R Notes - Learn the course

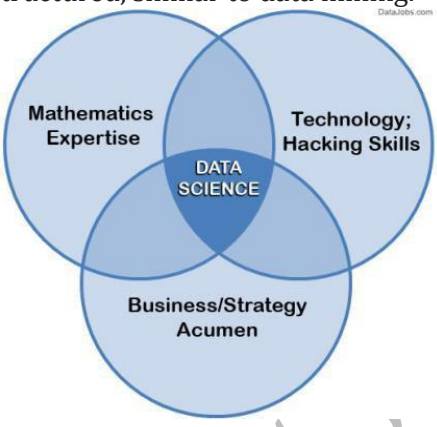
Data Communication (Manipur University)

UNIT-1
Introduction to Data Science

Data Science Process - roles , stages in data science project - working with data from files - working with relational data bases - exploring data - managing data - cleaning and sampling for modelling and validation - Introduction to NoSQL

Introduction:

Data science, also known as data-driven science, is an interdisciplinary field about scientific methods, processes, and systems to extract knowledge or insights from data in various forms, either structured or unstructured, similar to data mining.



Data scientists learn by doing, ensuring that every iteration and hypothesis improves the practitioner's knowledge base. By taking multiple datasets through the data science pipeline using two different programming languages (R and Python). Data Scientists also want to know that

- Data science recipes are ambiguous.
  Ex: When chefs begin a particular dish, they have a very clear picture in mind of what the finished product will look like.
- For data scientists, the situation is often different.
  Ex :One does not always know what the dataset in question will look like, and what might or might not be possible, given the amount of time and resources. Recipes are essentially a way to dig into the data and get started on the path towards asking the right questions to complete the best dish possible.
- If a Person(Data Practitioners) from a statistical or mathematical background, the modeling techniques on display might not excite.
  Ex: Pay attention to how many of the recipes overcome practical issues in the data science pipeline, such as loading large datasets and working with scalable tools to adapting known techniques to create data applications, interactive graphics, and web pages rather than reports and papers.
- Practicing data scientists require a great number and diversity of tools to get the job done.
  Ex : Data practitioners scrape, clean, visualize, model, and perform a million different tasks with a wide array of tools.

Data Science Process or Data Science Life Cycle

When a non-technical supervisor asks us to solve a data problem, the description of our task can be quite ambiguous at first. As the data scientist, to translate the task into a concrete problem. In the following figure , how to solve it and present the solution back to all of our stakeholders.

There are various steps involved in this workflow of the “Data Science Process.” This process involves several important steps:

Frame the problem:

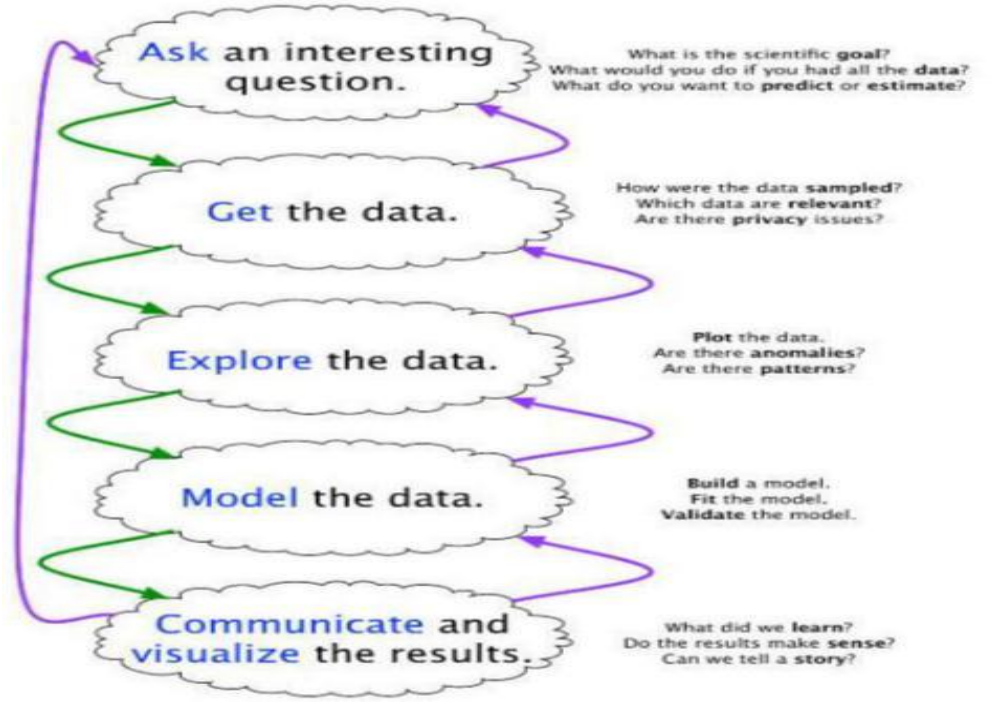
The first thing we have to do before we solve a problem is to define exactly what it is. We need to be able to translate data questions into something actionable.

Collect the raw data needed to solve the problem:

Once we have defined the problem, we need data to give the insights(understanding) needed to turn the problem around with a solution. This part of the process involves thinking through what data we need and finding ways to get that data, whether it's querying internal databases, or purchasing external datasets.

Process the data (data wrangling):

Real, raw data is rarely usable out of the box. There are errors in data collection, corrupt records, missing values and many other challenges which have to manage. In this first need to clean the data to convert it to a form that can be further analyze.



Explore the data:

Once we have cleaned the data, we have to understand the information contained within at a high level. This is to find out and get familiar with the data, understand what are the patterns in the data and at this stage we usually do missing data analysis, correlations, and distribution analysis, scatter plots, frequency analysis and so on. Through the EDA, we also lookout for data errors.

For Example, the value of Gender is "M" and "F" but we see the value of "f" and "m" as well, there might be some errors in the way the gender data is captured and if that is the case, it should be flagged out.

Perform in-depth analysis (machine learning, statistical models, algorithms):

This step of the process is where we are going to have to apply our statistical, mathematical and technological knowledge and leverage(hold) all of the data science tools at our disposal to crunch the data and find every insight.

Communicate results of the analysis:

All the analysis and technical results that we come up with little value unless we can explain to our stakeholders what they mean, in what way that is comprehensible and compelling. Data storytelling is a critical and underrated skill that we will build and use here.

Roles of Data Scientist:

Data science is not performed in a vacuum. It's a collaborative effort that draws on a number of roles, skills, and tools. In the data science process, the roles must be filled in a successful project.The roles of the data scientist can be shown in the following figure:

1.Data Engineer:

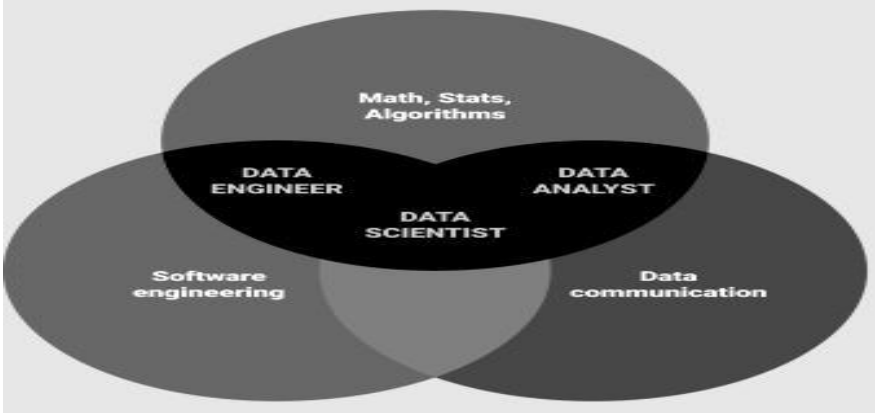
A Data Engineer is a person, fully equipped with knowledge of hardware, databases, data processing at scale and computer engineering and who can build data infrastructure, manage data storage and use and Implement production tools.

2.Data Scientist:

A data scientist is responsible for pulling and cleaning data, designing experiments, analyzing data and communicating result. He should have stronger statistics and presentation skills than a data analyst and data engineer. A data scientist would have strong skills of Inferential Statistics, Machine Learning, Data Analysis, Data Communication.

3.Data Science Manager:

A Data Science Manager is a person who builds a data team, manages the whole data science process, set goals and priorities and interact with other groups and higher management. He should be strong knowledge of
software and hardware, knowledge of roles, strong communication and he knows what can and can't be achieved. A Data Manger can be any background like: Data science plus management skills or Data engineering plus management skills or Management skills plus got certain training in data science.



4.Data Architect:

A Data Architect understand all the sources of data and responsible for integrating, centralizing and maintaining all the data. He has strong knowledge of how the data relates to the current operations and the effects that any future process changes will have on the use of data in the organization. The role may include things like designing relational databases, developing strategies for data acquisitions, archive recovery, and implementation of a database, cleaning and maintaining the database by removing and deleting old data etc.

5.Data Analyst:

Data analysts need to have a good understanding of programming, statistics, machine learning, data managing, and data visualization. The Analyst may not have the mathematical or research background to invent new algorithms, but they have a strong understanding of how to use existing tools to solve problems and get new useful insights from data.

6.Business Analyst:

Business Analyst performs the task of understanding business change needs, assessing the business impact of those changes, capturing, analyzing and documenting requirements and supporting the communication and delivery of requirements with relevant stakeholders. The business analyst role is often seen as a communication bridge between IT and the business stakeholders. Business analysts must be great verbal and written communicators, tactful diplomats, problem solvers, thinkers and analyzers – with the ability to engage with

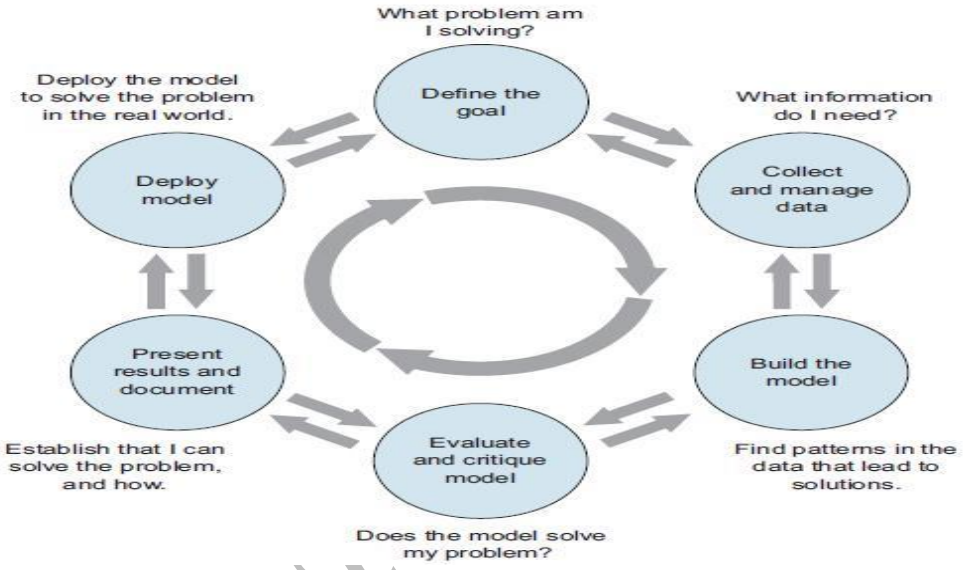
stakeholders to understand and respond to their needs in rapidly changing business environments.

7.Software Engineer:

Software engineers are also needed in data science team because Software is the generalization of a specific aspect of a data analysis. If specific parts of a data analysis require implementing or applying a number of procedures or tools together then we need to build a piece of software to reduce the repeated work.

Stages in data science project or Standard Lifecycle of Data Science Projects:

The data science environment is one that encourages feedback and iteration between the data scientist and all other stakeholders. This is reflected in the lifecycle of a data science project. The data science process breaks up the cycle into distinct stages, in reality the boundaries between the stages are fluid, and the activities of one stage will often overlap into other stages.



[Fig:The Lifecycle of a Data Science Project]

1.Defining the goal:

The first task in a data science project is to define a measurable and quantifiable goal. The goal should be specific and measurable. The less specific the goal, the likelier that the project will go unbounded, because no result will be "good enough."Once we have a good idea of the project's goals, we can focus on collecting data to meet those goals.

2.Data collection and management:

This step encompasses identifying the data we need, exploring it, and conditioning it to be suitable for analysis. This stage is often the most time-consuming step in the process.

This is the stage where we conduct initial exploration and visualization of the data. We will also clean the data: repair data errors and transform variables, as needed. In the process of exploring and cleaning the data, we may discover that it isn't suitable for our problem, or that we need other types of information as well. While collecting the data, we use information that can be directly measured, rather than information that is inferred from another measurement.

3.Modelling:

Many modeling procedures make specific assumptions about data distribution and relationships, there will be overlap and back-and-forth between the modeling stage and the data cleaning stage as you try to find the best way to represent the data and the best form in which to model it. The most common data science modeling tasks are:

- Classification—Deciding if something belongs to one category or another
- Scoring—Predicting or estimating a numeric value, such as a price or probability
- Ranking—Learning to order items by preferences

- Clustering—Grouping items into most-similar groups
- Finding relations—Finding correlations or potential causes of effects seen in the data
- Characterization—Very general plotting and report generation from data.

4.Model evaluation and critique:

Once we have a model, we need to determine our goals. In this defining more realistic goals or gathering the additional data or other resources that we need to achieve our original goals.

5.Presentation and documentation:

Once we have a model that meets our success criteria, we will present our results to our project sponsor and other stakeholders. we must also document the model for those in the organization who are responsible for using, running, and maintaining the model once it has been deployed.

6.Model deployment and maintenance:

Finally, the model is put into operation. In many organizations, the data scientist no longer has primary responsibility for the day-to-day operation of the model. But we still should ensure that the model will run smoothly and won't make disastrous unsupervised decisions. We also want to make sure that the model can be updated as its environment changes. In many situations, the model will initially be deployed in a small program. The test might bring out issues that didn't anticipate, and to adjust the model accordingly.

Working with data from files

The most common ready-to-go data format is a family of tabular formats called structured values. Most of the data find in (or nearly in) one of these formats. When you can read such files into R, you can analyze data from an incredible range of public and private data sources.

1.Working with well-structured data from files or URLs:

The easiest data format to read is table-structured data with headers. As shown in figure

	A	B	C	D	E	F	G
1	buying	maint	doors	persons	lug_boot	safety	rating
2	vhigh	vhigh	2	2	small	low	unacc
3	vhigh	vhigh	2	2	small	med	unacc
4	vhigh	vhigh	2	2	small	high	unacc
5	vhigh	vhigh	2	2	med	low	unacc
6	vhigh	vhigh	2	2	med	med	unacc

Car data viewed as a table

This data is arranged in rows and columns where the first row gives the column names. Each column represents a different fact or measurement; each row represents an instance or datum about which we know the set of facts. A lot of public data is in this format, so being able to read it opens up a lot of opportunities.

2.Loading well-structured data from files or URLs:

The read.table() command is powerful and flexible; it can accept many different types of data separators (commas, tabs, spaces, pipes, and others) and it has many options for controlling quoting and escaping data.read.table() can read from local files or remote URLs. If a resource name ends with the .gz suffix, read.table() assumes the file has been compressed in gzip style and will automatically decompress it while reading.

3.Working with other data formats:

.csv is not the only common data file format you'll encounter. Other formats include .tsv (tab-separated values), pipe-separated files, Microsoft Excel workbooks, JSON data,and XML. R's built-in read.table() command can be made to read most separated value formats.

4.Working data on less-structured data:

Data isn't always available in a ready-to-go format. Data curators often stop just short of producing a ready-to-go machine-readable format. This data is stored as tabular data without headers; it uses a cryptic encoding of values that requires the dataset's accompanying documentation to untangle. The data is an incomprehensible block of codes with no meaningful explanations:

A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 ...
A12 48 A32 A43 5951 A61 A73 2 A92 A101 2 ...
A14 12 A34 A46 2096 A61 A74 2 A93 A101 3 ...

Working with relational databases

Working with relational databases In many production environments, the data we get in a relational or SQL database, not in files. Public data is often in files (as they are easier to share), but our most important client data is often in databases. Relational databases scale easily to the millions of records and supply important production features such as parallelism, consistency, transactions, logging, and audits. When we are working with transaction data, we are likely to find it already stored in a relational database, as relational databases excel at online transaction processing (OLTP).

When we can export the data into a structured file and use the methods then transfer the data into R. But this is generally not the right way to do things. Exporting from databases to files is often unreliable and different due to variations in database tools and the typically poor job these tools do when quoting and escaping characters that are confused with field separators. Data in a database is often stored in a normalized form, which requires relational preparations called joins before the data is ready for analysis.

1.Staging the data into a database:

Structured data at a scale of millions of rows is best handled in a database. When we are working with text-processing tools, but a database is much better at representing the fact that our data is arranged in both rows and columns (not just lines of text).

2.Loading data from a database into R:

To load data from a database, we use a database connector. Then we can directly issue SQL queries from R. SQL is the most common database query language and allows us to specify arbitrary joins and aggregations. SQL is called a declarative language (as opposed to a procedural language) because in SQL we specify what relations we would like our data sample to have, not how to compute them.

Exploring data

Data Exploration is about describing the data by means of statistical and visualization techniques. Explore data in order to bring important aspects of that data into focus for further analysis. There are various techniques

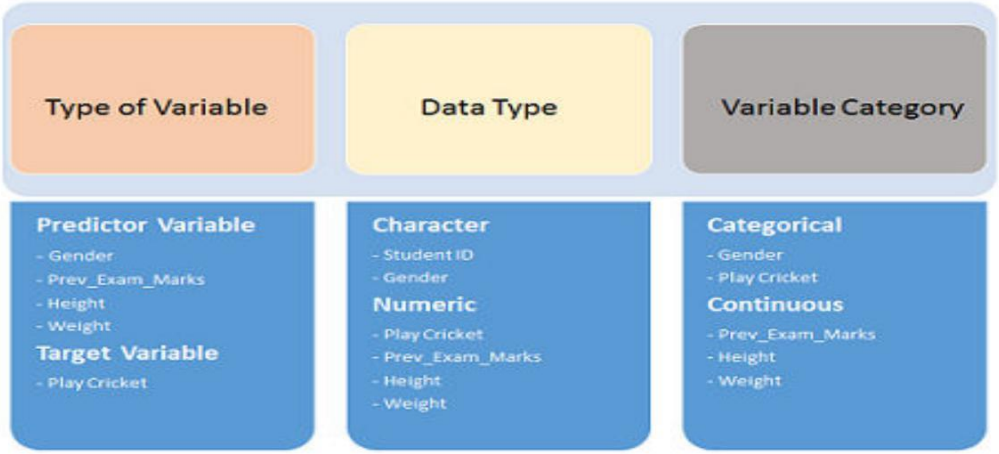
1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Missing values treatment
5. Outlier treatment
6. Variable transformation
7. Variable creation

1.Variable Identification:

First, identify Predictor (Input) and Target (output) variables. Next, identify the data type and category of the variables.

Example: Suppose, we want to predict, whether the students will play cricket or not (refer below data set). Here you need to identify predictor variables, target variable, data type of variables and category of variables.

Student_ID	Gender	Prev_Exam_Marks	Height (cm)	Weight Caregory (kgs)	Play Cricket
S001	M	65	178	61	1
S002	F	75	174	56	0
S003	M	45	163	62	1
S004	M	57	175	70	0
S005	F	59	162	67	0



2. Univariate Analysis:

Univariate analysis explores variables (attributes) one by one. Variables could be either categorical or numerical. There are different statistical and visualization techniques of investigation for each type of variable. Numerical variables can be transformed into categorical counterparts by a process called binning or discretization. It is also possible to transform a categorical variable into its numerical counterpart by a process called encoding. Finally, proper handling of missing values is an important issue in mining data.

a.Categorical Variables:

A categorical or discrete variable is one that has two or more categories (values). There are two types of categorical variable, nominal and ordinal. A nominal variable has no intrinsic ordering to its categories.

