Data Analysis using R and Python

DISSERTATION
SUBMITTED TO
CENTRAL UNIVERSITY OF HARYANA,
MAHENDERGARH, HARYANA - 123031.
FOR THE AWARD OF THE DEGREE OF MASTER OF SCIENCE
IN STATISTICS

SUPERVISED BY:
Dr. Kapil Kumar
Assistant Professor

SUBMITTED BY:
Jogesh Dhiman
Roll No. : 8952

DEPARTMENT OF STATISTICS
CENTRAL UNIVERSITY OF HARYANA,
MAHENDERGARH, HARYANA - 123031.
June 2018

DECLARATION

I hereby declare that the dissertation work entitled "Data Analysis using R and Python " submitted to the Department of Statistics, Central University of Haryana for the partial fulfillment of award of the degree of Master of Science in Statistics, is done by me under the supervision of Dr. Kapil Kumar, Assistant Professor, Department of Statistics, Central University of Haryana. The content of this dissertation has not been submitted so far in full or part for any degree or diploma in any other University or Institution.

Place: Central University of Haryana
Date :

Jogesh Dhiman
M.Sc. Statistics
Roll No. 8952

CERTIFICATE

This is to certify that the dissertation entitled "Data Analysis using R and Python " is being submitted to the Department of Statistics, Central University of Haryana, Mahendergarh, as dissertation work for partial fulfillment of requirements for the award of the degree of Master of Science in Statistics.

This work has been completed by Mr. Jogesh Dhiman under my supervision and has not been submitted elsewhere for any degree or diploma in any form.

Dr. Kapil Kumar
Assistant Professor
Department of Statistics
Central University of Haryana
Mahendergarh-123031

ACKNOWLEDGEMENT

This dissertation work owes its existence to the help, support and inspiration of several people including my well wishers and my friends. I would like to express my gratitude to all those who have helped me directly or indirectly to complete this work.

It is a great pleasure to express my gratitude and everlasting indebtedness to my dissertation supervisor Dr. Kapil Kumar, Department of Statistics, Central University of Haryana, Mahendergarh for his guidance during dissertation work. His support and inspiring suggestions have been precious not only for the development of this dissertation work but also to achieve my goals.

I am also indebted to Dr. Devendra Kumar and Dr. Manoj Kumar. They have been a constant source of encouragement and enthusiasm, not only during this dissertation but also during the two years of my Master programme.

I would like to express my heartful gratitude to my friends and classmates for the inspiration, support, cooperation and patience which I received from them.

My gratitude goes to my parents for their blessings, love, patience and moral support. They always encouraged me to achieve my goals.

Place: Central University of Haryana — Jogesh Dhiman
Date : — M.Sc. Statistics

Contents

Chapter 1

Introduction

Python and R are two very popular open source software's used for data analysis. However both software's are of equal importance and can be used as complement of each other. When it comes to choose one, R offers more depth when it comes to data analysis, data modelling while Python is easier to learn and tends to represent graphs in more polished way.

1.1 Why R?

- Open Source: R is an open source software and can be installed easily.
- Learning: R requires you to learn and understand coding. It's a low level programming which can take longer codes.
- Statistical Analysis: Designed by statisticians for doing statistical analysis. In R, statistical analysis is made strong using large number of packages. Currently, on dated 09^{th} June 2018, the CRAN package repository features 12611 available packages. Some of the well known packages are:
  For Visualization: ggplot2 (See Hadley Wickham, Winston Chang, RStudio (2016))
  For Reporting: knitr (See Yihui Xie (2018))
  For Geography: maps, ggmaps (See David Kahle, Hadley Wickham (2016))

1.2 Why Python?

- Open source: Python is also a open source software and can be installed easily.
- Learning: Python is easy to learn. Python is known for its simplicity in programming world.
- Speed: People are inappropriately obsessed with speed. Python is a high-level language, which means it has number of benefits to accelerate codes. Another benefit is that it is easy to learn.

- Statistical Analysis: Python is widely used in scientific computing and statistical analysis in industry and academics. A large number of analytics libraries are available, including numerical analysis, statistical analysis, data analysis, visualization. (See https://docs.python.org/3/library/)

1.3 Installing R software

R is an open source and can obtained from **Comprehensive R Archive Network (CRAN)**, which can be done as follows:

- Open an internet browser and go to www.r-project.org.
- Click the "download R" link in the middle of the page under "Getting Started."
- Select a CRAN location (a mirror site) and click the corresponding link.
- Click on the "Download R for (Mac) OS X" link at the top of the page.
- Click on the file containing the latest version of R under "Files."
- Save file, double-click it to open, and follow the installation instructions.

1.4 Installing Python Software

Python is an open Source and can be downloaded from website www.python.org , which can be done as follows:

- Go to the python website www.python.org and click on the 'Download' menu choice.
- Next click on the Python 3.7 (note that the version number may change) 'Windows Installer' to download the installer. If you know you're running a 64-bit OS, you can choose the x86-64 installer.
- Be sure to save the file that you are downloading.
- Once you have downloaded the file, open it. (You can also double-click on it to open it.)
- Now follow the installation instructions.
- Now for doing data analysis. We need to install packages.

Note: For new users of python, Anaconda (from https://www.continuum.io) is recommended due to complication in installing packages in python used in 'Data Analysis'. Anaconda provides you almost all necessary packages you need at work.
Otherwise, you would need to install every package separately and many times you will run into installation errors because of incompatible packages' versions.

Once the software is installed, now it can be executed by launching corresponding executable. The prompt, by default '>' and '>>>' in R and Python respectively, indicates that software is waiting for your commands. At this stage, new user will think, "what do I do now?" Using for the first time, let's start with their basic uses i.e. use these software's as a calculator. We can write commands in window at command prompt and use both software as powerful calculators.

1.5 Basic Operators in R

R language have usual basic operators. The common operators are:

1.5.1 Arithmetic Operators:

Following are the arithmetic operators in R.

Arithmetic Operators

Operators	Meaning	Examples
+	Addition	>7+6 [1] 13
-	Subtraction	> 9-5 [1] 4
*	Multiplication	> 6*2 [1] 12
/	Division	> 9/3 [1] 3
%%	Modulus i.e. remainder of the division of left operand by the right	> 5%%2 [1] 1 (i.e. remainder of 5/2)
^	Exponent - left operand raised to the power of right	> 2^3 [1] 8 (i.e. 2 to the power 3)
pi	Pi (R knows about pi i.e.π)	> pi [1] 3.141593
sin	Sin trigonometric function	> sin(90*(pi/180)) [1] 1 (i.e. converts angle radians then take sin())
log	Logarithm	> log(25) [1] 3.218876

```
4.12e-2

## [1] 0.0412
```

```
cos(120*pi/180)

## [1] -0.5

log(100,base=10)  #Takes the logarithm of x with base y;

## [1] 2

#if base is not specified, returns the natural logarithm
exp(25)  #Returns the exponential of x

## [1] 72004899337

sqrt(25)  #Returns the square root of x

## [1] 5

factorial(4)  #Returns the factorial of x (x!)

## [1] 24

choose(5,3)  #Returns the number of possible combinations

## [1] 10

#when drawing y elements at a time from x possibilities
```

1.5.2 Logical Operators

Following are logical operators in R:

Logical operators

Operators	**Meaning**
$<$	Less than
$>$	Greater than
$<=$	Less than equal to
$>=$	Greater than equal to
==	Exactly equal to
!=	Not equal to
! x	Not x
x | y	x or y
x&y	x and y

```
x=c(1:5)
y=2
x<y

## [1]  TRUE FALSE FALSE FALSE FALSE

x>y

## [1] FALSE FALSE  TRUE  TRUE  TRUE

x<=y

## [1]  TRUE  TRUE FALSE FALSE FALSE

x>=y

## [1] FALSE  TRUE  TRUE  TRUE  TRUE

x==y

## [1] FALSE  TRUE FALSE FALSE FALSE

x!=y

## [1]  TRUE FALSE  TRUE  TRUE  TRUE

!x

## [1] FALSE FALSE FALSE FALSE FALSE

!y

## [1] FALSE
```

1.6 Basic Operators in Python

Python also have usual basic arithmetic operators. The common operators are:

1.6.1 Arithmetic Operators

Some arithmetic operators used in Python are given below in table.

```
In [1]: import math

In [2]: math.log10(100)#Return the base-10 logarithm
```

Arithmetic operators

Operators	Meaning	Examples
+	Addition	>>> 3+5 8
-	Subtraction	>>> 6+4 10
*	Multiplication	>>> 5*2 10
/	division	>>> 8/4 2.0
%	Modulus - remainder of the division of left operand by the right	>>> 5%2 1 (remainder of 5/2)
**	Exponent - left operand raised to the power of right	>>> 2**3 8 (i.e. 2 to the power 3)
math.sin	Sin trigonometric function	>>> math.sin(90*(math.pi/180)) 1.0
math.log	logarithm	>>> math.log(25) 3.2188758248682006

```
Out[2]: 2.0

In [3]: math.cos(25)#return the cosine of 25 radians

Out[3]: 0.9912028118634736

In [4]: math.factorial(5) #return the value of 5!

Out[4]: 120

In [5]: math.gamma(5) #return the gamma function at 5

Out[5]: 24.0

In [6]: math.degrees(90) #return  angle 90 from radians to degree

Out[6]: 5156.620156177409
```

1.6.2 Logical Operators

The following are Logical operators used in Python:

```
In [1]: x=2
        y=7
```

Logical operators

Operators	Meaning
$<$	Less than
$>$	Greater than
$<=$	Less than equal to
$>=$	Greater than equal to
$==$	Exactly equal to
!=	Not equal to
!x	Not x
x | y	x or y
x&y	x and y

```
In [2]: if(x==y):
            print("x is equal to y")
        else:
            print("x is not equal to y")

x is not equal to y


In [3]: if (x>y):
            print("x is greater than y")
        else:
            print("x is not greater than y")

x is not greater than y


In [4]: if (x<y):
            print("x is less than y")
        else:
            print("x is not less than y")

x is less than y


In [5]: if (x>=y):
            print("x is greater than equal to y")
        else:
            print("x is not greater than equal to y")

x is not greater than equal to y
```

```
In [6]: if (x<=y):
            print("x is less than equal to y")
        else:
            print("x is not less than equal to y")

x is less than equal to y
```

Since every value has a data type. Data type are classes and variables are instances of these classes.

1.7 Data types in R

R has various type of 'data type' which includes vectors, matrices, array, data frame and lists.

- Vectors: when we have to create a vector with more than one variable, then we use c() function which means combining elements into vector. For example:

```
#create a vector
X<- c(1,2,3,5,6,7)
X

## [1] 1 2 3 5 6 7

print(class(X))# to get class of vector

## [1] "numeric"
```

- Matrices: Matrices is 2 dimensional data structure. It consists of elements of same class. It can be created using vector input to a matrix function. For example:

```
Y<- matrix(1:6, nrow=2, ncol=3)
Y

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

Z<- matrix(c("a","b","c","d"), byrow=T, nrow=2, ncol=2)
Z

##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "c"  "d"
```

- List: List is a special type of vector which contain elements of different data types. For example:

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)

# Print the list.
print(list1)

## [[1]]
## [1] 2 5 3
##
## [[2]]
## [1] 21.3
##
## [[3]]
## function (x)  .Primitive("sin")
```

- Dataframe: Data frame is commonly used data type to store tabular data. It is different from matrices data type. In matrix, every element should be from same class. But in data frame you can use list of vector of different class. Every column act as a list. Every time you read data in R will be stored in a data frame. For example:

```
df <- data.frame(name = c("ash","jane","paul","mark"),
                 score = c (67,56,87,91))
df

##   name score
## 1  ash    67
## 2 jane    56
## 3 paul    87
## 4 mark    91
```

1.8 Data types in Python

Python has five standard type data.

- Number: Number variables are created by the standard Python method. For example:

```
In [1]: a=5
```

```
In [2]: print(a,type(a))

5 <class 'int'>


In [3]: b=0.265

In [4]: print(b,type(b))

0.265 <class 'float'>
```

- String: String can be defined by using single('), double(") or triple(''') inverted commas. For example:

```
In [5]: greeting='greeting'

In [6]: print(greeting,type(greeting))

greeting <class 'str'>
```

- List: List are most useful data structures in Python. List can be simply defined by using comma separated values in square brackets. List can contain items of different type. But usually items have same type. Python lists are mutable and individual elements of a list can be changed. For example:

```
In [7]:  square_list=[0,1,2,4,5,15,25,30]

In [8]: print(square_list,type(square_list))

[0, 1, 2, 4, 5, 15, 25, 30] <class 'list'>


In [9]:  square_list[0]

Out[9]: 0

In [10]: square_list[2:6]

Out[10]: [2, 4, 5, 15]
```

- Tuple: Tuple is represented by number of values separated by commas. Tuple are immutable and can not change, they are faster in processing as compared to list. Hence, if your list is unlikely to change, you should use letups, instead of list. For example:

```
In [11]:  tulle_ex=1,2,3,5,9,77,44,55,21,36,59

In [12]: print(tulle_ex,type(tulle_ex))

(1, 2, 3, 5, 9, 77, 44, 55, 21, 36, 59) <class 'tulle'>
```

- Dictionary: Dictionaries in Python are lists of key:value pair. Dictionaries can be used to sort, iterate and compare data. Dictionaries are created using braces({}) with pairs separated by commas( ) and key values associated with a colon(:). In dictionaries key must be unique. For example:
  In [13]: dict={"A":1,"B":2,"C":3}

```
In [13]: dict={"A":1,"B":2,"C":3}

In [14]:  dict.keys()

Out[14]: dict_keys(['A', 'B', 'C'])

In [15]:  dict.values()

Out[15]: dict_values([1, 2, 3])
```

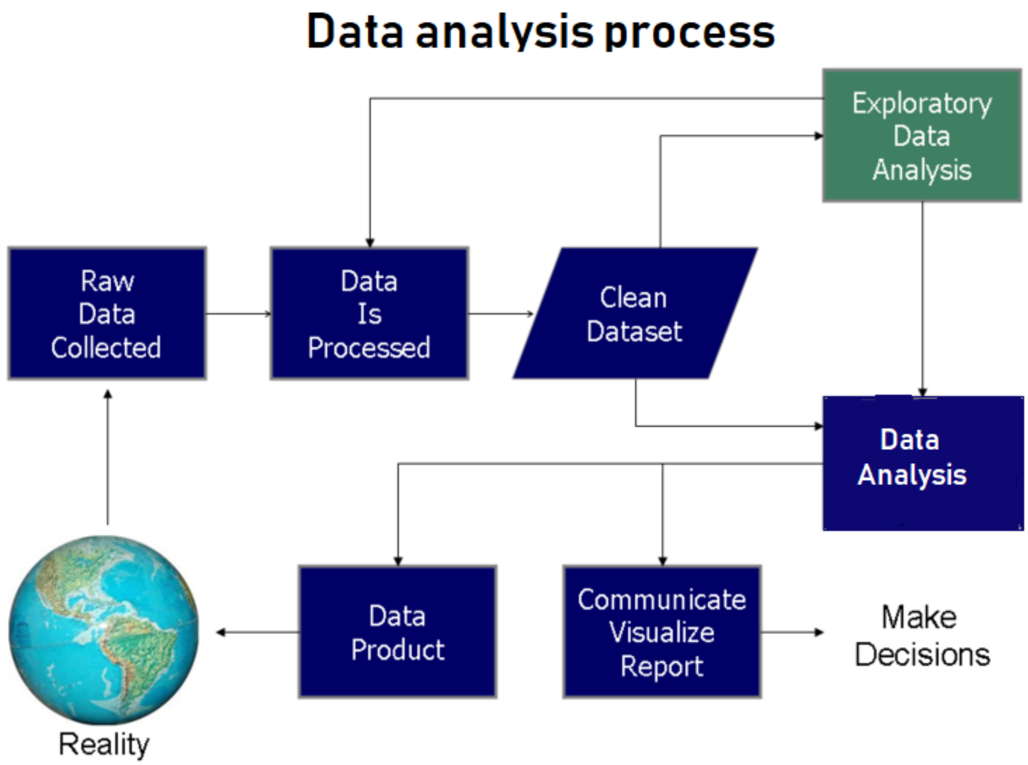
1.9 Data Analysis

Data analysis is a process of cleaning, transforming and modelling data with goal of getting a useful information, suggesting conclusions and making decision from data. In other words main purpose of data analysis is to look at what the data is trying to tell us. Data analysis with good statistical program is not difficult. Data analysis doesn't require much knowledge of mathematics or formulas that program uses for analysis. Data analysis requires few things:

- A clean data that is ready for analysis.
- A clear idea that what questions you want the data to answer.

Process of data analysis have following steps:

- Data cleaning
- Explanatory data dnalysis
- Data analysis
- Visualize report
- Decision making


Data analysis process
Exploratory Data Analysis
Raw Data Collected
Data Is Processed
Clean Dataset
Data Analysis
Data Product
Communicate Visualize Report
Make Decisions
Reality

Chapter 2

Graphics And Visualization

Graphics and visualization are powerful tool for describing and assisting analysis of data. The power of graphics and visualization arises from the fact that they describes the large quantities of information quickly. Graphics play an important role in good data analysis. They are useful for storing large data sets during data analysis, assist in describing and summarizing the data, and they can be tightly integrated with formal analytical statistical tools such as model-fitting techniques so that the analysis process can be refined.

2.1 Graphics and Visualization using R

R offers a variety of powerful tools for graphics and visualization. Each graphical function have a large number of options for producing graphs making it flexible. It is possible to display data and outcomes in wide variety of different ways. Base R plotting commands are used to display a variety of graphs and is divided into two basic groups:

- High-level plotting function: High-level plotting function create a new graph on the device, possibly with axis, labels, etc. Function such as hist(), plot(), boxplot() produces a entire plot or initialize a plot. High level potting function starts a new plot, erasing the current plot if following. Some of standard plot functions are:

  ```
  S.No  Function      Name of the plot
  1     plot()        Scatter plot
  2     hist()        Histogram
  3     boxplot()     Box plot or Box-and-whiskers plot
  4     stripchart()  Strip-chart
  5     barplot()     Bar-diagram
  6     stem()        Stem and leaf display
  ```

  The number of arguments can be passed to high-level plotting function, as follows:

