ॐ

SASURIE INSTITUTIONS

COIMBATORE | TIRUPPUR | VIJAYAMANGALAM

SASURIE COLLEGE OF ENGINEERING

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

UNIT I

Introduction to Statistics : Introduction to Statistics, Difference between inferential statistics and descriptivestatistics, Inferential Statistics- Drawing Inferences fromData, RandomVariables, Normal ProbabilityDistribution, Sampling, Sample Statistics and SamplingDistributions.

R overview and Installation- Overview and About R, R and R studio Installation, Descriptive Data analysis using R, Description of basic functions used to describe data in R.

UNIT II

Data manipulation withR: Data manipulation packages-dplyr,data.table, reshape2, tidyr, Lubridate, Data visualization withR.

Data visualization in Watson Studio: Adding data to datarefinery, Visualization of Data on WatsonStudio.

UNIT III

Python: Introduction toPython, How toInstall, Introduction to JupyterNotebook, Python scriptingbasics, Numpy andPandas-Creating and Accessing Numpy Arrays, Introduction to pandas, read and write csv, Descriptive statistics using pandas, Working with text data and datetime columns, Indexing and selecting data, groupby, Merge / Join datasets

UNIT IV

Data Visualization Tools inPython- Introduction to Matplotlib, Basic plots using matplotlib, Specialized Visualization Tools usingMatplotlib, Advanced Visualization Tools usingMatplotlib- WaffleCharts, WordClouds.

UNIT V

Introduction to Seaborn: Seaborn functionalities and usage, Spatial Visualizations and Analysis
in Python with Folium, Case Study.

TEXT BOOKS:

1. Core Python Programming - Second Edition,R. Nageswara Rao, Dreamtech Press.
2. Hands on programming with R by Garrett Grolemund,Shroff/O&#39;Reilly; First edition
3. Fundamentals of Mathematical Statistics by S.C. Gupta, Sultan Chand &amp; Sons

REFERENCE BOOKS:

1. Learn R for Applied Statistics: With Data Visualizations, Regressions, and Statistics by Eric Goh Ming Hui, Apress
2. Python for Data Analysis by William McKinney, Second Edition, O’Reilly Media Inc. \
3. The Comprehensive R Archive Network- https://cran.r-project.org
4. https://seaborn.pydata.org/
5. https://dataplatform.cloud.ibm.com/

INDEX

UNIT – I

Introduction to Statistics

Statistics is a mathematical science that includes methods for collecting, organizing, analyzing and visualizing data in such a way that meaningful conclusions can be drawn.
Statistics is also a field of study that summarizes the data, interpret the data making decisions based on the data.
Statistics is composed of two broad categories:

1. Descriptive Statistics
2. Inferential Statistics

1. Descriptive Statistics

Descriptive statistics describes the characteristics or properties of the data. It helps to summarize the data in a meaningful data in a meaningful way. It allows important patterns to emerge from the data. Data summarization techniques are used to identify the properties of data. It is helpful in understanding the distribution of data. They do not involve in generalizing beyond the data.

Two types of descriptive statistics

1. Measures of Central Tendency: (Mean , Median , Mode)
2. Measures of data spread or dispersion (range, quartiles, variance and standard deviation)

Measures of Central Tendency: (Mean , Median , Mode)

A measure of central tendency is a single value that attempts to describe a set of data by identifying the central position within that set of data. The mean, median and mode are all valid measures of central tendency.

Mean (Arithmetic)

The mean (or average) is the most popular and well known measure of central tendency. It can be used with both discrete and continuous data, although its use is most often with continuous data.
The mean is equal to the sum of all the values in the data set divided by the number of values in the data set. So, if we have values in a data set and they have values $x_1, x_2, \ldots x_n$, the sample mean, usually denoted by $\bar{x}$.

$\bar{\bar{x}} = (x_1,x_2,\ldots x_n)/ n.$

An important property of the mean is that it includes every value in the data set as part of the calculation. In addition, the mean is the only measure of central tendency where the sum of the deviations of each value from the mean is always zero.