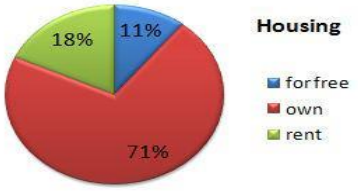
For example, gender is a categorical variable having two categories (male and female) with no intrinsic ordering to the categories. An ordinal variable has a clear ordering.

Univariate Analysis - Categorical		
Statistics	**Visualization**	**Description**
Count	Bar Chart	The number of values of the specified variable.
Count%	Pie Chart	The percentage of values of the specified variable.

Example: The housing variable with three categories (for free, own and rent).

Frequency Table

Housing	Count	Count%
for free	96	10.67%
own	641	71.22%
rent	163	18.11%



b.Numerical Variables:

First, identify Predictor (Input) and Target (output) variables. Next, identify the data type and category of the variables.

Example: Suppose, we want to predict, whether the students will play cricket or not (refer below data set). Here we need to identify predictor variables, target variable, data type of variables and category of variables.

3.Bivariate Analysis:

Bivariate analysis is the simultaneous analysis of two variables (attributes). It explores the concept of relationship between two variables, whether there exists an association and the strength of this association, or whether there are differences between two variables and the significance of these differences. There are three types of bivariate analysis.

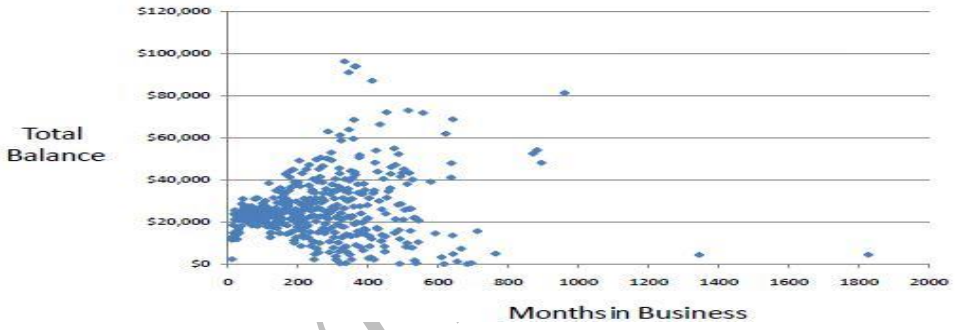
a. Numerical & Numerical
b. Categorical & Categorical
c. Numerical & Categorical

a. Numerical & Numerical

i. Scatter plot:

A scatter plot is a useful visual representation of the relationship between two numerical variables (attributes) and is usually drawn before working out a linear correlation or fitting a regression line. The resulting pattern indicates the type (linear or non-linear) and strength of the relationship between two variables. More information can be added to a two-dimensional scatter plot,

For example: Label points with a code to indicate the level of a third variable. If we are dealing with many variables in a data set, a way of presenting all possible scatter plots of two variables at a time is in a scatter plot matrix.



ii.Linear Correlation:

Linear correlation quantifies the strength of a linear relationship between two numerical variables. When there is no correlation between two variables, there is no tendency for the values of one quantity to increase or decrease with the values of the second quantity.

$$r = \frac{Covar(x,y)}{\sqrt{Var(x)Var(y)}}$$

$$Covar(x,y) = \frac{\Sigma(x-\bar{x})(y-\bar{y})}{n}$$

$$Var(x) = \frac{\Sigma(x-\bar{x})^2}{n}$$

$$Var(y) = \frac{\Sigma(y-\bar{y})^2}{n}$$

r : Linear Correlation

$Covar$: Covariance

Var : Variance

R only measures the strength of a linear relationship and is always between -1 and 1 where -1 means perfect negative linear correlation and +1 means perfect positive linear correlation and zero means no linear correlation.

Temperature	83	64	72	81	70	68	65	75	71	85	80	72	69	75
Humidity	86	65	90	75	96	80	70	80	91	85	90	95	70	70

	Variance	Covariance	Correlation
Temperature	40.10	19.78	**0.32**
Humidity	98.23		

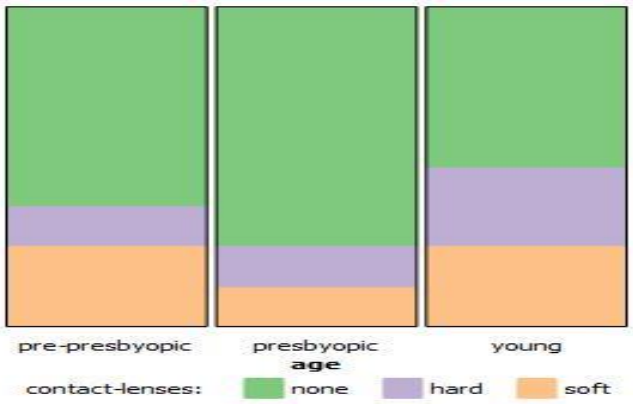
There is a weak linear correlation between Temperature and Humidity.

Prepare

b.Categorical & Categorical:

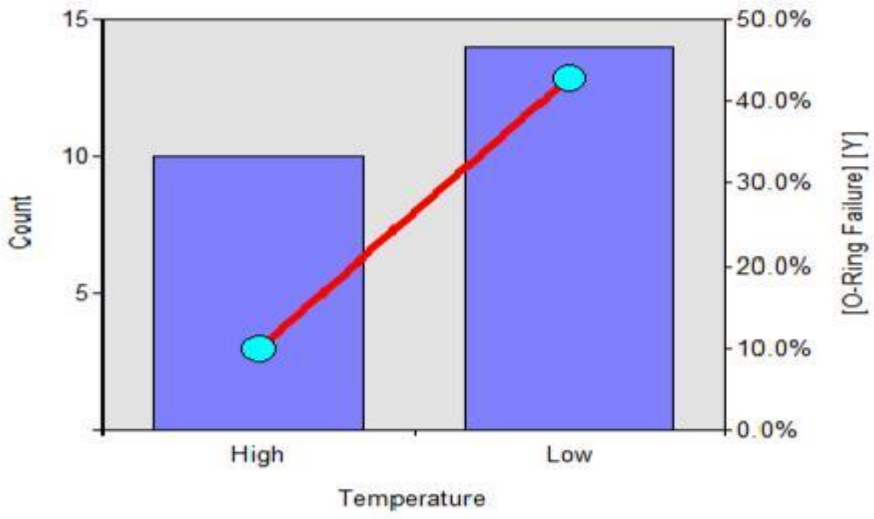
i.Stacked Column Chart:

Stacked Column chart is a useful graph to visualize the relationship between two categorical variables. It compares the percentage that each category from one variable contributes to a total across categories of the second variable.



ii.Combination Chart:

A combination chart uses two or more chart types to emphasize that the chart contains different kinds of information. Here, we use a bar chart to show the distribution of one categorical variable and a line chart to show the percentage of the selected category from the second categorical variable. The combination chart is the best visualization method to demonstrate the predictability power of a predictor (X-axis) against a target (Y-axis).



iii.Chi-square Test:

The chi-square test can be used to determine the association between categorical variables. It is based on the difference between the expected frequencies (e) and the observed frequencies (n) in one or more categories in the frequency table. The chi-square distribution returns a probability for the computed chi-square and the degree of freedom. A probability of zero shows a complete dependency between two categorical variables and a probability of one

means that two categorical variables are completely independent. Tchouproff Contingency Coefficient measures the amount of dependency between two categorical variables.

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(n_{ij}-e_{ij})^2}{e_{ij}}$$

Tchouproff Contingency Coefficient

$$\rho_\tau = \sqrt{\frac{\chi^2}{n\sqrt{(c-1)(r-1)}}}$$

$$e_{ij} = \frac{n_{i.}n_{.j}}{n}$$

$$df = (r-1)(c-1)$$

Example:The following frequency table (contingency table) with a chi-square of 10.67, degree of freedom (df) of 2 and probability of 0.005 shows a significant dependency between two categorical variables (hair and eye colors).

e=(44 * 52)/95=24.1

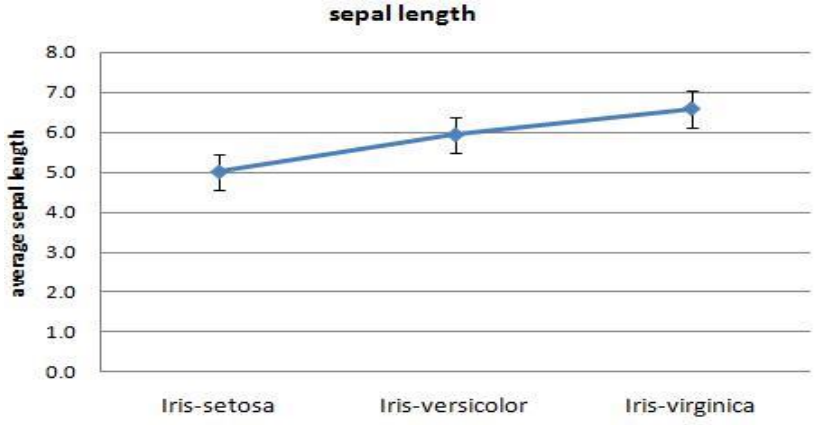
Eye	Hair: Light	Hair: Dark	
Black	32 (24.1)	12 (19.9)	44
Green/Blue	14 (19.7)	22 (16.3)	36
Others	6 (8.2)	9 (6.8)	15
	52	43	95

$$\chi^2 = 10.67$$
$$df = (r-1)(c-1) = (3-1)(2-1) = 2$$
$$p = 0.005$$
$$\rho_\tau = \sqrt{\frac{10.67}{95\sqrt{(3-1)(2-1)}}} = 0.28$$

c.Numerical & Categorical:

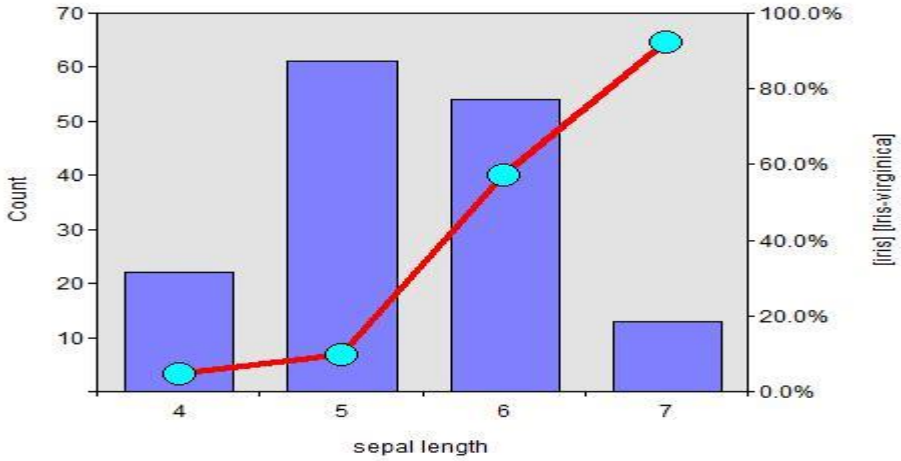
i.Line Chart with Error Bars:

A line chart with error bars displays information as a series of data points connected by straight line segments. Each data point is average of the numerical data for the corresponding category of the categorical variable with error bar showing standard error. It is a way to summarize how pieces of information are related and how they vary depending on one another (iris_linechart.xlsx).



ii.Combination Chart:

A combination chart uses two or more chart types to emphasize that the chart contains different kinds of information. Here, we use a bar chart to show the distribution of a binned numerical variable and a line chart to show the percentage of the selected category from the categorical variable. The combination chart is the best visualization method to demonstrate the predictability power of a predictor (X-axis) against a target (Y-axis).



iii.Z-test and t-test:

Z-test and t-test are basically the same. They assess whether the averages of two groups are statistically different from each other. This analysis is appropriate for comparing the averages of a numerical variable for two categories of a categorical variable.

$$Z = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}}}$$

where:

- $\overline{X}_1, \overline{X}_2$: Averages
- S_1^2, S_2^2: Variances
- N_1, N_2: Counts
- Z: Standard Normal Distribution

If the probability of Z is small, the difference between two averages is more significant.

t-test:

When the n1 or n2 is less than 30 we use the t-test instead of the Z-test.

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{S^2\left(\frac{1}{N_1} + \frac{1}{N_2}\right)}}$$

where:

- $\overline{X}_1, \overline{X}_2$: Averages
- S_1^2, S_2^2: Variances
- N_1, N_2: Counts
- t: has t distribution with $N_1 + N_2 - 2$ degree of freedom

$$S^2 = \frac{(N_1 - 1)S_1^2 + (N_2 - 1)S_2^2}{N_1 + N_2 - 2}$$

Example:There is a significant difference between the means (averages) of the numerical variable (Temperature) in two different categories of the categorical variable (O-Ring Failure). O-Ring Failure Temperature

53 56 57 70 70 70 75

63 66 67 67 67 68 69 70 72 73 75 76 76 78 79 80 81

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{S^2\left(\frac{1}{N_1} + \frac{1}{N_2}\right)}}$$

where:

- $\overline{X}_1, \overline{X}_2$: Averages
- S_1^2, S_2^2: Variances
- N_1, N_2: Counts
- t: has t distribution with $N_1 + N_2 - 2$ degree of freedom

$$S^2 = \frac{(N_1 - 1)S_1^2 + (N_2 - 1)S_2^2}{N_1 + N_2 - 2}$$

The low probability (0.0156) means that the difference between the average temperature for failed O-Ring and the average temperature for intact O-Ring is significant.

iv.Analysis of Variance (ANOVA):

The ANOVA test assesses whether the averages of more than two groups are statistically different from each other. This analysis is appropriate for comparing the averages of a numerical variable for more than two categories of a categorical variable.

Source of Variation	Sum of Squares	Degree of Freedom	Mean Square	F	P
Between Groups	SS_B	df_B	$MS_B = SS_B/df_B$	$F=MS_B/MS_W$	P(F)
Within Groups	SS_W	df_w	$MS_W = SS_W/df_w$		
Total	SS_T	df_T			

Example: There is a significant difference between the averages of the numerical variable (Humidity) in the three categories of the categorical variable (Outlook).

Outlook		Humidity			
Overcast	86	65	90	75	
rainy	96	80	70	80	91
sunny	85	90	95	70	70

Outlook	Count	Mean	Variance
overcast	4	75	57.5
rainy	5	69.8	10.96
sunny	5	76.2	32.56

Source of Variation	Sum of Squares	Degree of freedom	Mean Square	F Value	Probability
Between Groups	113.83	2	56.91	1.3987	0.288
Within Groups	447.60	11	40.69		
Total	561.43	13			

There is no significant difference between the averages of Humidity in the three categories of Outlook.

4.Missing values treatment:

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analyzed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55		Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57		Y
Mr. Kunal	57	M	N

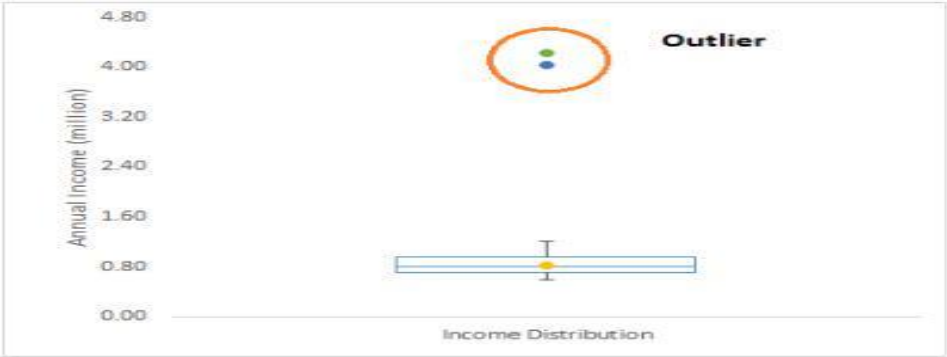
Name	Weight	Gender	Play Cricket/ Not
Mr. Amit	58	M	Y
Mr. Anil	61	M	Y
Miss Swati	58	F	N
Miss Richa	55	F	Y
Mr. Steve	55	M	N
Miss Reena	64	F	Y
Miss Rashmi	57	F	Y
Mr. Kunal	57	M	N

Notice the missing values in the image shown above: In the left scenario, we have not treated missing values. The inference from this data set is that the chances of playing cricket by males is higher than females. On the other hand, if you look at the second table, which shows data after treatment of missing values (based on gender), we can see that females have higher chances of playing cricket compared to males.

5.Outlier treatment:

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

Example: Customer profiling and find out that the average annual income of customers is $0.8 million. But, there are two customers having annual income of $4 and $4.2 million. These two customers annual income is much higher than rest of the population. These two observations will be seen as Outliers.



6.Variable transformation:

In data modelling, transformation refers to the replacement of a variable by a function. For instance, replacing a variable x by the square / cube root or logarithm x is a transformation. In other words, transformation is a process that changes the distribution or relationship of a variable with others.

7.Variable creation:

Variable creation is a process to generate a new variables / features based on existing variable(s).

For example, we have date(dd-mm-yy) as an input variable in a data set. We can generate new variables like day, month, year, week, weekday that may have better relationship with target variable. This step is used to highlight the hidden relationship in a variable:

Emp_Code	Gender	Date	New_Day	New_Month	New_Year
A001	Male	21-Sep-11	21	9	2011
A002	Female	27-Feb-13	27	2	2013
A003	Female	14-Nov-12	14	11	2012
A004	Male	07-Apr-13	7	4	2013
A005	Female	21-Jan-11	21	1	2011
A006	Male	26-Apr-13	26	4	2013
A007	Male	15-Mar-12	15	3	2012

Managing data or Manage Data:

Good data management includes developing effective processes for consistently collecting and recording data, storing data securely, backing up data, cleaning data, and modifying data so it can be transferred between different types of software for analysis.
Good data management is inextricably linked to data quality assurance–the processes and procedures that are used to ensure data quality. Using data of unknown or low quality may result in making the wrong decisions about policies and programmes. Data quality assurance (DQA) should be built into each step in the data cycle – data collection, aggregation and reporting, analysis and use, and dissemination and feedback.

Even when data have been collected using well-defined procedures and standardized tools, they need to be checked for any inaccurate or missing data. This "data cleaning" involves finding and dealing with any errors that occur during writing, reading, storage, transmission, or processing of computerized data.

Ensuring data quality also extends to presenting the data appropriately in the evaluation report so that the findings are clear and conclusions can be substantiated. Often, this involves making the data accessible so that they can be verified by others and/or used for additional purposes such as for synthesizing results across different evaluations. Commonly referred to aspects of data quality are:

- **Validity:** The degree to which the data measure what they are intended to measure.
- **Reliability:** Data are collected consistently; definitions and methodologies are the same when doing repeated measurements over time.
- **Completeness:** Data are complete (i.e., no missing data or data elements).
- **Precision:** Data have sufficient detail.
- **Integrity:** Data are protected from deliberate bias or manipulation for political or personal reasons
- **Availability:** Data are accessible so they can be validated and used for other purposes.
- **Timeliness:** Data are up-to-date current and available on time. Options
- **Consistent Data Collection and Recording:** processes to ensure data are collected consistently across different sites and different data collectors.
- **Data Backup:** onsite and offsite, automatic and manual processes to guard against the risk of data being lost or corrupted.
- **Data Cleaning:** detecting and removing (or correcting) errors and inconsistencies in a data set or database due to the corruption or inaccurate entry of the data.
- **Effective Data Transfer:** processes to move data between systems, including between software packages, to avoid the need to rekey data.
- **Secure Data Storage:** processes to protect electronic and hard copy data in all forms, including questionnaires, interview tapes and electronic files from being accessed without authority or damaged.
- **Archive Data for Future Use:** systems to store de-identified data so that they can be accessed for verification purposes or for further analysis and research in the future.