  ```
  S.No. Arguments     Explanation
  1     main=" "      Title of the plot
  2     xlab=" "      Label for x axis
  ```

3	ylab=" "	Label for y axis
4	xlim=	Specify x limit
5	ylim=	Specify y limit
6	type="p/l/o"	Style of plotting symbol
7	pch=" "	Shape of the points
8	lty=" "	Style of the line

- Low-Level plotting function: Sometimes high-level plotting functions do not produce the graphs which we desire or we want to add more information to the graph then we use low-level plotting functions those add more information to an existing plot, such as lines, labels, extra points. Some low-level ploting functions are:

S.No.	Function	Explanation
1	lines()	Lines
2	abline()	lines given by intercept and slope
3	points()	points
4	text()	Text in the plot
5	legend()	List of symbols

R has many powerful packages for graphical representation and visualization. Two most widely used packages are:

- lattice: lattice was created by Deepayan Sarkar, see Sarkar(2017) lattice is a powerful and high-level plotting data visualization system. The lattice is based on Grid-graphics engine and requires grid add on package. lattice provides its own interface for modification of set of graphical and non-graphical settings.

- ggplot2: ggplot2 was created by Hadley Wickham in 2005, see Wickham(2016). Since 2005 ggplot2 has been grown in use to become most powerful R package. In comparison to R base, ggplot2 allows user to add or remove components at high-level of abstraction. This abstraction comes at a cost, that ggplot2 is being slower than lattice graphics.

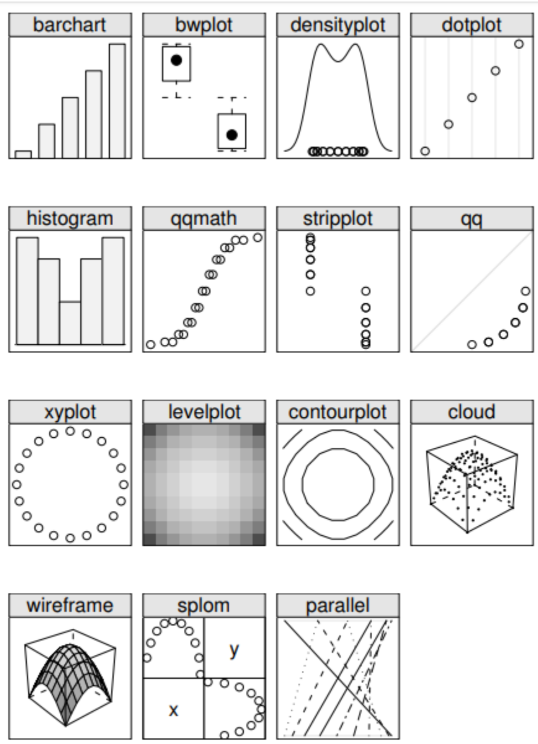
Dataset:
We are using Motor Trend Car Road tests datasets in R dataset package.
Description: The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973-74 models).

2.1.1 Application of lattice

The lattice package created by Deepayan Sarkar is an attempt to improve R base graphics. Lattice package provides better default and ability to simply display multivariate analysis.(See Sarkar D.(2017)) The plotting functions available in lattice:

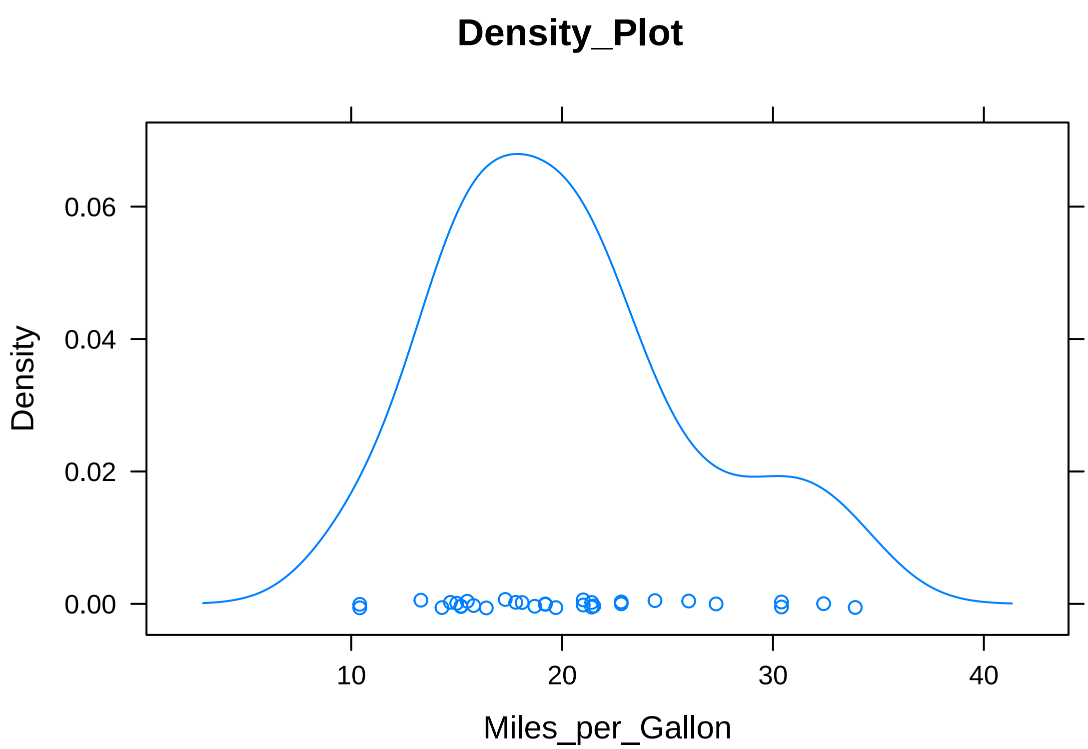
Lattice Function	Description	R base analogue
bwplot()	Boxplots	boxplot()

barchart()	Barcharts	barplot()
histogram()	Histograms	hist()
densityplot()	density plots	none
qqmath()	Quantile-quantile plot (Data set vs theoretical distribution)	qqnorm()
dotplot()	Dotplots	dotchart()
xyplot()	Scatterplots	plot()
stripplot()	Stripplots	stripchart()
qq()	Quantile-quantile plots(Data set vs data set)	qqplot()
cloud()	3-D scatterplot	none
wireframe()	3-D surfaces	persp()
levelplot()	Level plots	image()
contourplot()	Contour plots	contour()
splom()	Scatterplot matrices	pairs()
parallel()	Parallel coordinate plots	none

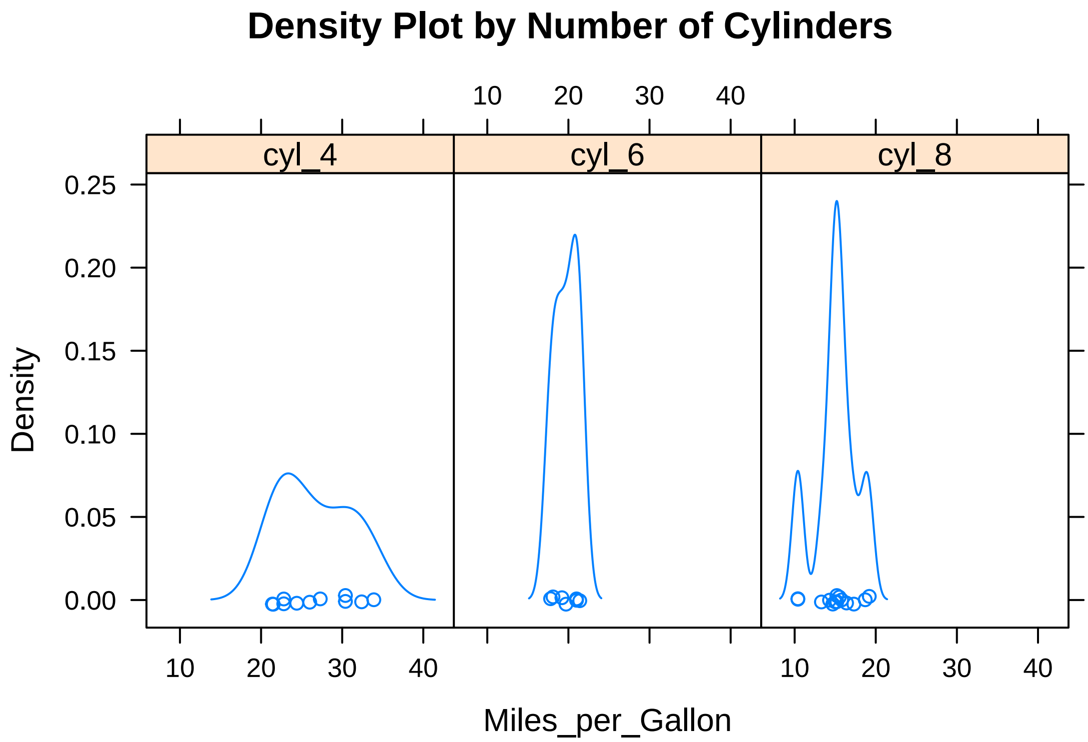


The lattice package supports the trellis graph that shows the relationship variables. The basic format of lattice is:
graph_type(formula, data=)
Here we show some applications of lattice function with example of dataset of mtcars:

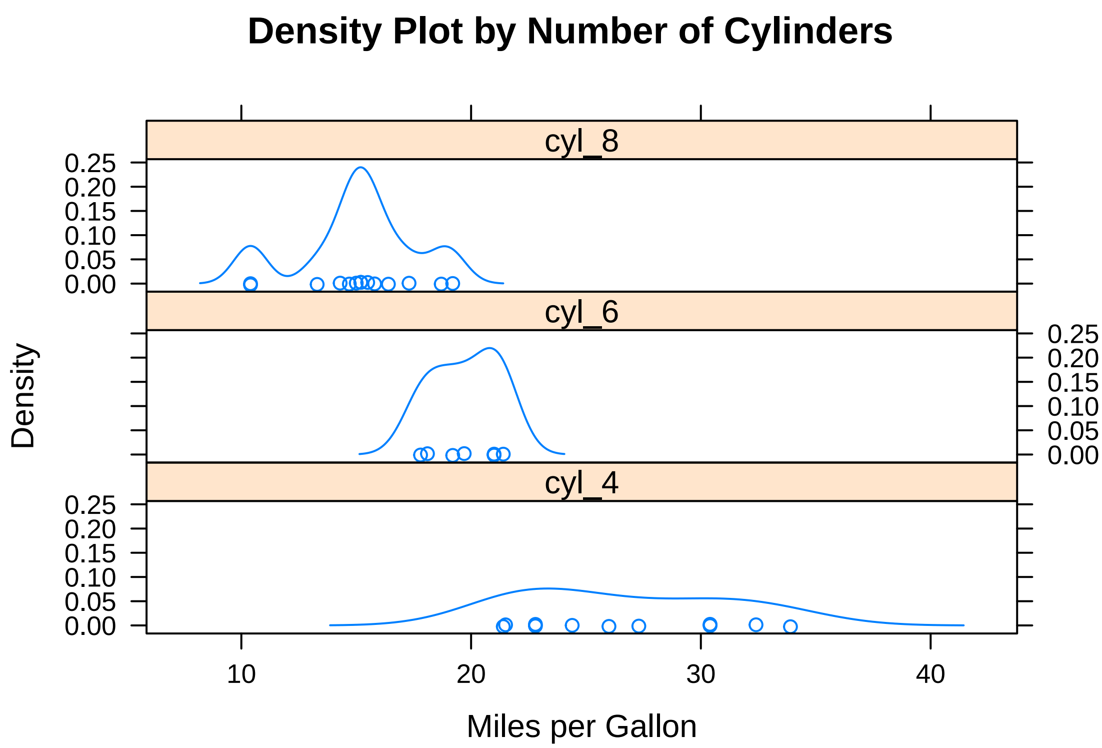
```
# Lattice Examples
library(lattice)
attach(mtcars)

# create factors with value labels
cyl.factor <-factor(cyl,levels=c(4,6,8),
   labels=c("cyl_4","cyl_6","cyl_8"))
gear.factor<-factor(gear,levels=c(3,4,5),
        labels=c("gears_3","gears_4","gears_5"))


# kernel density plot
densityplot(~mpg,
        main="Density_Plot",
        xlab="Miles_per_Gallon")
```



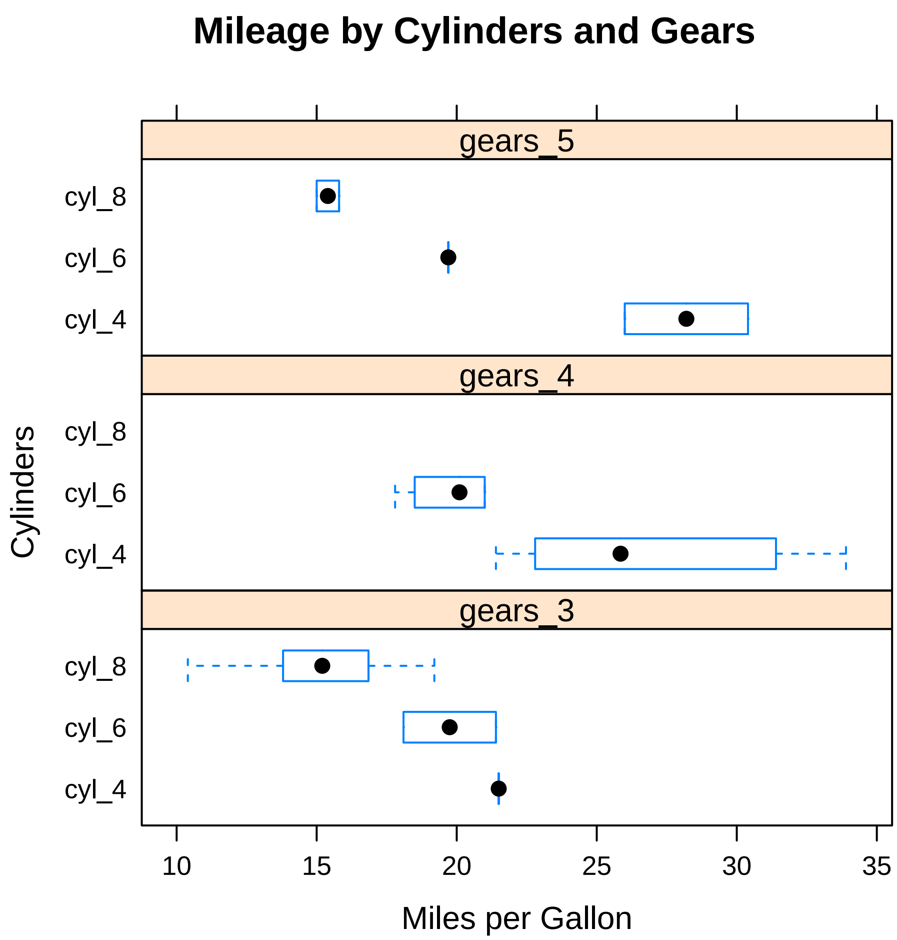

```
# kernel density plots by factor level
densityplot(~mpg|cyl.factor,
        main="Density Plot by Number of Cylinders",
   xlab="Miles_per_Gallon")
```



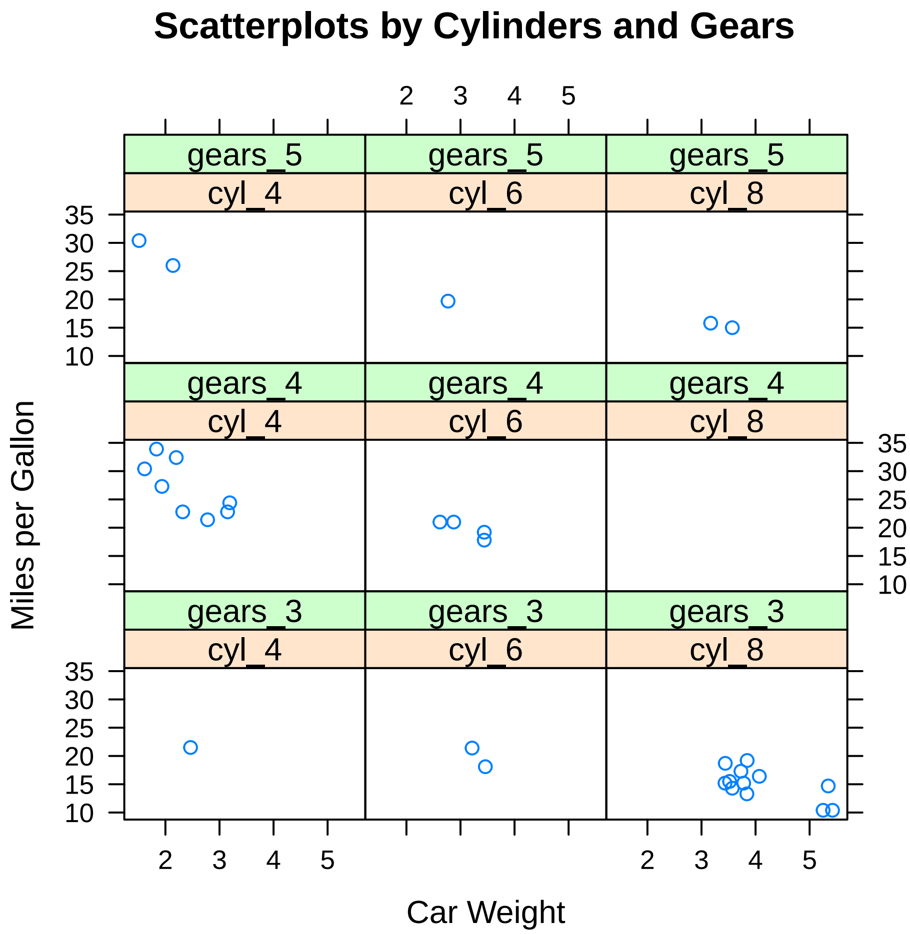

```
# kernel density plots by factor level (alternate layout)
densityplot(~mpg|cyl.factor,
        main="Density Plot by Number of Cylinders",
   xlab="Miles per Gallon",
   layout=c(1,3))
```



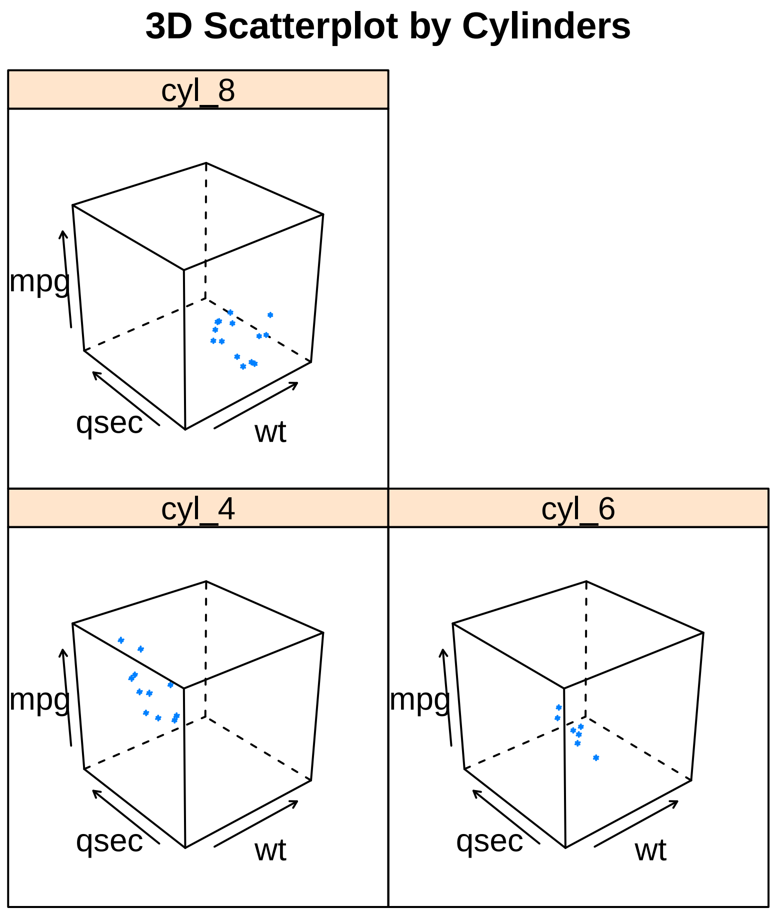

```
# boxplots for each combination of two factors
bwplot(cyl.factor~mpg|gear.factor,
        ylab="Cylinders", xlab="Miles per Gallon",
   main="Mileage by Cylinders and Gears",
   layout=(c(1,3)))
```



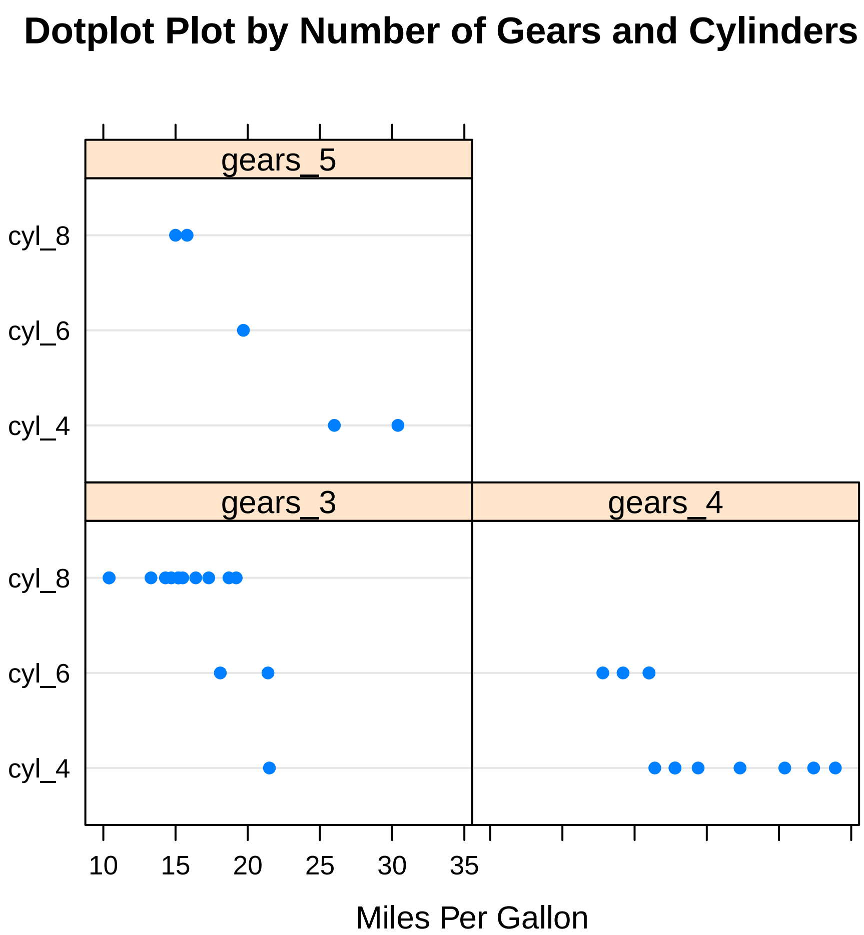

```
# scatterplots for each combination of two factors
xplot(mpg~wt|cyl.factor*gear.factor,
        main="Scatterplots by Cylinders and Gears",
   ylab="Miles per Gallon", xlab="Car Weight")
```



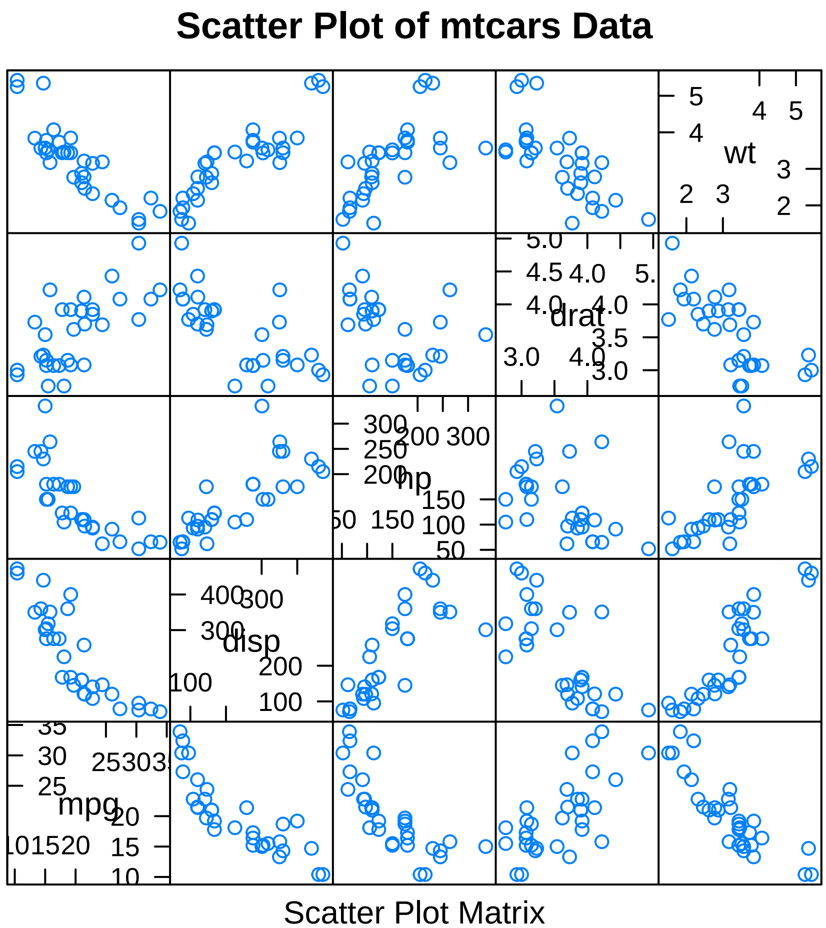

```
# 3d scatterplot by factor level
cloud(mpg~wt*qsec|cyl.factor,
        main="3D Scatterplot by Cylinders")
```




```r
# dotplot for each combination of two factors
dotplot(cyl.factor~mpg|gear.factor,
        main="Dotplot Plot by Number of Gears and Cylinders",
   xlab="Miles Per Gallon")
```




```r
# scatterplot matrix
splom(mtcars[c(1,3,4,5,6)],
      main="Scatter Plot of mtcars Data")
```



2.1.2 Applications of ggplot2

Here we show some applications of ggplot2 package

Advantages of ggplot2

- Plot at high-level of abstraction.
- Very flexible
- Consistent grammar of graphics

Grammar of graphics

- data
- aesthetic mapping
- geometric objects
- scales

- coordinate system
- position adjustments
- statistical transformation

Basic format of ggplot2: ggplot(data = , aes(x =, y =, ...)) + geom_xxx()

- ggplot(): start object and specify the data.
- geom_xxx(): There are many types of geometric objects some are as follow:
  - geom_bar: bars with bases on the x-axis
  - geom_boxplot: boxes-and-whiskers
  - geom_histogram: histogram
  - geom_smooth: smoothed conditional means (e.g. loess smooth)
  - geom_line: lines
  - geom_ribbon: bands spanning y-values across a range of x-values
  - geom_point: points (scatterplot)
  - geom_errorbar: T-shaped error bars
- aes(): specify the aesthetic elements.

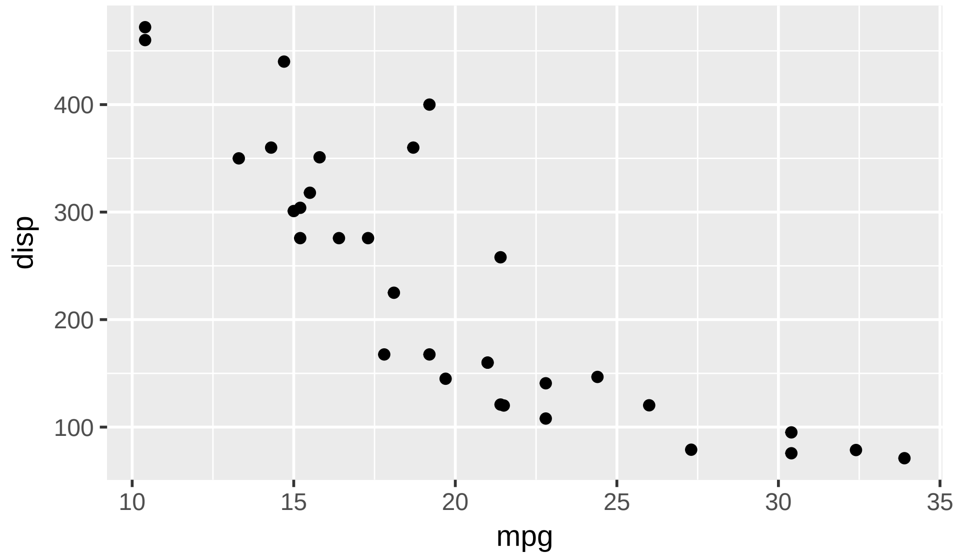
Installing and attaching package ggplot2

Like other package ggplot2 can be installed using function install.package(" ") and can be attached using attach() or library function.
installing package
> install.packages("ggplot2")

```
#attaching library ggplot
library("ggplot2")