Median:

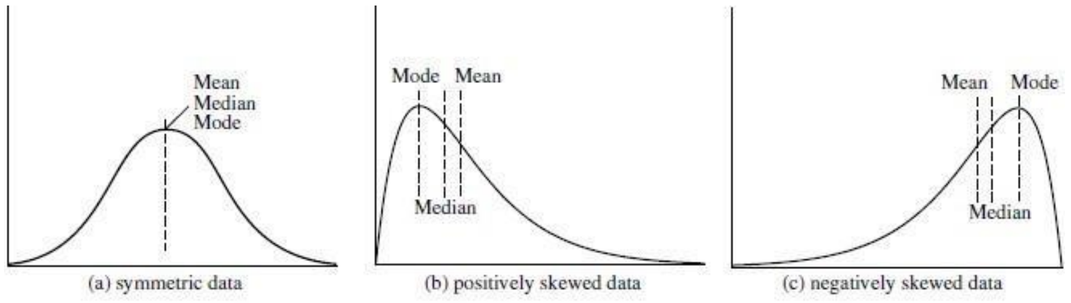
The median is the middlescore for a set of data that has been arranged in order of magnitude. The median is less affected by outliers and skewed data. It is a holistic measure. It is easy method of approximation of median value of a large data set.

Mode

The mode is the most frequent score in our data set. The mode is used for categorical data where we want to know which is the most common category occurring in the population. There are possibilities for the greatest frequency to correspond to different values. This results in more than one,two or more modes in a dataset. They are called as unimodal, bimodal and multimodal datasets. If each data occurs only once then the mode is equal to zero.

Unimodal frequency curve with symmetric data distribution , the mean median and mode are all the same.

In real applications the data is not symmetrical and they are asymmetric.It might be positively skewed or negatively skewed. If positively skewed then mode is smaller than median and in negatively skewed the mode occurs at a value greater than the median.



Mean, median, and mode of symmetric versus positively and negatively skewed data.

Measures of spread:

Measures of spread are the ways of summarizing a group of data by describing how scores are spread out. To describe this spread, a number of statistics are available to us, including the range, quartiles, absolute deviation, variance and standard deviation.

- The degree to which numerical data tend to spread is called the dispersion, or variance of the data. The common measures of data dispersion: Range, Quartiles, Outliers, and Boxplots.

Range : Range of the set is the difference between the largest (max()) and smallest (min()) values. Ex:Step 1: Sort the numbers in order, from smallest to largest: 7, 10, 21, 33, 43, 45, 45, 65, 67, 87, 98, 99

Step 2: Subtract the smallest number in the set from the largest number in the set:
99 – 7 = 92
The range is 92

Quartiles : Percentile : *k*th percentile of a set of data in numerical order is the value *xi h*aving the property that *k* percent of the data entries lie at or below *xi*

- The first quartile (*Q*1) is the 25th percentile;
- The third quartile (*Q*3) is the 75th percentile
- The distance between the first and third quartiles is the range covered by the middle half of the data.
- Interquartile range (*IQR*) and is defined as *IQR* = *Q*3 - Q1.
- Outliers is to single out values falling at least 1.*5 *IQR* above the third quartile or below the first quartile.
- *Five-number summary*: median, the quartiles *Q*1 and *Q*3, and the smallest and largest individual observations comprise the five number summary: *Minimum*; *Q*1; *Median*; *Q*3; *Maximum*

Example : Quartiles

- Start with the following data set:
- 1, 2, 2, 3, 4, 6, 6, 7, 7, 7, 8, 11, 12, 15, 15, 15, 17, 17, 18, 20
- There are a total of twenty data points in the set. There is an even number of data values, hence the median is the mean of the tenth and eleventh values.
- the median is: (7 + 8)/2 = 7.5.
- The median of the first half of the set is found between the fifth and sixth values of:
- 1, 2, 2, 3, 4, 6, 6, 7, 7, 7
- Thus the first quartile is found to equal *Q*1 = (4 + 6)/2 = 5
- To find the third quartile, examine the top half of the original data set. The median of
- 8, 11, 12, 15, 15, 15, 17, 17, 18, 20
- is (15 + 15)/2 = 15. Thus the third quartile *Q*3 = 15.

A small interquartile range indicates data that is clumped about the median. A larger interquartile range shows that the data is more spread out

Variance and Standard Deviation

The variance of N observations, $x_1, x_2, \ldots, x_N$, is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 = \frac{1}{N}\left[\sum x_i^2 - \frac{1}{N}\left(\sum x_i\right)^2\right],$$

where ẍ is the mean value of the observations
The standard deviation, Ϭ, of the observations is the square root of the variance, $Ϭ^2$

measures spread about the mean and should be used only when the mean is chosen Ϭ as the measure of center.
Ϭ =0 only when there is no spread, that is, when all observations have the same value.
Otherwise Ϭ > 0.

