois        Indiana           Iowa         Kansas       Kentucky      Louisiana
             4              3              2              3              2              1
         Maine       Maryland  Massachusetts       Michigan      Minnesota    Mississ
ippi
             2              4              3              4              2              1
      Missouri        Montana       Nebraska         Nevada  New Hampshire     New J
ersey
             4              2              2              4              2              3
    New Mexico       New York North Carolina   North Dakota           Ohio       Okla
homa
             4              4              1              2              3              3
        Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota      Te
nnessee
             3              3              3              1              2              1
         Texas           Utah        Vermont       Virginia     Washington  West Virginia
             4              3              2              3              3              2
     Wisconsin        Wyoming
             2              3

Within cluster sum of squares by cluster:
[1]  8.316061 11.952463 16.212213 19.922437
 (between_SS / total_SS =  71.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
```

```
[7] "size"        "iter"        "ifault"

> autoplot(km.res,df, frame=TRUE)
```

#Its possible to compute the mean of each variables by clusters using the original data:

```
> aggregate(USArrests, by=list(cluster=km.res$cluster), mean)
  cluster   Murder   Assault UrbanPop     Rape
1       1 13.93750 243.62500 53.75000 21.41250
2       2  3.60000  78.53846 52.07692 12.17692
3       3  5.65625 138.87500 73.87500 18.78125
4       4 10.81538 257.38462 76.00000 33.19231
```

#kmeans() function returns a list of components, including:
#cluster: A vector of integers (from 1:k) indicating the cluster to which each point is allocated
#centers: A matrix of cluster centers (cluster means)
#totss: The total sum of squares (TSS), i.e â^‘(xiâ^’xÂ¯)2. TSS measures the total variance in the data.
#withinss: Vector of within-cluster sum of squares, one component per cluster
#tot.withinss: Total within-cluster sum of squares, i.e. sum(withinss)
#betweenss: The between-cluster sum of squares, i.e. totssâ^’tot.withinss
#size: The number of observations in each cluster

 # Cluster number for each of the observations

```
> km.res$cluster
       Alabama         Alaska        Arizona       Arkansas     California       Colorado 
             1              4              4              1              4              4 
   Connecticut       Delaware        Florida        Georgia         Hawaii          Idaho 
             3              3              4              1              3              2 
      Illinois        Indiana           Iowa         Kansas       Kentucky      Louisiana 
             4              3              2              3              2              1 
         Maine       Maryland  Massachusetts       Michigan      Minnesota    Mississippi 
             2              4              3              4              2              1 
      Missouri        Montana       Nebraska         Nevada  New Hampshire     New Jersey 
             4              2              2              4              2              3 
```

```
   New Mexico      New York North Carolina   North Dakota         Ohio      Okla
homa
          4           4           1           2           3           3
      Oregon   Pennsylvania   Rhode Island South Carolina   South Dakota     Te
nnessee
          3           3           3           1           2           1
       Texas        Utah       Vermont      Virginia    Washington  West Virginia
          4           3           2           3           3           2
    Wisconsin       Wyoming
          2           3
> head(km.res$cluster, 4)
 Alabama   Alaska  Arizona Arkansas
      1        4        4        1
```

 # Cluster size

```
> km.res$size
[1]  8 13 16 13
```

 # Cluster means

```
> km.res$centers
     Murder    Assault   UrbanPop       Rape
1  1.4118898  0.8743346 -0.8145211  0.01927104
2 -0.9615407 -1.1066010 -0.9301069 -0.96676331
3 -0.4894375 -0.3826001  0.5758298 -0.26165379
4  0.6950701  1.0394414  0.7226370  1.27693964
```