Data Cleaning

Data cleansing is the process of detecting and correcting data quality issues. It typically includes both automatic steps such as queries designed to detect broken data and manual steps such as data wrangling.
Examples:

- **Corrupt Data** – corrupt data due to bad transfers
- **Inconsistent Data** – Differences in values in different tables.
  For eg. age is different for same person
- **Inaccurate Data** – inaccuracies because of improper maintenances For eg. Apple Phone is termed Sony
- **Irrelevant Data** – data out of context and purpose.
  For eg. personal information for giving grades
- **Dirty Data** - variety of data improperly handles.
  For eg. pdf, doc, html, json etc.
- **Typographical Errors** – eg. Bengaluru and Bangalore Standardization – eg. Percentages and Grades
- **Referential Integrity** – eg. migration of old data to new platform
- **Completeness** – eg. filling missing postal codes by using telephone numbers
- **Characters** – eg. Special Characters create problems while parsing

Sampling for modeling and validation:

Sampling is the process of selecting a subset of a population to represent the whole, during analysis and modeling. In the current era of big datasets, some people argue that computational power and modern algorithms are analyze the entire large dataset without the need to sample.

When we are in the middle of developing or refining a modeling procedure, it is easier to test and debug the code on small subsamples before training the model on the entire dataset. Visualization can be easier with a subsample of the data; ggplot runs faster on smaller datasets, and too much data will often unclear the patterns in a graph.

1.Test and training splits:

The data used to build the final model usually comes from multiple datasets. There are three data sets commonly used in different stages of the creation of the model.

- **Train Dataset**
  - ✓ Used for training algorithms.
  - ✓ Labels for records are used
- **Validation Dataset:**
  - ✓ Used in iterative process
  - ✓ Used to check accuracy in each training step
- **Test Dataset**
  - ✓ To measure performance of final model
  - ✓ Can be used only once
  - ✓ Should not be touched till final model

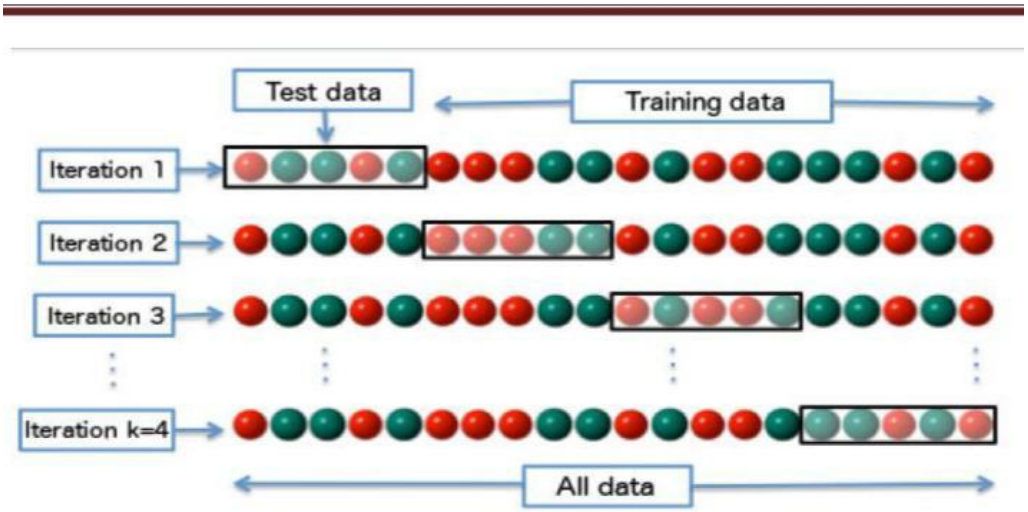
2.Creating a sample group column:

A convenient way to manage random sampling is to add a sample group column to the data frame. The sample group column contains a number generated uniformly from zero to one, using the runif function.

3.Validation:

Cross Validation is a technique which involves reserving a particular sample of a data set on which you do not train the model. Later, we test the model on this sample before finalizing the model. There are various steps involved in cross validation:

- We reserve a sample data set.
- Train the model using the remaining part of the data set.
- Use the reserve sample of the data set test (validation) set.



Introduction to NoSQL:

1.History of NoSQL:

The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.In the early 2009, when last.fm wanted to organize an event on open-source distributed databases, Eric Evans, a Rackspace employee, reused the term to refer databases which are non-relational, distributed, and does not conform to atomicity, consistency, isolation, durability - four obvious features of traditional relational database systems.

2.Definition of NoSQL:

NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways. It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users). These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

3.Uses of NoSQL:

In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others. Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially. To avail the above service properly, it is required to process huge amount of data. The evolution of NoSql databases is to handle these huge data properly.

4.NoSQL pros/cons:

Advantages :

- ✓ High scalability
- ✓ Distributed Computing
- ✓ Lower cost
- ✓ Schema flexibility, semi-structure data
- ✓ No complicated Relationships
- ✓ Disadvantages:
- ✓ No standardization
- ✓ Limited query capabilities (so far)
- ✓ Eventual consistent is not intuitive to program for

4.NoSQL Categories

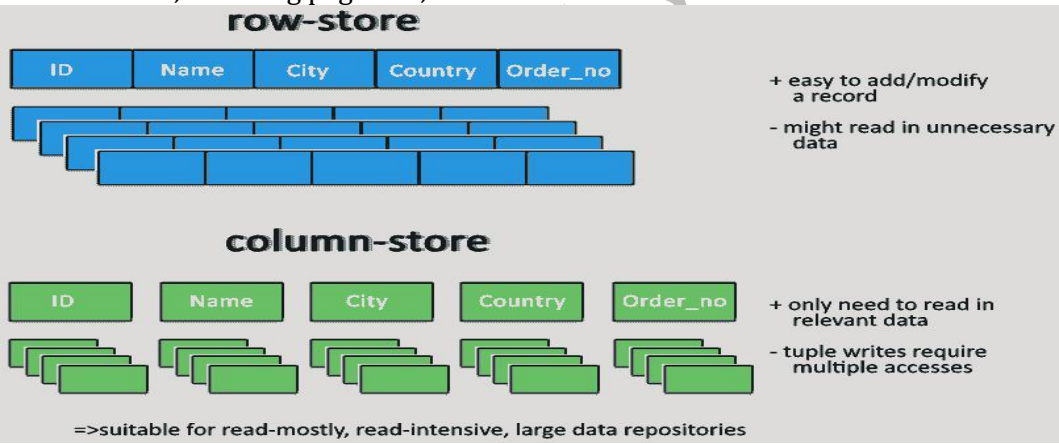
There are four general types (most common categories) of NoSQL databases. Each of these categories has its own specific attributes and limitations. There is not a single solutions

which is better than all the others,however there are some databases that are better to solve specific problems.
The most common categories of NoSQL databases are:

- Key-value stores
- Column-oriented
- Graph
- Document oriented

a.Key-value stores

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data.
- Based on Amazon's Dynamo paper.
- Key value stores allow developer to store schema-less data.
- In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON, BLOB (Binary Large OBjec) etc.
- A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
  For example a key-value pair might consist of a key like "Name" that is associated with a value like "Robin".
- Key-Value stores can be used as collections, dictionaries, associative arrays etc.
- Key-Value stores follow the 'Availability' and 'Partition' aspects of CAP theorem.
- Key-Values stores would work well for shopping cart contents, or individual values like color schemes, a landing page URI, or a default account number.

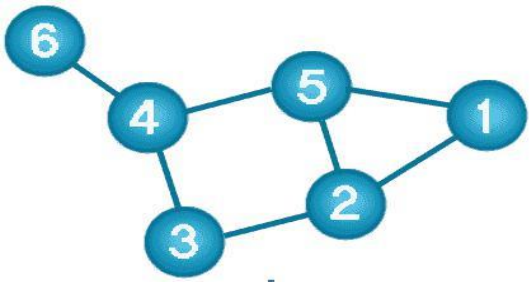


b. Column-oriented databases:

- ✓ Column-oriented databases primarily work on columns and every column is treated individually.
- ✓ Values of a single column are stored contiguously.
- ✓ Column stores data in column specific files.
- ✓ In Column stores, query processors work on columns too.
- ✓ All data within each column datafile have the same type which makes it ideal for compression.
- ✓ Column stores can improve the performance of queries as it can access specific column data.
- ✓ High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).
- ✓ Works on data warehouses and business intelligence, customer relationship management (CRM), Library card catalogs etc.

c.Graph databases:

A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called edges or arcs, of certain entities called nodes or vertices. The following picture presents a labeled graph of 6 vertices and 7 edges.



d. Document Oriented databases:

- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.
- Documents are stored into collections in order to group different kinds of data.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

5. RDBMS vs NoSQL

RDBMS

- Structured and organized data
- Structured query language (SQL)
- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency

NoSQL

- Stands for Not Only SQL
- No declarative query language
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data
- CAP Theorem
- Prioritizes high performance, high availability and scalability
- BASE Transaction

UNIT-2
Modelling Methods

Choosing and Evaluating models – Mapping problems to machine learning, Evaluating clustering models , Validating models –Cluster analysis - K-means algorithm, Naive Bayes – Memorization methods – Linear and Logistic regression –Unsupervised methods

1.Choosing and Evaluating Models

As a data scientist, the ultimate goal is to solve a concrete business problem: increase look-to-buy ratio, identify fraudulent transactions, predict and manage the losses of a loan portfolio, and so on. Many different statistical modeling methods can be used to solve any given problem. Each statistical method will have its advantages and disadvantages for a given business goal and business constraints.

To make progress, must be able to measure model quality during training and also ensure that model will work as well in the production environment as on training data. In general, there are two tasks. They are model evaluation, model validation.

To prepare for these statistical tests, always split our data into training data and test data, as illustrated in figure

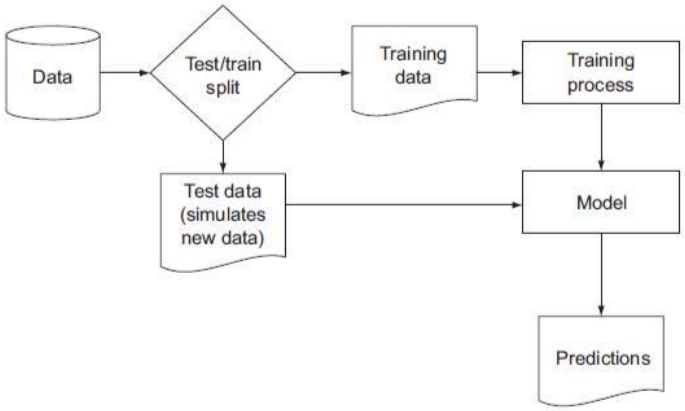


Figure 5.1 **Schematic model construction and evaluation**

Model evaluation as quantifying the performance of a model. In this we must find a measure of model performance that is appropriate to both the original.

Model validation as the generation of an assurance that the model will work in production as well as it worked during training. business goal and the chosen modeling technique.

2.Mapping problems to machine learning:

The task is to map a business problem to a good machine learning method. There are a number of business problems that our team might be called on to address:

- Predicting what customers might buy, based on past transactions Identifying fraudulent transactions
- Determining price elasticity (the rate at which a price increase will decrease sales, and vice versa) of various products or product classes
- Determining the best way to present product listings when a customer searches for an item
- Customer segmentation: grouping customers with similar purchasing behavior
- AdWord valuation: how much the company should spend to buy certain. AdWords on search engines
- Evaluating marketing campaigns Organizing new products into a product catalog Solving Classification Problems
- Solving Scoring Problems

1.Linear regression:

Linear regression builds a model such that the predicted numerical output is a linear additive function of the inputs. This can be a very effective approximation, even when the underlying situation is in fact nonlinear. The resulting model also gives an indication of the relative impact of each input variable on the output. Linear regression is good first model when trying to predict a numeric value.

2.Logistic regression

Logistic regression always predicts a value between 0 and 1, making it suitable for predicting probabilities (when the observed outcome is a categorical value) and rates (when the observed outcome is a rate or ratio). Logistic regression is an appropriate approach to the fraud detection problem.

3. Working without known targets:

The preceding methods require that we have a training dataset of situations with known outcomes. In some situations, these are not (yet) a specific outcome that we want to predict. Instead, we may be looking for patterns and relationships in the data that can be help for understand our customers or business better. These situations correspond to a class of approaches called unsupervised learning rather than predicting outputs based on inputs, the objective of unsupervised learning is to discover similarities and relationships in the data. Some common clustering methods include these:

- K-means clustering
- Apriori algorithm for finding association rules
- nearest neighbour

3.Evaluating clustering models

Clustering models are hard to evaluate because they are unsupervised. The clusters that items are assigned to and generated by the modeling procedure, not supplied in a series of annotated examples. Evaluation is largely checking observable summaries about the clustering. For Example: Evaluating a division of 100 random points in a plane into five clusters.

1.Intra-cluster distances versus cross-cluster distances:

The traditional measure of comparing the typical distance between two items in the same cluster to the typical distance between two items from different clusters. For Example: The mean distance from points in one cluster to points in another.

2.Treating clusters as classifications or scores:

Distance metrics are good for checking the performance of clustering algorithms, but they don't always translate to business needs. When sharing a clustering with our project sponsor or client, we advise treating the cluster assignment as if it were a classification.

For each cluster label, generate an outcome assigned to the cluster (such as all email in the cluster is marked as spam/non-spam, or all accounts in the cluster are treated as having a revenue value equal to the mean revenue value in the cluster).Then use either the classifier or scoring model evaluation ideas to evaluate the value of the clustering.

4.Validating models

We choose a modelling technique and evaluate the performance of the model on training data. The testing of a model on new data (or a simulation of new data from our test set) is called model validation.

1.Identifying common model problems:

some list of common modelling problems are:

- **Bias:** Systematic error in the model, such as always under predicting.
- **Variance:** Undesirable (but non-systematic) distance between predictions and Actual values. Often due to oversensitivity of the model training procedure to small variations in the distribution of training data.
- **Over fitting:** A lot of modelling problems are related to over fitting.
  For Example: diagnosing models.

An overfit model looks great on the training data and performs poorly on new data. A model's prediction error on the data that it trained from is called training error. A model's prediction error on new data is called generalization error. Usually, training error will be smaller than generalization error (no big surprise). Ideally, though, the two error rates should be close. If generalization error is large, then the model has probably over fit—it's memorized the training data instead of discovering generalizable rules or patterns.

- **Non-significance:** A model that appears to show an important relation when in fact the relation may not hold in the general population, or equally good predictions can be made without the relation.

2.Quantifying model soundness:

A single evaluation of model performance gives only a point estimate of performance. We need a good characterization of how much potential variation there is in our model production and measurement procedure, and how well our model is likely to perform on future data.

3.Ensuring model quality:

The standards of scientific presentation are always share variations on data and procedures.

For example: The likely distribution of the statistics under variations on modeling procedure or data. We say something like "We have an accuracy of 90% on our training data," but instead we'd run additional experiments or "We see an accuracy of 85% on hold-out data." Or even better: "We saw accuracies of at least 80% on all but 5% of our returns."

a. Testing on held-out data:

Testing on hold-out data, while useful, only gives a single-point estimate of model performance.

b. K-FOLD Cross-Validation:

The k-fold cross-validation is to repeat the construction of the model on different subsets of the available training data and then evaluate the model only on data not seen during construction.

c. Significance testing

Statisticians have a powerful idea related to cross-validation called significance testing.

d. Confidence intervals:

An important and very technical frequentist statistical concept is the confidence interval.

For Example: a 95% confidence interval is an interval from an estimation procedure such that the procedure is thought to have a 95% (or better) chance of catching the true unknown value to be estimated in an interval. It is not the case that there is a 95% chance that the unknown true value is actually in the interval at hand.

5.Cluster Analysis

In cluster analysis, the goal is to group the observations in our data into *clusters* such that every datum in a cluster is more similar to other datum's in the same cluster than it is to datum's in other clusters.

For example, a company that offers guided tours might want to cluster its clients by behaviour and tastes: which countries they like to visit; whether they prefer adventure tours, luxury tours, or educational tours;

Distances :

In order to cluster, you need the notions of similarity and dissimilarity. Dissimilarity can be thought of as distance, so that the points in a cluster are closer to each other than they are to the points in other clusters.

Different application areas will have different notions of distance and dissimilarity. A few of the most common ones:

- Euclidean distance

- Hamming distance
- Manhattan (city block) distance
- Cosine similarity

EUCLIDEAN DISTANCE :

The most common distance is Euclidean distance. The Euclidean distance between two vectors x and y is defined as

edist(x, y) <- sqrt((x[1]-y[1])^2 + (x[2]-y[2])^2 + ...)

This is the measure people tend to think of when they think of "distance." Optimizing squared Euclidean distance is the basis of k-means. Euclidean distance only makes sense when all the data is real-valued (quantitative). If the data is categorical (in particular, binary), then other distances can be used.

HAMMING DISTANCE :

For categorical variables (male/female, or small/medium/large), you can define the distance as 0 if two points are in the same category, and 1 otherwise. If all the variables are categorical, then you can use Hamming distance, which counts the number of mismatches:

hdist(x, y) <- sum((x[1] != y[1]) + (x[2] != y[2]) + ...)

Here, a != b is defined to have a value of 1 if the expression is true, and a value of 0 if the expression is false.

MANHATTAN (CITY BLOCK) DISTANCE :

Manhattan distance measures distance in the number of horizontal and vertical units it takes to get from one (real-valued) point to the other (no diagonal moves):

mdist(x, y) <- sum(abs(x[1]-y[1]) + abs(x[2]-y[2]) + ...)

This is also known as L1 distance (and squared Euclidean distance is L2 distance).

COSINE SIMILARITY :

Cosine similarity is a common similarity metric in text analysis. It measures the smallest angle between two vectors (the angle theta between two vectors is assumed to be between 0 and 90 degrees). Two perpendicular vectors (theta = 90 degrees) are the most dissimilar; the cosine of 90 degrees is 0. Two parallel vectors are the most similar(identical, if you assume they're both based at the origin); the cosine of 0 degrees is 1.

From elementary geometry, you can derive that the cosine of the angle between two vectors is given by the normalized dot product between the two vectors:

dot(x, y) <- sum(x[1]*y[1] + x[2]*y[2] + ...)

cossim(x, y) <- dot(x, y)/(sqrt(dot(x,x)*dot(y,y)))

Preparing the data :

To demonstrate clustering, we'll use a small dataset from 1973 on protein consumption from nine different food groups in 25 countries in Europe.1 The goal is to group the countries based on patterns in their protein consumption. The dataset is loaded into R as a data frame called protein, as shown in the next listing.

UNITS AND SCALING :

The documentation for this dataset doesn't mention what the units of measurement are, though we can assume all the columns are measured in the same units. This is important: units (or more precisely, disparity in units) affect what clusterings an algorithm will discover. You can scale the data in R using the function scale().

Hierarchical clustering with hclust() :

The hclust() function takes as input a distance matrix (as an object of class dist), which records the distances between all pairs of points in the data (using any one of a variety of metrics). It returns a dendrogram: a tree that represents the nested clusters.

hclust() uses one of a variety of clustering methods to produce a tree that records the nested cluster structure. You can compute the distance matrix using the function dist().

dist() will calculate distance functions using the (squared) Euclidean distance (method="euclidean"), the Manhattan distance (method="manhattan"), and something like the Hamming distance, when categorical variables are expanded to indicators (method="binary"). If

you want to use another distance metric, you'll have to compute the appropriate distance matrix and convert it to a dist object using the as.dist() call (see help(dist) for further details).

```
d <- dist(pmatrix, method="euclidean")
pfit <- hclust(d, method="ward")
plot(pfit, labels=protein$Country)
```

The dendrogram suggests five clusters .You can draw the rectangles on the dendrogram using the function rect.hclust():

```
rect.hclust(pfit, k=5)
```

6.K-Means Algorithm

K-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as bary center of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where,
'||xi - vj||' is the Euclidean distance between xi and vj.
'ci' is the number of data points in ith cluster.
'c' is the number of cluster centers.