## Warning:  package 'ggplot2' was built under R version 3.4.4
##
## Attaching package:  'ggplot2'
## The following object is masked from 'mtcars':
##
##     mpg
```

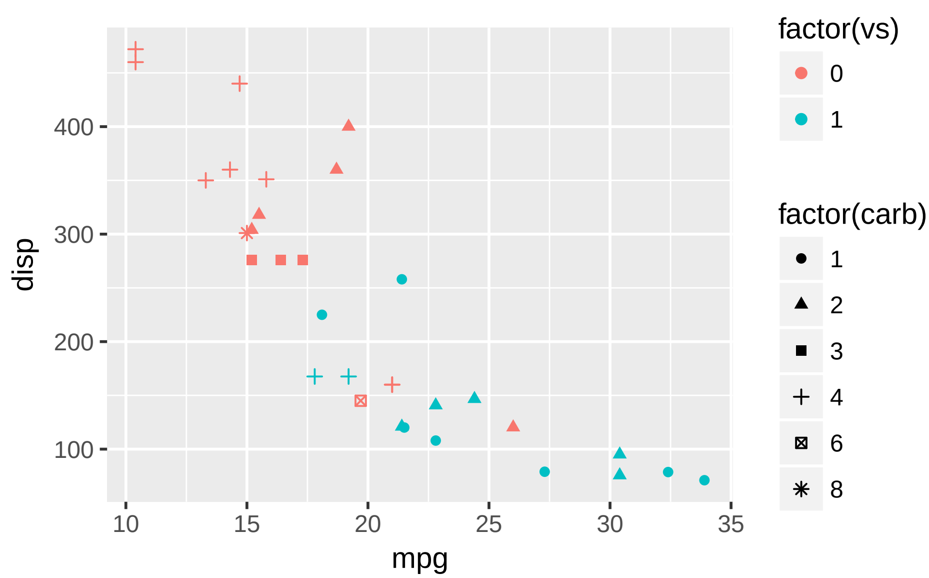
- How to make scatter plot

```
ggplot(mtcars, aes(x=mpg,y=disp))+geom_point()
```



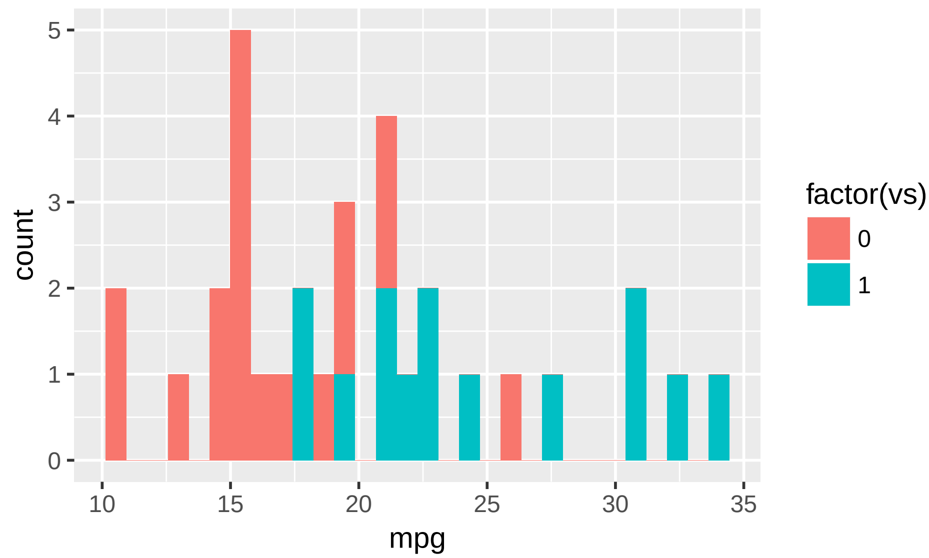
Distinguish groups:To distinguish first change integers into factors. To distinguish we can use two methods either distinguish by color or by shape. We will use both in same graph using two different groups as follow:

```
ggplot(mtcars, aes(x=mpg,y=disp, shape=factor(carb), colour=factor(vs)))+
  geom_point()
```

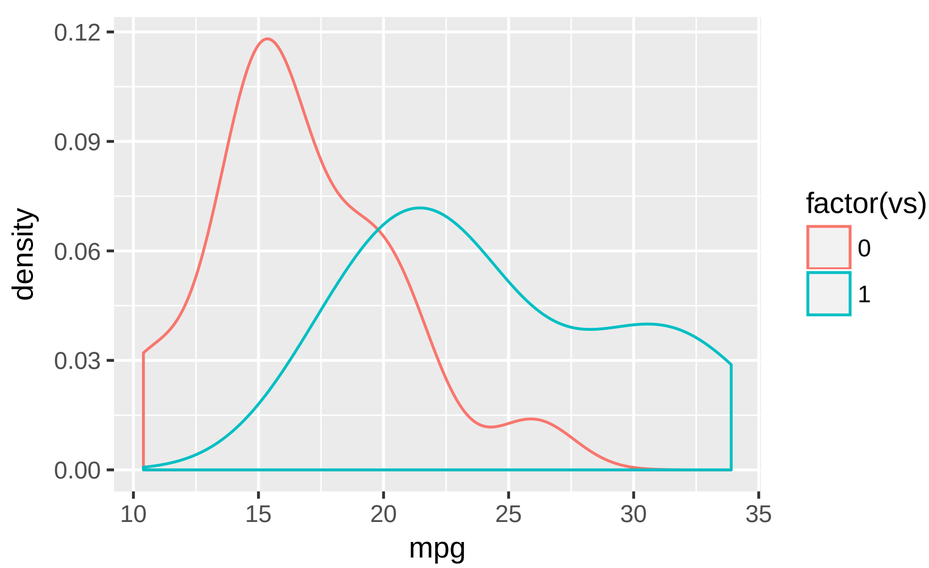


- How to make Histogram, density and boxplot: If we want to plot the distribution of mpg in mtcars and want to show trends by different groups. We can also show density graph by simply adding a geom_density() function as follow:

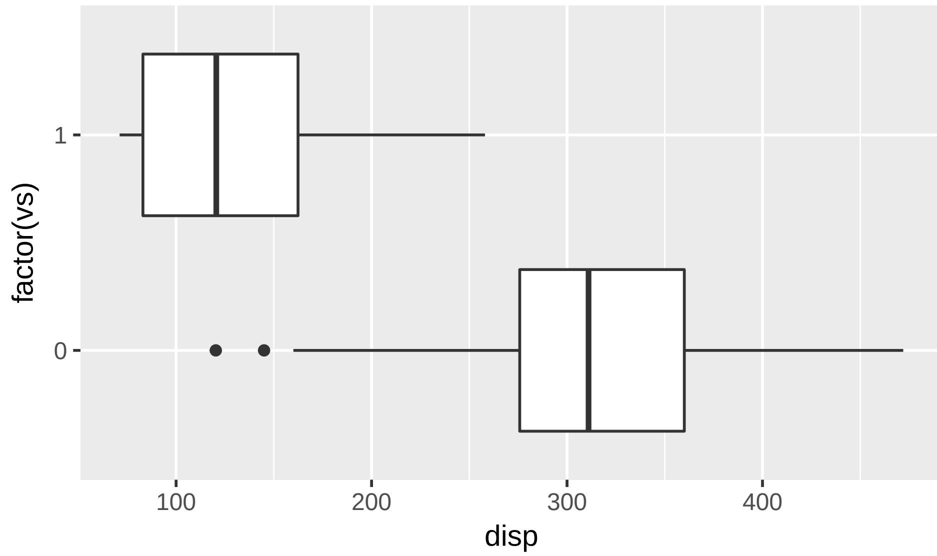
```
ggplot(mtcars, aes(x=mpg, fill=factor(vs)))+geom_histogram()

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(mtcars, aes(x=mpg, colour=factor(vs)))+geom_density()
```



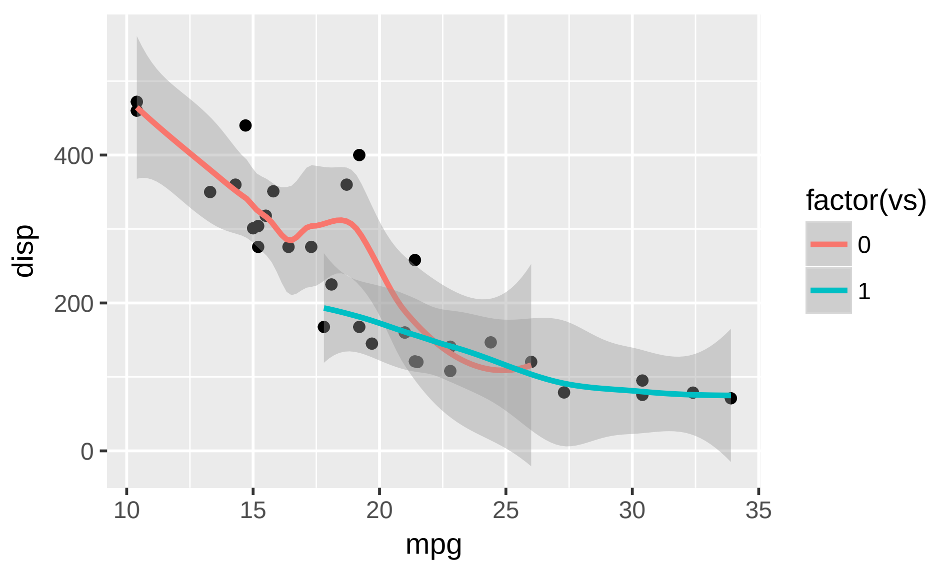

```
ggplot(mtcars, aes(x=factor(vs),y=disp))+geom_boxplot()+coord_flip()
```




```
#coord_flip() function to flip the coordinates
```

- How to make Trend line: Trend line can aid the eye in seeing patterns in the presence of over plotting. Using geom_smooth() to add Trend line to your graph:

```
ggplot(mtcars, aes(x=mpg,y=disp))+geom_point()+
  geom_smooth(aes(colour=factor(vs)))#aes() will show the trend line

## `geom_smooth()` using method = 'loess'
```

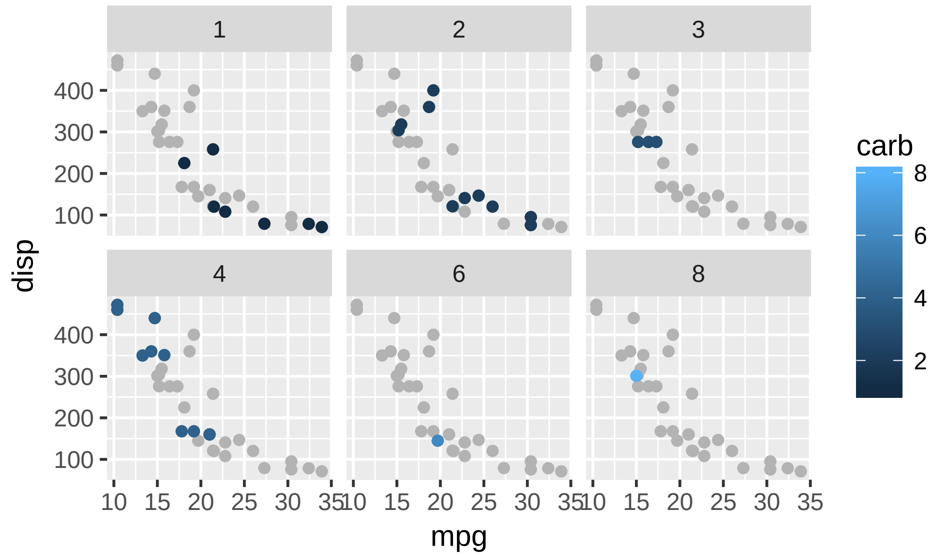


```
#by groups trend line is not necessarily describing the regression
#results of your data. It may be very DIFFERENT from the regression
#line of your model.
```

- Faceting: Faceting generates small multiples each showing a different subset of the data.
  - facet_null(): a single plot, the default.
  - facet_wrap(): "wraps" a 1 dimensional ribbon of panels into 2 dimension, facet_wrap(GI)
  - facet_grid(): produces a 2D grid of panels defined by variables which form the rows and columns. facet_grid(row col)

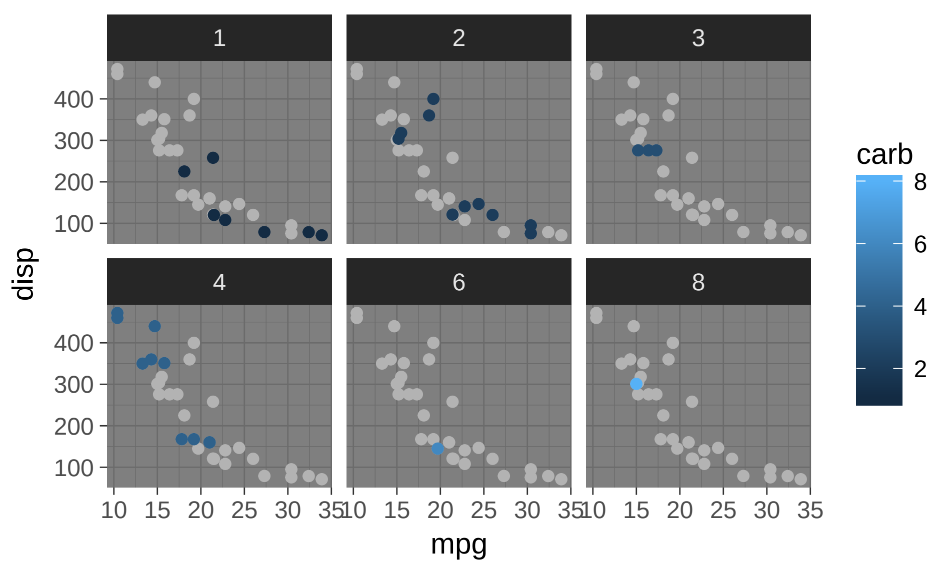
Faceting is an alternative approach to use aesthetics (like color, shape or size) to differentiate groups. It is good when groups overlap a lot, but it make small differences that are harder to observe.

```
library(dplyr)

##
## Attaching package:  'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

mtcars_new<-select(mtcars,-carb)
ggplot(mtcars, aes(x=mpg,y=disp))+geom_point(data=mtcars_new,
                                              colour="grey70")+
  geom_point(aes(colour = carb))+facet_wrap(~carb)
```



- Theme: The theme system in ggplot2 enables a user to control non-data elements of a ggplot object. It makes the ggplot2 a flexible and powerful graphing tool for data visualization.

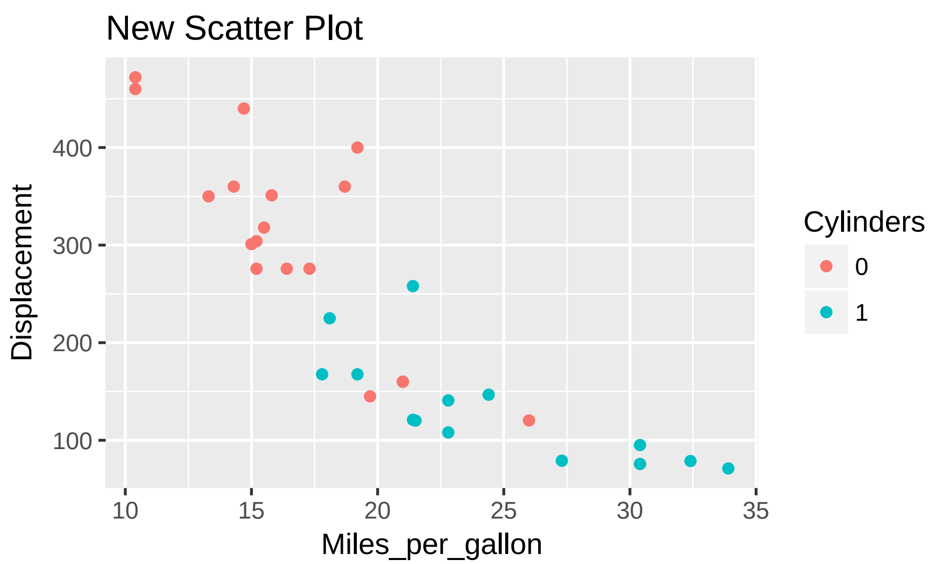
```
ggplot(mtcars, aes(x=mpg,y=disp))+geom_point(data=mtcars_new,
                                             colour="grey70")+
  geom_point(aes(colour = carb))+facet_wrap(~carb)+theme_dark()
```



- Setting axis limits and labeling scales: We commonly need to adjust axis so ggplot2 provides several convenient functions to label axis and adjust axis and other aesthetics:
  - lims, xlim, ylim: set axis limits

- expand_limits: extend limits of scales for various aethetics
- xlab, ylab, ggtitle, labs: give labels (titles) to x-axis, y-axis, or graph; labs can set labels for all aesthetics and title

```
ggplot(mtcars, aes(mpg, disp, colour=factor(vs))) + geom_point()+
  labs(colour ="Cylinders")+labs(x = "Miles_per_gallon",
                                  y="Displacement")+
  ggtitle("New Scatter Plot")
```



2.2 Graphics and Visualization using Python

Python has many visualization tools for building an interactive visualization. It is possible to make beautiful plots for display in python. There are number of other visualization tools in wide use. Few of them are:

- matplotlib
- seaborn
- ggplot
- mayavi
- chaos

But here we'll be focused on two of above that is matplotlib and seaborn.

2.2.1 Applications of matplotlib

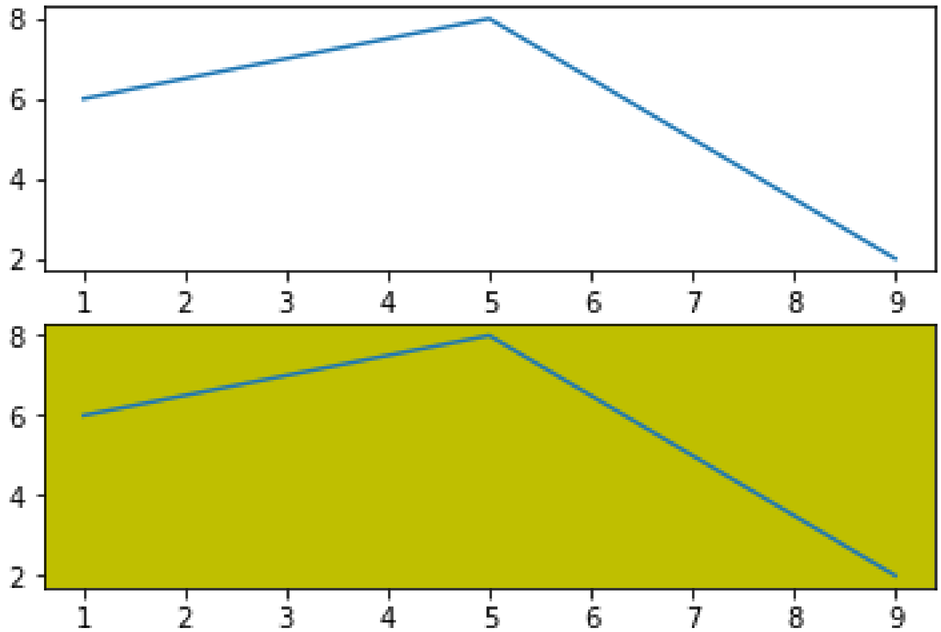
matplotlib is a plotting package designed for creating quality plot. matplotlib was created by John Hunter in 2002 to enable a MATLAB like plotting in python. matplotlib has a number of add-on toolkits, such as mplot3d for 3D plots and basemap for mapping. There are several ways to interact with matplotlib and the most common is through pylab mode in ipython. matplotlib is probably the single most used Python package for 2D-graphics. It provides both a very quick way to visualize data from Python and publication-quality figures in many formats. pyplot provides a convenient interface to the matplotlib object-oriented plotting library. It is modeled closely after Matlab(TM). Firstly we will import (See https://matplotlib.org/) all the library needed by using following commands

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

- Simple plotting a line for two array using plot() function and subplot() function.

```
In [2]: # plot a line, implicitly creating a subplot(111)
        plt.plot([1,2,3],[6,7,8])
        # now create a subplot which represents the top plot of a grid
        # with 2 rows and 1 column. Since this subplot will overlap the
        # first, the plot (and its axes) previously created, will be removed
        plt.subplot(211)
        plt.plot([1,5,9],[6,8,2])
        plt.subplot(212, facecolor='y')
        # creates 2nd subplot with yellow background
        plt.plot([1,5,9],[6,8,2])
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x202a31f93c8>]
```




```
In [3]: data=pd.read_csv("E:/jimmy/J project/mtcars.csv",header=0)
        print(data.head())
```

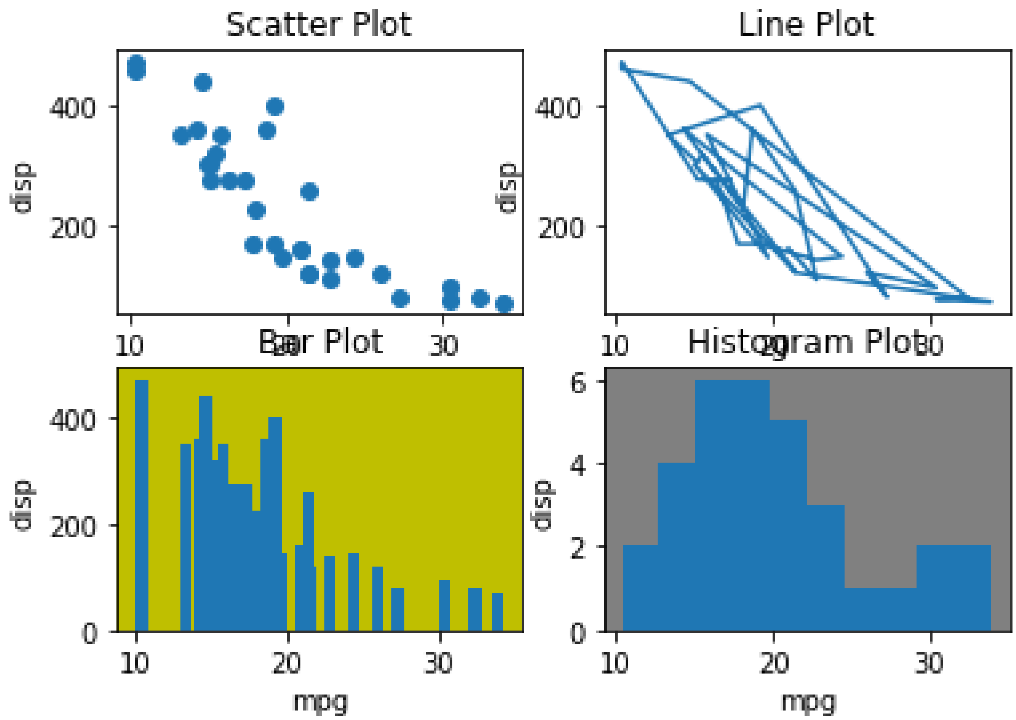
```
          Unnamed: 0   mpg  cyl   disp   hp  drat     wt   qsec  vs  am  gear  \
0          Mazda RX4  21.0    6  160.0  110  3.90  2.620  16.46   0   1     4
1      Mazda RX4 Wag  21.0    6  160.0  110  3.90  2.875  17.02   0   1     4
2         Datsun 710  22.8    4  108.0   93  3.85  2.320  18.61   1   1     4
3     Hornet 4 Drive  21.4    6  258.0  110  3.08  3.215  19.44   1   0     3
4  Hornet Sportabout  18.7    8  360.0  175  3.15  3.440  17.02   0   0     3

   carb
0     4
1     4
2     1
3     1
4     2
```

- Scatter plot, line plot, barplot and histogram in matplotlib:
- scatter() function for plotting scatter plot
- plot() function for plotting line plot
- bar() function for plotting bar plot
- hist() function for histogram plot
- plt.xlabel() function is used to labeling x-axis.

- plt.ylabel() function is used to labeling y-axis.
- plt.title() function is used to give title to graph.