Inferential Statistics – Definition and Types

Inferential statistics is generally used when the user needs to make a conclusion about the whole population at hand, and this is done using the various types of tests available. It is a technique which is used to understand trends and draw the required conclusions about a large population by taking and analyzing a sample from it. Descriptive statistics, on the other hand, is only about the smaller sized data set at hand – it usually does not involve large populations. Using variables and the relationships between them from the sample, we will be able to make generalizations and predict other relationships within the whole population, regardless of how large it is.

With inferential statistics, data is taken from samples and generalizations are made about a population.Inferential statistics use statistical models to compare sample data to other samples or to previous research.

There are two main areas of inferential statistics:

1. **Estimating parameters:**

This means taking a statistic from the sample data (for example the sample mean) and using it to infer about a population parameter (i.e. the population mean).There may be sampling variations because of chance fluctuations, variations in sampling techniques, and other sampling errors. Estimation about population characteristics may be influenced by such factors. Therefore, in estimation the important point is that to what extent our estimate is close to the true value. Characteristics of Good Estimator: A good statistical estimator should have the following characteristics, (i) Unbiased (ii) Consistent (iii) Accuracy

i) Unbiased

An unbiased estimator is one in which, if we were to obtain an infinite number ofrandom samples of a certain size, the mean of the statistic would be equal to theparameter. The sample mean, (x) is an unbiased estimate of population mean (μ)because if we look at possible random samples of size N from a population, thenmean of the sample would be equal to μ.

ii) Consistent

A consistent estimator is one that as the sample size increased, the probability thatestimate has a value close to the parameter also increased. Because it is a consistentestimator, a sample mean based on 20 scores has a greater probability of beingcloser to (μ) than does a sample mean based upon only 5 scores

iii) Accuracy

The sample mean is an unbiased and consistent estimator of population mean (μ).But we should not over look the fact that an estimate is just a rough or approximatecalculation. It is unlikely in any estimate that (x) will be exactly equal to populationmean (μ). Whether or not x is a good estimate of (μ) depends upon the representativeness of sample, the sample size, and the variability of scores in the population.

2. **Hypothesis tests.** This is where sample data can be used to answer research questions. For example, we might be interested in knowing if a new cancer drug is effective. Or if breakfast helps children perform better in schools.

Inferential statistics is closely tied to the logic of hypothesis testing. We hypothesize that this value characterise the population of observations. The question is whether that hypothesis is reasonable evidence from the sample. Sometimes hypothesis testing is referred to as statistical decision-making process. In day-to-day situations we are required to take decisions about the population on the basis of sample information.

Statement of Hypothesis

A statistical hypothesis is defined as a statement, which may or may not be true about the population parameter or about the probability distribution of the parameter that we wish to validate on the basis of sample information. Most times, experiments are performed with random samples instead of the entire population and inferences drawn from the observed results are then generalised over to the entire population. But before drawing inferences about the population it should be always kept in mind that the observed results might have come due to chance factor. In order to have an accurate or more precise inference, the chance factor should be ruled out.

Null Hypothesis

The probability of chance occurrence of the observed results is examined by the null hypothesis (H0). Null hypothesis is a statement of no differences. The other way to state null hypothesis is that the two samples came from the same population. Here, we assume that population is normally distributed and both the groups have equal means and standard deviations.
Since the null hypothesis is a testable proposition, there is counter proposition to it known as alternative hypothesis and denoted by H1 . In contrast to null hypothesis, the alternative hypothesis (H1) proposes that

i) the two samples belong to two different populations,

ii) their means are estimates of two different parametric means of the respective population, and

iii) there is a significant difference between their sample means.

The alternative hypothesis (H1) is not directly tested statistically; rather its acceptance or rejection is determined by the rejection or retention of the null hypothesis. The probability ‘p’ of the null hypothesis being correct is assessed by a statistical test. If probability ‘p’ is too low, H0 is rejected and H1 is accepted.

It is inferred that the observed difference is significant. If probability ‘p’ is high, H0 is accepted and it is inferred that the difference is due to the chance factor and not due to the variable factor.