Algorithmic steps for k-means clustering:

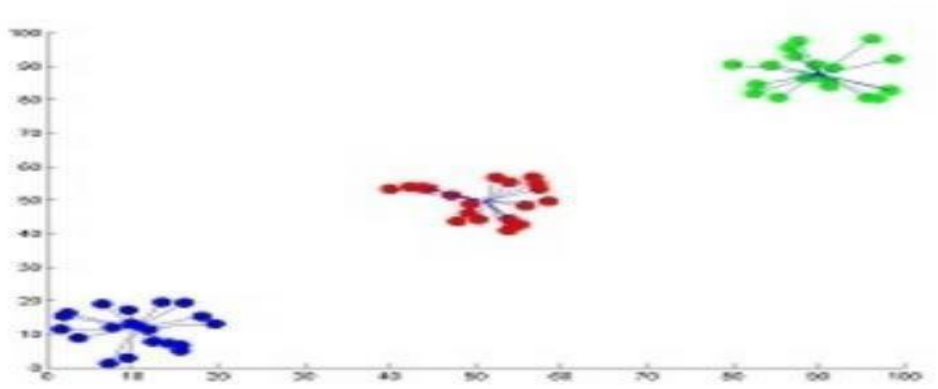
- Let X = {x1,x2,x3,……..,xn} be the set of data points and
- V = {v1,v2,……..,vc} be the set of centers.
- Randomly select 'c' cluster centers.
- Calculate the distance between each data point and cluster centers.
- Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
- Recalculate the new cluster center using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i$$

  where, 'ci' represents the number of data points in ith cluster.
- Recalculate the distance between each data point and new obtained cluster centers.
- If no data point was reassigned then stop, otherwise repeat from step 3).

Advantages

- Fast, robust and easier to understand.
- Relatively efficient.
- Gives best result when data set are distinct or well separated from each other.



Disadvantages

- The learning algorithm requires apriori specification of the number of cluster centers.
- If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
- The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- Euclidean distance measures can unequally weight underlying factors.
- The learning algorithm provides the local optima of the squared error function.
- Randomly choosing of the cluster center cannot lead us to the fruitful result.
- Applicable only when mean is defined i.e. fails for categorical data.
- Unable to handle noisy data and outliers.
- Algorithm fails for non-linear data set.

7.Naïve Bayes:

The Naive Bayesian classifier is based on Bayes' theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

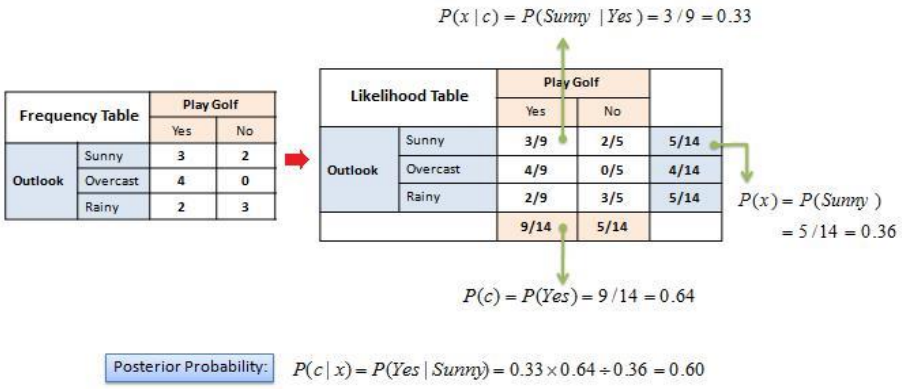
Algorithm:

Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assume that the effect of the value of a predictor on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

- P(c|x) is the posterior probability of class (target) given predictor (attribute). P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class. P(x) is the prior probability of predictor.

Example:

The posterior probability can be calculated by first, constructing a frequencytable for each attribute against the target. Then, transforming the frequency tables to likelihood tables and finally use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome .



The zero-frequency problem:

Add 1 to the count for every attribute value-class combination (Laplace estimator) when an attribute value (Outlook=Overcast) doesn't occur with every class value (Play Golf=no).

Numerical Predictors:

Numerical variables need to be transformed to their categorical counterparts (binning) before constructing their frequency tables. The other option we have using the distribution of the numerical variable to have a good guess of the frequency.

For example, one common practice is to assume normal distributions for numerical variables. The probability density function for the normal distribution is defined by two parameters (mean and standard deviation).

8.Memorization Methods:

The simplest methods in data science are called as memorization methods. These are the methods that generate answers by returning a majority category (in case of classification) or average value (in case of scoring) of a subset of the original training data. These methods can vary from models depending on a single variable (similar to the analyst's pivot table), to decision trees (similar to business rules), to nearest neighbour and Naive Bayes methods. These memorization methods used for solving classification problems. They are

1.KDD and KDD Cup 2009:

The Conference on Knowledge Discovery and Data Mining (KDD) is the premier conference on machine learning methods. Every year KDD hosts a data mining cup, where teams analyze a dataset and then are ranked against each other. The KDD Cup is a huge deal and the inspiration for the famous Netflix Prize and even Kaggle competitions.

The KDD Cup 2009 provided a dataset about customer relationship management. The contest supplied 230 facts about 50,000 credit card accounts. From these features, the goal was to predict account cancellation (called churn), the innate tendency to use new products and services (called appetency), and willingness to respond favorably to marketing pitches (called up selling).

In many score-based competitions, this contest concentrated on machine learning and deliberately abstracted or skipped over a number of important data science issues, such as defining goals, requesting new measurements, collecting data, and quantifying classifier performance in terms of business goals. For this contest we don't have names or definitions for any of the independent (or input) variables and no real definition of the dependent (or outcome) variables. The advantage is that the data is already in a ready-to-model format (all input variables and the results arranged in single rows).

2.Building single-variable models:

Single-variable models are simply models built using only one variable at a time.The single-variable models are build from both categorical and numeric variables. We should be able to build, evaluate, and cross-validate single-variable models with confidence.

a. Using categorical features:

A single-variable model based on categorical features is to describe as a table. For this the business analysts use as a pivot table (which promotes values or levels of a feature of new

columns) and statisticians use as a contingency table (where each possibility is given a column name).
Example: In both cases, the R command to produce a table is table().

b.Using numeric features

There are a number of ways to use a numeric feature to make predictions. A common method is to bin the numeric feature into a number of ranges and then use the range labels as a new categorical variable.
Example: quantile() and cut() commands in R.

c.Using cross-validation to estimate effects of over fitting

Cross-validation applies in all modeling situations. Using cross-validation to estimate effects of over fitting. The cross-validation is a procedure to estimate the degree of overfit which is hidden in our models.

In repeated cross-validation, a subset of the training data is used to build a model,and a complementary subset of the training data is used to score the model. Automatic cross-validation is extremely useful in all modeling situations.

3.Building models using many variables:

Models that combine the effects of many variables are much more powerful than models that use only a single variable. Some of the most fundamental multiple-variable models are: decision trees, nearest neighbour, and Naive Bayes.

a. Variable selection:

A key part of building many variable models is selecting what variables to use and how the variables are to be transformed or treated. When variables are available has a huge impact on model utility. Each variable use to represent a chance of explaining more of the outcome variation (a chance of building a better model) but also represent a possible source of noise and over fitting.

b.Using decision trees

Decision trees are a simple model type and we can view a decision tree as a procedure to split the training data into pieces and use a simple memorized constant on each piece. Decision trees (especially a type called classification and regression trees) can be used to quickly predict either categorical or numeric outcomes. The concept of decision trees is to think of them(categorical or numeric outcomes) as machine-generated business rules.

c.Using nearest neighbour methods:

A k-nearest neighbor (KNN) method scores an example(X) by finding the k training examples nearest to the example(X) and then taking the average of their outcomes as the score. The concept of nearness is basic Euclidean distance, so it can be useful to select nonduplicate variables, rescale variables, and orthogonalize variables.

d.Using Naive Bayes:

Naive Bayes is an interesting method that memorizes how each training variable is related to outcome, and then makes predictions by multiplying together the effects of each variable.

9.Linear and logistic regression:

Linear and logistic regression can also provide advice by quantifying the relationships between the outcomes and the model's inputs. Both linear and logistic regression assume that the outcome is a function of a linear combination of the inputs. Since the models are expressed completely by their coefficients, they're small, portable, and efficient—all valuable qualities when putting a model into production. If the model's errors are homoscedastic(sequence or vector of random variables), the model might be trusted to extrapolate predictions outside the training range. Extrapolation is never completely safe, but it's sometimes necessary.

1.Using linear regression:

Linear regression is the bread and butter prediction method for statisticians and data scientists. If we're trying to predict a numerical quantity like profit, cost, or sales volume, we should always try linear regression first. If it works well, we're done; if it fails, the detailed diagnostics produced give us a good clue as to what methods we should try next.

a. Understanding linear regression:

Linear regression models the expected value of a numeric quantity (called the dependent or response variable) in terms of numeric and categorical inputs (called the independent or explanatory variables).

For example: suppose we're trying to predict how many pounds a person on a diet and exercise plan will lose in a month.

b. Building a linear regression model:

The first step in either prediction or finding relations (advice) is to build the linear regression model. The lm() command is used to build the linear regression model.

c. Making predictions:

Once we have called lm() to build the model, our first goal is to predict income. To predict, we pass data into the predict() method. The dtest and dtrain data are used for both the test and training data frames.

d. Finding relations and extracting advice:

All of the information in a linear regression model is stored in a block of numbers called the coefficients. The coefficients are available through the coefficients(model) command.

e. Reading the model summary and characterizing coefficient quality:

It is to check whether model coefficients are reliable or not and shows the coefficients' relations. When presenting results, we demonstrate how all of these fields are derived and what the fields mean by using summary() command.

f. Linear regression takeaways:

- Linear regression is the go-to statistical modeling method for quantities.
- Linear regression will have trouble with problems that have a very large number of variables, or categorical variables with a very large number of levels
- Enhance linear regression by adding new variables or transforming variables.
- Linear regression can predict well even in the presence of correlated variables, but correlated variables lower the quality of the advice.
- Linear regression packages have some of the best built-in diagnostics available, but rechecking our model on test data is still the most effective safety check.

2. Logistic regression:

Logistic regression is the most important (and probably most used) member of a class of models called generalized linear models. Unlike linear regression, logistic regression can directly predict values that are restricted to the (0,1) interval, such as probabilities. It's the go-to method for predicting probabilities. Like linear regression, the coefficients of a logistic regression model can be treated as advice. It's also a good choice for binary classification problems.

a. Understanding logistic regression:

Like linear regression, logistic regression will findthe best coefficients to predict y, including finding advantageous combinations and cancellations when the inputs are correlated.

b. Building a logistic regression model:

The command to build a logistic regression modelin R is glm(). The (Generalized Linear Models) glm() will work with prediction numeric values between 0 and 1.0 in addition to predicting classifications.

c. Making predictions:

Making predictions with a logistic model is similar to making predictions with a linear model—use the predict() function.

d. Finding relations and extracting advice from logistic models:

The coefficients of a logistic regression model encode the relationships between the input variables and the output in a way similar to how the coefficients of a linear regression model. We can get the model's coefficients by using coefficients(model) command in R.

e. Reading the model summary and characterizing coefficients:

The coefficient values are only to be trusted if the coefficient values are statistically significant and determine some facts about model quality using summary(model) in R.

f. Logistic regression takeaways:

- Logistic regression is the go-to statistical modeling method for binary classification.
- Logistic regression will have trouble with problems with a very large number of variables, or categorical variables with a very large number of levels.
- Logistic regression is well calibrated: it reproduces the marginal probabilities of the data.
- Logistic regression can predict well even in the presence of correlated variables, but correlated variables lower the quality of the advice.
- glm() provides good diagnostics, but rechecking the model on test data is still most effective diagnostic.

10.Unsupervised methods:

Methods to discover unknown relationships in data are called unsupervised methods. With unsupervised methods, there's no outcome that we are trying to predict; instead, we want to discover patterns in the data.

For example, we may want to find groups of customers with similar purchase patterns, or correlations between population movement and socioeconomic factors. Unsupervised analyses are often not ends in themselves; rather, they're ways of finding relationships and patterns that can be used to build predictive models. There are two classes of unsupervised methods

1. **Cluster analysis:** finds groups in your data with similar characteristics.
2. **Association rule mining:** finds elements or properties in the data that tend to occur together.

1.Cluster analysis:

In cluster analysis, the goal is to group the observations in our data into clusters such that every datum(a piece of information)in a cluster is more similar to other datums in the same cluster than it is to datums in other clusters.

For example, a company that offers guided tours might want to cluster its clients by behaviour and tastes: which countries they like to visit; whether they prefer adventure tours, luxury tours, or educational tours; what kinds of activities they participate in; and what sorts of sites they like to visit. Such information can help the company design attractive travel packages and target the appropriate segments of their client base with them. Cluster analysis is a topic worthy of a book in itself.

Here we have two approaches

- Hierarchical clustering finds nested groups of clusters. An example of hierarchical clustering might be the standard plant taxonomy, which classifies plants by family, then genus, then species, and so on.
- k-means, which is a quick and popular way of finding clusters inquantitative data.

2. Association rules:

Association rule mining is used to find objects or attributes that frequently occur together.

For example: products that are often bought together during a shopping session, or queries that tend to occur together during a session on a website's search engine.

a. Overview of association rules:

The unit of "togetherness" when mining association rules is called a transaction.

Depending on the problem, a transaction could be a single shopping basket, a single user session on a website, or even a single customer. The objects that comprise a transaction are referred to as items in an item set: the products in the shopping basket, the pages visited during a website session, the actions of a customer. Sometimes transactions are referred to as baskets, from the shopping basket analogy.

b. Mining for association rules occurs in two steps:

- Look for all the item sets (subsets of transactions) that occur more often than in a minimum fraction of the transactions.
- Turn those item sets into rules.
- Mining association rules with the a rules package:
  a rules includes an implementation of the popular association rule algorithm apriori, as well as implementations to read in and examine transaction data. The package uses special data types to hold and manipulate the data.

c. Association rule takeaways:

- The goal of association rule mining is to find relationships in the data: items or attributes that tend to occur together.
- When a large number of different possible items can be in a basket (thousands of different books), most events will be rare.
- Association rule mining is often interactive, as there can be many rules to sort.

UNIT-3
Introduction to R Language

Reading and getting data into R - Ordered and Unordered factors - Arrays and Matrices - Lists and Data frames - Reading data from files

Introduction

- R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.
- The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
- R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- R is free software distributed under a GNU-style copy left, and an official part of the GNU project called GNU S .

3.1 Reading and getting data into R

If you do not have any data for analyse so we getting data into R is a very important task. This focuses on ways to create these complex samples and get data into R, where you are able to undertake further analyses.

Using the combine Command for Making Data

The simplest way to create a sample is to use the c() command. You can think of this as short for combine or concatenate, which is essentially what it does. The command takes the following form:

c(item.1, item.2, item.3, item.n)

Everything in the parentheses is joined up to make a single item. More usually you will assign the joined-up items to a named object:

sample.name = c(item.1, item.2, item.3, item.n)

This is much like you did when making simple result objects, except now your sample objects consist of several bits rather than a single value.

Entering Numerical Items as Data

Numerical data do not need any special treatment; you simply type the values, separated by commas, into the c() command.

In the following example, imagine that you have collected some data (a sample) and now want to get the values into R:

>data1 = c(3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9)

Now just create a new object to hold your data and then type the values into the parentheses. The values are separated using commas.

The “result” is not automatically displayed; to see the data you must type its name:

```
> data1
[1] 3 5 7 5 3 2 6 8 5 6 9
```

Previously the named objects contained single values

In the following example you can see that there are 41 values in the sample:

```
[1] 582 132 716 515 158 80 757 529 335 497 3369 746 201 277 593
[16] 361 905 1513 744 507 622 347 244 116 463 453 751 540 1950 520
[31] 179 624 448 844 1233 176 308 299 531 71 717
```

The second row starts with [16], which tells you that the first value in that row is the 16th in the sample. This simple index system makes it a bit easier to pick out specific items.

You can incorporate existing data objects with values to make new ones simply by incorporating them as if they were values themselves (which of course they are). In this

example you take the numerical sample that you made earlier and incorporate it into a larger sample:

```
> data1
[1] 3 5 7 5 3 2 6 8 5 6 9
> data2 = c(data1, 4, 5, 7, 3, 4)
> data2
[1] 3 5 7 5 3 2 6 8 5 6 9 4 5 7 3 4
```

Entering Text Items as Data

If the data you require are not numerical, you simply use quotes to differentiate them from numbers. There is no difference between using single and double quotes; R converts them all to double. You can use either or both as long as the surrounding quotes for any single item match, as shown in the following:

our.text = c("item1", "item2", 'item3')

In practice though, it is a good habit to stick to one sort of quote; single quote marks are easier to type.

The following example shows a simple text sample comprising of days of the week:

```
> day1 = c('Mon', 'Tue', 'Wed', 'Thu')
> day1
[1] "Mon" "Tue" "Wed" "Thu"
```

3.2 Ordered and Unordered Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modelling.

Factors are created using the **factor ()** function by taking a vector as input.

Example:

```
# Create a vector as input.
data <-c("East","West","East","North","North","East","West","West","West","East","North")
print(data)
print(is.factor(data))
# Apply the factor function.
factor_data <- factor(data)
print(factor_data)
print(is.factor(factor_data))
```

When we execute the above code, it produces the following result –

```
[1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"  "East"  "North"
[1] FALSE
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East North West
[1] TRUE
```

Factors in Data Frame:

On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```
# Create the vectors for data frame.
height <- c(132,151,162,139,166,147,122)
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")
# Create the data frame.
input_data <- data.frame(height,weight,gender)
print(input_data)
# Test if the gender column is a factor.
print(is.factor(input_data$gender))
# Print the gender column so see the levels.
print(input_data$gender)
```

When we execute the above code, it produces the following result –

```
  height weight gender
1    132     48   male
2    151     49   male
3    162     66 female
4    139     53 female
5    166     67   male
6    147     52 female
7    122     40   male
[1] TRUE
[1] male   male   female female male   female male
Levels: female male
```

Changing the Order of Levels:

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

```
data <- c("East","West","East","North","North","East","West","West","West","East","North")
# Create the factors
factor_data <- factor(data)
print(factor_data)

# Apply the factor function with required order of the level.
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

When we execute the above code, it produces the following result –

```
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East North West
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East West North
```

Generating Factor Levels

We can generate factor levels by using the **gl()** function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

```
gl(n, k, labels)
```

Following is the description of the parameters used –

n is a integer giving the number of levels.
k is a integer giving the number of replications.
labels is a vector of labels for the resulting factor levels.

Example :

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

When we execute the above code, it produces the following result –

```
Tampa   Tampa   Tampa   Tampa   Seattle Seattle Seattle Seattle Boston
[10] Boston  Boston  Boston
Levels: Tampa Seattle Boston
```

3.3 Arrays and Matrices

R-Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the array function. It takes vectors as input and uses the values in the dim parameter to create an array.

Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```
# Create two vectors of different lengths.
vector1 <- c ( 5 , 9 , 3 )
 vector2 <- c ( 10 , 11 , 12 , 13 , 14 , 15 )
# Take these vectors as input to the array.
 result <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ))
print ( result )
```

When we execute the above code, it produces the following result –

```
, , 1
    [,1] [,2] [,3]
[1,]   5   10   13
[2,]   9   11   14
[3,]   3   12   15
, , 2
    [,1] [,2] [,3]
[1,]   5   10   13
[2,]   9   11   14
[3,]   3   12   15
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the dimnames parameter.

```
# Create two vectors of different lengths.
vector1 <- c ( 5 , 9 , 3 )
vector2 <- c ( 10 , 11 , 12 , 13 , 14 , 15 )
column . names <- c ( "COL1" , "COL2" , "COL3" )
row . names <- c ( "ROW1" , "ROW2" , "ROW3" )
matrix . names <- c ( "Matrix1" , "Matrix2" )
# Take these vectors as input to the
array. result <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ), dimnames = list ( column . names ,
row . names ,   matrix . names ))
print ( result )
```

When we execute the above code, it produces the following result –

```
, , Matrix1
     ROW1 ROW2 ROW3
COL1    5   10   13
COL2    9   11   14
COL3    3   12   15
, , Matrix2
     ROW1 ROW2 ROW3
COL1    5   10   13
COL2    9   11   14
COL3    3   12   15
```

Accessing Array Elements

```
# Create two vectors of different lengths.
vector1 <- c ( 5 , 9 , 3 )
vector2 <- c ( 10 , 11 , 12 , 13 , 14 , 15 )
column . names <- c ( "COL1" , "COL2" , "COL3" )
row . names <- c ( "ROW1" , "ROW2" , "ROW3" )
matrix . names <- c ( "Matrix1" , "Matrix2" )
# Take these vectors as input to the array.
```

```
result <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ), dimnames = list ( column . names , row . names , matrix . names ))
# Print the third row of the second matrix of the array.
print ( result [ 3 ,, 2 ])
# Print the element in the 1st row and 3rd column of the 1st matrix.
 print ( result [ 1 , 3 , 1 ])
# Print the 2nd Matrix.
print ( result [,, 2 ])
```

When we execute the above code, it produces the following result –

```
ROW1 ROW2 ROW3
  3   12   15
[1] 13
          ROW1 ROW2 ROW3
COL1    5   10   13
COL2    9   11   14
COL3    3   12   15
```

Manipulating Array Elements

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.
 vector1 <- c ( 5 , 9 , 3 )
vector2 <- c ( 10 , 11 , 12 , 13 , 14 , 15 )
# Take these vectors as input to the array.
array1 <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ))
# Create two vectors of different lengths.
vector3 <- c ( 9 , 1 , 0 )
 vector4 <- c ( 6 , 0 , 11 , 3 , 14 , 1 , 2 , 6 , 9 )
array2 <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ))
# create matrices from these arrays.
matrix1 <- array1 [,, 2 ]
matrix2 <- array2 [,, 2 ]
# Add the matrices.
result <- matrix1 +matrix2
print ( result )
```

When we execute the above code, it produces the following result –

```
     [,1] [,2] [,3]
[1,]   10   20   26
[2,]   18   22   28
[3,]    6   24   30
```

Calculations Across Array Elements:

We can do calculations across the elements in an array using the apply function.

Syntax apply(x, margin, fun)

Following is the description of the parameters used –

- x is an array.
- margin is the name of the data set used.
- fun is the function to be applied across the elements of the array.

Example

We use the apply function below to calculate the sum of the elements in the rows of an array across all the matrices.

```
# Create two vectors of different lengths.
vector1 <- c ( 5 , 9 , 3 )
 vector2 <- c ( 10 , 11 , 12 , 13 , 14 , 15 )
# Take these vectors as input to the array.
new . array <- array ( c ( vector1 , vector2 ), dim = c ( 3 , 3 , 2 ))
print ( new . array )
# Use apply to calculate the sum of the rows across all the matrices.
result <- apply ( new . array , c ( 1 ), sum )
print ( result )
```

When we execute the above code, it produces the following result –

```
, , 1
     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
 [3,]    3   12   15
, , 2
     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

R-Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the matrix function.

Syntax: The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames)

Following is the description of the parameters used −

- data is the input vector which becomes the data elements of the matrix.
- nrow is the number of rows to be created.
- ncol is the number of columns to be created.
- byrow is a logical clue. If TRUE then the input vector elements are arranged by row.
- dimname is the names assigned to the rows and columns.

Example

Create a matrix taking a vector of numbers as input

```
# Elements are arranged sequentially by row.
M <- matrix ( c ( 3 : 14 ), nrow = 4 , byrow = TRUE )
print ( M )
# Elements are arranged sequentially by column.
 N <- matrix ( c ( 3 : 14 ), nrow = 4 , byrow = FALSE )
print ( N )
# Define the column and row names.
 rownames = c ( "row1" , "row2" , "row3" , "row4" )
 colnames = c ( "col1" , "col2" , "col3" )
P <- matrix ( c ( 3 : 14 ), nrow = 4 , byrow = TRUE ,
dimnames = list ( rownames , colnames ))
print ( P )
```

When we execute the above code, it produces the following result –

```
    [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
 [4,]    6   10   14
          col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

```
# Define the column and row names.
 rownames = c ( "row1" , "row2" , "row3" , "row4" )
colnames = c ( "col1" , "col2" , "col3" )
# Create the matrix.
P <- matrix ( c ( 3 : 14 ), nrow = 4 , byrow = TRUE ,
dimnames = list ( rownames , colnames ))
# Access the element at 3rd column and 1st row.
print ( P [ 1 , 3 ])
# Access the element at 2nd column and 4th row.
print ( P [ 4 , 2 ])
# Access only the  2nd row.
print ( P [ 2 ,])
# Access only the 3rd column.
print ( P [, 3 ])
```

When we execute the above code, it produces the following result –

```
[1] 5
[1] 13
col1 col2 col3
  6   7   8
row1 row2 row3 row4
  5   8  11  14
```

Matrix Computations

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix.

The dimensions should be same for the matrices involved in the operation.

Matrix Operations :

```
# Create two 2x3 matrices.
matrix1 <- matrix ( c ( 3 , 9 ,- 1 , 4 , 2 , 6 ), nrow = 2 )
print ( matrix1 )
matrix2 <- matrix ( c ( 5 , 2 , 0 , 9 , 3 , 4 ), nrow = 2 )
print ( matrix2 )
# Add the matrices.
result <- matrix1 + matrix2
cat ( "Result of addition" , "\n" )
print ( result )
# Subtract the matrices
result <- matrix1 - matrix2
cat ( "Result of subtraction" , "\n" )
print ( result )
# Multiply the matrices.
result <- matrix1 * matrix2
cat ( "Result of multiplication" , "\n" )
print ( result )
# Divide the matrices
result <- matrix1 / matrix2
cat ( "Result of division" , "\n" )
print ( result )
```

3.4 Lists and Data Frames

R-Lists

Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using list function.

Creating a List:

Following is an example to create a list containing strings, numbers, vectors and a logical values

```
# Create a list containing strings, numbers, vectors and a logical values.
 list_data <- list ( "Red" , "Green" , c ( 21 , 32 , 11 ), TRUE , 51.23 , 119.1 )
print ( list_data )
```

When we execute the above code, it produces the following result –

```
[[1]]
[1] "Red"
[[2]]
[1] "Green"
 [[3]]
[1] 21 32 11
[[4]]
[1] TRUE
[[5]]
[1] 51.23
[[6]]
[1] 119.1
```

Naming List Elements:

The list elements can be given names and they can be accessed using these names.

```
# Create a list containing a vector, a matrix and a list.
list_data <- list ( c ( "Jan" , "Feb" , "Mar" ), matrix ( c ( 3 , 9 , 5 , 1 ,- 2 , 8 ), nrow = 2 ),   list (
"green" , 12.3 ))
```

```
# Give names to the elements in the list.
names ( list_data )<- c ( "1st Quarter" , "A_Matrix" , "A Inner list" )
# Show the list.
print ( list_data )
```

When we execute the above code, it produces the following result –

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Matrix
[,1] [,2] [,3]
[1,]   3   5  -2
[2,]   9   1   8
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
$A_Inner_list[[2]]
[1] 12.3
```

Accessing List Elements:

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example –

```
# Access the first element of the list.
print ( list_data [ 1 ])
# Access the thrid element. As it is also a list, all its elements will be printed.
print ( list_data [ 3 ])
# Access the list element using the name of the element.
print ( list_data$A_Matrix )
```

When we execute the above code, it produces the following result –

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
$A_Inner_list[[2]]
[1] 12.3
    [,1] [,2] [,3]
[1,]   3   5  -2
[2,]   9   1   8
```

Manipulating List Elements:

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```
# Add element at the end of the list.
list_data [ 4 ]<- "New element" print ( list_data [ 4 ])
# Remove the last element.
 list_data [ 4 ]<- NULL
# Print the 4th Element.
print ( list_data [ 4 ])
# Update the 3rd Element.
list_data [ 3 ]<- "updated element"
print ( list_data [ 3 ])
```

When we execute the above code, it produces the following result –

```
[[1]] [1] "New element"
$
NULL
$`A Inner list` [1] "updated element"
```

Merging Lists

You can merge many lists into one list by placing all the lists inside one list function.

```
# Create two lists.
list1 <- list ( 1 , 2 , 3 )
list2 <- list ( "Sun" , "Mon" , "Tue" )
# Merge the two lists. merged .
list <- c ( list1 , list2 )
# Print the merged list.
print (merged . list )
```

When we execute the above code, it produces the following result –

```
 [[1]] [1] 1
[[2]] [1] 2
[[3]] [1] 3
[[4]] [1] "Sun"
[[5]] [1] "Mon"
[[6]] [1] "Tue"
```

Converting List to Vector:

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the unlist function. It takes the list as input and produces a vector.

```
# Create lists. list1 <- list ( 1 : 5 )
print ( list1 )
list2 <- list ( 10 : 14 )
print ( list2 )
# Convert the lists to vectors.
v1 <- unlist ( list1 )
v2 <- unlist ( list2 )
print ( v1 ) print ( v2 )
# Now add the vectors
result <- v1 +v2 print ( result )
```

When we execute the above code, it produces the following result –

```
[[1]]
[1] 1 2 3 4 5
[[1]]
[1] 10 11 12 13 14
[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19
```

R - Data Frames

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

```
# Create the data frame.
emp . data <- data . frame (   emp_id = c ( 1 : 5 ),
        emp_name = c ( "Rick" , "Dan" , "Michelle" , "Ryan" , "Gary" ),
        salary = c ( 623.3 , 515.2 , 611.0 , 729.0 , 843.25 ),
        start_date = as . Date ( c ( "2012-01-01" , "2013-09-23" , "2014-11-15" , "2014-05-11" ,
        "2015-03-27" )),
        stringsAsFactors = FALSE )
# Print the data frame.
print ( emp . data )
```

When we execute the above code, it produces the following result –

```
 emp_id   emp_name    salary    start_date
1    1    Rick       623.30    2012-01-01
2    2    Dan        515.20    2013-09-23
3    3    Michelle   611.00    2014-11-15
4    4    Ryan       729.00    2014-05-11
5    5    Gary       843.25    2015-03-27
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using str function.

```
# Get the structure of the data frame.
str ( emp . data )
```

When we execute the above code, it produces the following result –

```
'data.frame':   5 obs. of  4 variables:
$ emp_id    : int  1 2 3 4 5
$ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
$ salary    : num  623 515 611 729 843
$ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying summary function.

```
# Print the summary.
print ( summary ( emp . data ))
```

When we execute the above code, it produces the following result –

```
   emp_id    emp_name          salary       start_date
Min.  :1   Length:5         Min.  :515.2   Min.  :2012-01-01
1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
Median :3   Mode  :character   Median :623.3   Median :2014-05-11
Mean  :3                 Mean  :664.4   Mean  :2014-01-14
 3rd Qu.:4                 3rd Qu.:729.0   3rd Qu.:2014-11-15
Max.  :5                 Max.  :843.2   Max.  :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

```
# Extract Specific columns.
 result <- data . frame ( emp . data$emp_name , emp . data$salary )
 print ( result )
```

When we execute the above code, it produces the following result –

```
  emp.data.emp_name emp.data.salary
1           Rick        623.30
2            Dan        515.20
3        Michelle        611.00
4           Ryan        729.00
5           Gary        843.25
```

Extract the first two rows and then all columns

```
# Extract first two rows.
result <- emp . data [ 1 : 2 ,]
print ( result )
```

When we execute the above code, it produces the following result –

```
  emp_id   emp_name   salary   start_date
1     1    Rick     623.3    2012-01-01
2     2    Dan      515.2    2013-09-23
```

```
# Extract 3rd and 5th row with 2nd and 4th column.
result <- emp . data [ c ( 3 , 5 ), c ( 2 , 4 )]
print ( result )
```

When we execute the above code, it produces the following result –

```
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27
```

Expand Data Frame:

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```
# Add the "dept" coulmn.
emp . data$dept <- c ( "IT" , "Operations" , "IT" , "HR" , "Finance" )
v <- emp . data
print ( v )
```

When we execute the above code, it produces the following result –

```
  emp_id   emp_name    salary   start_date      dept
1     1    Rick       623.30    2012-01-01      IT
2     2    Dan        515.20    2013-09-23      Operations
3     3    Michelle    611.00    2014-11-15       IT
4     4    Ryan        729.00    2014-05-11       HR
5     5    Gary        843.25    2015-03-27       Finance
```

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the rbind function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```
# Create the first data frame.
emp . data <- data . frame (   emp_id = c ( 1 : 5 ),   emp_name = c ( "Rick" , "Dan" , "Michelle" ,
"Ryan" , "Gary" ),   salary = c ( 623.3 , 515.2 , 611.0 , 729.0 , 843.25 ),
    start_date = as . Date ( c ( "2012-01-01" , "2013-09-23" , "2014-11-15" , "2014-05-11" , "2015-
03-27" )),   dept = c ( "IT" , "Operations" , "IT" , "HR" , "Finance" ),   stringsAsFactors = FALSE )
# Create the second data frame
emp . newdata <- data . frame (   emp_id = c ( 6 : 8 ),   emp_name = c ( "Rasmi" , "Pranab" , "Tusar"
),   salary = c ( 578.0 , 722.5 , 632.8 ),   start_date = as . Date ( c ( "2013-05-21" , "2013-07-30" ,
"2014-06-17" )),  dept = c ( "IT" , "Operations" , "Fianance" ),   stringsAsFactors = FALSE )
# Bind the two data frames.
emp . finaldata <- rbind ( emp . data , emp . newdata )
print ( emp . finaldata )
```

When we execute the above code, it produces the following result –

	emp_id	emp_name	salary	start_date	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	5	Gary	843.25	2015-03-27	Finance
6	6	Rasmi	578.00	2013-05-21	IT
7	7	Pranab	722.50	2013-07-30	Operations
8	8	Tusar	632.80	2014-06-17	Fianance

3.5 Reading data from files

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

1.Input as CSV File:

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named input.csv.

You can create this file using windows notepad by copying and pasting this data. Save the file as input.csv using the save As All files option in notepad.

```
id , name , salary , start_date , dept
1 , Rick , 623.3 , 2012 - 01 - 01 , IT
2 , Dan , 515.2 , 2013 - 09 - 23 , Operations
3 , Michelle , 611 , 2014 - 11 - 15 , IT
4 , Ryan , 729 , 2014 - 05 - 11 , HR
5, Gary , 843.25 , 2015 - 03 - 27 , Finance
6 , Nina , 578 , 2013 - 05 - 21 , IT
7 , Simon , 632.8 , 2013 - 07 - 30 , Operations
8 , Guru , 722.5 , 2014 - 06 - 17 , Finance
```

Reading a CSV File

Following is a simple example of read.csv function to read a CSV file available in your current working directory –

```
data <- read . csv ( "input.csv" )
print ( data )
```

When we execute the above code, it produces the following result –

```
  id,  name,  salary,  start_date,  dept
```

```
1    1   Rick    623.30   2012-01-01    IT
2    2   Dan     515.20   2013-09-23    Operations
3    3   Michelle 611.00  2014-11-15    IT
4    4   Ryan    729.00   2014-05-11    HR
5    5   Gary    843.25   2015-03-27    Finance
6    6   Nina    578.00   2013-05-21    IT
7    7   Simon   632.80   2013-07-30    Operations
8    8   Guru    722.50   2014-06-17    Finance
```

Get the maximum salary

```
# Create a data frame.
        data <- read . csv ( "input.csv" )
# Get the max salary from data frame.
        sal <- max ( data$salary )
        print ( sal )
```

When we execute the above code, it produces the following result –

```
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Get the max salary from data frame.
        sal <- max ( data$salary )
# Get the person detail having max salary.
        retval <- subset ( data , salary == max ( salary ))
        print ( retval )
```

When we execute the above code, it produces the following result –

```
   id   name  salary  start_date    dept
5    5   Gary  843.25  2015-03-27    Finance
```

Get all the people working in IT department

```
# Create a data frame.
        data <- read . csv ( "input.csv" )
        retval <- subset ( data , dept == "IT" )
        print ( retval )
```

```
   id   name    salary   start_date   dept
1    1   Rick    623.3    2012-01-01   IT
3    3   Michelle  611.0  2014-11-15   IT
6    6   Nina    578.0    2013-05-21   IT
```

2.EXCEL FILES:

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. Few such packages are - XLConnect, xlsx, gdata etc. We will be using xlsx package. R can also write into excel file using this package.

Input as xlsx File:

Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

```
id name     salary   start_date dept
1Rick  623.3   1/1/2012 IT
2Dan     515.2    9/23/2013   Operations
3Michelle  611   11/15/2014 IT
4Ryan  729   5/11/2014 HR
5Gary  843.25   3/27/2015 Finance
6Nina  578     5/21/2013 IT
7Simon  632.8   7/30/2013 Operations
8Guru  722.5   6/17/2014 Finance
```

Also copy and paste the following data to another worksheet and rename this worksheet to "city".

```
name city
```

Rick Seattle
Dan Tampa
Michelle Chicago
Ryan Seattle
Gary Houston
Nina Boston
Simon Mumbai
Guru Dallas

Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.

Reading the Excel File

The input.xlsx is read by using the read.xlsx function as shown below. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file input.xlsx.
data <- read . xlsx ( "input.xlsx" , sheetIndex = 1 )
print ( data )
```

When we execute the above code, it produces the following result –

	id,	name,	salary,	start_date,	dept
1	1	Rick	623.30	2012-01-01	IT
2	2	Dan	515.20	2013-09-23	Operations
3	3	Michelle	611.00	2014-11-15	IT
4	4	Ryan	729.00	2014-05-11	HR
5	NA	Gary	843.25	2015-03-27	Finance
6	6	Nina	578.00	2013-05-21	IT
7	7	Simon	632.80	2013-07-30	Operations
8	8	Guru	722.50	2014-06-17	Finance

UNIT-4
Probability Distributions

Statistical models in R - Binomial , Poisson , Normal distributions - Manipulating objects - data distribution

4.1 Statistical models in R

Basic statistical descriptions can used to identify properties of the data and highlight which data values should be treated as noise or outliers.

In this data can describe the three areas of basic statistical descriptions. That is mean, median and mode. The most common data dispersion measures are the range, quartiles and inter-quartile range. The five number summary and dot plots and the variance and standard deviation of the data. We can use many graphic displays of basic statistical description to visually inspect our data. We discusses with measures of central tendency.

Measuring the central tendency: Mean, Median, Mode

Mean: The "Mean" is computed by adding all of the numbers in the data together and dividing by the number elements contained in the data set.

Example:

Data Set = 2, 5, 9, 3, 5, 4, 7
Number of Elements in Data Set = 7
Mean = (2 + 5 + 9 + 7 + 5 + 4 + 3) / 7 = 5

Median: The "Median" of a data set is dependant on whether the number of elements in the data set is odd or even. First reorder the data set from the smallest to the largest then if the number of elements is odd, then the Median is the element in the middle of the data set.
If the number of elements is even, then the Median is the average of the two middle terms.

Examples: Odd Number of Elements

Data Set = 2, 5, 9, 3, 5, 4, 7
Reordered = 2, 3, 4, 5, 5, 7, 9
^
Median = 5

Mode: The "Mode" for a data set is the element that occur the most often. It is not uncommon for a data set to have more than one mode.
This happens when two or more elements occur with equal frequency in the data set. A data set with two modes is called bimodal. A data set with three modes is called tri-modal.

Examples: Single Mode

Data Set = 2, 5, 9, 3, 5, 4, 7
Mode = 5

Measuring the dispersion of data: range, inter-quartile range. Variance and standard deviation

There are many ways to describe variability including:

- Range
- Inter-quartile range (IQR)
- Variance and Standard deviation

Let's look at each of these in turn.