```
In [4]: fig=plt.figure()
        fig.add_subplot(2,2,1)#axis of first graph
        #scatter plot can be drawn using scatter() function
        plt.scatter(data.mpg, data.disp)
        plt.ylabel('disp')
        plt.title('Scatter Plot')
        fig.add_subplot(2,2,2)#axis of second graph
        #line plot can be drawn using plot() function
        plt.plot(data.mpg, data.disp)
        plt.ylabel('disp')
        plt.title('Line Plot')
        fig.add_subplot(2,2,3,facecolor='y')#axis of third graph
        #bar plot can be drawn using bar() function
        plt.bar(data.mpg, data.disp)
        plt.xlabel('mpg')
        plt.ylabel('disp')
        plt.title('Bar Plot')
        fig.add_subplot(2,2,4,facecolor='gray')#axis of second graph
        #bar plot can be drawn using hist() function
        plt.hist(data.mpg)
        plt.xlabel('mpg')
        plt.ylabel('disp')
        plt.title('Histogram Plot')
```

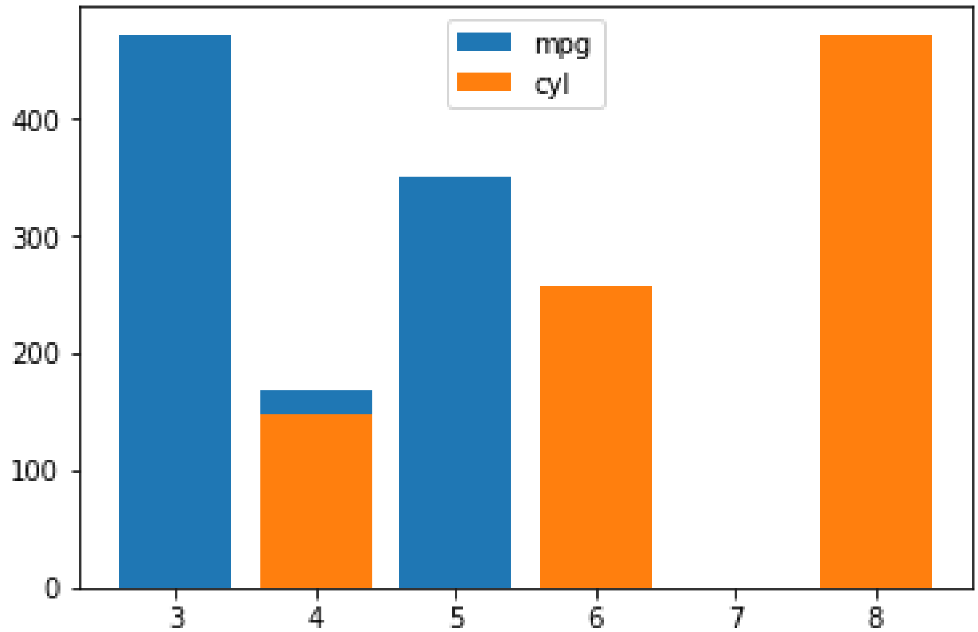
```
Out[4]: Text(0.5,1,'Histogram Plot')
```



- Plotting bar plot to distinguish displacement by different groups.

```
In [5]: plt.bar(data.gear, data.disp, label='mpg')
        plt.bar(data.cyl, data.disp, label='cyl')
        plt.legend()
```

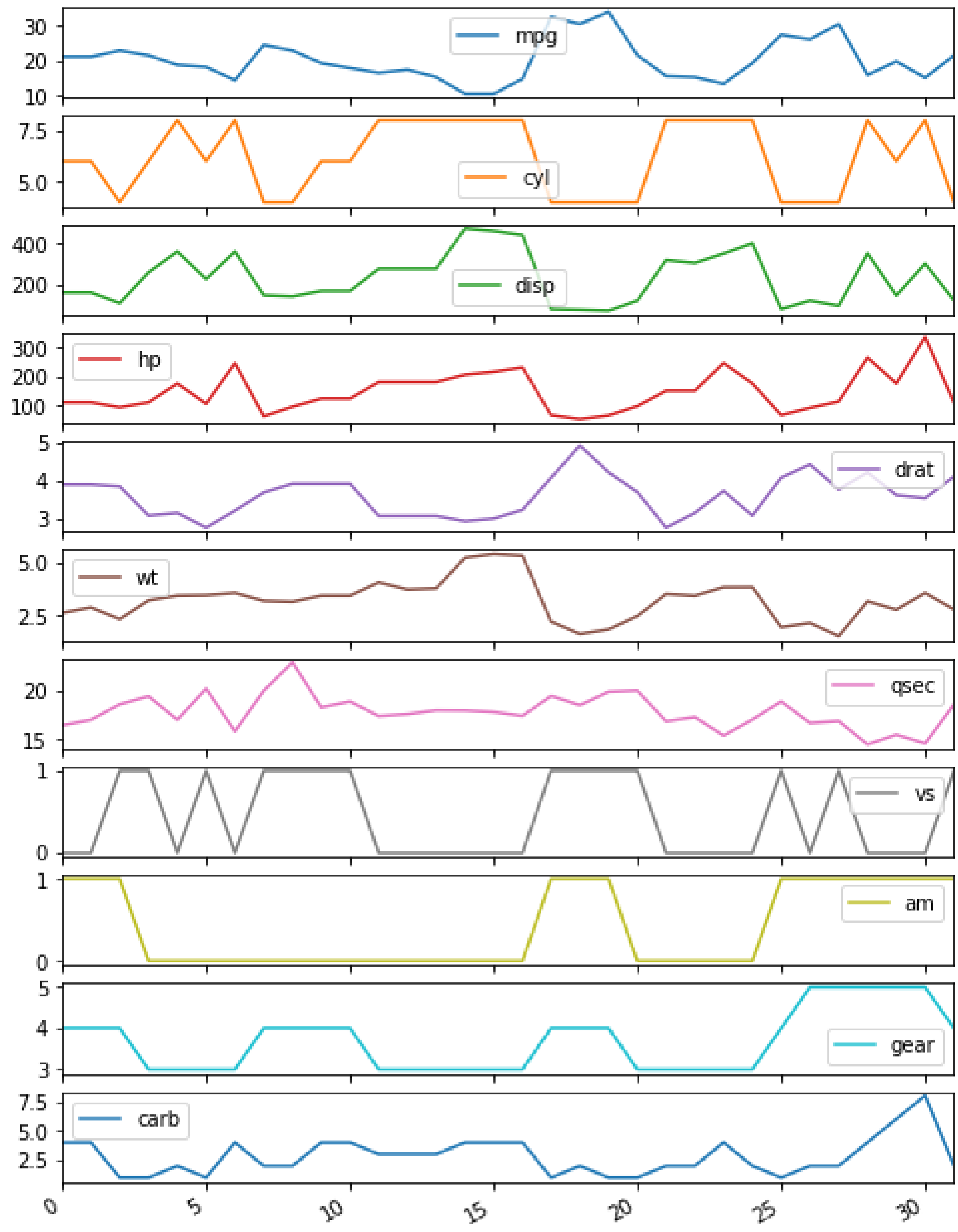
```
Out[5]: <matplotlib.legend.Legend at 0x202a38ccb00>
```



- To plot different pline plot in single command for all the variables

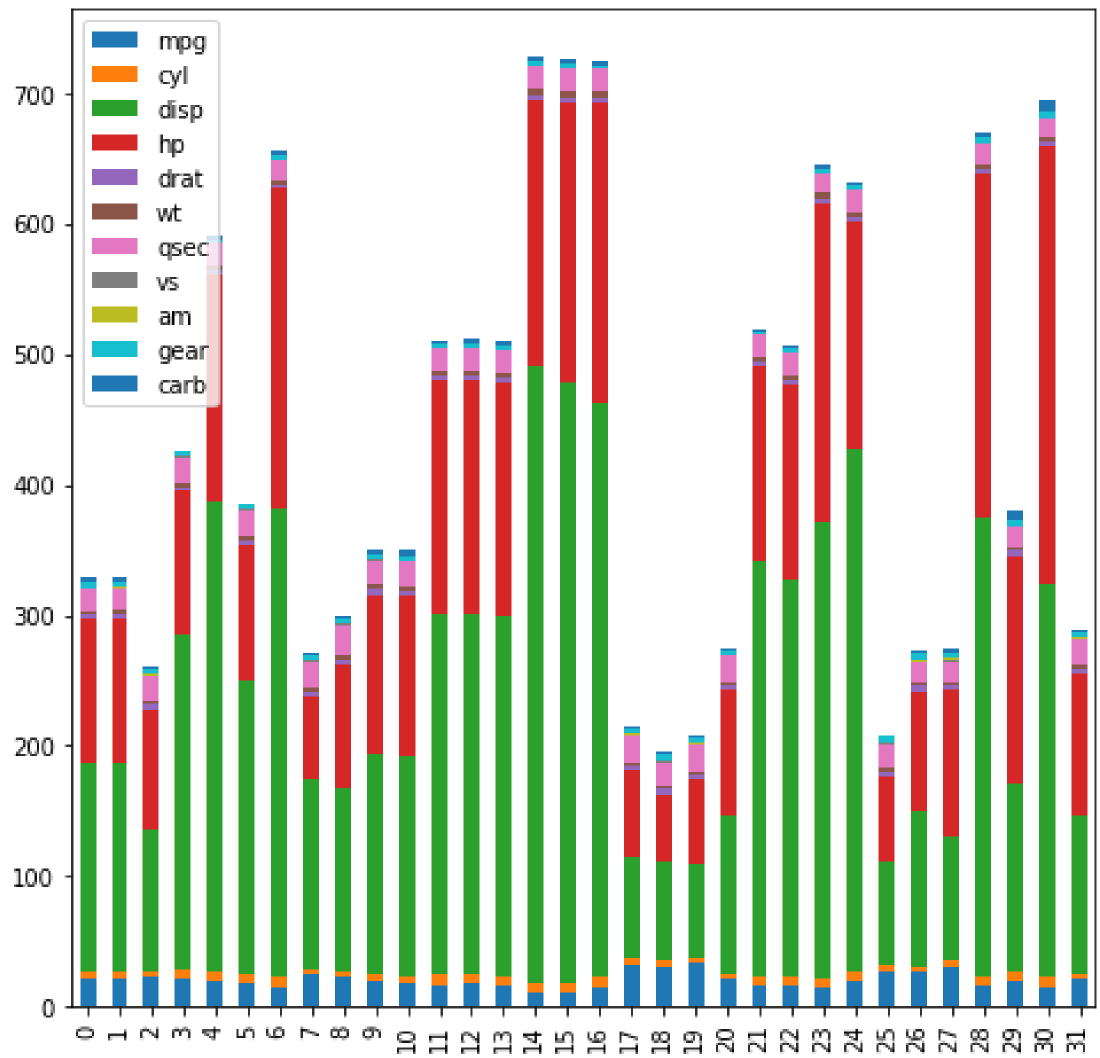
```
In [6]: data.plot(subplots=True, figsize=(8,12)); plt.legend(loc='best')
```

```
Out[6]: <matplotlib.legend.Legend at 0x202a3973278>
```




```
In [7]: data.plot(kind='bar',stacked=True, figsize=(8,8))
```

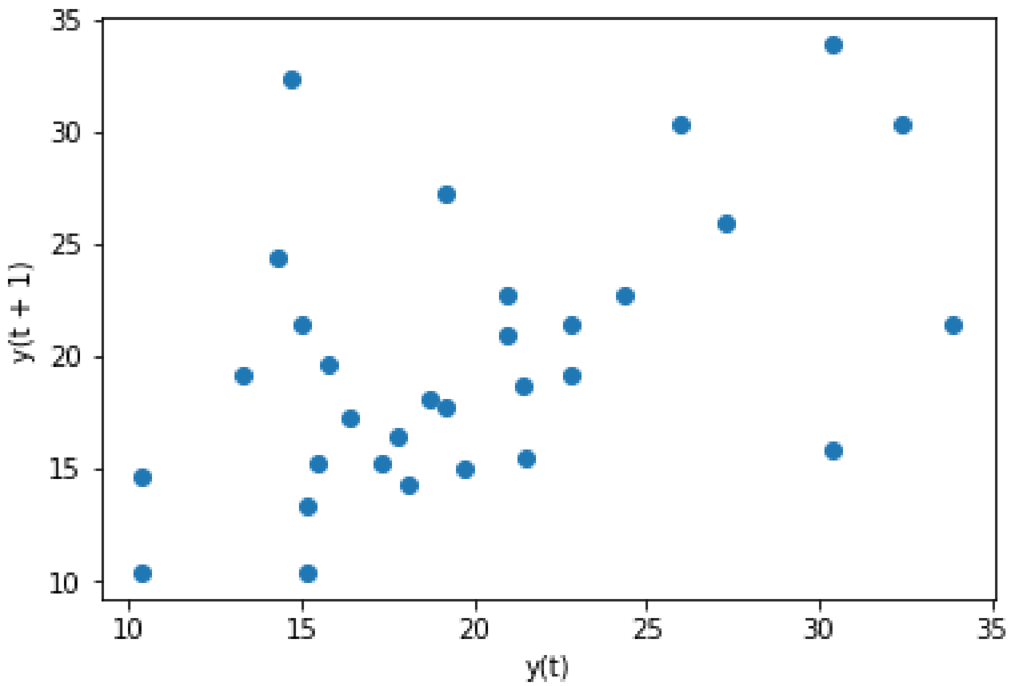
```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x202a4d61828>
```



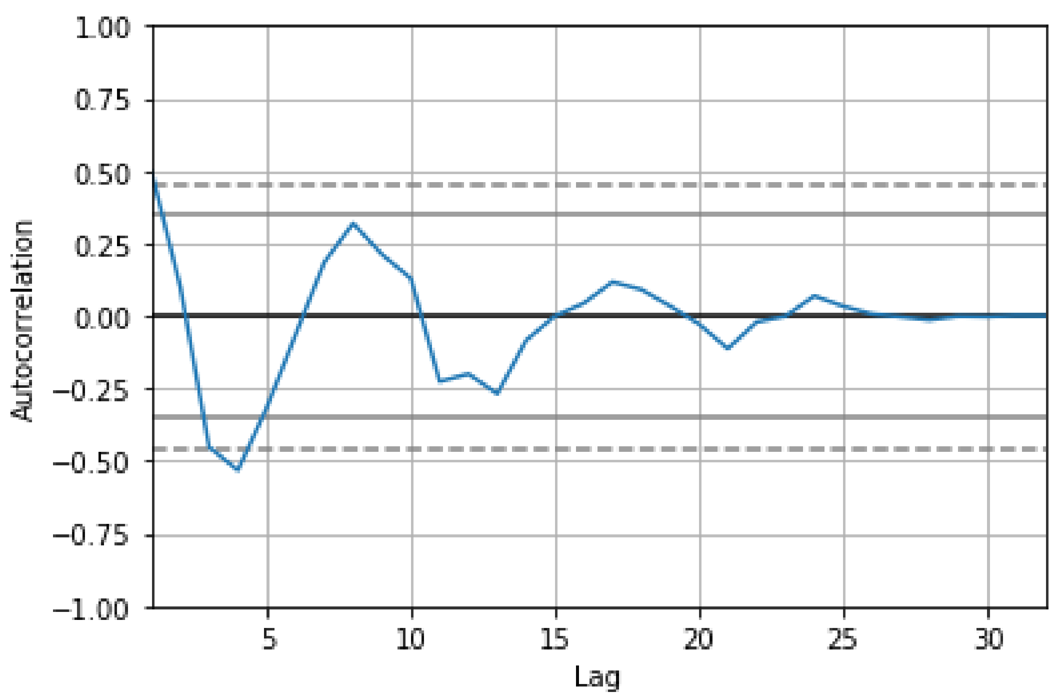
- Auto-correlation plots which are a commonly-used for checking randomness in a data set. This randomness is ascertained by computing auto correlation for data values at varying time lags. If random, such autocorrelation should be near zero for any and all time-lag separations. If non-random, then one or more of the autocorrelation will be significantly non-zero. It can be computed using following functions:

```
In [8]: from pandas.plotting import lag_plot, autocorrelation_plot
        lag_plot(data.mpg)
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x202a5314a58>
```

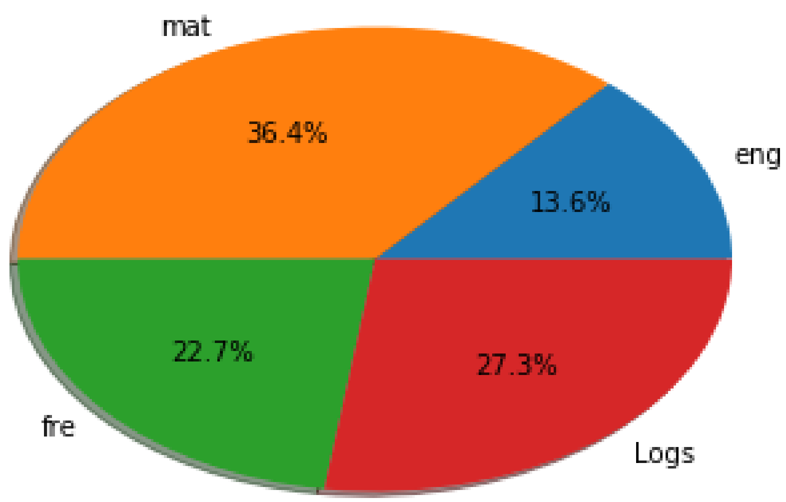



```
In [9]: autocorrelation_plot(data.mpg)

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x202a4df8ef0>
```



- Pie chart: Pie chart can be easily drawn using pie() function.

```
In [10]: labels = 'eng', 'mat', 'fre', 'Logs'
         fracs = [15,40,25,30]
         explode = (0, 0.05, 0, 0)

         plt.pie(fracs, labels=labels, autopct='%1.1f%%', shadow=True)
         #autopct : None (default), string, or function, optional
         #If not None, is a string or function used to label the wedges with their numer
         #The label will be placed inside the wedge. If it is a format string, the label
         #If it is a function, it will be called.
         plt.show()
```



Now using iris dataset to show some more attractive plots in matplotlib

```
In [11]: iris=pd.read_csv("E:/jimmy/J project/Book1.csv",header=0)

In [12]: iris.head()
```

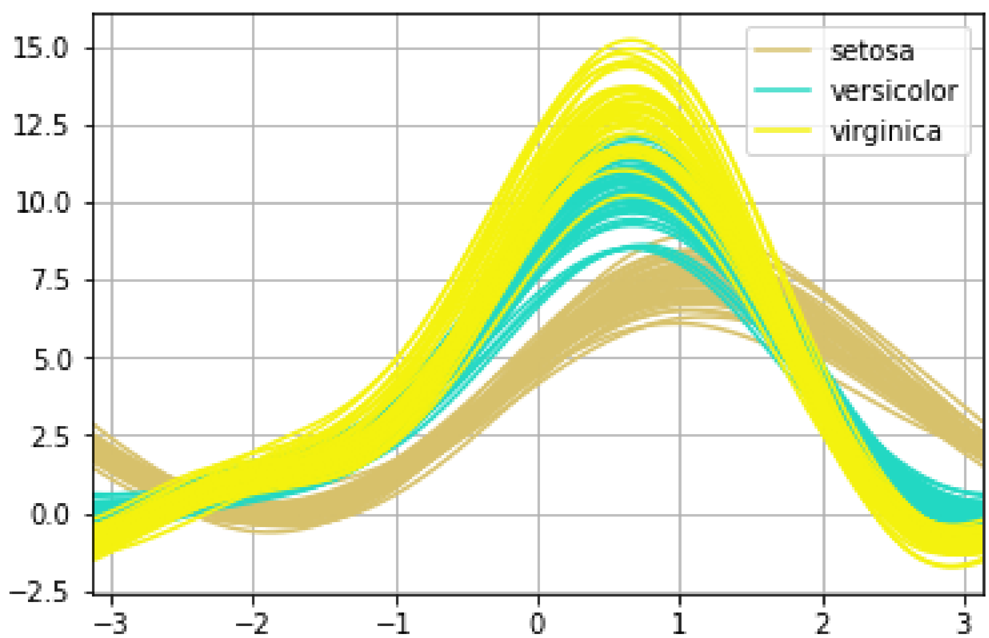
Out[12]:

	sepal_length	sepal_width	petal_length	petal_width	Name
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- Andrews plot: Andrews plot or Andrews curve is a way to visualize structure in high-dimensional data.

```
In [13]: from pandas.plotting import andrews_curves
         plt.figure()
         andrews_curves(iris, 'Name')
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x202a550b6d8>
```

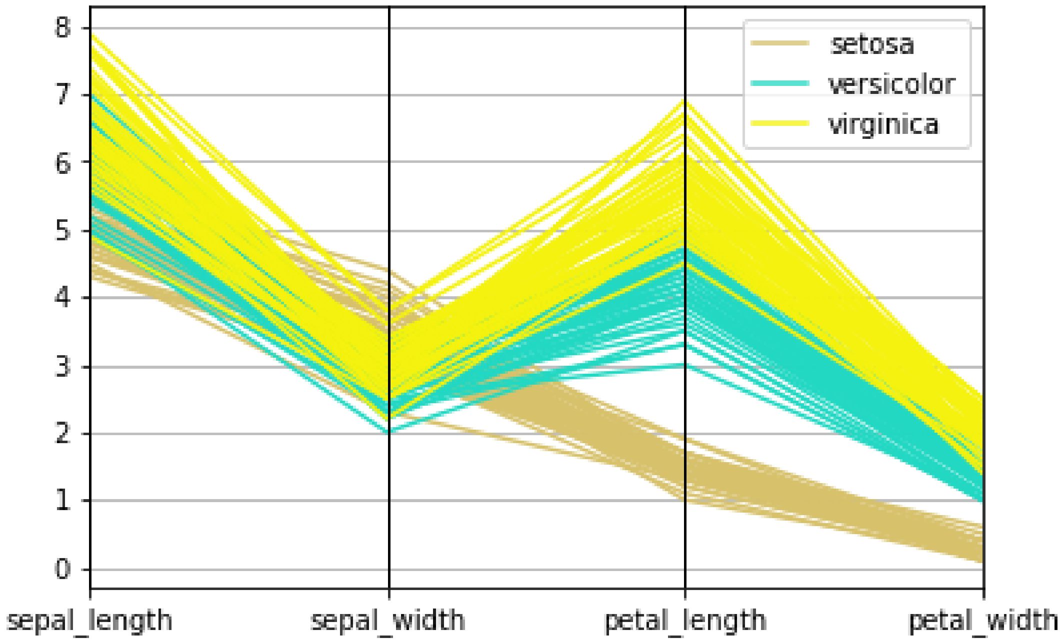


- Parallel coordinate plot: A parallel coordinate plot maps each row in the data table as a line, or profile. Each attribute of a row is represented by a point on the line. This makes parallel coordinate plots similar in appearance to line charts, but the way data is translated into a plot is substantially different.

```
In [14]: from pandas.plotting import parallel_coordinates
```

```
In [15]: plt.figure()
         parallel_coordinates(iris, 'Name')
```

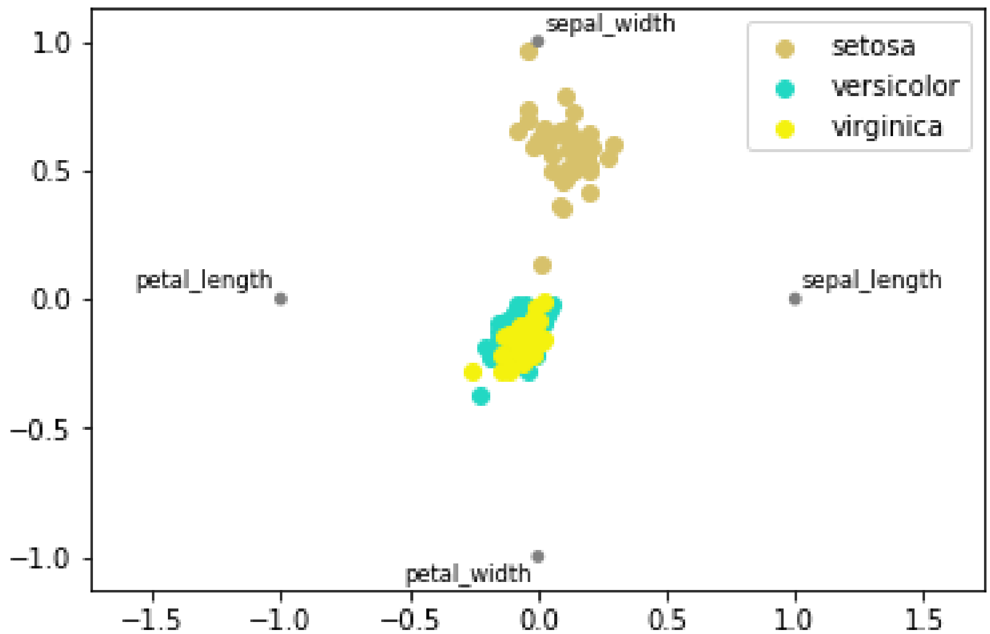
```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x202a56ad518>
```



- RadViz is based on a simple spring tension minimization algorithm. Basically you set up a bunch of points in a plane. In our case, they are equally spaced on a unit circle. Each point represents a single attribute. You then pretend that each sample in the data set is attached to each of these points by a spring, the stiffness of which is proportional to the numerical value of that attribute (they are normalized to unit interval). The point in the plane, where our sample settles to (where the forces acting on our sample are at an equilibrium) is where a dot representing our sample will be drawn. Depending on which class that sample belongs it will be colored differently.