Level of Significance

The level of significance is defined as the probability of rejecting a null hypothesis by the test when it is really true, which is denoted as α. That is, P (Type I error) = α.

Confidence level:

Confidence level refers to the possibility of a parameter that lies within a specified range of values, which is denoted as c. Moreover, the confidence level is connected with the level of significance. The relationship between level of significance and the confidence level is $c=1-\alpha$.

The common level of significance and the corresponding confidence level are given below:

- The level of significance 0.10 is related to the 90% confidence level.
- The level of significance 0.05 is related to the 95% confidence level.
- The level of significance 0.01 is related to the 99% confidence level.

The rejection rule is as follows:

- If p-value $\leq$ level of significance(α), then reject the null hypothesis H_0.
- If p-value $>$ level of significance(α), then do not reject the null hypothesis H_0.

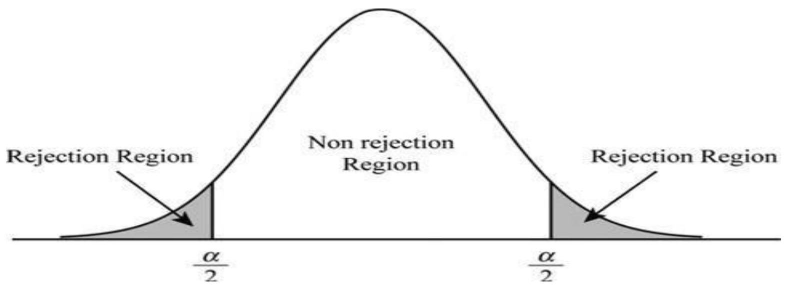
Rejection region:

The rejection region is the values of test statistic for which the null hypothesis is rejected.

Non rejection region:

The set of all possible values for which the null hypothesis is not rejected is called the rejection region.

The rejection region for two-tailed test is shown below:



The rejection region for one-tailed test is given below:

- In the left-tailed test, the rejection region is shaded in left side.
- In the right-tailed test, the rejection region is shaded in right side.

One-tail and Two-tail Test

Depending upon the statement in alternative hypothesis (H1), either a one-tail or twotail test is chosen for knowing the statistical significance. A one-tail test is a directional test. It is formulated to find the significance of both the magnitude and the direction (algebraic sign) of the observed difference between two statistics. Thus, in two-tailed tests researcher is interested in testing whether one sample mean is significantly higher (alternatively lower) than the other sample mean.

Types of Inferential Statistics Tests

There are many tests in this field, of which some of the most important are mentioned below.

1. Linear Regression Analysis

In this test, a linear algorithm is used to understand the relationship between two variables from the data set. One of those variables is the dependent variable, while there can be one or more independent variables used. In simpler terms, we try to predict the value of the dependent variable based on the available values of the independent variables. This is usually represented by using a scatter plot, although we can also use other types of graphs too.

2. Analysis of Variance

This is another statistical method which is extremely popular in data science. It is used to test and analyse the differences between two or more means from the data set. The significant differences between the means are obtained, using this test.

3. Analysis of Co-variance

This is only a development on the Analysis of Variance method and involves the inclusion of a continuous co-variance in the calculations. A co-variate is an independent variable which is continuous, and is used as regression variables. This method is used extensively in statistical modelling, in order to study the differences present between the average values of dependent variables.

4. Statistical Significance (T-Test)

A relatively simple test in inferential statistics, this is used to compare the means of two groups and understand if they are different from each other. The order of difference, or how significant the differences are can be obtained from this.

5. Correlation Analysis

Another extremely useful test, this is used to understand the extent to which two variables are dependent on each other. The strength of any relationship, if they exist, between the two variables can be obtained from this. You will be able to understand whether the variables have a strong correlation or a weak one. The correlation can also be negative or positive, depending upon the variables. A negative correlation means that the value of one variable decreases while the value of the other increases and positive correlation means that the value both variables decrease or increase simultaneously.

Differences between Descriptive and Inferential Statistics

Descriptive Statistics	**Inferential Statistics**
Concerned with describing the target population	Make inferences from the sample and generalize them to the population
Organise, analyse, present the data in a meaningful way	Compare, tests and predicts future outcomes
The analysed results are in the form of graphs, charts etc	The analysed results are the probability scores
Describes the data which is already known	Tries to make conclusions about the population beyond the data available
Tools: Measures of central tendency and measures of spread	Tools: Hypothesis tests, analysis of variance etc

Random Variables

A random variable, X, is a variable whose possible values are numerical outcomes of a random phenomenon. There are two types of random variables, discrete and continuous.