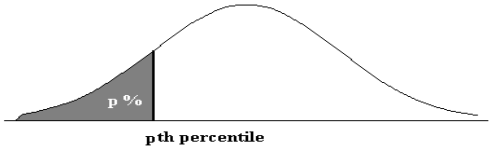
A. Range: *R* = maximum - minimum

Easy to calculate
Very much affected by extreme values (range is not a resistant measure of variability)

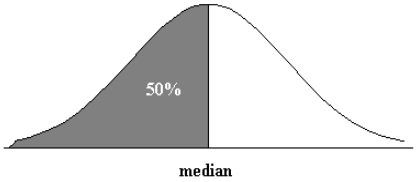
B. Inter-quartile range (IQR)

In order to talk about inter-quartile range, we need to first talk about percentiles.
The ***p*th percentile** of the data set is a measurement such that after the data are ordered from smallest to largest, at most, *p*% of the data are at or below this value and at most, (100 - *p*)% at or above it.

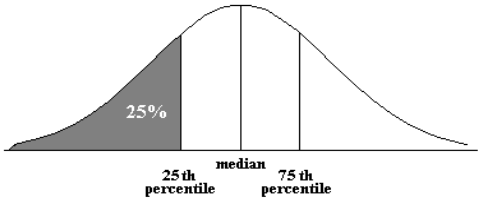
.



Thus, the median is the 50th percentile. Fifty percent or the data values fall at or below the median
Also, Q_1 = lower quartile = the 25th percentile and Q_3 = upper quartile = the 75th percentile.

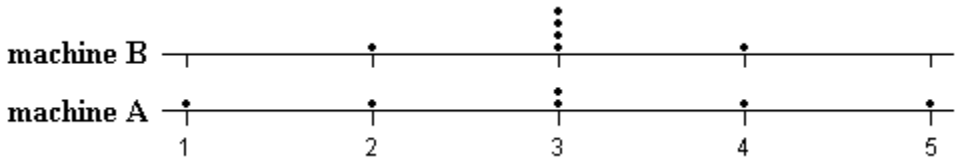


The **inter-quartile range** is the difference between upper and lower quartiles and denoted as **IQR**.



IQR = Q_3 - Q_1 = upper quartile - lower quartile = 75th percentile - 25th percentile.
Note: *IQR* is not affected by extreme values. It is thus a resistant measure of variability.

C. Variance and Standard Deviation
Two transaction machines *A* and *B* drop candies when a quarter is inserted. The number of pieces of candy one gets is random. The following data are recorded for six trials at each transaction machine:
Pieces of candy from vending machine *A*:
1, 2, 3, 3, 5, 4
mean = 3, median = 3, mode = 3
Pieces of candy from vending machine *B*:
2, 3, 3, 3, 3, 4
mean = 3, median = 3, mode = 3
Dot plots for the pieces of candy from transaction machine *A* and transaction machine *B*:



They have the same centre; one way to compare their spreads is to compute their standard deviations. In the following section, we are going to compute the sample variance and the sample standard deviation for a data set.

4.2 Binomial distribution

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.
R has four in-built functions to generate binomial distribution. They are described below.

1. dbinom(x, size, prob)
2. pbinom(x, size, prob)
3. qbinom(p, size, prob)
4. rbinom(n, size, prob)

Following is the description of the parameters used –

- x is a vector of numbers.
- p is a vector of probabilities.
- n is number of observations.

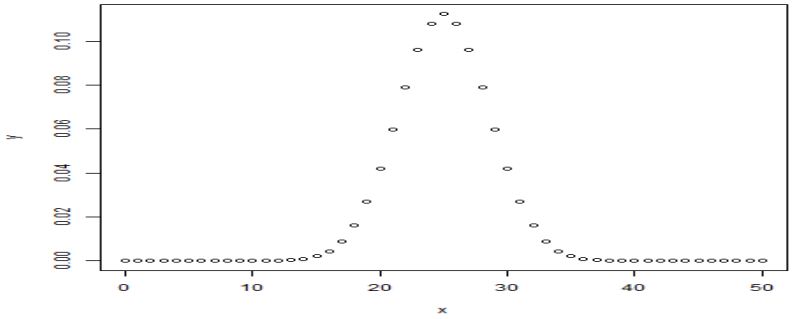
- size is the number of trials.
- prob is the probability of success of each trial.

1.dbinom():

This function gives the probability density distribution at each point.

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)
# Create the binomial distribution.
y <- dbinom(x,50,0.5)
# Give the chart file a name.
png(file = "dbinom.png")
# Plot the graph for this sample.
plot(x,y)
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –



2.pbinom() :

This function gives the cumulative probability of an event. It is a single value representing the probability.

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)
print(x)
```

When we execute the above code, it produces the following result –

```
[1] 0.610116
```

3.qbinom() :

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# How many heads will have a probability of 0.25 will come out when a coin is tossed 51 times.
x <- qbinom(0.25,51,1/2)
print(x)
```

When we execute the above code, it produces the following result –

```
[1] 23
```

4.rbinom() :

This function generates required number of random values of given probability from a given sample.

```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)
print(x)
```

When we execute the above code, it produces the following result –

```
[1] 58 61 59 66 55 60 61 67
```

4.3 Poisson distribution

Poisson Regression involves regression models in which the response variable is in the form of counts and not fractional numbers. For example, the count of number of births or

number of wins in a football match series. Also the values of the response variables follow a Poisson distribution.

The general mathematical equation for Poisson regression is –

log(y) = a + b1x1 + b2x2 + bnxn.....

Following is the description of the parameters used –

- **y** is the response variable.
- **a** and **b** are the numeric coefficients.
- **x** is the predictor variable.

The function used to create the Poisson regression model is the **glm()** function.

Syntax

The basic syntax for **glm()** function in Poisson regression is –

glm(formula,data,family)

Following is the description of the parameters used in above functions –

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is 'Poisson' for Logistic Regression.

Example

We have the in-built data set "warpbreaks" which describes the effect of wool type (A or B) and tension (low, medium or high) on the number of warp breaks per loom. Let's consider "breaks" as the response variable which is a count of number of breaks. The wool "type" and "tension" are taken as predictor variables.

Input Data

```
input <- warpbreaks
print(head(input))
```

When we execute the above code, it produces the following result –

```
  breaks  wool  tension
1   26     A    L
2   30     A    L
3   54     A    L
4   25     A    L
5   70     A    L
6   52     A    L
```

Create Regression Model

```
output <-glm(formula = breaks ~ wool+tension,
             data = warpbreaks,
            family = poisson)
print(summary(output))
```

When we execute the above code, it produces the following result –

```
Call:
glm(formula = breaks ~ wool + tension, family = poisson, data = warpbreaks)

Deviance Residuals:
   Min      1Q    Median      3Q     Max
 -3.6871  -1.6503  -0.4269    1.1902   4.2616

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 297.37 on 53 degrees of freedom
Residual deviance: 210.39 on 50 degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
In the summary we look for the p-value in the last column to be less than 0.05 to consider an impact of the predictor variable on the response variable. As seen the wooltype B having tension type M and H have impact on the count of breaks.

4.4 Normal distribution

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.
R has four in built functions to generate normal distribution. They are described below.

1. dnorm(x, mean, sd)
2. pnorm(x, mean, sd)
3. qnorm(p, mean, sd)
4. rnorm(n, mean, sd)

Following is the description of the parameters used in above functions –

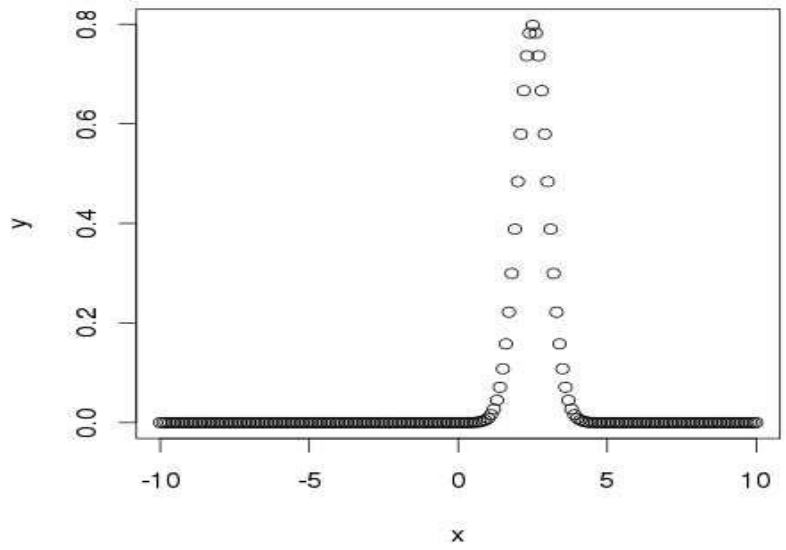
- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations(sample size).
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

1.dnorm():

This function gives height of the probability distribution at each point for a given mean and standard deviation.

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.
x <- seq(-10, 10, by = .1)
# Choose the mean as 2.5 and standard deviation as 0.5.
y <- dnorm(x, mean = 2.5, sd = 0.5)
# Give the chart file a name.
png(file = "dnorm.png")
plot(x,y)
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –

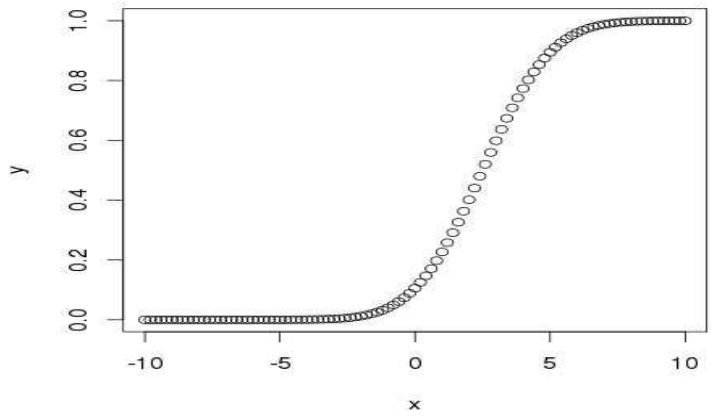


2.pnorm():

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.
x <- seq(-10,10,by = .2)
 # Choose the mean as 2.5 and standard deviation as 2.
y <- pnorm(x, mean = 2.5, sd = 2)
# Give the chart file a name.
png(file = "pnorm.png")
# Plot the graph.
plot(x,y)
# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –



3.qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)
```

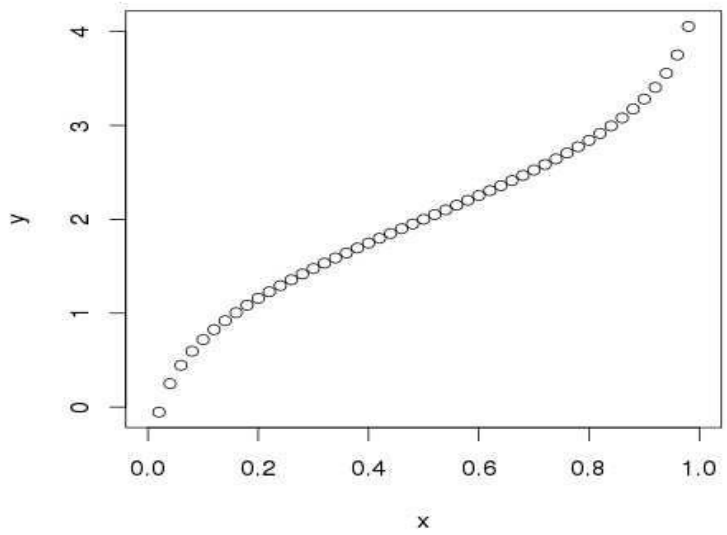
```
# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)

# Give the chart file a name.
png(file = "qnorm.png")

# Plot the graph.
plot(x,y)

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –



4.rnorm():

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

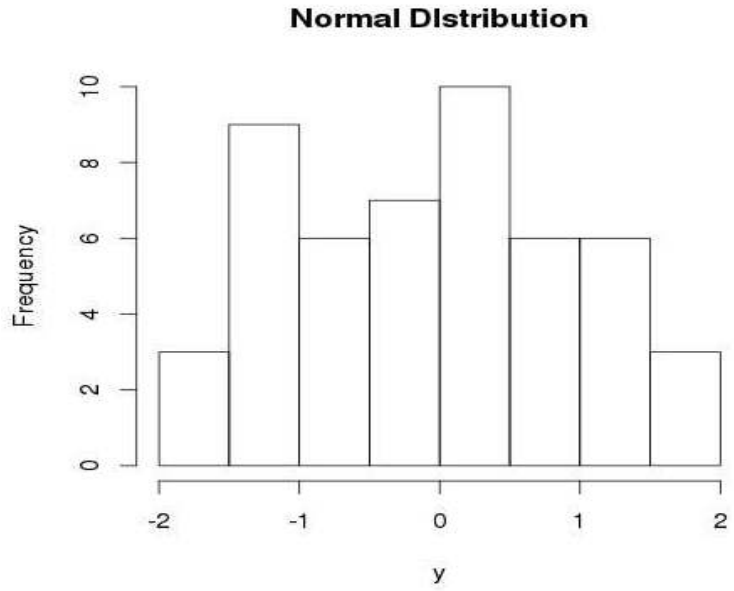
```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)

# Give the chart file a name.
png(file = "rnorm.png")

# Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")

# Save the file.
dev.off()
```

When we execute the above code, it produces the following result –



4.5 Manipulating objects

Probability distribution objects allow you to fit a probability distribution to sample data, or define a distribution by specifying parameter values. You can then perform a variety of analyses on the distribution object.

Create Probability Distribution Objects :

Estimate probability distribution parameters from sample data by fitting a probability distribution object to the data using fitdist. You can fit a single specified parametric or non-parametric distribution to the sample data. You can also fit multiple distributions of the same type to the sample data based on grouping variables. For most distributions, **fitdist** uses maximum likelihood estimation (MLE) to estimate the distribution parameters from the sample data.

fitdist() :

Fit probability distribution object to data

Syntax :

pd = fitdist(x,distname)
pd = fitdist(x,distname,Name,Value)
[pdca,gn,gl] = fitdist(x,distname,'By',groupvar)
[pdca,gn,gl] = fitdist(x,distname,'By',groupvar,Name,Value)

Description

It creates a probability distribution object by fitting the distribution specified by distname to the data in column vector x. It creates the probability distribution object with additional options specified by one or more name-value pair arguments. creates probability distribution objects by fitting the distribution specified by distname to the data in x based on the grouping variable groupvar. It returns a cell array of fitted probability distribution objects, pdca, a cell array of group labels, gn, and a cell array of grouping variable levels, gl. it returns the above output arguments using additional options specified by one or more name-value pair arguments.

EX:

```
load hospital
x = hospital.Weight;
pd = fitdist(x,'Normal')
gender = hospital.gender;
[pdca,gn,gl] = fitdist(x,'Normal','By',gender)
```

Alternatively, you can create a probability distribution object with specified parameter values using makedist().

makedist() :

Create probability distribution object

Syntax

pd = makedist(distname)
pd = makedist(distname,Name,Value)
list = makedist
makedist -reset

Work with Probability Distribution Objects :

Once you create a probability distribution object, you can use object functions to:

- Compute confidence intervals for the distribution parameters **paramci()**
- Compute summary statistics, including mean **mean()** , median **median()** ,inter quartile range **iqr()** , variance **var(),** and standard deviation **std()**.
- Evaluate the probability density function **pdf()** .
- Evaluate the cumulative distribution function **cdf()** or the inverse cumulative distribution function **icdf()** .
- Compute the negative log likelihood **negloglik()** and profile likelihood function **proflik()** for the distribution.
- Generate random numbers from the distribution **random() .**
- Truncate the distribution to specified lower and upper limits **truncate() .**

Save a Probability Distribution Object :

To save your probability distribution object to a .R file:

In the toolbar, click **Save Workspace**. This option saves all of the variables in your workspace, including any probability distribution objects.

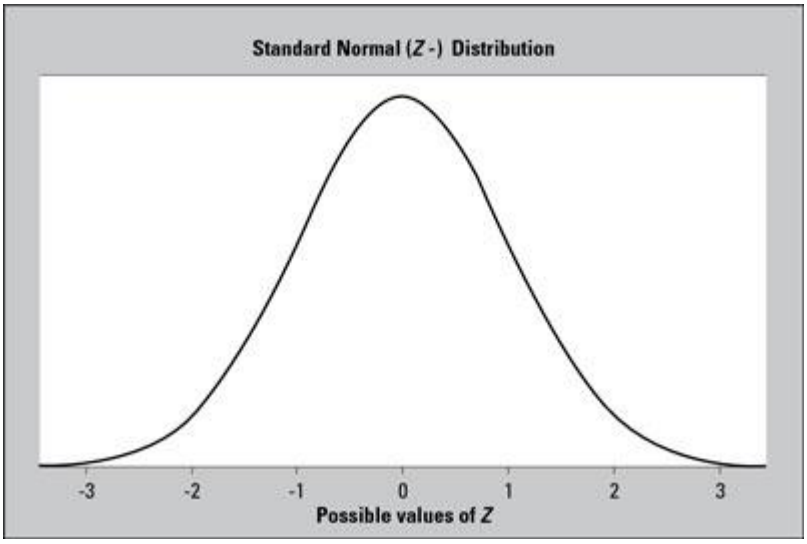
In the workspace browser, right-click the probability distribution object and select **Save as**. This option saves only the selected probability distribution object, not the other variables in your workspace.

4.6Data distribution

The *distribution* of a statistical data set (or a population) is a listing or function showing all the possible values (or intervals) of the data and how often they occur. When a distribution of categorical data is organized, you see the number or percentage of individuals in each group. When a distribution of numerical data is organized, they're often ordered from smallest to largest, broken into reasonably sized groups (if appropriate), and then put into graphs and charts to examine the shape, center, and amount of variability in the data.

The world of statistics includes dozens of different distributions for categorical and numerical data; the most common ones have their own names. One of the most well-known distributions is called the *normal distribution,* also known as the *bell-shaped curve.* The normal distribution is based on numerical data that is continuous; its possible values lie on the entire real number line. Its overall shape, when the data are organized in graph form, is a symmetric bell-shape. In other words, most (around 68%) of the data are centered around the mean (giving you the middle part of the bell), and as you move farther out on either side of the mean, you find fewer and fewer values (representing the downward sloping sides on either side of the bell).

Due to symmetry, the mean and the median lie at the same point, directly in the center of the normal distribution. The standard deviation is measured by the distance from the mean to the *inflection point* (where the curvature of the bell changes from concave up to concave down).



Z-) distribution has a bell-shaped curve with mean 0 and standard deviati"/>

A standard normal (*Z*-) distribution has a bell-shaped curve with mean 0 and standard deviation 1.

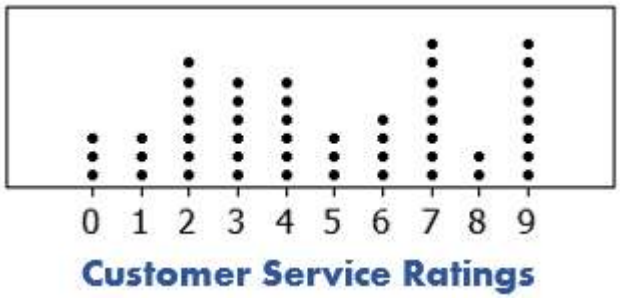
This figure shows a graph of a normal distribution with mean 0 and standard deviation 1 (this distribution has a special name, the *standard normal distribution* or *Z-distribution*). The shape of the curve resembles the outline of a bell.

Because every distinct population of data has a different mean and standard deviation, an infinite number of normal distributions exist, each with its own mean and its own standard deviation to characterize it.

Data distributions are used often in statistics. They are graphical methods of organizing and displaying useful information. There are several types of data distributions. In this lesson, we will focus on dot plots, histograms, box plots, and tally charts.

Dot Plots

Dot plots show numerical values plotted on a scale. Each dot represents one value in the set of data. In the example below, the customer service ratings range from 0 to 9. The dots tell us the **frequency**, or rate of occurrence, of customers who gave each rating. If you look at the 5 rating, you can see that three customers gave that rating, and if you look at a score of 9, eight customers gave that rating. We can also see that ratings were provided by fifty customers, one dot for each customer.



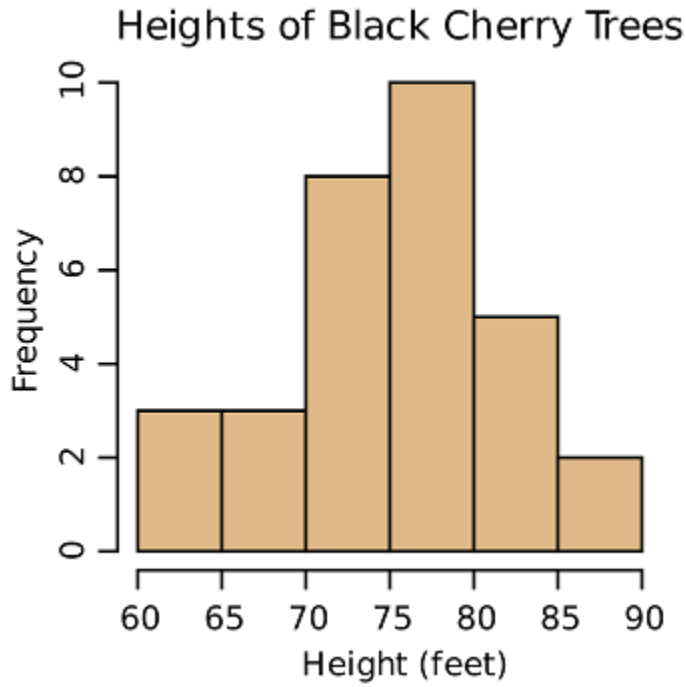
Example of a dot plot

Now imagine that ratings were provided by five hundred customers. It would not be practical or useful to have a distribution of five hundred dots. For this reason, dot plots are used for data that have a relatively small number of values.

Histograms

Histograms display data in ranges, with each bar representing a range of numeric values. The height of the bar tells you the frequency of values that fall within that range. In the

example below, the first bar represents black cherry trees that are between 60 and 65 feet in height. The bar goes up to three, so there are three trees that are between 60 and 65 feet.



Example of a histogram

Histograms are an excellent way to display large amounts of data. If you have a set of data that includes thousands of values, you can simply adjust the frequency interval to accommodate a larger scale, rather than just 0-10.

UNIT-5
Delivering Results

Documentation & Deployment - Producing Effective Presentations - Introduction to Graphical Analysis - plot() function - Displaying Multivariate data - matrix plots - multiple plots in one window - exporting graph - using graphics parameters.

5.1 Documentation and Deployment

Buzz dataset :

The original supplied documentation tells us the buzz data is structured as shown in below table

Buzz Data	Description
Rows	Each row represents many different measurements of the popularity of a technical personal computer discussion topic.
Topics	Topics include technical issues about personal computers such as brand names, memory, over clocking, and so on.
Measurement types	For each topic, measurement types are quantities such as the number of discussions started, number of posts, number of authors, number of readers, and so on. Each measurement is taken at eight different times.
Times	The eight relative times are named 0 through 7 and are likely days (the original variable documentation is not completely clear and the matching paper has not yet been released). For each measurement type all eight relative times are stored in different columns in the same data row.
Buzz	The quantity to be predicted is called buzz and is defined as being true or 1 if the ongoing rate of additional discussion activity is at least 500 events per day averaged over a number of days after the observed days. Likely buzz is a future average of the seven variables labelled NAC (the original documentation is unclear on this).

The buzz problem demonstrates some features that are common in actual data science projects:

- ✓ This is a project where we're trying to predict the future from past features.
- ✓ The quantity to be predicted is a function of future values of variables we're measuring.
- ✓ A domain-specific reshaping of the supplied variables would be appropriate.

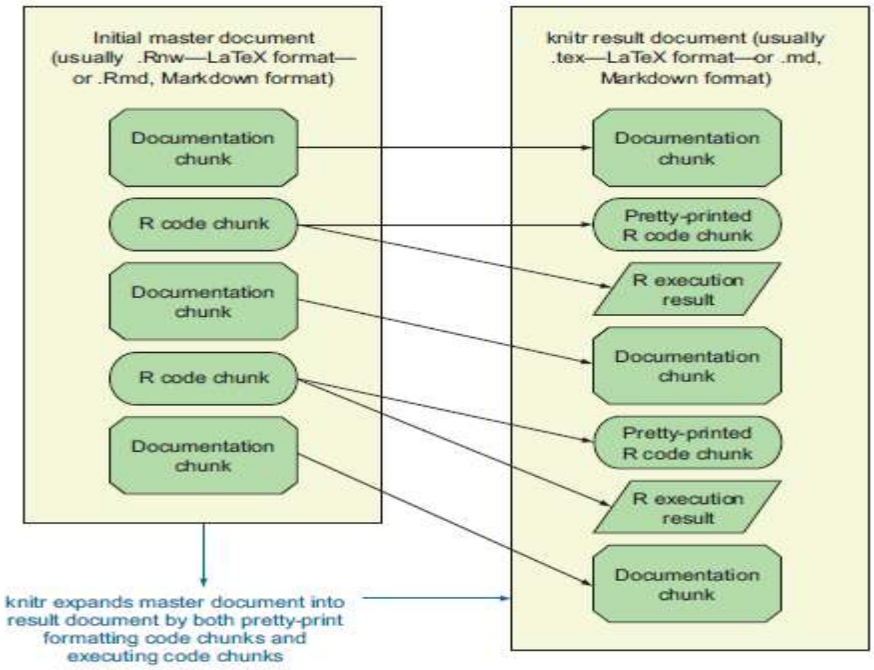
Using knitr to produce milestone documentation

The first sort of documentation we recommend is project milestone or checkpoint documentation. At major steps of the project you should take some time out to repeat your work in a clean environment

What is knitr?

knitr is an R package that allows the inclusion of R code and results inside documents. knitr's operation is similar in concept to Knuth's literate programming and to the R Sweave package. In practice you maintain a master file that contains both user readable documentation and chunks of program source code. The document types supported by knitr include LaTeX, Markdown, and HTML. LaTeX format is a good choice for detailed typeset technical documents. Markdown format is a good choice for online documentation and wikis. Direct HTML format may be appropriate for some web applications.

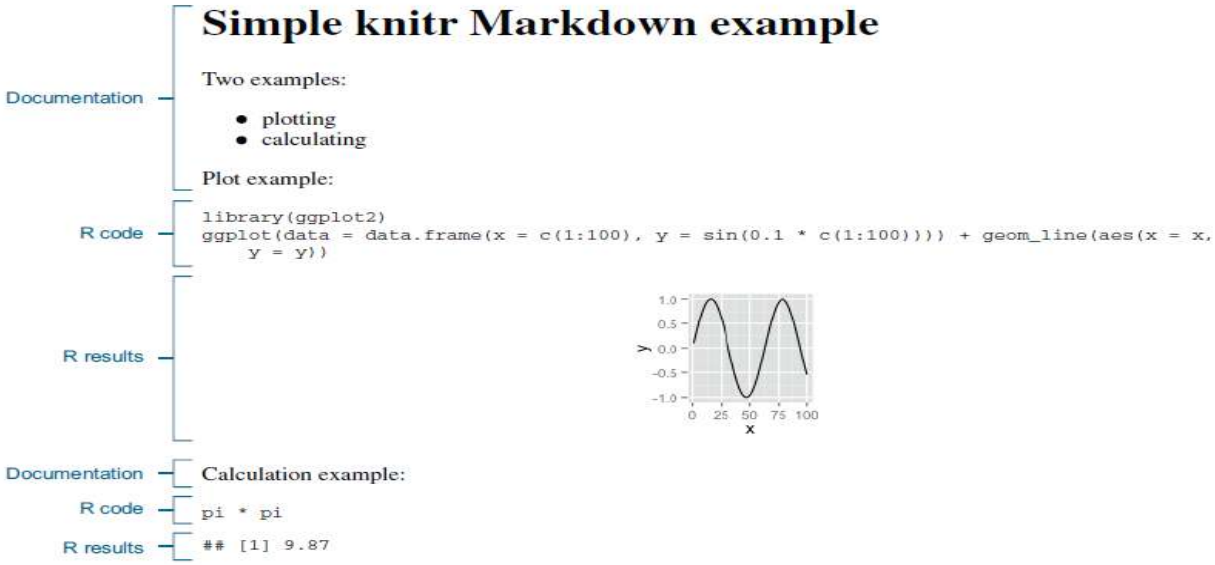
knitr's main operation is called a knit: knitr extracts and executes all of the R code and then builds a new result document that assembles the contents of the original document plus pretty-printed code and results (see figure).



[Fig: Knitr Process schematic]

The process is best demonstrated with a few examples.

```
# Simple knitr Markdown example
Two examples:
* plotting
* calculating
Plot example:
```{r plotexample, fig.width=2, fig.height=2, fig.align='center'}
library(ggplot2)
ggplot(data=data.frame(x=c(1:100),y=sin(0.1*c(1:100)))) +
geom_line(aes(x=x,y=y))
```
Calculation example:
```{r calcexample}
pi*pi
```


```

[Fig : Simple Knitr Markdown Result]

Project Directory Structure:

A possible project directory structure can be having the following things :

Directory	Description
Data	Where we save original downloaded data. This directory must usually be excluded from version control (using the .gitignore feature) due to file sizes, so you must ensure it's backed up. We tend to save each data refresh in a separate subdirectory named by date.
Scripts	Where we store all code related to analysis of the data.
Derived	Where we store intermediate results that are derived from data and scripts. This directory must be excluded from source control. You also should have a master script that can rebuild the contents of this directory in a single command (and test the script from time to time). Typical contents of this directory are compressed files and file-based databases (H2, SQLite).
Results	Similar to derived, but this directory holds smaller later results (often based on derived) and hand-written content. These include important saved models, graphs, and reports. This directory is under version control, so collaborators can see what was said when. Any report shared with partners should come from this directory.

5.2 Producing Effective Presentations

To produce effective presentations of our day to day project work and how to deploy our model into production. This included the additional documentation needed to support the operational designs. For effective presentations we have the following.....

1. Presenting your results to the project sponsors.
2. Presenting your model to end users.
3. Presenting your work to other data scientists.

1.Presenting your results to the project sponsors :

The project sponsor is the person who wants the data science result—generally for the business need that it will fill. The project sponsor primary interest is business-orientation. To cover these considerations, we recommend a structure similar to the following:

- **Summarize the motivation behind the project, and its goals.**

  This section of the presentation is intended to provide context for the rest of the talk, especially if it will be distributed to others in the company who weren't as closely involved as your project sponsor was. we provide background for the motivation behind the project by showing the business need and how the project will address that need.
- **State the project's results.**

  This section of the presentation briefly describes what you did, and what the results were, in the context of the business need.
- **Back up the results with details, as needed.**

Once your audience knows what you've done, why, and how well you've succeeded (from a business point of view), you can fill in details to help them understand more.

- **Discuss recommendations, outstanding issues, and possible future work.**

No project ever produces a perfect outcome, and you should be up-front (but optimistic) about the limitations of your results. As a data scientist, you're of course interested in improving the model's performance, but to the audience, improving the model is less important than improving the process (and better meeting the business need). Frame the discussion from that perspective

Project sponsor presentation takeaways:

Here's what you should remember about the project sponsor presentation:

- ✓ Keep it short.
- ✓ Keep it focused on the business issues, not the technical ones.
- ✓ Your project sponsor might use your presentation to help sell the project or its results to the rest of the organization. Keep that in mind when presenting background and motivation.
- ✓ Introduce your results early in the presentation, rather than building up to them.

2. Presenting your model to end users :

No matter how well your model performs, it's important that the people who will actually be using it have confidence in its output and are willing to adopt it. Otherwise, the model won't be used, and your efforts will have been wasted.

For an end user presentation, we recommend a structure similar to the following:

- **Summarize the motivation behind the project, and its goals.**

With the model's end users, it's less important to discuss business motivations and more important to focus on how the model affects them.

- **Show how the model fits into the users' workflow (and how it improves that workflow).**

In this of the presentation, you explain how the model helps the users do their job. A good way to do this is to give before-and-after scenarios of a typical user workflow, Presumably, the before process and its minuses are already obvious to the users.

- **Show how to use the model.**

In this section is likely the bulk of the presentation, where you'll teach the users how to use the model. It describes how a product manager will interact with the buzz model.

End user presentation takeaways:

Here's what you should remember about the end user presentation:

- ✓ Your primary goal is to convince the users that they want to use your model.
- ✓ Focus on how the model affects (improves) the end users' day-to-day processes.
- ✓ Describe how to use the model and how to interpret or use the model's outputs.

3 Presenting your work to other data scientists :

A presentation to your peers generally has the following structure:

- **Introduce the problem.**

we start off by introducing the concept of buzz and why it's important, then go straight into the prediction task.

- **Discuss related work.**

Discuss others who have done research on problems related to your problem, what approach they took, and how their approach is similar to or different from yours.

- **Discuss your approach.**

Talk about what you did in lots of detail, including compromises that you had to make and setbacks that you had.

- **Give results and findings. Discuss future work.**

In this we discuss our model's performance (precision/recall) and also confirm that representative end users did find the model's output useful to their jobs.

Peer presentation takeaways :

Here's what you should remember about your presentation to fellow data scientists:

- ✓ A peer presentation can be motivated primarily by the modelling task.
- ✓ Unlike the previous presentations, the peer presentation can (and should) be rich in technical details.
- ✓ Be up-front about limitations of the model and assumptions made while building it. Your audience can probably spot many of the limitations already.

5.3 Introduction to Graphics Analysis

Graphs are a powerful way to present your data and results in a concise manner. Whatever kind of data you have, there is a way to illustrate it graphically. A graph is more readily understandable than words and numbers, and producing good graphs is a vital skill. R has powerful and flexible graphical capabilities. In general terms, R has two kinds of graphical commands: some commands generate a basic plot of some sort, and other commands are used to tweak the output and to produce a more customized finish.

In order to produce graphical output, the user calls a series of graphics functions, each of which produces either a complete plot, or adds some output to an existing plot.

R graphics follows a painters model," which mean s that graphics output occurs in steps,with later output obscuring any previous output that it overlaps.

There are very many graphical functions provided by R and the add-on packages for R.

The organization of R graphics :

The R graphics system can be broken into four distinct levels: graphics packages; graphics systems; a graphics engine, including standard graphics devices; and graphics device packages (see Figure).

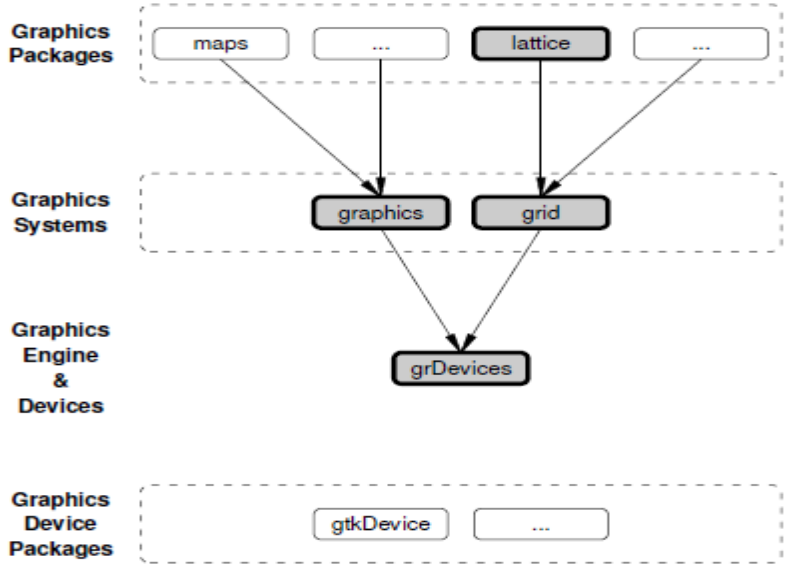


Figure is The structure of the R graphics system showing the main packages that provide graphics functions in R . Arrows indicate where one package builds on the functions in another package. The packages described in this book are highlighted with thicker borders and grey backgrounds.

The core R graphics functionality described in this book is provided by the graphics engine and the two graphics systems, traditional graphics and grid.

The graphics engine consists of functions in the grDevices package and provides fundamental support for handling such things as colours and fonts and graphics devices for producing output in different graphics

formats .

The traditional graphics system consists of functions in the graphics package and the grid graphics system consists of functions in the grid package .

There are many other graphics functions provided in add-on graphics packages, which build on the functions in the graphics systems. Only one such package, the lattice package. The lattice package builds on the grid system to provide Trellis plots .

There are also add-on graphics device packages that provide additional graphical output formats.

Types of graphics functions :

Plotting commands divided into three basic groups

1. High-level plotting functions create a new plot on the graphics device, possibly with axes, labels, titles and so on.
2. Low-level plotting functions add more information to an existing plot, such as extra points, lines and labels.
3. Interactive graphics functions allow you to interactively add information to, or extract information from the plots.

In addition, R maintains a list of graphical parameters which can be manipulated to customize your plots.

5.4 plot() function

- It is the generic function for plotting of R objects. This can be used in scatter plots.
- Scatter plots show many points plotted in the Cartesian plane.
- Each point represents the values of two variables.
- One variable is chosen in the horizontal axis and another in the vertical axis.
- The simple scatter plot is created using the plot() function.

Syntax :

The basic syntax for creating scatterplot in R is –

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

Following is the description of the parameters used –

- ✓ x is the data set whose values are the horizontal coordinates.
- ✓ y is the data set whose values are the vertical coordinates.
- ✓ main is the tile of the graph.
- ✓ xlab is the label in the horizontal axis.
- ✓ ylab is the label in the vertical axis.
- ✓ xlim is the limits of the values of x used for plotting.
- ✓ ylim is the limits of the values of y used for plotting.
- ✓ axes indicates whether both axes should be drawn on the plot.

Example

We use the data set "mtcars" available in the R environment to create a basic scatterplot. Let's use the columns "wt" and "mpg" in mtcars.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
```

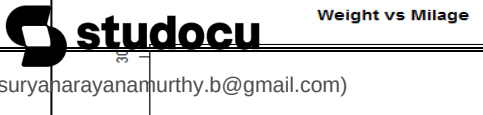
When we execute the above code, it produces the following result –

	wt	mpg
Mazda RX4	2.620	21.0
Mazda RX4 Wag	2.875	21.0
Datsun 710	2.320	22.8
Hornet 4 Drive	3.215	21.4
Hornet Sportabout	3.440	18.7
Valiant	3.460	18.1

Creating the Scatter plot :

The below script will create a scatter plot graph for the relation between wt(weight) and mpg(miles per gallon).