```
In [16]: from pandas.plotting import radviz
         plt.figure()
         radviz(iris,'Name')
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x202a53c2080>
```



2.2.2 Applications of seaborn

Seaborn is a library for making attractive and informative statistical graphics in python. It is built on top of matplotlib and is integrated with pyData(See https://seaborn.pydata.org/). Some of the features that seaborn offers are:

- Several built-in themes for styling matplotlib graphics
- Tools for choosing color palettes to make beautiful plots that reveal patterns in your data
- Functions for visualizing uni-variate and bi-variate distributions for comparing them between subsets of data
- Tools that fit and visualize linear regression models for different kinds of independent and dependent variables
- Functions that visualize matrices of data and use clustering algorithms to discover structure in those matrices
- A function to plot statistical time series data with flexible estimation and representation of uncertainty around the estimate
- High-level abstractions for structuring grids of plots that let you easily build complex visualizations

Chapter 3

Data Cleaning

Most statistical theory focuses on data modelling, prediction and inference, while it is assumed that our data is inaccurate for analysis. Here, inaccurate data is data that is incorrect, incomplete, out-of-date, or wrongly formatted. In practice, it is very rare that raw data one works with is in correct format or without error. Often our data can be quite messy, if we will do direct analysis it might corrupt our analysis. So we need to process our data before doing any further analysis. A data analyst spend its most of the time in doing data cleaning i.e preparing the data for statistical analysis. Data cleaning is process of correcting the errors and transforming raw data into consistent data that can be analysed. R and python provides good environment for data cleaning.

3.1 Why data cleaning is important?

Activity of transforming raw data into consistent data without errors duplicates and inconsistencies i.e.

- Cleaning and transforming data into high quality data.
- To get reliable and unbiased data.
- To get valid, accurate and complete data.

No quality data, No quality decision: Quality decision must be based on good quality data(data with errors may lead to misleading statistics). The ultimate goal of data cleaning is to make inconsistent data get ready for analysis.

3.2 Problem if we don't clean our data

- Inaccurate or biased conclusions: If we don't clean our data then conclusion made on that data may be inaccurate or biased.
- Violation of statistical assumption: Statistical assumption may violate in raw data. Leading to robust conclusions.

3.3 Data cleaning process

3.3.1 ETL process

ETL i.e. extract transform and load

- Extract: Extract data from original data source
- Transform: Manipulate the data into useful format and clean the data.
- Load: Load the data into data warehouse intended for analysis

3.3.2 Extraction of data using R

R offers wide range of packages to import data in any format such as .txt, .csv, .XLXS/ .XLS(Excel)

- Import data from excel We have to use 'readxl' package to access excel files. In excel file first row should contain variable/column names.
  library(readxl)
  $dataset = read\_excel("C : /Users/example2.xls")$
- Import data from csv In csv file first row should contain variable/column names. Where each element is comma separated and header is true. We use command as follow:

  $mydata = read.table("c : /mydata.csv", header = TRUE, sep = ",", row.names = "id")$

3.3.3 Extraction data using Python

To import data in python we use pandas package. Pandas is a powerful data analysis package. It has several function to read data(if you are using Anaconda, Pandas must be pre installed). Data can be in any of the popular formats - CSV, TXT, XLS/XLSX (Excel), sas7bdat (SAS), Rdata (R) etc. To import data in python we use pandas.
using pandas:
Firstly you need to need to load pandas by running command as follow:

import pandas as pd

- To import CSV file import pandas as pd
  $mydata = pd.read\_csv("C : /Users/Documents/file1.csv")$
- To import excel file import pandas as pd
  $mydata = pd.read\_excel("C : /User/file1.xls")$

- To import text file import pandas as pd
$mydata = pd.read\_table("C : /Users/example2.txt")$
$mydata = pd.read\_csv("C : /example2.txt", sep = "\backslash t")$ (if tab separated file)

3.3.4 Transformation

- Rebuild Missing Data: Recreating missing information as and when possible, such as Post codes, states, country, phone area codes, gender, web address from email addresses etc.

- Standardize and Normalize Data: The entries in fields or the categories of the given data set must be homogeneous i.e. all the entries must have the same format for Name, Address, Email, Contact Number, abbreviated/full name of provinces, titles. Moreover, this step ensures that similar information e.g. sir, Mr., Mr are altogether changed over to Mr. Or, on the other hand road, st., strt. are altogether changed over to St.). Convert telephone numbers to their standard format, or as required.

- De-Duplicate data: Identify potential duplicates. Seek high accuracy matches with a tolerance for misspelling, missing values or different address orders. For mission critical data, these results should be manually reviewed and then update the database accordingly.

- Verification to enrich data: Validate the data against internal and external data sources to append value adding info. I.e., business contacts can be validated against yellow pages to verify their current phone number and addresses. Same goes for various other fields including credit ratings, geo-coords, key contacts, employee size, profit, revenue, time zones etc., can be fetched for each company.

3.3.5 Transformation using R

- **Merging Data**: merging two datasets require atleast one variable in common. In R we use merge() function to merge two datasets.

```
mydata1<-data.frame(
   id = c (1:6),
   name = c("Rick","Dan","Michelle","Ryan","Gary","Ryan"),
   salary = c(623.3,515.2,611.0,729.0,843.25,552.1))
mydata2<-data.frame(
   id = c (1:6),
   name = c("Rick","Dan","Michelle","Ryan","Gary","Ryan"),
   age = c(32,31,26,29,36,29))
mydata3<-merge(mydata1,mydata2)
mydata3

##   id     name salary age
```

```
## 1  1     Rick 623.30  32
## 2  2      Dan 515.20  31
## 3  3 Michelle 611.00  26
## 4  4     Ryan 729.00  29
## 5  5     Gary 843.25  36
## 6  6     Ryan 552.10  29
```

By default the data frames are merged on the columns with names they both have, but separate specifications of the columns can be given by by.x and by.y. The rows in the two data frames that match on the specified columns are extracted, and joined together.

```
mydata4<-merge(mydata1,mydata2, by.name="Ryan")
mydata4

##   id     name salary age
## 1  1     Rick 623.30  32
## 2  2      Dan 515.20  31
## 3  3 Michelle 611.00  26
## 4  4     Ryan 729.00  29
## 5  5     Gary 843.25  36
## 6  6     Ryan 552.10  29
```

- **Removing Duplicates**: In R removing duplicates can be done by using unique() function, but another related and interesting function to achieve the same end is duplicated(). dplyr::distinct() : Keep only unique element and is more efficient than unique(). distinct() is best-suited for interactive use.

```
x<-c(0,1,1,1,2,3,6,5,5)
unique(x)

## [1] 0 1 2 3 6 5

df = data.frame(
  A=c("foo", "foo","foo", "foo", "bar"),
  B=c(0,1,1,1,1),
  C=c("A","A","A","B","A"))
print(df)

##     A B C
## 1 foo 0 A
## 2 foo 1 A
```

```
## 3 foo 1 A
## 4 foo 1 B
## 5 bar 1 A

#The dplyr package can be loaded as follow:
# Load
library(dplyr)
#Remove duplicate rows based on all columns:
distinct(df)

##     A B C
## 1 foo 0 A
## 2 foo 1 A
## 3 foo 1 B
## 4 bar 1 A

#Remove duplicate rows based on certain columns (variables):
distinct(df,A)

##     A
## 1 foo
## 2 bar
```

The function distinct() in dplyr package can be used to keep only unique/distinct rows from a data frame. If there are duplicate rows, only the first row is preserved. It's an efficient version of the R base function unique().

- **Missing Observations**: In R, missing values are represented by the symbol NA (not available). Impossible values (e.g., dividing by zero) are represented by the symbol NaN (not a number).
  - **Detecting missing values**: Missing values can be detected using is.na() function.

```
df1 = data.frame(A=c(1,2,3,4,5),
                 B=c('a', 'c', 'e', 'f', 'h'),
                 C=c('one', 'two', NA, 'three',NA))
print(df1)

##   A B     C
## 1 1 a   one
## 2 2 c   two
## 3 3 e  <NA>
## 4 4 f three
## 5 5 h  <NA>
```

```
is.na(df1) # is.na is used to detect which is NA with TRUE or FALSE

##          A     B     C
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE
## [3,] FALSE FALSE  TRUE
## [4,] FALSE FALSE FALSE
## [5,] FALSE FALSE  TRUE

which(is.na(df1))

## [1] 13 15
```

– **Ways to exclude missing values** : Math functions generally have a way to exclude missing values in their calculations. mean(), median(), colSums(), var(), sd(), min() and max() all take the na.rm argument. When this is TRUE, missing values are omitted. The default is FALSE, meaning that each of these functions returns NA if any input number is NA.
  More functions used to exclude missing values. If you have large number of observations in your dataset, then try deleting (or not to include missing values while model building, for example by setting na.action=na.omit) those observations (rows) that contain missing values.
  * na.omit: Drop out any rows with missing values anywhere in them and forgets them forever.
  * na.exclude: Drop out rows with missing values, but keeps track of where they were (so that when you make predictions, for example, you end up with a vector whose length is that of the original response.)
  * na.pass: returns the object unchanged
  * na.fail: returns the object only if it contains no missing values

```
x <- c(1,2,NA,3)
mean(x) # returns NA

## [1] NA

mean(x, na.rm=TRUE) #return 2

## [1] 2

na.omit(df1) # Drop out rows with missing values

##   A B     C
## 1 1 a   one
## 2 2 c   two
## 4 4 f three
```

A couple of other packages supply more efficient results:

– Hmisc library can be used to replace missing values with mean, median and mode

```
df2 = data.frame(A=c(1,2,3,4,5,NA),
                 B=c(0.5,0.8,1.2,NA,0.1,1.5),
                 C=c(15,26,NA,12,NA,NA))
df2

##    A   B  C
## 1  1 0.5 15
## 2  2 0.8 26
## 3  3 1.2 NA
## 4  4  NA 12
## 5  5 0.1 NA
## 6 NA 1.5 NA

library(Hmisc)

## Loading required package:  survival
## Loading required package:  Formula
##
## Attaching package:  'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##    src, summarize
## The following objects are masked from 'package:base':
##
##    format.pval, units

impute(df2$A, mean)  # replace with mean

##  1  2  3  4  5  6
##  1  2  3  4  5 3*

impute(df2$B, median)  # replace with median

##    1    2    3    4    5    6
##  0.5  0.8  1.2 0.8*  0.1  1.5

impute(df2$C,0)  # replace specific number

##   1   2   3   4   5   6
##  15  26  0*  12  0*  0*
```

3.3.6 Transformation using Python

- **Merging data**: Merge or Join operation will combine data sets by linking rows using one or more keys. few merge function arguments
  - left: DataFrame to be merged on the left side.
  - right: DataFrame to be merged on the right side.
  - on: Column names to join on. Must be found on both DataFrame objects.
  - left on: Columns in left DataFrame to use as a join keys.
  - right on: Columns in right DataFrame to use as a join keys.

```
In [10]: from pandas import Series, DataFrame
         import pandas as pd
```

```
In [11]: df_1=DataFrame({'key': ['b','b','a','c','a','a','b'],
                        'd':['0','1','2','3','4','5','6']})
         print(df_1)
```

```
   d key
0  0   b
1  1   b
2  2   a
3  3   c
4  4   a
5  5   a
6  6   b
```

```
In [12]: df_2=DataFrame({'key': ['a','b','d'],
                        'd2':['0','1','2']})
         print(df_2)
```

```
  d2 key
0  0   a
1  1   b
2  2   d
```

```
In [13]: pd.merge(df_1,df_2)
```

```
Out[13]:    d key d2
         0  0   b  1
         1  1   b  1
         2  6   b  1
         3  2   a  0
         4  4   a  0
         5  5   a  0
```

```
In [14]: df_3=DataFrame({'lkey':['b','b','a','c','a','a','b'],
                         'data1':range(7)})
         print(df_3)

   data1 lkey
0      0    b
1      1    b
2      2    a
3      3    c
4      4    a
5      5    a
6      6    b


In [15]: df_4=DataFrame({'rkey':['a','b','d'],
                         'data2': range(3)})
         print(df_3)

   data1 lkey
0      0    b
1      1    b
2      2    a
3      3    c
4      4    a
5      5    a
6      6    b


In [16]: pd.merge(df_3,df_4,left_on='lkey',right_on='rkey',how='outer')

Out[16]:    data1 lkey  data2 rkey
         0    0.0    b    1.0    b
         1    1.0    b    1.0    b
         2    6.0    b    1.0    b
         3    2.0    a    0.0    a
         4    4.0    a    0.0    a
         5    5.0    a    0.0    a
         6    3.0    c    NaN  NaN
         7    NaN  NaN    2.0    d
```

- **Removing Duplicates**: Duplicate rows may be found in DataFrame for number of reasons. In python removing duplicates can be done using drop_duplicates(). DataFrame.drop_duplicates() return DataFrame with duplicate rows removed, optionally only considering certain columns

```
In [8]: import pandas as pd
        df = pd.DataFrame({"A":["foo", "foo", "foo", "bar"],
                            "B":[0,1,1,1], "C":["A","A","B","A"]})
        print(df)

     A  B  C
0  foo  0  A
1  foo  1  A
2  foo  1  B
3  bar  1  A
```

```
In [9]: df.drop_duplicates(subset=['A', 'C'], keep=False)

Out[9]:      A  B  C
        2  foo  1  B
        3  bar  1  A
```

where **keep** : {'first', 'last', False}, default 'first'

– first : Drop duplicates except for the first occurrence.
– last : Drop duplicates except for the last occurrence.
– False : Drop all duplicates.

- **Missing Observations**: By "missing" we simply mean NA ("not available") or "not present for whatever reason". Many data sets simply arrive with missing data, either because it exists and was not collected or it never existed. In pandas, one of the most common ways that missing data is introduced into a data set is by re-indexing

```
In [28]: import pandas as pd
         import numpy as np
         df = pd.DataFrame(np.random.randn(5, 3),
                           index=['a', 'c', 'e', 'f', 'h'],
                           columns=['one', 'two', 'three'])
         df['four'] = 'bar'
         df['five'] = df['one'] > 0
         print(df)

        one       two     three four   five
a -0.336057  0.512864 -0.854062  bar  False
c -0.424267 -0.101321  0.948349  bar  False
e  0.957720 -0.602851  0.859344  bar   True
f  0.734610 -0.769397  0.355850  bar   True
h -1.733099 -0.451442 -0.785071  bar  False
```

```
In [29]: df2 = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
         print(df2)

        one       two     three four   five
a -0.336057  0.512864 -0.854062  bar  False
b       NaN       NaN       NaN  NaN    NaN
c -0.424267 -0.101321  0.948349  bar  False
d       NaN       NaN       NaN  NaN    NaN
e  0.957720 -0.602851  0.859344  bar   True
f  0.734610 -0.769397  0.355850  bar   True
g       NaN       NaN       NaN  NaN    NaN
h -1.733099 -0.451442 -0.785071  bar  False
```

pandas objects are equipped with various data manipulation methods for dealing with missing data.

- Filling missing values: fillna
  * The fillna function can "fill in" NA values with non-NA data in a couple of ways, which we illustrate:
  * Replace NA with a scalar value
  * Fill gaps forward or backward

```
In [30]: df2.fillna(0)

Out[30]:        one       two     three four   five
         a -0.336057  0.512864 -0.854062  bar  False
         b  0.000000  0.000000  0.000000    0      0
         c -0.424267 -0.101321  0.948349  bar  False
         d  0.000000  0.000000  0.000000    0      0
         e  0.957720 -0.602851  0.859344  bar   True
         f  0.734610 -0.769397  0.355850  bar   True
         g  0.000000  0.000000  0.000000    0      0
         h -1.733099 -0.451442 -0.785071  bar  False

In [31]: df.fillna(method='pad')

Out[31]:        one       two     three four   five
         a -0.336057  0.512864 -0.854062  bar  False
         c -0.424267 -0.101321  0.948349  bar  False
         e  0.957720 -0.602851  0.859344  bar   True
         f  0.734610 -0.769397  0.355850  bar   True
         h -1.733099 -0.451442 -0.785071  bar  False
```

To remind you, these are the available filling methods:

- pad / ffill:Fill values forward.
- bfill / backfill:Fill values backward.

You may wish to simply exclude labels from a data set which refer to missing data. To do this, use the dropna method:

```
In [36]: df2.dropna(axis=0)

Out[36]:         one       two     three four   five
         a -0.336057  0.512864 -0.854062  bar  False
         c -0.424267 -0.101321  0.948349  bar  False
         e  0.957720 -0.602851  0.859344  bar   True
         f  0.734610 -0.769397  0.355850  bar   True
         h -1.733099 -0.451442 -0.785071  bar  False
```

You can also fillna using a dict or Series that is align-able. The labels of the dict or index of the Series must match the columns of the frame you wish to fill. The use case of this is to fill a DataFrame with the mean of that column.

```
In [38]: df2.fillna(df2.mean())

Out[38]:         one       two     three four   five
         a -0.336057  0.512864 -0.854062  bar  False
         b -0.160219 -0.282429  0.104882  NaN    0.4
         c -0.424267 -0.101321  0.948349  bar  False
         d -0.160219 -0.282429  0.104882  NaN    0.4
         e  0.957720 -0.602851  0.859344  bar   True
         f  0.734610 -0.769397  0.355850  bar   True
         g -0.160219 -0.282429  0.104882  NaN    0.4
         h -1.733099 -0.451442 -0.785071  bar  False
```

Chapter 4

Explanatory Data Analysis

Explanatory Data Analysis (EDA) is an approach to data analysis. EDA is a critical step in analyzing data. Its where the experimenter takes the bird's eye of the data and tries to make some sense of it. Exploratory Data Analysis takes place after gathering and cleaning data, and is often implemented before any formal statistical technique is applied. Among the main purposes of this type of analysis are of course getting to know our data, its tendencies and its quality, and also to check or even start formulating our hypothesis. Here are some reasons why we use EDA:

- Detection of mistakes.
- Gain maximum insight into the dataset and its underlying structure.
- Determining relationships among the explanatory variables.
- Check assumptions associated with any model fitting or hypothesis test.
- Detection of outliers

Most EDA techniques are graphical in nature with a few quantitative techniques. The reason for the heavy reliance on graphics is that by its nature the main role of EDA is to open-mindedly explore. The particular graphical techniques employed in EDA are often quite simple consisting of various techniques of:

- Plotting the raw data (such as data traces, histograms, bihistograms, probability plots, lag plots, block plots, and Youden plots.
- Plotting simple statistics such as mean plots, standard deviation plots, box plots, and main effects plots of the raw data.

Types of Exploratory Data Analysis:
EDA falls into four main areas:

- Univariate EDA- Looking at one variable of interest, like age, height, income level etc.
- Multivariate EDA- Analysis of multiple variables at the same time.

4.1 Univarate Explanatory Data Analysis

In univariate EDA our interest is analyzing each variable, like age, gender, income etc. The usual goal of univariate EDA is to better appreciate the "sample distribution". Outlier detection is also a part of this analysis. Below are some techniques used in univariate EDA:

- Summary statistics: Summary statistics summarize and provide information about your sample data. It includes where the average lies and whether your data is skewed. Summary statistics fall into three main categories:

  – Measures of location and central tendency(e.g. mean, median, mode etc.).

  – Measure of dispersion(e.g. Standard deviation)

  – Measures of shape(e.g. skewness and kurtosis)

  A common collection of statistics used as summary statistics are the five-number summary i.e. the minimum, 25th percentile, median, 75th percentile, maximum of the data

- Histogram: The purpose of a histogram is to graphically summarize the distribution of a univariate data set. The histogram graphically shows the following:

  – Center (i.e., the location) of the data.

  – Spread (i.e., the scale) of the data.

  – Skewness of the data.

  – Presence of outliers.

  – Presence of multiple modes in the data.

- Stem and leaf plots: A simple substitute to histogram is stem and leaf plot. Nevertheless, a histogram is generally considered better for estimating the shape of a sample distribution than the stem and leaf plot.

- Boxplots: Boxplot is visualization of five-number summary with more information. Boxplot graphically shows the following:

  – Displays variable's location and spread.

  – Provide indication of data symmetry and skewness.

  – Shows outliers

- Density plot: A Density Plot visualizes the distribution of data over a continuous interval or time period.

4.2 Multivariate Explanatory Data Analysis

Multivariate EDA techniques generally show the relationship between two or more variables in the form of either cross-tabulation or statistics. Below are some techniques used in univariate EDA:

- Correlation matrix: Correlation matrix measures degree of relationship between the variables under consideration. The degree of relationship is expressed by coefficient which range from correlation ($-1 \leq r \leq +1$). It deals with the association between two or more variables.

- Scatter plot: Scatter plot is a diagrammatic representation of bivariate data. It is used to plot data points on a horizontal and a vertical axis in the attempt to show how much one variable is affected by another. Scatter plots are important in statistics because they can show the extent of correlation. Besides showing the extent of correlation, a scatter plot shows the sense of the correlation:
  - If the vertical (or y-axis) variable increases as the horizontal (or x-axis) variable increases, the correlation is positive.
  - If the y-axis variable decreases as the x-axis variable increases or vice-versa, the correlation is negative.
  - If it is impossible to establish either of the above criteria, then the correlation is zero.

- Multiple Boxplot: Unlike regular box plots in which the range of values of one variable is represented, the multiple box plot represents ranges of values of multiple variables. Multiple Boxplot can be used to visualize multiple variables together. It can be used for comparing two or more variables.

- Multiple histogram: A panel of histograms enables you to compare the data distributions of different groups. You can create the histograms in a column (stacked vertically) or in a row.

4.3 Explanatory Data Analysis using R

We will use a popular dataset in R library "dataset" and the dataset used is cars.
Description: The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.
Data is imported using read.csv() function of pandas module.

```
data<-read.csv("E:/jimmy/J project/cars.csv", header = T,sep = ',')
head(data)

##   speed dist
## 1     4    2
```

```
## 2      4   10
## 3      7    4
## 4      7   22
## 5      8   16
## 6      9   10
```

We can see some basic characteristics of the dataset using dim(), str(), names(), head(), tail(), summary() functions.