Example of Random variable

- A person’s blood type
- Number of leaves on a tree
- Number of times a user visits LinkedIn in a day
- Length of a tweet.

Discrete Random Variables :

A discrete random variable is one which may take on only a countable number of distinct values such as 0,1,2,3,4,........ Discrete random variables are usually counts. If a random variable can take only a finite number of distinct values, then it must be discrete. Examples of discrete random variables include the number of children in a family, the Friday night attendance at a cinema, the number of patients in a doctor's surgery, the number of defective light bulbs in a box of ten.

The ***probability distribution*** of a discrete random variable is a list of probabilities associated with each of its possible values. It is also sometimes called the probability function or the probability mass function

Suppose a random variable X may take k different values, with the probability that $X = x_i$ defined to be $P(X = x_i) = p_i$. The probabilities p_i must satisfy the following:

1: $0 \leq p_i \leq 1$ *for each i*

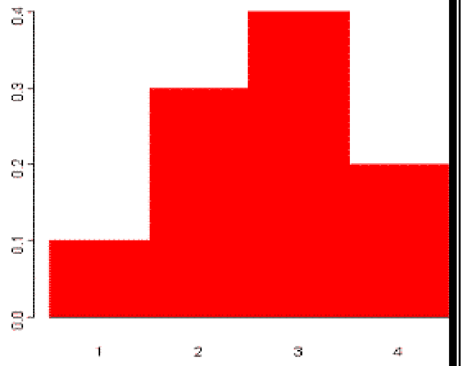
2: $p_1 + p_2 + ... + p_k = 1.$

Example

Suppose a variable X can take the values 1, 2, 3, or 4. The probabilities associated with each outcome are described by the following table:

Outcome	1	2	3	4
Probability	0.1	0.3	0.4	0.2

The probability that X is equal to 2 or 3 is the sum of the two probabilities: $P(X = 2 \text{ or } X = 3) = P(X = 2) + P(X = 3) = 0.3 + 0.4 = 0.7$. Similarly, the probability that X is greater than 1 is equal to 1 - $P(X = 1)$ = 1 - 0.1 = 0.9, by the complement rule.



Continuous Random Variables

A ***continuous random variable*** is one which takes an infinite number of possible values. Continuous random variables are usually measurements. Examples include height, weight, the amount of sugar in an orange, the time required to run a mile.

A continuous random variable is not defined at specific values. Instead, it is defined over an *interval* of values, and is represented by the ***area under a curve*** (known as an *integral*). The probability of observing any single value is equal to 0, since the number of values which may be assumed by the random variable is infinite.

Suppose a random variable *X* may take all values over an interval of real numbers. Then the probability that *X* is in the set of outcomes *A*, *P(A)*, is defined to be the area above *A* and under a curve. The curve, which represents a function *p(x)*, must satisfy the following:

1: *The curve has no negative values (p(x) $\geq$ 0 for all x)*

2: *The total area under the curve is equal to 1.*

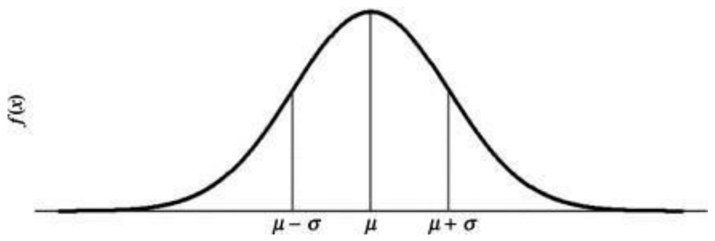
A curve meeting these requirements is known as a ***density curve***.

All random variables (discrete and continuous) have a ***cumulative distribution function***. It is a function giving the probability that the random variable *X* is less than or equal to *x*, for every value *x*. For a discrete random variable, the cumulative distribution function is found by summing up the probabilities.