When we execute the above code, it produces the following result –



```
# Get the input values.
input <- mtcars[,c('wt','mpg')]
# Give the chart file a name.
png(file = "scatterplot.png")
# Plot the chart for cars with weight
#between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
  xlab = "Weight",
  ylab = "Milage",
 xlim = c(2.5,5),
ylim = c(15,30),
 main = "Weight vs Milage"
)
# Save the file.
dev.off()
```

5.5 Displaying multivariate data

5.6 Matrix plots

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is −

pairs(formula, data)

Following is the description of the parameters used −

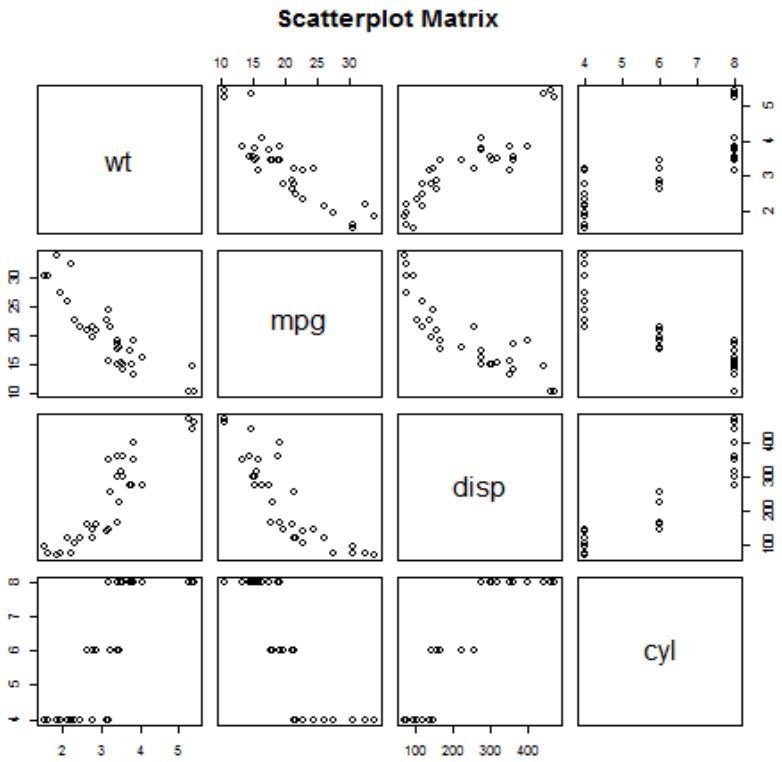
- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.

```
# Give the chart file a name.
png(file = "scatterplot_matrices.png")
# Plot the matrices between 4 variables giving 12 plots.
# One variable with 3 others and total 4 variables.
pairs(~wt+mpg+disp+cyl,data = mtcars,
  main = "Scatterplot Matrix")
# Save the file.
dev.off()
```

When the above code is executed we get the following output.



Plot Matrices Using a Color/Intensity Grid :

Plots a matrix, m, associating the magnitude of the i,jth cell of m with the color of the i,jth cell of an nrow(m) by ncol(m) grid.

Usage

plot.matrix(x, labels=list(seq(1:dim(x)[1]), seq(1:dim(x)[2])),
drawlab=TRUE, diaglab=TRUE, ...)

Arguments

- x — An input matrix
- labels — A list containing the vectors of row and column labels (respectively)
- drawlab — Add labels to the plot?
- diaglab — Label the diagonal?
- type — what type of plot should be drawn. Possible types are
  "p" for **p**oints,
  "l" for **l**ines,
  "b" for **b**oth,
  "c" for the lines part alone of "b",
  "o" for both '**o**verplotted',
  "h" for '**h**istogram' like (or 'high-density') vertical lines,
  "s" for stair **s**teps,
  "S" for other **s**teps, see 'Details' below,
  "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility.

plot.matrix is particularly valuable for examining large adjacency matrices, whose structure can be non-obvious otherwise.

Examples

#Plot a small adjacency matrix
plot.matrix(rgraph(5))

#Plot a much larger one

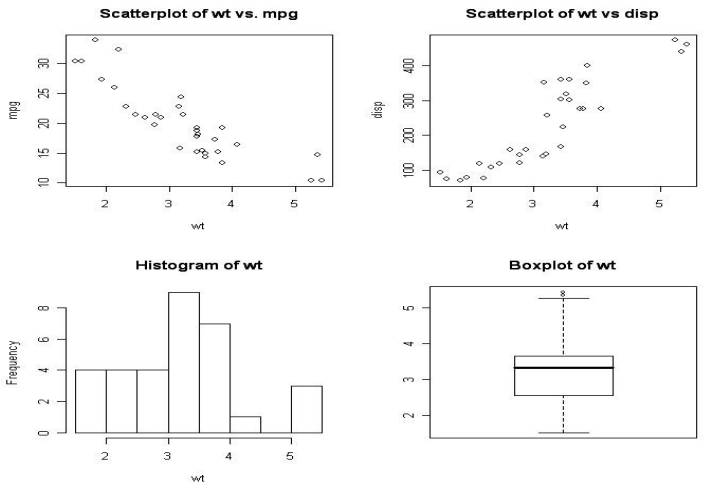
```
plot.matrix(rgraph(100),drawlab=FALSE,diaglab=FALSE)
```

5.7 Multiple plots in one window

R makes it easy to combine multiple plots into one overall graph, using either the par( ) or layout( ) function.

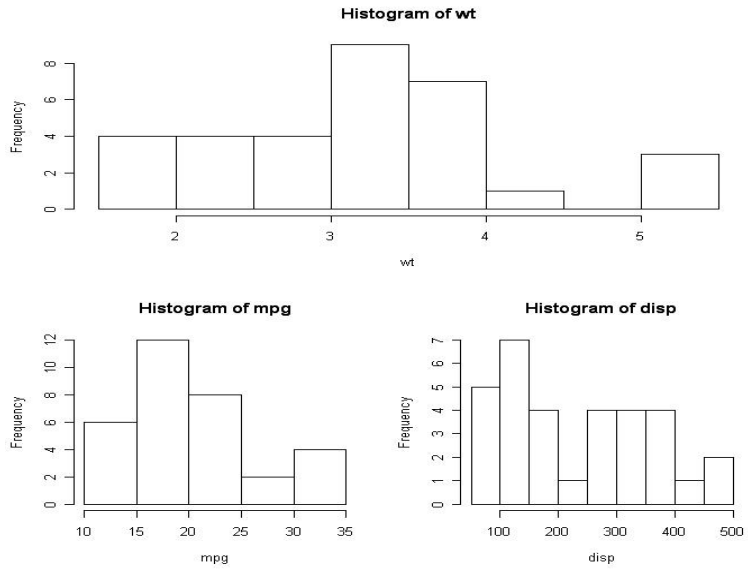
With the par( ) function, you can include the option mfrow=c(nrows, ncols) to create a matrix of nrows x ncols plots that are filled in by row. mfcol=c(nrows, ncols) fills in the matrix by columns.

```
# 4 figures arranged in 2 rows and 2 columns
attach(mtcars)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```



The layout( ) function has the form layout(mat) where mat is a matrix object specifying the location of the N figures to plot.

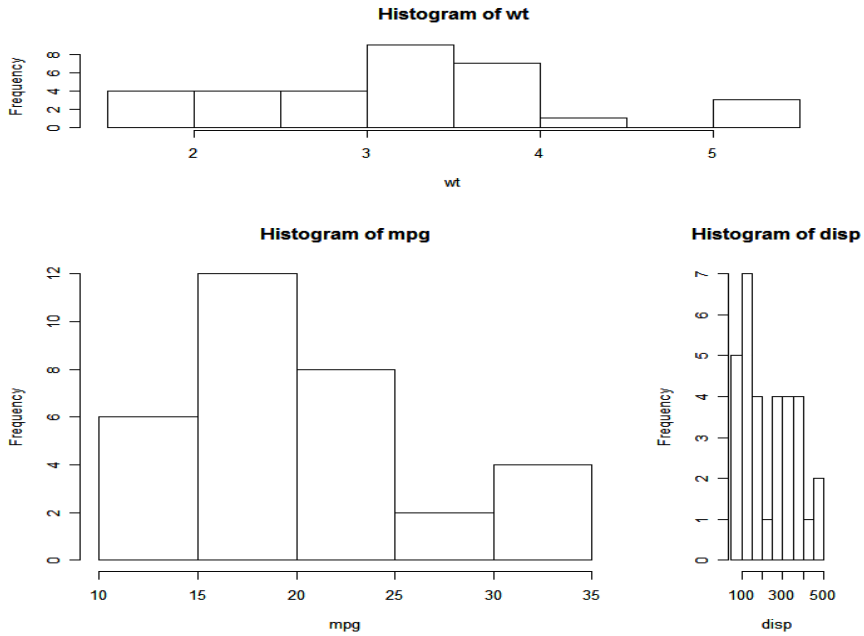
```
# One figure in row 1 and two figures in row 2
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
hist(wt)
hist(mpg)
hist(disp)
```



Optionally, you can include widths= and heights= options in the layout( ) function to control the size of each figure more precisely. These options have the form
widths= a vector of values for the widths of columns
heights= a vector of values for the heights of rows.

Relative widths are specified with numeric values. Absolute widths (in centimetres) are specified with the lcm() function.

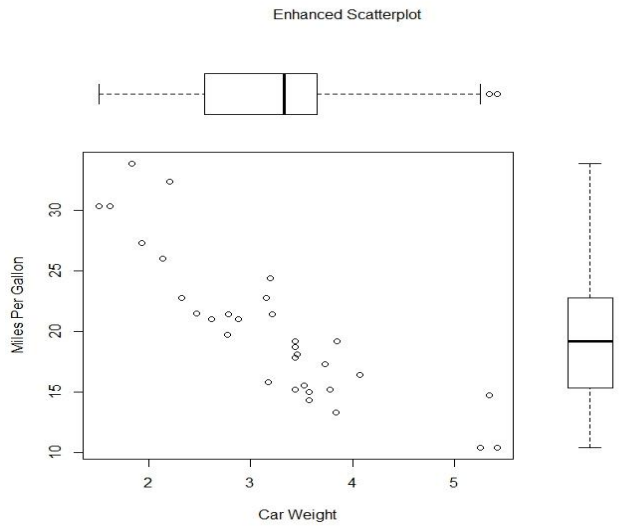
```
# One figure in row 1 and two figures in row 2
# row 1 is 1/3 the height of row 2
# column 2 is 1/4 the width of the column 1
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE),
 widths=c(3,1), heights=c(1,2))
hist(wt)
hist(mpg)
hist(disp)
```



Creating a figure arrangement with fine control :

In the following example, two box plots are added to scatterplot to create an enhanced graph.

```
# Add boxplots to a scatterplot
par(fig=c(0,0.8,0,0.8), new=TRUE)
plot(mtcars$wt, mtcars$mpg, xlab="Car Weight",
  ylab="Miles Per Gallon")
par(fig=c(0,0.8,0.55,1), new=TRUE)
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE)
par(fig=c(0.65,1,0,0.8),new=TRUE)
boxplot(mtcars$mpg, axes=FALSE)
mtext("Enhanced Scatterplot", side=3, outer=TRUE, line=-3)
```



To understand this graph, think of the full graph area as going from (0,0) in the lower left corner to (1,1) in the upper right corner. The format of the fig= parameter is a numerical vector of the form c(x1, x2, y1, y2). The first fig= sets up the scatterplot going from 0 to 0.8 on the x axis and 0 to 0.8 on the y axis. The top boxplot goes from 0 to 0.8 on the x axis and 0.55 to 1 on the y axis. I chose 0.55 rather than 0.8 so that the top figure will be pulled closer to the scatter plot. The right hand boxplot goes from 0.65 to 1 on the x axis and 0 to 0.8 on the y axis. Again, I chose a value to pull the right hand boxplot closer to the scatterplot. You have to experiment to get it just right.

fig= starts a new plot, so to add to an existing plot use new=TRUE.

You can use this to combine several plots in any arrangement into one graph.

5.8 Exporting graph

Since R runs on so many different operating systems, and supports so many different graphics formats, it's not surprising that there are a variety of ways of saving your plots, depending on what operating system you are using, what you plan to do with the graph, and whether you're connecting locally or remotely.

The first step in deciding how to save plots is to decide on the output format that you want to use. The following table lists some of the available formats, along with guidance as to when they may be useful.

Format	Driver	Notes
JPG	Jpeg	Can be used anywhere, but doesn't resize
PNG	Png	Can be used anywhere, but doesn't resize
WMF	win.metafile	Windows only; best choice with Word; easily resizable
PDF	Pdf	Best choice with pdflatex; easily resizable

Postscript	Postscript	Best choice with latex and Open Office; easily resizable

1 A General Method :

- First, here's a general method that will work on any computer with R, regardless of operating system or the way that you are connecting.
- Choose the format that you want to use. In this example, I'll save a plot as a JPG file, so I'll use the jpeg driver.
- The only argument that the device drivers need is the name of the file that you will use to save your graph. Remember that your plot will be stored relative to the current directory. You can find the current directory by typing getwd() at the R prompt.
- You may want to make adjustments to the size of the plot before saving it. Consult the help file for your selected driver to learn how.
- Now enter your plotting commands as you normally would. You will **not** actually see the plot - the commands are being saved to a file instead.
- When you're done with your plotting commands, enter the dev.off() command. This is very important - without it you'll get a partial plot or nothing at all.

So if I wanted to save a jpg file called "rplot.jpg" containing a plot of x and y, I would type the following commands:

```
> jpeg('rplot.jpg')
> plot(x,y)
> dev.off()
```

2 Another Approach :

If you follow the process in the previous section, you'll first have to make a plot to the screen, then re-enter the commands to save your plot to a file. R also provides the dev.copy command, to copy the contents of the graph window to a file without having to re-enter the commands. For most plots, things will be fine, but sometimes translating what was on the screen into a different format doesn't look as nice as it should.

To use this approach, first produce your graph in the usual way. When you're happy with the way it looks, call dev.copy, passing it the driver you want to use, the file name to store it in, and any other arguments appropriate to the driver.

For example, to create a png file called myplot.png from a graph that is displayed by R, type

```
> dev.copy(png,'myplot.png')
> dev.off()
```

Remember that when you save plots this way, the plot isn't actually written to the file until you call dev.off.

3 Local Sessions with Windows or OS X

If you're actually sitting in front of a Windows or Mac computer (i.e. not using ssh to connect), the graphical user interface makes it easy to save files. Under Windows, right click inside the graph window, and choose either "Save as metafile ..." or "Save as postscript ..." If using Word, make sure to save as a metafile.

On a Mac, click on the graphics window to make sure it's the active one, then go to File -> Save in the menubar, and choose a location to save the file. It will be saved as a pdf file, which you can double click to open in Preview, and then use the File -> Save As menu choice to convert to another format.

5.9 Using graphics parameters

The following is a list of arguments available to control the appearance of graphs. This list is not meant to be studied, but rather to serve as a reference throughout the section *Graphical Methods* .

The following arguments can be specified within any graphical function to specify titles, where *string* is the title inserted in quotes, or a character vector containing the title:

- ✓ **main=**"*string*"
  main title, above plot, in enlarged characters
- ✓ **sub=**"*string*"
  sub-title for bottom of plot
- ✓ **main=**"# of chin ups \n Females"
  for titles and sub-titles to run over more than one line, use \n in the character string to start a new line

The following parameters may only be used in high-level graphical functions (those that set up coordinate systems, e.g., plot, qqplot).

- ✓ **xlab=**"*string*", **ylab=**"*string*"
  labels for the x an y axes
- ✓ **axes=**
  logical flag specifying *axes=F* forces Splus to omit drawing the axes
- ✓ **xlim=, ylim=**
  vectors giving the min and max values for the x and y axes
  Splus may extend the axes further to produce pretty tick labels
- ✓ **log=**
  specifies the axes to be logarithmic (ie.: log="xy")
- ✓ **type=**
  type of plot:
  "p" points
  "l" lines
  "b" both
  "o" both - overstruck
  "h" high density
  "n" null (sets up the axes to be filled in later)

The function par() can be used to set graphical parameters which stay in effect throughout all plotting until they are reset in another call to par or by invoking a new graphic device.

- ✓ **par(mfrow=c(2,3))**
  * it is possible to specify the number and arrangement of graphs on a page using the *mfrow=* option
  * *mfrow=c(2,3)* allows 6 plots to appear on a page (2 rows of 6 plots each)
  * because the *mfrow* argument was used within the par function, the set up will stay in effect for all subsequent graphs
- ✓ **par(mfrow=c(1,1))**
  * restores the normal one-graph-per-page form
  * this could also be done by invoking a new graphic device
  > **X11()**
- ✓ **par(mfcol=c(3,2))**
  * the page is set up as above except that the graphs are printed columnwise, *mfrow* prints graphs row by row

The following arguments can be specified either by the par() function or within a graphics function (for temporary parameter settings which stay in effect only for the call to the function and are then reset to their previous values).

- ✓ **pch=**
  plotting character, the default is *pch=*"*"

it is possible to specify either a character, using quotes, or a numeric value, causing Splus to use one of a set of predefined values:

Basic symbols	Superimposed symbols	Filled in symbols
0 square	7 0 and 4	15 square
1 octagon	8 3 and 4	16 octagon
2 triangle	9 3 and 5	17 triangle
3 cross	10 1 and 3	18 diamond
4 X	11 2 and 6	
5 diamond	12 0 and 3	
6 inverted triangle	13 1 and 4	
	14 0 and 2	

- ✓ **cex=**
  character expansion, ie.: cex=0.5 gives half the standard character size
- ✓ **lty=**
  line type: a value of 1 gives solid lines, values of 2,3... will select a non-solid line type
- ✓ **lwd=**
  line width: the default value is 1. Increasing values produce thicker lines
- ✓ **col=**
  numeric color specification (default is 1)
- ✓ **adj=**
  string justification:
  0 left justify
  1 right justify
  0.5 centre or any other fraction
- ✓ **oma=**
  a vector specifying the margins on each of the four sides of a multiple figure region:
  c(bottom, left, top, right)
  where the values specified are the number of lines of text
- ✓ **ask=**
  logical value
  ask=T means graphical device will prompt before going to the next page/screen of output (prevents plots from being erased before they are studied)
- ✓ **lab=**
  controls number of tick intervals and length of labels
  x= *number of tick intervals on x axis (default = 5)*
  y= *number of tick intervals on y axis (default = 5)*
  llen= *length of labels on both axes (default = 7)*
  eg.: lab=c(x=6, y=4, llen=9)
- ✓ **las=**
  type of axis labels:
  0 parallel to axis
  1 horizontal
  2 perpendicular to axis
  For a list of all the available graphical parameters, type **help(par)**
  Other Useful FUnctions
  The arguments *main=* and *sub=* cannot be used from within these functions with the exception of title()
- ✓ **points(x,y,pch=)**
  the function points() allows you to add variables to the current plot
- ✓ **lines(x,y,lty=)**
  the function lines() allows you to add lines to the current plot
- ✓ **abline**
  adds a line to a plot

arguments can be specified as:
(a,b) where a = intercept and b = slope
(coef) vector containing the intercept a and slope b
(reg) a regression object
(h=) vector of y-coordinates for horizontal lines
(v=) vector of x-coordinates for vertical lines

- ✓ **legend(x,y,legend=)**

puts a box at the specified x,y coordinates with examples (if possible) of the lines/points/shading used in the graph identified by user-defined text

- ✓ **identify(x,y,labels=)**

similar to legend(), identify() prints text (labels) at the specified points

- ✓ **locator(n= ,type=)**

returns the coordinates specified interactively on a plot
default is n=500, *type="n"*
by changing type, points and/or lines may be added to the plot
the locator() function can be used within the function legend() or identify() instead of specifying the x and y coordinates: **legend(locator(1),legend)**
the point is identified by placing the cursor over the point and pressing the left mouse button
to exit locator without identifying all "n" points, press the middle mouse button

- ✓ **title(main= , sub= , xlab= , ylab=)**

adds titles and/or axis labels to the current plot
will overwrite any existing titles and/or axis labels

- ✓ **segments(x1,y1,x2,y2)**

adds a line segment with endpoints (x1,y1) and (x2,y2) to the current plot

- ✓ **arrows(x1,y1,x2,y2)**

adds an arrow to the current plot
for a description of optional arguments, type **help(segments)**

- ✓ **text(x,y,labels)**

adds text to the current plot
the coordinates can be given by two arguments (x and y) or by a single argument x which is a univariate time series, a complex vector, a matrix with 2 columns, or a list containing components named x and y

- ✓ **mtext("string",side=)**

adds text in the margins of the current plot
side specifies the side on which the text is to be placed
bottom
left (default)
top
right
other options are available for this function, use help(mtext)

- ✓ **axis**

adds an axis to the current plot

- ✓ **box**

surrounds the current plot with a box