```
dim(data)

## [1] 50  2

str(data)

## 'data.frame': 50 obs. of  2 variables:
##  $ speed: int  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : int  2 10 4 22 16 10 18 26 34 17 ...

names(data)

## [1] "speed" "dist"

head(data)

##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10

tail(data)

##    speed dist
## 45    23   54
## 46    24   70
## 47    24   92
## 48    24   93
## 49    24  120
## 50    25   85

summary(data)

##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

In R we are using dplyr package for doing EDA.
Some of the key "verbs" provided by the dplyr package are

- select: return a subset of the columns of a data frame, using a flexible notation
- filter: extract a subset of rows from a data frame based on logical conditions
- arrange: reorder rows of a data frame
- rename: rename variables in a data frame
- mutate: add new variables/columns or transform existing variables
- summaries / summarize: generate summary statistics of different variables in the data frame, possibly within strata

Installing the dplyr package
The dplyr package can be installed from CRAN. To install from CRAN, just run
> install.packages("dplyr")

```
#After installing the package it is important to load it into your R session
library(dplyr)
```

The arrange() function is used to reorder rows of a data frame according to one of the variables/columns.

```
head(arrange(data,speed))

##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

The select() function can be used to select columns of a data frame that you want to focus on. Often you'll have a large data frame containing "all" of the data, but any given analysis might only use a subset of variables or observations.

```
head(select(data,speed))

##   speed
## 1     4
## 2     4
## 3     7
## 4     7
## 5     8
## 6     9
```

```
tail(select(data,speed))

##    speed
## 45    23
## 46    24
## 47    24
## 48    24
## 49    24
## 50    25
```

Renaming a variable in a data frame in R is surprisingly hard to do! The rename() function is designed to make this process easier.

```
d=head(rename(data,velocity=speed))
d

##   velocity dist
## 1        4    2
## 2        4   10
## 3        7    4
## 4        7   22
## 5        8   16
## 6        9   10
```

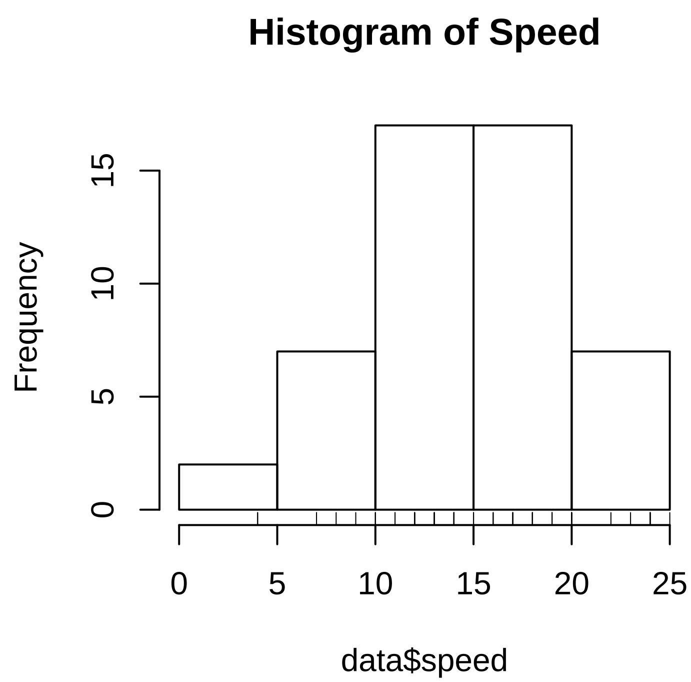
Univariate Explanatory data analysis

- five-number summary: five-number summary can be computed using fivenum() function. It's often a bit nice to use the summary() function.

```
fivenum(data$speed)

## [1]  4 12 15 19 25

summary(data$speed)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    15.0    15.4    19.0    25.0
```

- Histogram: Histogram can be drawn using hist() function. We can get a little more detail by using the rug() function to show us the actual data points.

```
hist(data$speed, main="Histogram of Speed")
rug(data$speed)
```

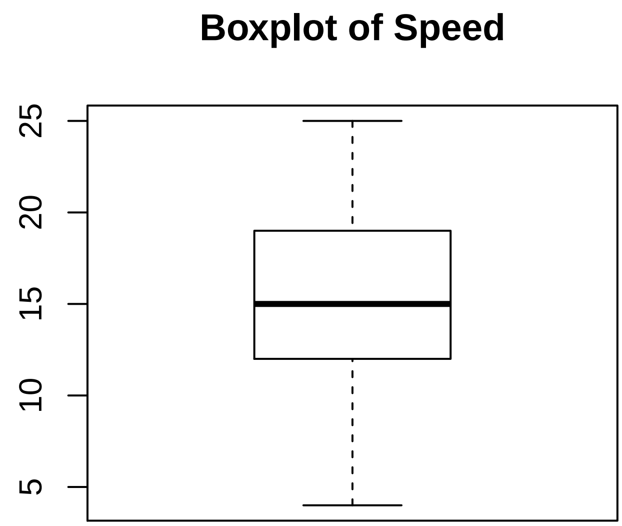


- Stem-leaf plot: Stem-leaf plot can be drawn using stem() function.

```
stem(data$speed)

##
##   The decimal point is at the |
##
##    4 | 00
##    6 | 00
##    8 | 00
##   10 | 00000
##   12 | 00000000
##   14 | 0000000
##   16 | 00000
##   18 | 0000000
##   20 | 00000
##   22 | 00
##   24 | 00000
```

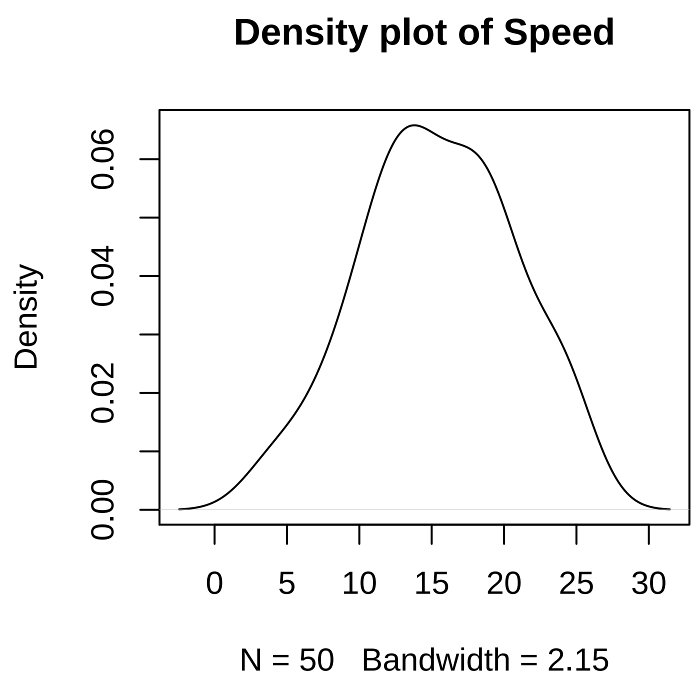
- Box plot: Box-plot can be drawn using boxplot() function.

```
boxplot(data$speed, main="Boxplot of Speed")
```



- Density plot: Density plot can be drawn using plot(density()) where density() function return the density data and plot() function returns the result.

```
plot(density(data$speed), main="Density plot of Speed")
```



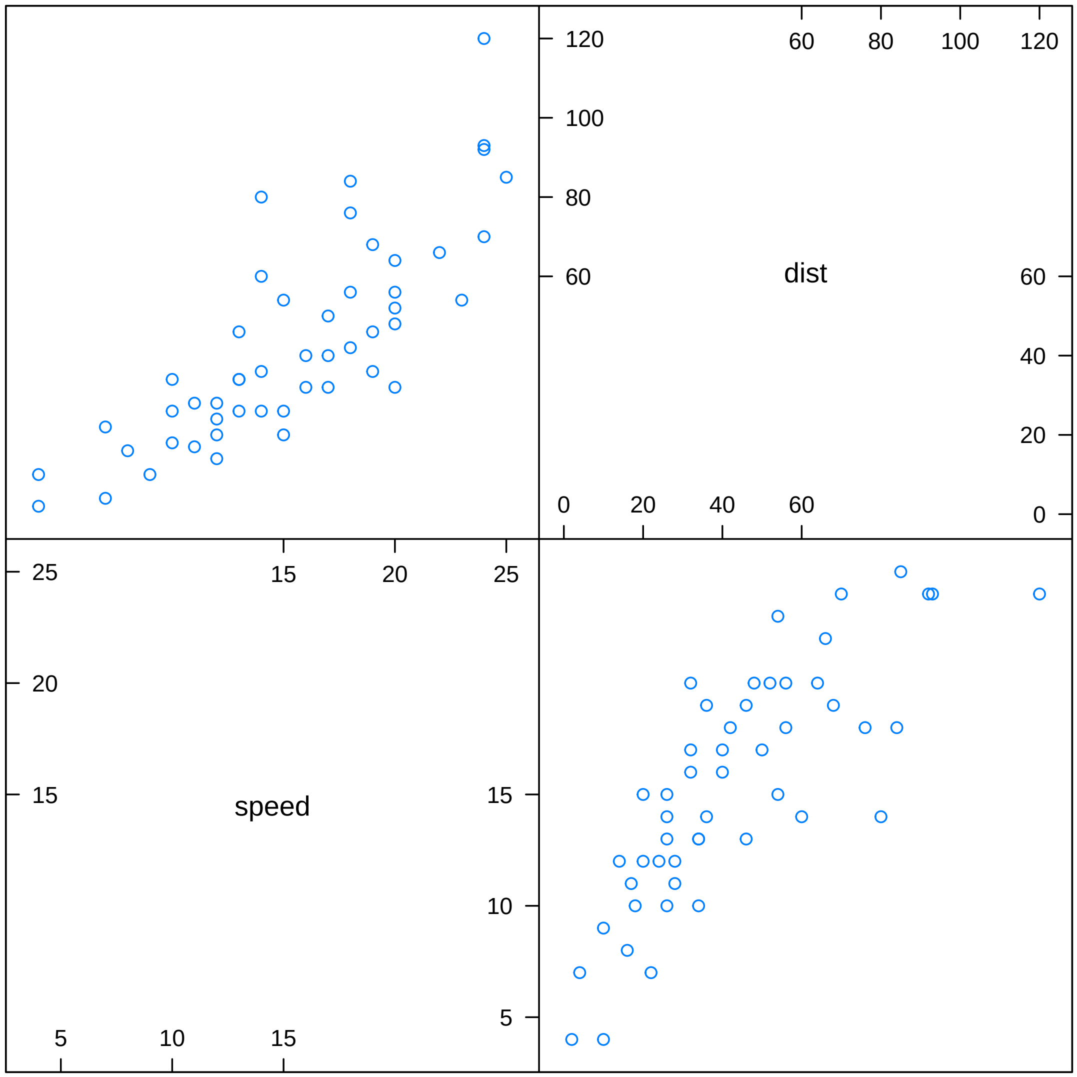
Multivariate Explanatory data analysis

- Correlation matrix:The function cor() can be used to compute a correlation matrix. The function rcorr() [in Hmisc package] can be used to compute the significance levels for pearson and spearman correlations.

```
cor(data)

##           speed      dist
## speed 1.0000000 0.8068949
## dist  0.8068949 1.0000000
```

- Scatter plot: The function splom()[in the package lattice], can be used to display a scatter plot. The function chart.Correlation()[in the package PerformanceAnalytics], can be used to display a chart of a scatter plot and correlation between variables.

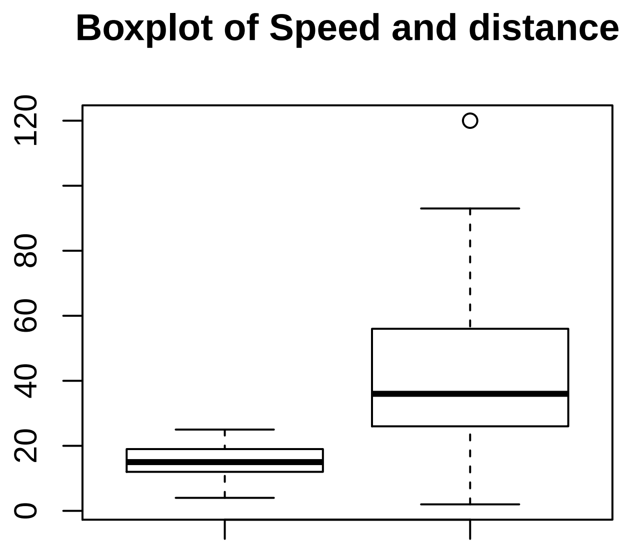
```
library(lattice)
splom(data)
```



Scatter Plot Matrix

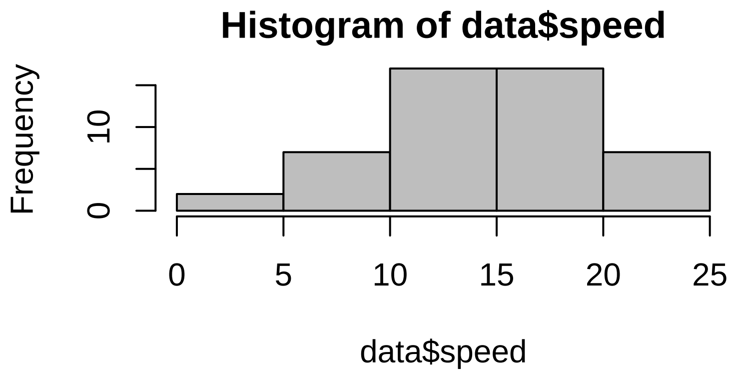
- Multiple boxplot: Multiple boxplot can be simply drawn by adding up variable in boxplot() function.

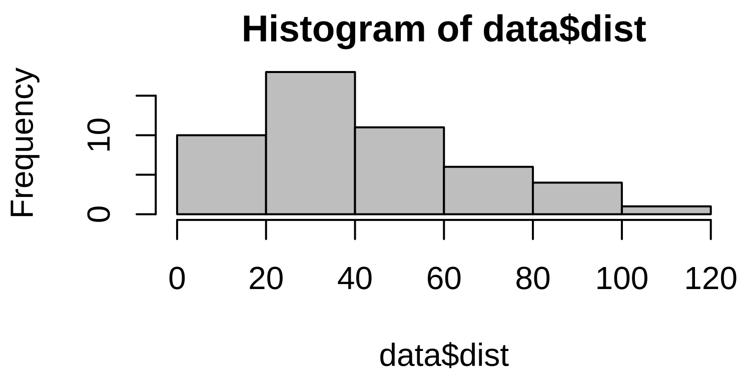
```
boxplot(data$speed,data$dist, main="Boxplot of Speed and distance")
```



- Multiple Histogram: Mulptiple histogram can be simply drawn using hist() function for different variables of interest after par() function which divides the graph window in different rows and columns.

```
par(mfrow = c(2, 1), mar = c(4, 4, 2, 1))
hist(data$speed, col = "gray")
hist(data$dist, col = "gray")
```





4.4 Explanatory Data Analysis using Python

We will use a popular dataset in R library "dataset" and the dataset used is cars.
Description: The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.
Data is imported using read_csv() function of pandas module.

```
In [1]: import pandas as pd
        data=pd.read_csv("E:/jimmy/J project/cars.csv")
        data
```

```
Out[1]:     speed  dist
        1       4     2
        2       4    10
        3       7     4
        4       7    22
        5       8    16
        6       9    10
        7      10    18
        8      10    26
        9      10    34
        10     11    17
        11     11    28
```

```
12    12    14
13    12    20
14    12    24
15    12    28
16    13    26
17    13    34
18    13    34
19    13    46
20    14    26
21    14    36
22    14    60
23    14    80
24    15    20
25    15    26
26    15    54
27    16    32
28    16    40
29    17    32
30    17    40
31    17    50
32    18    42
33    18    56
34    18    76
35    18    84
36    19    36
37    19    46
38    19    68
39    20    32
40    20    48
41    20    52
42    20    56
43    20    64
44    22    66
45    23    54
46    24    70
47    24    92
48    24    93
49    24   120
50    25    85
```

- We can see some basic characteristics of the dataset using DataFrame.info(), DataFrame.tail(), DataFrame.head(), DataFrame.loc[], DataFrame.describe() functions.

```
In [2]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50 entries, 1 to 50
Data columns (total 2 columns):
speed    50 non-null int64
dist     50 non-null int64
dtypes: int64(2)
memory usage: 1.2 KB
```

```
In [3]: data.tail()

Out[3]:     speed  dist
        46     24    70
        47     24    92
        48     24    93
        49     24   120
        50     25    85

In [4]: data.head()

Out[4]:    speed  dist
        1      4     2
        2      4    10
        3      7     4
        4      7    22
        5      8    16

In [5]: data.loc[3:6]

Out[5]:    speed  dist
        3      7     4
        4      7    22
        5      8    16
        6      9    10

In [6]: data.describe()

Out[6]:            speed        dist
        count  50.000000   50.000000
        mean   15.400000   42.980000
        std     5.287644   25.769377
        min     4.000000    2.000000
        25%    12.000000   26.000000
        50%    15.000000   36.000000
        75%    19.000000   56.000000
        max    25.000000  120.000000
```

- **Univariate Explanatory data Analysis**
- five-number summary:five number summary can be computed using Data.Frame.variable.describe() function.

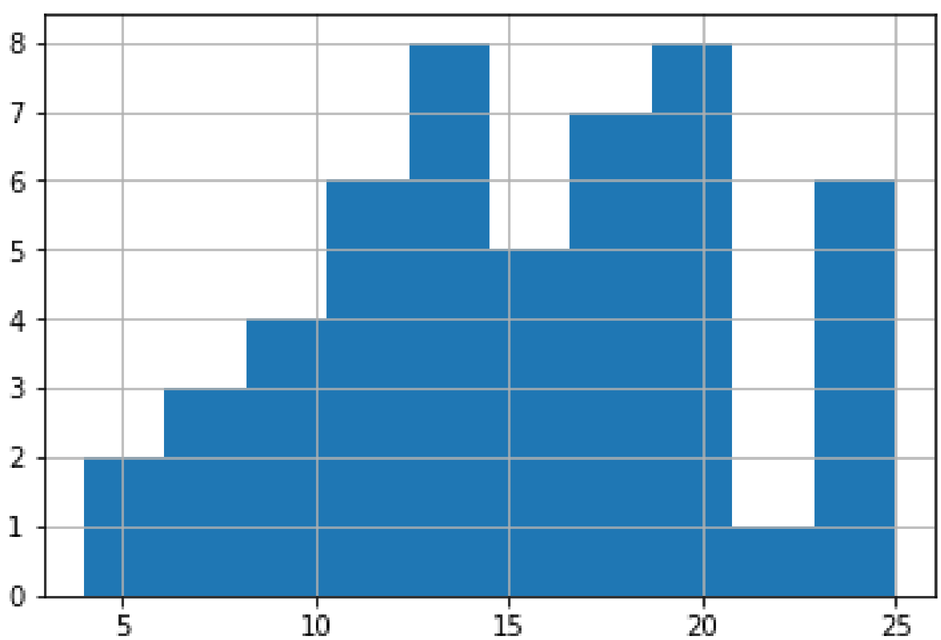
```
In [7]: data.speed.describe()
```

```
Out[7]: count    50.000000
        mean     15.400000
        std       5.287644
        min       4.000000
        25%      12.000000
        50%      15.000000
        75%      19.000000
        max      25.000000
        Name: speed, dtype: float64
```

- Histogram: Histogram can be drawn using hist() function from matplotlib module.

```
In [8]: import matplotlib.pyplot as plt
        %matplotlib inline
        data.speed.hist()
```

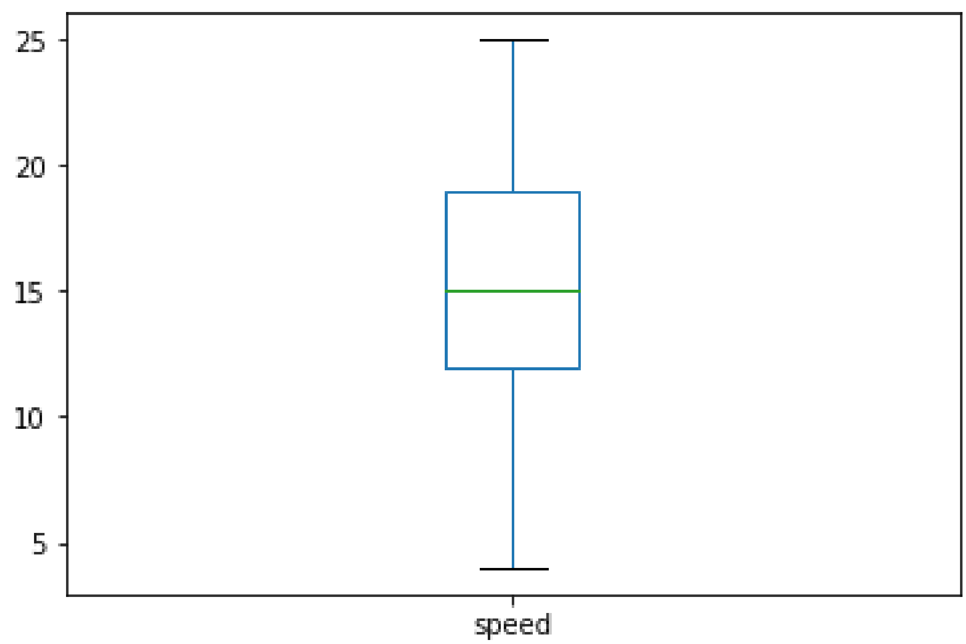
```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x27dd9d87cf8>
```



- Box-plot: Box-plot can be drawn using DataFrame.plot.box() function from matplotlib module.

```
In [9]: data['speed'].plot.box()
```

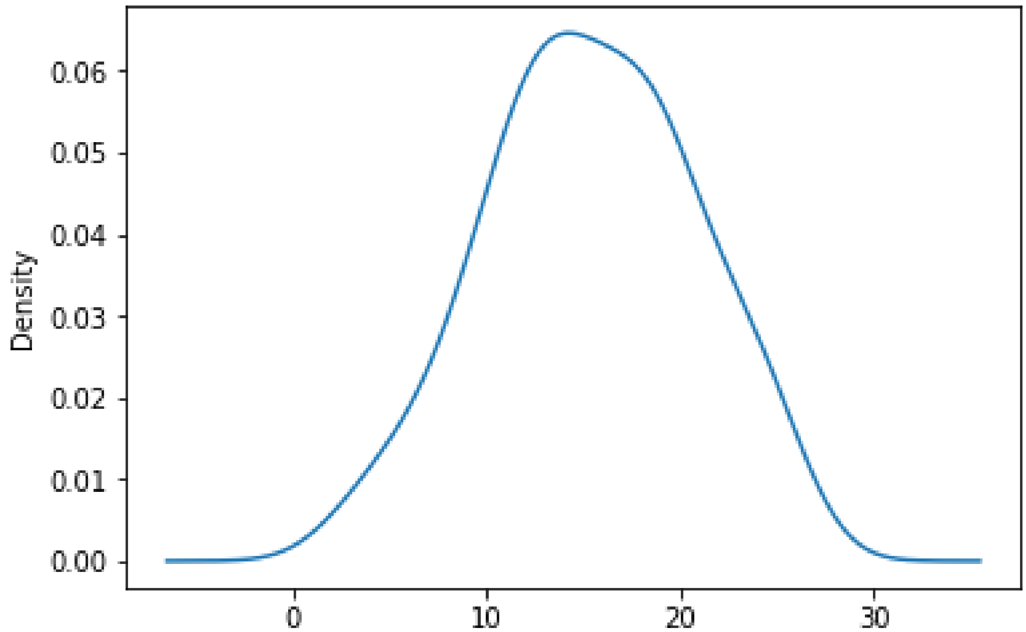
```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x27dd9dc2710>
```



- Density-plot: Density-plot can be drawn using DataFrame.plot.kde() function from matplotlib module.