Normal Probability Distribution

The Bell-Shaped Curve

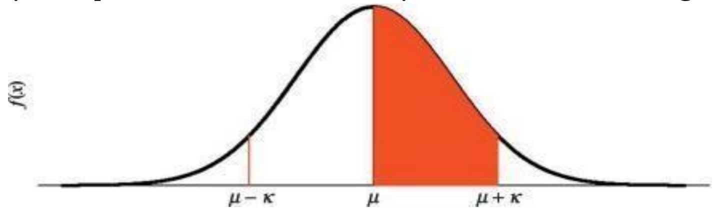
The **Bell-shaped Curve** is commonly called the **normal curve** and is mathematically referred to as the Gaussian probability distribution. Unlike Bernoulli trials which are based on discrete counts, the **normal distribution** is used to determine the probability of a continuous random variable.



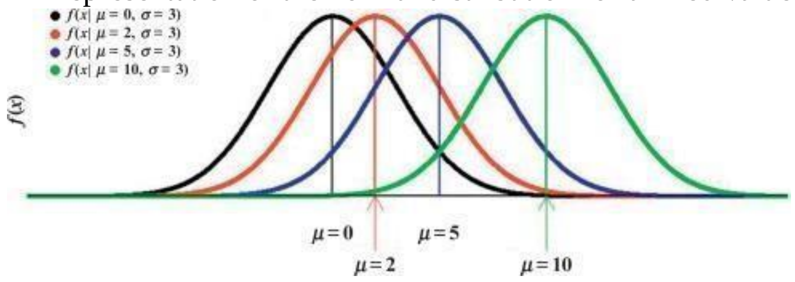
The **normal** or Gaussian Probability Distribution is most popular and important because of its unique mathematical properties which facilitate its application to practically any physical problem in the real world.The constants $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are the parameters;

- "$\boldsymbol{\mu}$" is the population true mean (or expected value) of the subject phenomenon characterized by the continuous random variable, ***X***,
- "$\boldsymbol{\sigma}^2$" is the population true variance characterized by the continuous random variable, ***X***.
- Hence, "$\boldsymbol{\sigma}$" the population standard deviation characterized by the continuous random variable ***X***;
- the points located at $\mu-\sigma$ and $\mu+\sigma$ are the points of inflection; that is, where the graph changes from cupping up to cupping down

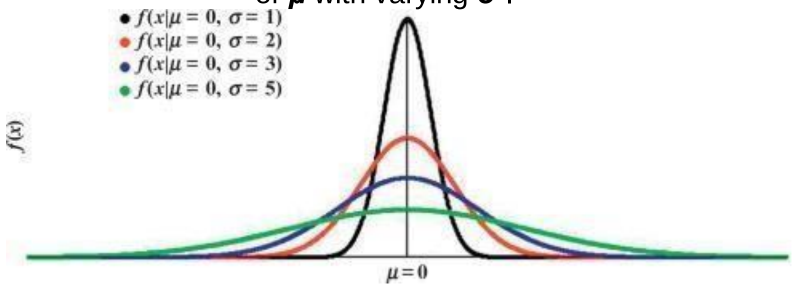
The **normal curve graph of the normal probability distribution**) is **symmetric** with respect to the mean ***μ*** as the **central position**. That is, the area between ***μ*** and ***κ*** units to the left of ***μ*** is equal to the area between ***μ*** and ***κ*** units to the right of ***μ***.



There is not a unique **normal probability distribution.**The figure below is a graphical representation of the normal distribution for a fixed value of σ2 with μ varying.



The figure below is a graphical representation of the **normal distribution** for a fixed value of ***μ*** with varying ***σ***^{2}.



SAMPLING and SAMPLING DISTRIBUTION

Sampling is a process used in statistical analysis in which a predetermined number of observations are taken from a larger population.It helps us to make statistical inferences about the population.A population can be defined as a whole that includes all items and characteristics of the research taken into study. However, gathering all this information is time consuming and costly. We therefore make inferences about the population with the help of samples.

Random sampling:

In data collection, every individual observation has equal probability to be selected into a sample. In random sampling, there should be no pattern when drawing a sample.

Probability sampling:

It is the sampling technique in which every individual unit of the population has greater than zero probability of getting selected into a sample.

Non-probability sampling:

It is the sampling technique in which some elements of the population have no probability of getting selected into a sample.

Cluster samples:

It divides the population into groups (clusters). Then a random sample is chosen from the clusters.

Systematic sampling : select sample elements from an ordered frame. A sampling frame is just a list of participants that we want to get a sample from.

Stratified sampling : sample each subpopulation independently. First, divide the population into homogeneous (very similar) subgroups before getting the sample. Each population member only belongs to one group. Then apply simple random or a systematic method within each group to choose the sample.

Sampling Distribution

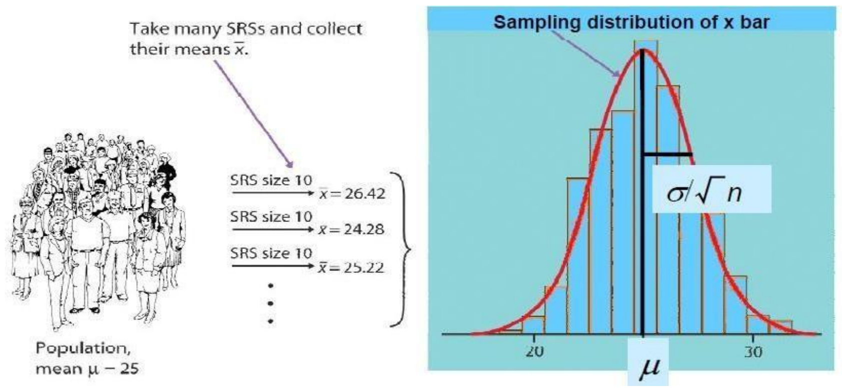
A sampling distribution is a probability distribution of a statistic. It is obtained through a large number of samples drawn from a specific population.It is the distribution of all possible values taken by the statistic when all possible samples of a fixed size n are taken from the population.

Sampling Distributions and Inferential Statistics

Sampling distributions are important for inferential statistics. A population is specified and the sampling distribution of the mean and the range were determined. In practice, the process proceeds the other way: the sample data is collected and from these data we estimate parameters of the sampling distribution. This knowledge of the sampling distribution can be very useful.

- Knowing the degree to which means from different samples would differ from each other and from the population mean (this would give an idea of how close the particular sample mean is likely to be to the population mean)
- The most common measure of how much sample means differ from each other is the standard deviation of the sampling distribution of the mean. This standard deviation is called the standard error of the mean.
- If all the sample means were very close to the population mean, then the standard error of the mean would be small. On the other hand, if the sample means varied considerably, then the standard error of the mean would be large.

Sampling distribution of the sample mean

1. We take many random samples of a given size n from a population with mean μ and standard deviation σ. 2. Some sample means will be above the population mean μ and some will be below, making up the sampling distribution.	 Take many SRSs and collect their means $\bar{x}$. SRS size 10 $\bar{x}$ = 26.42 SRS size 10 $\bar{x}$ = 24.28 SRS size 10 $\bar{x}$ = 25.22 Population, mean μ = 25 Sampling distribution of x bar $\sigma/\sqrt{n}$ 20 30 μ

For any population with mean μ and standard deviation σ:

- The **mean,** or center of the sampling distribution of $\bar{x}$, is equal to the population mean μ: $\mu_{\bar{x}} = \mu$.
- The **standard deviation** of the sampling distribution is $\sigma/\sqrt{n}$, where n is the sample size : $\sigma_{\bar{x}} = \sigma/\sqrt{n}$.

Application

Hypokalemia is diagnosed when blood potassium levels are below 3.5mEq/dl. Let's assume that we know a patient whose measured potassium levels vary daily according to a normal distribution $N(\mu = 3.8,\ \sigma = 0.2)$.

If only one measurement is made, what is the probability that this patient will be misdiagnosed with Hypokalemia?

$$z = \frac{(x-\mu)}{\sigma} = \frac{3.5-3.8}{0.2} = -1.5\ , \qquad P(z < -1.5) = 0.0668 \approx 7\%$$

Instead, if measurements are taken on 4 separate days, what is the probability of a misdiagnosis?

$$z = \frac{(\bar{x}-\mu)}{\sigma/\sqrt{n}} = \frac{3.5-3.8}{0.2/\sqrt{4}} = -3\ , \qquad P(z < -3) = 0.0013 \approx 0.1\%$$

R overview and Installation

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 **GNUs**.

Features of R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

To Install R:

1. Open an internet browser and go to www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for Windows" link at the top of the page.
5. Click on the "install R for the first time" link at the top of the page.
6. Click "Download R for Windows" and save the executable file somewhere on computer. Run the .exe file and follow the installation instructions.
7. Now that R is installed, next step is to download and install RStudio.