```
In [10]: data['speed'].plot.kde()
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x27dd94a0160>
```



Note: There are some of the tools in seaborn module for examining univariate and bivariate distributions. That can be used as follows:
*import seaborn as sns #for calling seaborn module
* sns.distplot() #will give density plot
* sns.distplot(x, kde=False, rug=True) #will give histogram plot
* sns.distplot(x, hist=False, rug=True) #will give density plot

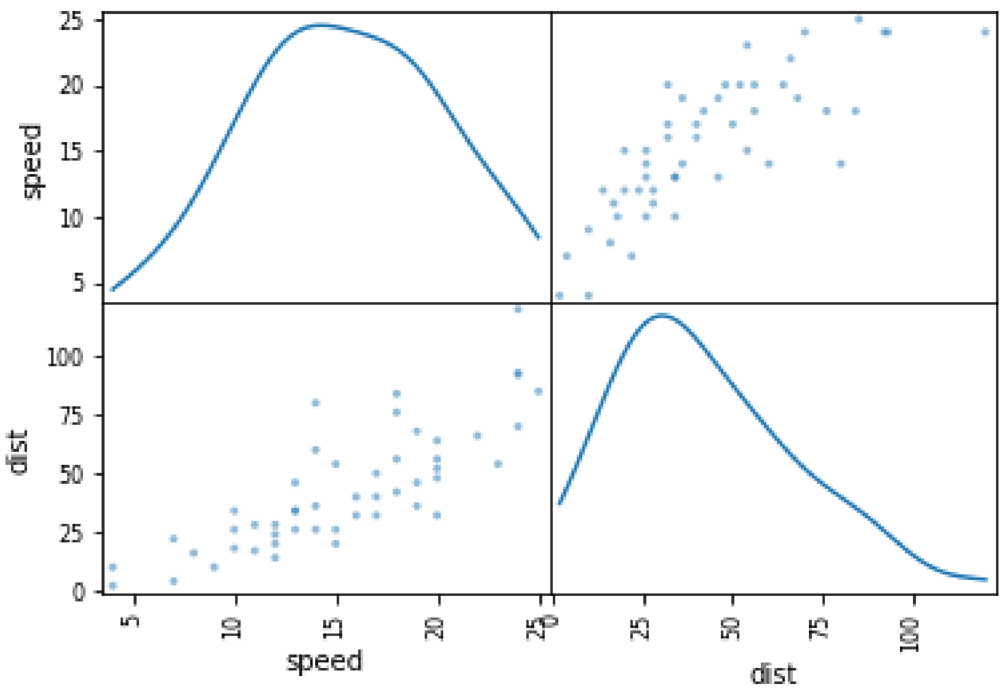
- **Multivariate Explanatory Data Analysis***
- Correlation matrix: DataFrame.corr(method='pearson') compute pairwise correlation of columns. where argument method:{"pearson,'kendall,'spearman'}

```
In [11]: data.corr(method='pearson')

Out[11]:           speed      dist
         speed  1.000000  0.806895
         dist   0.806895  1.000000
```

- Scatter-plot: scatter_matrix() function in pandas.plotting module can be used to plot Scatter-plot matrix.

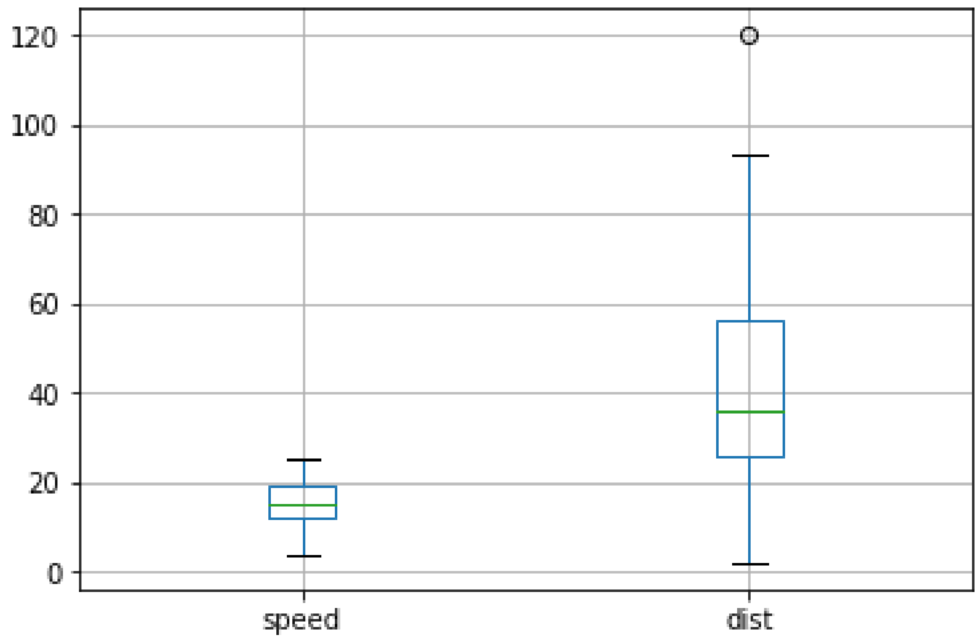
```
In [12]: from pandas.tools.plotting import scatter_matrix as scattermatrix

In [13]: scattermatrix(data, diagonal='kde') #diagonal='kde'
argument shows density plot in diagonal by default it will show histogram.
```

```
Out[13]: array([[<matplotlib.axes._subplots.AxesSubplot object at
0x0000027DDBD52438>,
                 <matplotlib.axes._subplots.AxesSubplot object at
                 0x0000027DDBDB0048>],
                [<matplotlib.axes._subplots.AxesSubplot object at
                0x0000027DDBDE7588>,
                 <matplotlib.axes._subplots.AxesSubplot object at
                 0x0000027DDBE20550>]], dtype=object)
```



- Multiple Box-plot: Multiple Box-plot can be drawn using DataFrame.boxplot(column=[]) as follow:

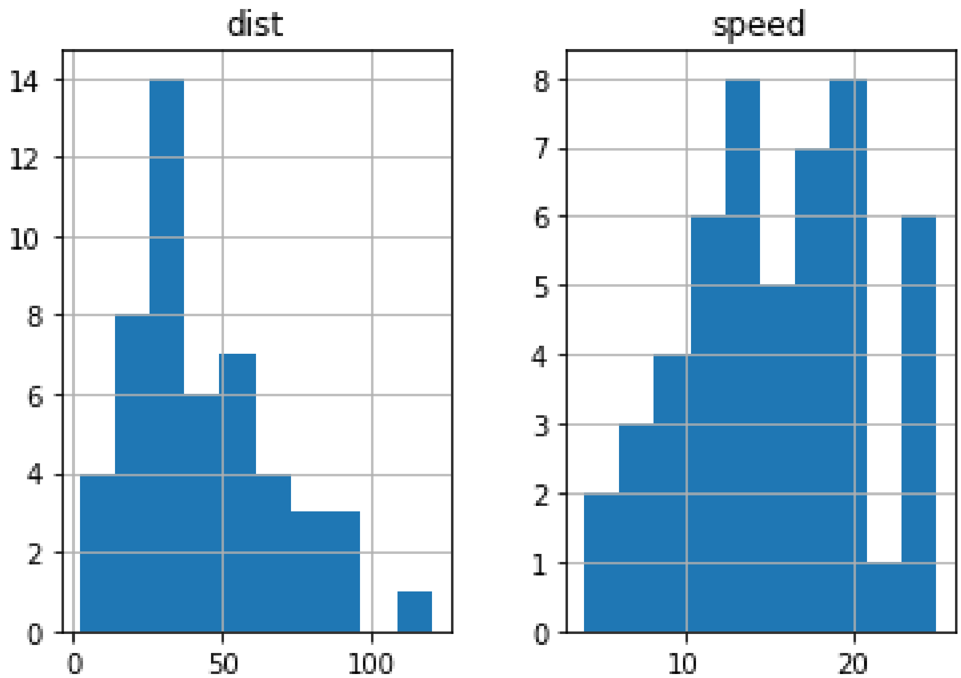
```
In [14]: plt.figure();
         BP=data.boxplot(column=['speed','dist'])
```



- Multiple Histogram: Multiple Histogram can be drawn similarly like multiple box-plot using DataFrame.hist(column=[]) as follow:

```
In [15]: data.hist(column=['speed','dist'])
```

```
Out[15]: array([[<matplotlib.axes._subplots.AxesSubplot object at
0x0000027DDBECA710>,
                 <matplotlib.axes._subplots.AxesSubplot object at
                 0x0000027DDBF66908>]], dtype=object)
```



Note: For more visualization tools on pandas refer to https://pandas.pydata.org/pandas-docs/stable/visualization.html

Chapter 5

Regression Analysis

Regression analysis is used to know the nature of relationship between two or more variables i.e. probable form of mathematical relation between X and Y (where X represent various explanatory variables and Y represents response variable). Regression is also used to predict or estimate the value of one variable(response or dependent variable) corresponding to given value of another variable (explanatory or independent variable).
Linear model: A model is said to be linear when it is linear in parameter. **Non-linear model**: A model is said to be non-linear when it is non-linear in parameter.

5.1 Linear Regression Analysis

In scatter diagram, quite often it is seen that there is a tendency for the point of two variable to cluster around some curve called the curve of regression. If curve is straight line it is called line of regression. And it tells, there is linear regression among the variables. If curve is a line, then it tells there is non linear regression among the variables. The Linear regression model is given by:

$$Y = X\beta + \varepsilon \tag{5.1}$$

where:

- Y denotes the dependent(or response) variable.
- X denotes the k independent(or explanatory) variable $x_1, x_2, ..., x_k$.
- β denotes the regression coefficient associated with $x_1, x_2, ..., x_k$ variables.

We write equation (1) as $y = x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \varepsilon$ for k explanatory variables. This is called as multiple linear regression model.

Example: *Income and education of a person are related it is expected that for an average higher level of education provides higher income. So, the simple linear regression model can be* *expressed as:*

$$Income = \beta_0 + \beta_1 education + \varepsilon_i \tag{5.2}$$

β_0 reflects income when education is zero as it is expected that even an illiterate person can also have some income and β_1 reflects average change in income with respect to per unit change in education. Further, this model neglects that most people have higher income when they are older then they are younger.
So, better model is multiple linear regression model and can be expressed as:

$$Income = \beta_0 + \beta_1 education + \beta_2 age + \varepsilon_i \tag{5.3}$$

5.1.1 Assumptions in Linear regression model

The linear regression has five key assumptions:

- There should be a linear relationship between dependent and independent variables.
- The error term should be normally distributed.
- The error term must have constant variance. The presence of constant variance among error term is known as homoskedasticity. And the absence of constant variance among the error term is known as heteroskedasticity.
- The independent variables should not be correlated. Absence of this phenomenon is called multicollinearity.
- There should be no correlation between the residual or error term. Absence of this phenomenon is known as Autocorrelation

Note: Normal probability plot and plot of residual versus corresponding fitted values is helpful in detecting several common type of model assumption.

5.2 Logistic Regression Analysis

When we have binary variable or categorical variable (dependent variable) we use logistic regression. Logistic model is used for prediction of probability of occurrence of an event by fitting data to a logistic curve. In this regression, the response variable has only two possible outcome coded as 0 or 1. It makes use of several predictor variables that may be either categorical or numerical.

Example: *The probability that a person has a heart attack within a specified time period can be predicted from knowledge of person's age, sex, cholesterol level, weight, etc.*

Logistic model belongs to a class of model known as **Generalized Linear Model**(GLM). The logistic regression model uses the odd ratio, which is given by:

$$Oddratio = \frac{probabilityofaneventofinterest}{1 - probabilityofaneventofinterest} \tag{5.4}$$

The logistic regression is based on log odd ratio, ln(odd ratio). Equation given below defines logistic regression model for k independent variables.

$$ln(Oddratio) = \beta_0 + x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \varepsilon_i \tag{5.5}$$

where:

- k= no. of independent variable in model
- ε_i=random error in observation i

5.3 Regression Analysis using R

Description of dataset: For analysis we are using a dataset in which record times in 1984 for Scottish Hill races are recorded having 35 observation and whose component variables are following:

- dist: distance in miles(on the map).
- climb: total height gained during the route, in feet.
- time: record time in minutes.

5.3.1 Multiple Linear Regression

Loading dataset in R

```
Hills_data<-read.csv("E:/jimmy/j2/Hills.csv",header=TRUE)
head(Hills_data)

##               X dist climb   time
## 1   Greenmantle  2.5   650 16.083
## 2      Carnethy  6.0  2500 48.350
## 3 Craig Dunain  6.0   900 33.650
## 4       Ben Rha  7.5   800 45.600
## 5    Ben Lomond  8.0  3070 62.267
## 6      Goatfell  8.0  2866 73.217

names(Hills_data)

## [1] "X"     "dist"  "climb" "time"
```

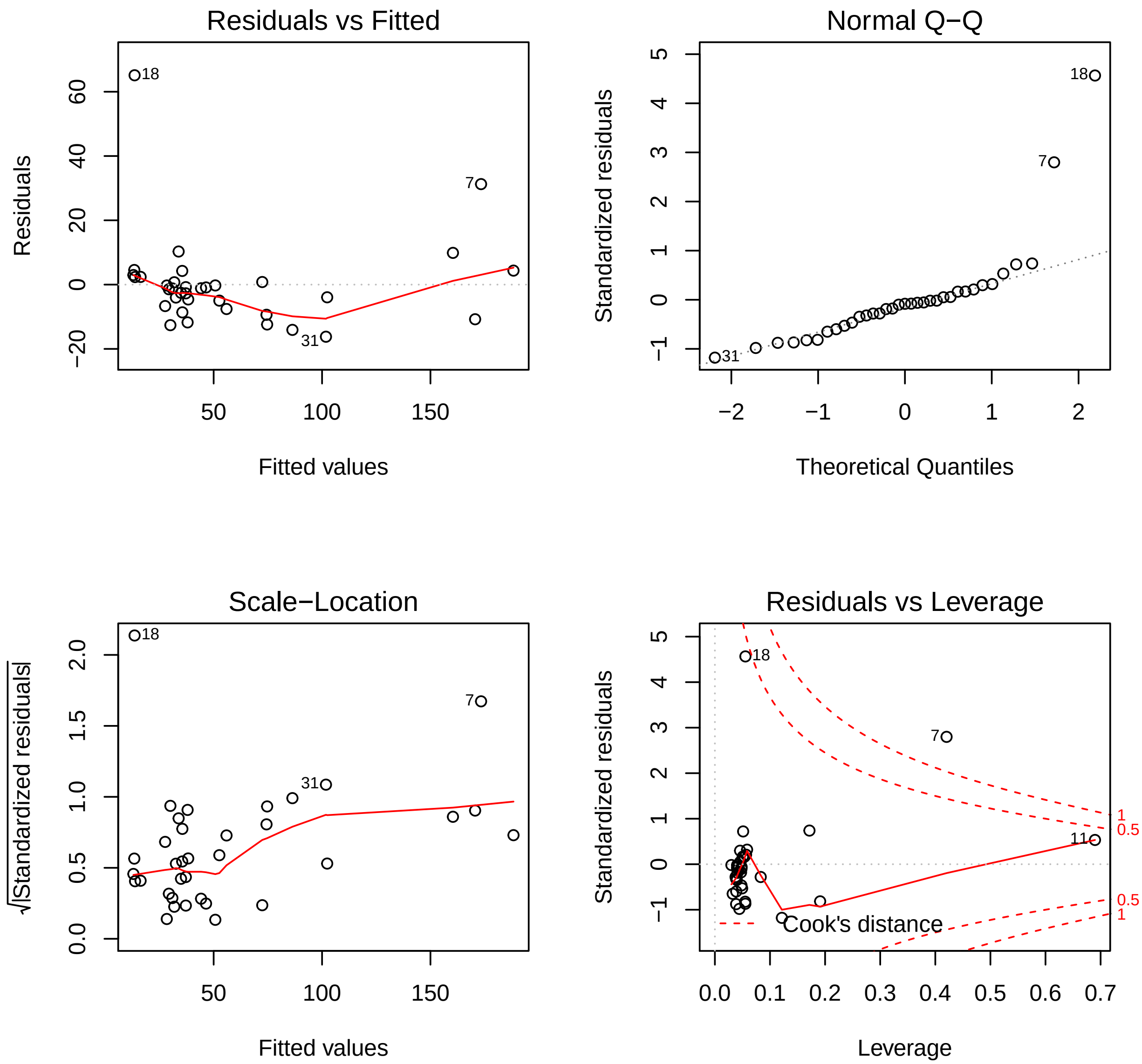
Fitting Multiple Linear Regression:

```
#model for multiple linear regression model
model1=lm(time~dist+climb,data=Hills_data)
summary(model1)

##
## Call:
## lm(formula = time ~ dist + climb, data = Hills_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.215  -7.129  -1.186   2.371  65.121
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.992039   4.302734  -2.090   0.0447 *
## dist         6.217956   0.601148  10.343 9.86e-12 ***
## climb        0.011048   0.002051   5.387 6.45e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.68 on 32 degrees of freedom
## Multiple R-squared:  0.9191,Adjusted R-squared:  0.914
## F-statistic: 181.7 on 2 and 32 DF,  p-value: < 2.2e-16
```

To check assumption of model we can use following steps:
For a quick check of model assumption we can use plot() function which give 2*2 plot containing following:

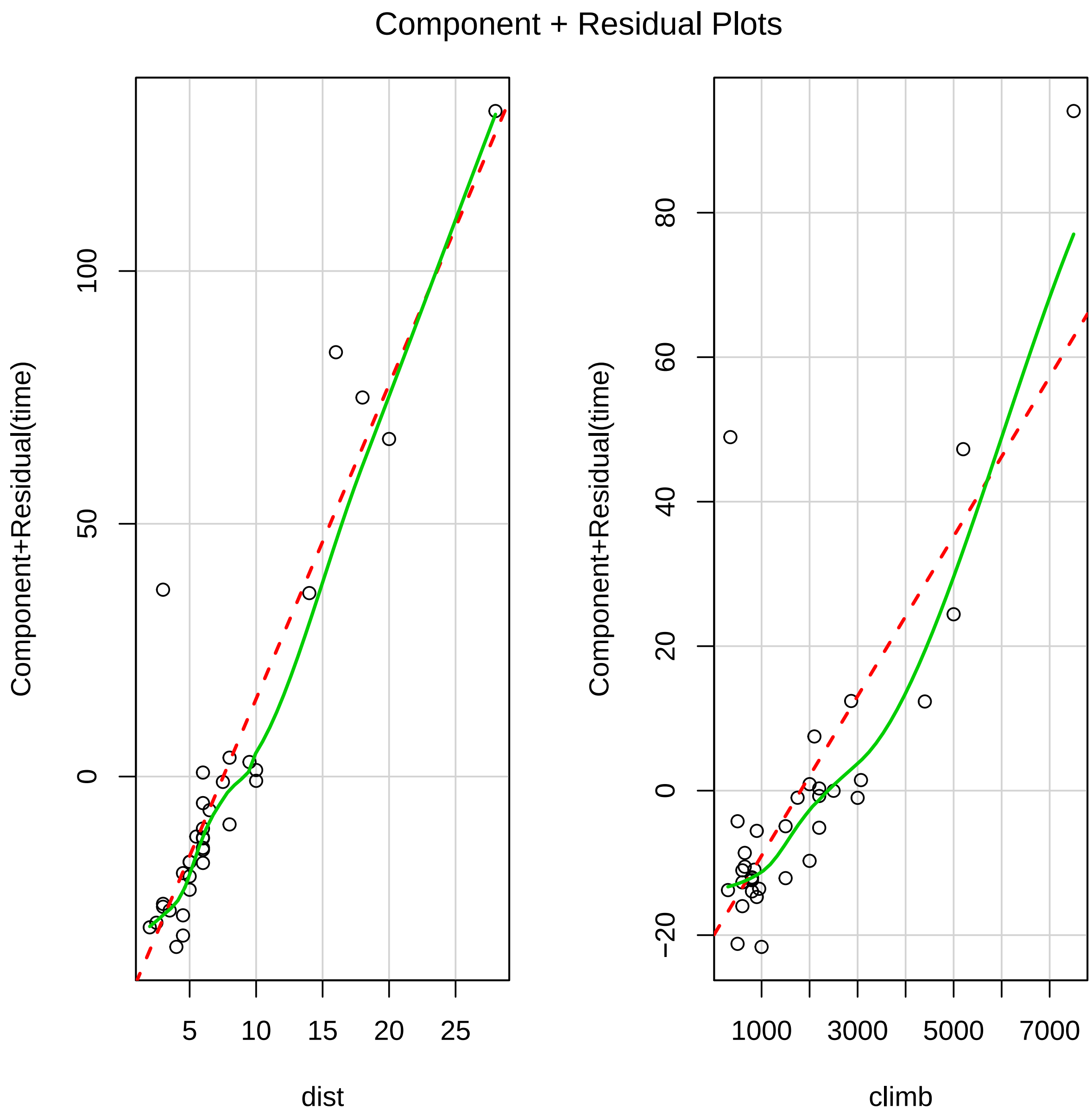
- Residual versus fitted values.
- Normal quantile-quantile plot.
- Standardized residual versus Fitted values.
- Residual versus Leverage.

```
par(mfrow=c(2,2))#used to partion are window into 2 rows and 2 columns
plot(model1)
```



- Component residual plot can be drawn using cr.Plots() function in library(car)

```
library(car)

## Warning:  package 'car' was built under R version 3.4.4
##
## Attaching package:  'car'
## The following object is masked from 'package:dplyr':
##
##     recode

crPlots(model1)
```



5.3.2 Logistic Regression

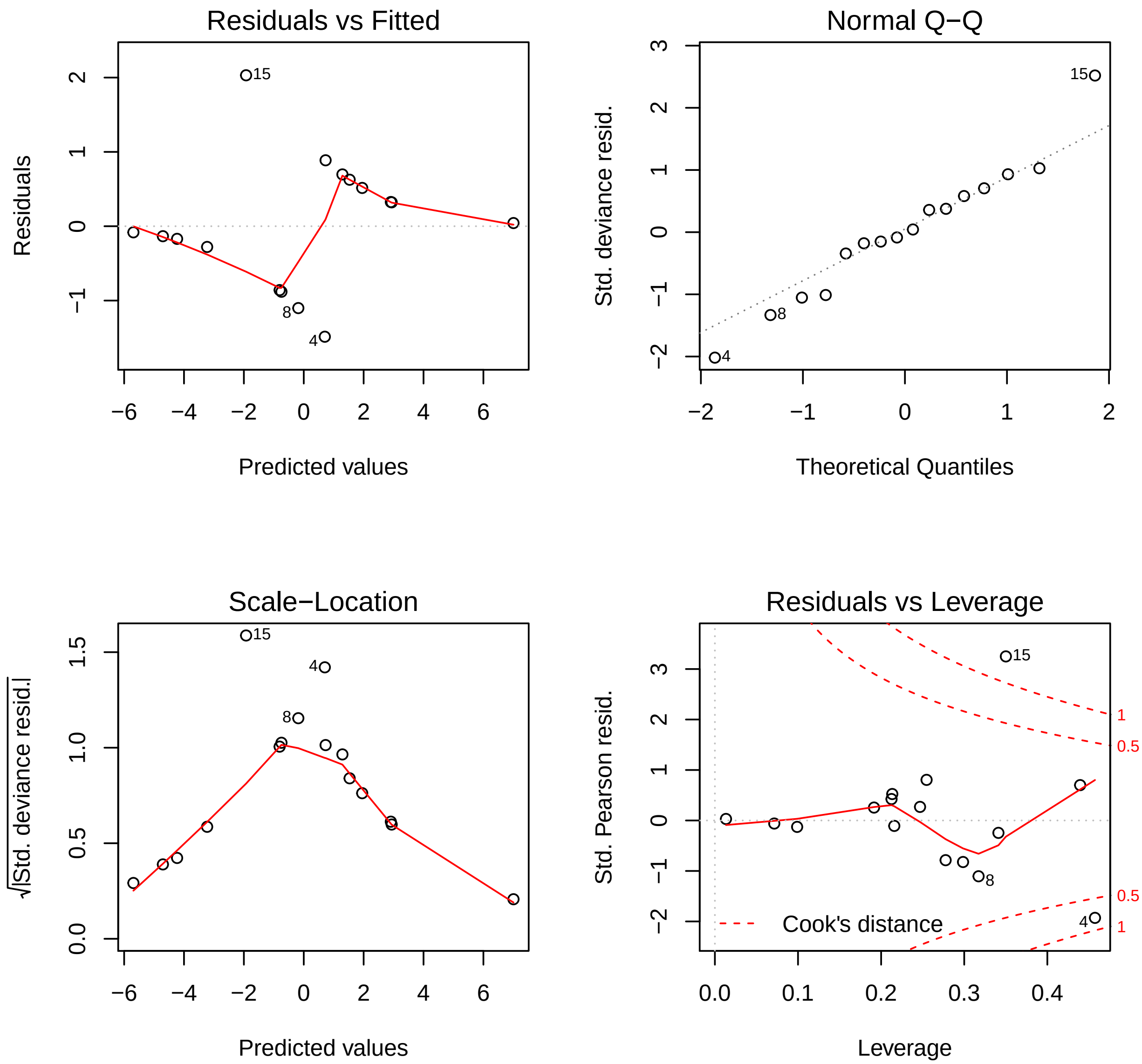
Description of data: For analysis we are using dataset of Contraceptive use data, showing the distribution of 1607 currently married and fecund women interviewed in Fiji Fertility Survey, according to age, education, desire for more children and current use of contraception. Loading dataset in R

```
c_data=read.csv("E:/jimmy/j2/contraceptive_use.xlsx", header=TRUE)
print(c_data)

##       age education wantsMore notUsing using
```

```
## 1    <25      low     yes     53    6
## 2    <25      low      no     10    4
## 3    <25     high     yes    212   52
## 4    <25     high      no     50   10
## 5  25-29      low     yes     60   14
## 6  25-29      low      no     19   10
## 7  25-29     high     yes    155   54
## 8  25-29     high      no     65   27
## 9  30-39      low     yes    112   33
## 10 30-39      low      no     77   80
## 11 30-39     high     yes    118   46
## 12 30-39     high      no     68   78
## 13 40-49      low     yes     35    6
## 14 40-49      low      no     46   48
## 15 40-49     high     yes      8    8
## 16 40-49     high      no     12   31
```

- contrasts() function shows how variable is dummyfied by R. str() function shows the structure of data.

```
contrasts(c_data$education)

##      low
## high   0
## low    1

contrasts(c_data$wantsMore)

##     yes
## no    0
## yes   1

str(c_data)

## 'data.frame': 16 obs. of  5 variables:
##  $ age      : Factor w/ 4 levels "<25","25-29",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ education: Factor w/ 2 levels "high","low": 2 2 1 1 2 2 1 1 2 2 ...
##  $ wantsMore: Factor w/ 2 levels "no","yes": 2 1 2 1 2 1 2 1 2 1 ...
##  $ notUsing : int  53 10 212 50 60 19 155 65 112 77 ...
##  $ using    : int  6 4 52 10 14 10 54 27 33 80 ...

mymodel<-glm(wantsMore~notUsing+using+education,data=c_data,family = binomial
             (link='logit'))
par(mfrow=c(2,2))
plot(mymodel)
```



5.4 Regression Analysis using Python

Description of dataset: For analysis we are using a dataset in which record times in 1984 for Scottish Hill races are recorded having 35 observation and whose component variables are following:

- dist: distance in miles(on the map).
- climb: total height gained during the route, in feet.
- time: record time in minutes.

Statsmodels is a python module that provide functions for estimation of many different statistical model, as well as conducting statistical test and statistical data exploration. We are using statsmodels for conducting multiple linear regression and checking assumptions of model by analyzing the residuals (See http://www.statsmodels.org/dev/regression.html) Loading all important modules used in regression analysis:

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import statsmodels.formula.api as sm
        %matplotlib inline
```

Loading Dataset for Multiple regression analysis:

```
In [2]: Hills_data=pd.read_csv("E:/jimmy/j2/Hills.csv",header=0)
        print(Hills_data)

           Unnamed: 0  dist  climb     time
0         Greenmantle   2.5    650   16.083
1            Carnethy   6.0   2500   48.350
2        Craig Dunain   6.0    900   33.650
3             Ben Rha   7.5    800   45.600
4          Ben Lomond   8.0   3070   62.267
5            Goatfell   8.0   2866   73.217
6        Bens of Jura  16.0   7500  204.617
7         Cairnpapple   6.0    800   36.367
8              Scolty   5.0    800   29.750
9            Traprain   6.0    650   39.750
10        Lairig Ghru  28.0   2100  192.667
11             Dollar   5.0   2000   43.050
12            Lomonds   9.5   2200   65.000
13        Cairn Table   6.0    500   44.133
14         Eildon Two   4.5   1500   26.933
15          Cairngorm  10.0   3000   72.250
16        Seven Hills  14.0   2200   98.417
17         Knock Hill   3.0    350   78.650
18         Black Hill   4.5   1000   17.417
19         Creag Beag   5.5    600   32.567
20       Kildcon Hill   3.0    300   15.950
21   Meall Ant-Suidhe   3.5   1500   27.900
22     Half Ben Nevis   6.0   2200   47.633
23           Cow Hill   2.0    900   17.933
24      N Berwick Law   3.0    600   18.683
25         Creag Dubh   4.0   2000   26.217
26          Burnswark   6.0    800   34.433
```

```
27         Largo Law   5.0    950   28.567
28           Criffel   6.5   1750   50.500
29            Acmony   5.0    500   20.950
30         Ben Nevis  10.0   4400   85.583
31       Knockfarrel   6.0    600   32.383
32     Two Breweries  18.0   5200  170.250
33         Cockleroi   4.5    850   28.100
34      Moffat Chase  20.0   5000  159.833
```

```
In [3]: print(Hills_data.shape, Hills_data.dtypes)
```

```
(35, 4) Unnamed: 0     object
dist           float64
climb            int64
time           float64
dtype: object
```

Fitting a Multiple regression model

```
In [4]: model=sm.ols(formula='time~dist+climb',data=Hills_data).fit()
```

```
In [5]: print(dir(model)) #gives number of objects in model.
```

```
['HC0_se', 'HC1_se', 'HC2_se', 'HC3_se', '_HCCM', '__class__', '__
delatt__', '__dict__', '__dir__', '__doc__', '__eq__', '_ _forma
__','__ge__', '__getattribute__', '__gt__', '__hash__
', '__init__','__init_subclass__', '__le__', '__lt__', '__
module__','__ne__','__new__', '__reduce__', '__reduce_ex__
', '__repr__', '__setattr__', '__sizeof__', '__str__', '__
subclasshook__', '__weakref__','_cache', '_data_attr', '_get_robust
cov_results', '_is_nested', '_wexog_singular_values', 'aic', 'bic', 'bse
', 'centered_tss', 'compare_f_test','compare_lm_test', 'compare_lr_test
', 'condition_number', 'conf_int', 'conf_int_el','cov_HC0', 'cov_HC1','
cov_HC2', 'cov_HC3', 'cov_kwds', 'cov_params','cov_type','df_model', 'df
_resid', 'eigenvals', 'el_test', 'ess', 'f_pvalue', 'f_test', 'fittedvalues
', 'fvalue', 'get_influence', 'get_prediction', 'get_robustcov_results','
initialize', 'k_constant', 'llf', 'load', 'model', 'mse_model', 'mse_resid',
'mse_total', 'nobs', 'normalized_cov_params', 'outlier_test', 'params',
'predict','pvalues', 'remove_data', 'resid', 'resid_pearson', 'rsquared',
'rsquared_adj','save', 'scale', 'ssr', 'summary', 'summary2', 't_test', '
tvalues', 'uncentered_tss','use_t', 'wald_test', 'wald_test_terms', 'wresid']
```

summary() function displays details of the result.

```
In [6]: print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   time   R-squared:                       0.919
Model:                            OLS   Adj. R-squared:                  0.914
Method:                 Least Squares   F-statistic:                     181.7
Date:                Sun, 25 Mar 2018   Prob (F-statistic):           3.40e-18
Time:                        17:04:52   Log-Likelihood:                -142.11
No. Observations:                  35   AIC:                             290.2
Df Residuals:                      32   BIC:                             294.9
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -8.9920      4.303     -2.090      0.045     -17.756      -0.228
dist           6.2180      0.601     10.343      0.000       4.993       7.442
climb          0.0110      0.002      5.387      0.000       0.007       0.015
==============================================================================
Omnibus:                       47.910   Durbin-Watson:                   2.249
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              233.976
Skew:                           3.026   Prob(JB):                     1.56e-51
Kurtosis:                      14.127   Cond. No.                     4.20e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 4.2e+03. This might indicate that there
are strong multicollinearity or other numerical problems.
```

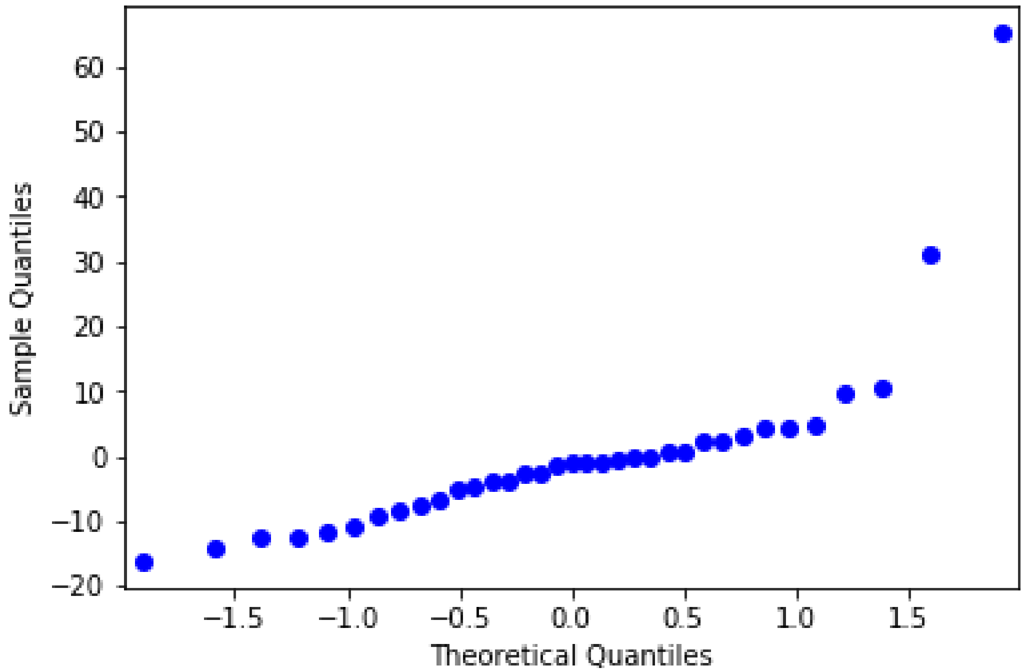
The coefficient of determination is equal to R-squared value i.e. 0.846. Warning message is indicating that there might be strong multicollinearity present.

To check assumption of model we can use following steps

- Normal Probability plot to show assumption of normality of residuals.

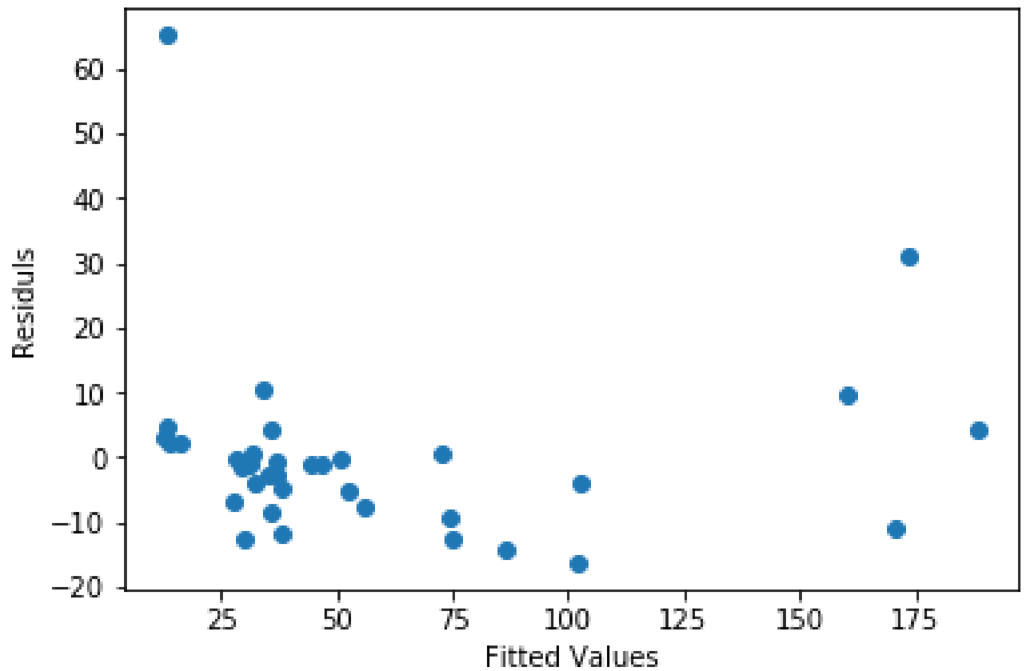
```
In [7]: import statsmodels.api as sma
        sma.qqplot(model.resid)
```

```
Out[7]:
```



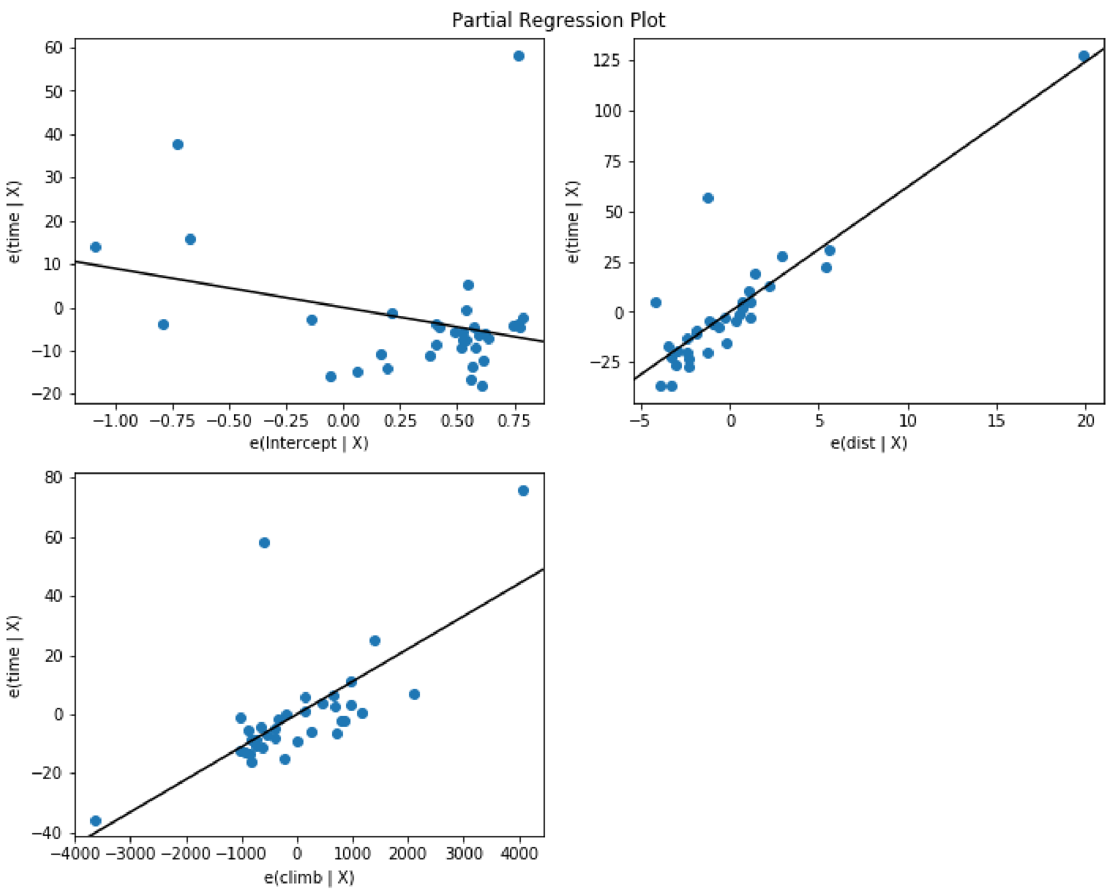
- residual vs fitted value plot to show holding assumptions

In [8]:
```
plt.scatter(model.fittedvalues.values,model.resid)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.show()
```



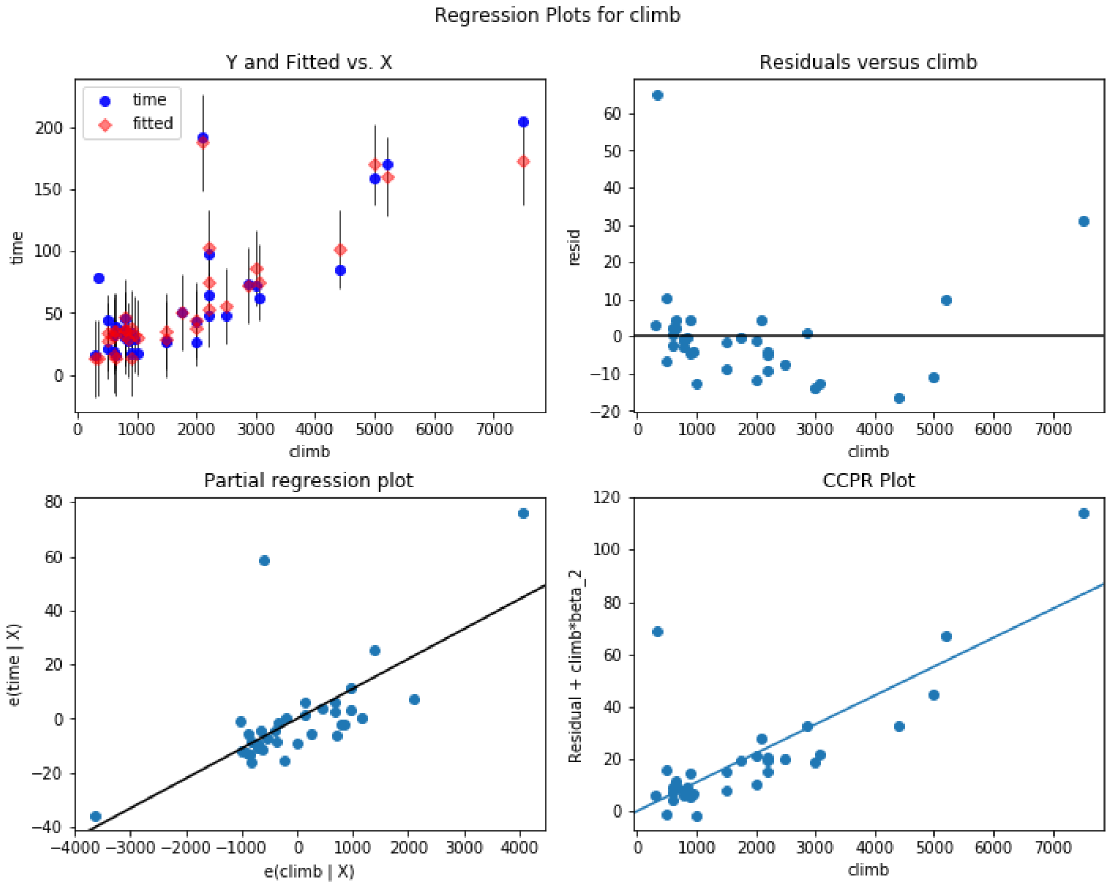
- For quick check of all the regressors, you can use plot_partregress_grid() function.

```
In [9]: fig=plt.figure(figsize=(10,8))
        fig=sma.graphics.plot_partregress_grid(model,fig=fig)
```



- The function plot_regress_exog() gives a 2*2 plot containing following:
  *Dependent variable and fitted values confidence interval versus independent variable chosen.
  * Residual versus independent variable chosen.
  * Partial regression plot.
  * Component-component plus residual plot. This function can be used quickly to check assumptions w.r.t a single regressor.

```
In [10]: fig = plt.figure(figsize=(10,8))
         fig = sma.graphics.plot_regress_exog(model, "climb", fig=fig)
```


Regression Plots for climb
Y and Fitted vs. X
time
fitted
climb
Residuals versus climb
resid
climb
Partial regression plot
e(time | X)
e(climb | X)
CCPR Plot
Residual + climb*beta_2
climb

Bibliography

[1] David Kahle, Hadley Wickham (2016), Spatial Visualization with ggplot2
https://cran.r-project.org/web/packages/ggmap/ggmap.pdf

[2] Hadley Wickham [aut, cre], Winston Chang [aut], RStudio [cph].(2016), Create Elegant Data Visualizations Using the Grammar of Graphics
https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf

[3] Introductory Statistics with R. P. Dalgaard. Springer. Covers material typical of an introductory statistics course, using R for examples. Assumes no advanced mathematics beyond algebra.

[4] Mixed Effects Models in S and S-Plus. J. C. Pinheiro, D. M. Bates. Springer. An advanced title covering liner and nonlinear mixed effect models.

[5] Modern Applied Statistics with S, 4th Ed.. B.D. Ripley and V.N.Venables. Springer. A more advanced book using S. Emphasis on linear models and multivariate data analysis. Includes some coverage of R but more specific to SPlus.

[6] R Bloggers (2017). Data Science Job Report 2017. Retrieved from: https://www.r-bloggers.com/

[7] R Development Core Team. (2007b). R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. Available from http://www.r-project.org

[8] Revolution Analytics. (2014, February 20). What is R? Retreived February 21, 2016, from: http://www.inside-r.org/what-is-r, February 20, 2014.

[9] Sarkar, D. (2017). lattice: Lattice graphics. Retrieved from https://cran.r-project.org/web/packages/lattice/lattice.pdf

[10] The R Foundation. (2017). The R Project for Statistical Computing. Retrieved from: https://www.r-project.org/

[11] Yihui Xie (2016), A General-Purpose Package for Dynamic Report Generation in R
https://cran.r-project.org/web/packages/knitr/knitr.pdf

[12] Wickham H (2015). R Packages: Organize, Test, Document, and Share Your Code. O'Reilly, Sebastopol.

[13] Allen B. Downey. Think Python. O'Reilly Media, first edition, August 2012.

[14] Fabian Pedregosa, Ga el Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: machine learning in Python. Journal of Machine Learning Research, 12:2825-2830, 2011.

[15] Guo, Philip. (2007, May). Why Python is a great language for teaching beginners in introductory programming classes. Retrieved from: http://pgbovine.net/python-teaching.htm

[16] Hinsen K (2007). ScientificPython Manual. URL http://dirac.cnrs-orleans.fr/ ScientificPython/ScientificPythonManual/.

[17] Hughes, Zachariah. (2015, March). Personal Experience Section in MatLab vs. Python vs. R.

[18] Kevin Sheppard. Introduction to Python for econometrics, statistics and data analysis. Self-published, University of Oxford, version 2.1 edition, February 2014.

[19] matplotlib documentation, https://matplotlib.org/

[20] Oliphant TE (2006). Guide to NumPy. Provo, UT. URL http://www.tramy.us/.

[21] pandas documentation, https://pandas.pydata.org/

[22] seaborn documentation, https://seaborn.pydata.org/

[23] statsmodels documentation, http://www.statsmodels.org/dev/regression.html

[24] Skipper Seabold and Josef Perktold. Statsmodels: econometric and statistical modeling with python. In Proceedings of the 9th Python in Science Conference, 2010. Annotation: Description of the statsmodels package for Python.

[25] Temple Lang D (2005). "R/S-PLUS-Python Interface." URL http://www.omegahat.org/ RSPython/.

[26] Van Rossum G, others (2016). Python Programming Language. URL http://www.python. org/

[27] Wes McKinney. Python for data analysis. O'Reilly, Sebastopol, California, first edition, October 2012. Annotation: Book on data analysis with Python introducing the Pandas library.