To Install RStudio

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions.

R Command Prompt

Once R environment setup is done, then it’s easy to start R command prompt by just typing the following command at command prompt – “$ R”

This will launch R interpreter and will get a prompt > where we can start typing your program as follows −

```
> myString <- "Hello, World!"
> print ( myString)

[1] "Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

R Script File

execute scripts at command prompt with the help of R interpreter called **Rscript**.

```
# My first program in R Programming

myString <- "Hello, World!"
print ( myString)
```

Save the above code in a file test.R and execute it at command prompt as given below.

```
$ Rscript test.R
```

When we run the above program, it produces the following result.

```
"Hello, World!"
```

Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing actual program. Single comment is written using # in the beginning of the statement as follows −

```
# My first program in R Programming
```

R does not support multi-line comments but they can be written as follows:

```
"This is a demo for multi-line comments and it should be put inside either a
single OR double quote"

myString <- "Hello, World!"
print ( myString)
```

Result for above code is:

"Hello, World!"

R data types:

The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are −

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify

Logical	TRUE, FALSE	v <- TRUE
		print(class(v))
		[1] "logical"

Numeric	12.3, 5, 999	v <- 23.5
		print(class(v))
		[1] "numeric"

Integer	2L, 34L, 0L	v <- 2L
		print(class(v))
		[1] "integer"

Complex	3 + 2i	v <- 2+5i
		print(class(v))
		[1] "complex"

Character	'a' , '"good", "TRUE", '23.4'	v <- "TRUE"
		print(class(v))
		[1] "character"

Raw	"Hello" is stored as 48 65 6c 6c 6f	v <- charToRaw("Hello")
		print(class(v))
		[1] "raw"

In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above.

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

```
                                    # Create a vector.
apple <- c('red','green',"yellow")
            print(apple)
  # Get the class of the vector.
        print(class(apple))
```

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

```
                    "red"    "green" "yellow"
                                      "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

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

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

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

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
                                    # Create a matrix.
 M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
                          print(M)
```

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

```
                          [,1] [,2] [,3]
                                  [1,] "a" "a" "b"
```

```
[2,] "c" "b" "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

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

```
, , 1
      [,1]     [,2]     [,3]
[1,] "green" "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green" "yellow" "green"
, , 2
      [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green" "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length. Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <- data.frame( gender = c("Male", "Male","Female"), height = c(152, 171.5, 165),
weight = c(81,93, 78), Age = c(42,38,26))
print(BMI)
```

Result −

```
gender height weight Age
1 Male 152.0      81  42
2 Male 171.5      93  38
3 Female 165.0     78  26
```

R - Variables

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

R - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

types of operators in R programming −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

Descriptive Data analysis using R:

R provides a wide range of functions for obtaining summary statistics. One method of obtaining descriptive statistics is to use the **sapply( )** function with a specified summary statistic.

```
sapply(mydata, mean, na.rm=TRUE)
```

Possible functions used in sapply include **mean, sd, var, min, max, median, range, and quantile**.

Check your data

You can inspect your data using the functions **head**() and **tails**(), which will display the first and the last part of the data, respectively.

```
# Print the first 6 rows
head(my_data, 6)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1      5.1         3.5         1.4      0.2    setosa
2      4.9         3.0         1.4      0.2    setosa
3      4.7         3.2         1.3      0.2    setosa
4      4.6         3.1         1.5      0.2    setosa
5      5.0         3.6         1.4      0.2    setosa
6      5.4         3.9         1.7      0.4    setosa
```

R functions for computing descriptive statistics

Some R functions for computing descriptive statistics:

Description	R function
Mean	**mean**()
Standard deviation	**sd**()
Variance	**var**()
Minimum	**min**()
Maximum	**maximum**()
Median	**median**()
Range of values (minimum and maximum)	**range**()
Sample quantiles	**quantile**()
Generic function	**summary**()
Interquartile range	**IQR**()

Descriptive statistics for a single group

Measure of central tendency: mean, median, mode

Roughly speaking, the central tendency measures the “average” or the “middle” of your data. The most commonly used measures include:

- the mean: the average value. It’s sensitive to outliers.
- the median: the middle value. It’s a robust alternative to mean.
- and the mode: the most frequent value

In R,

- The function **mean**() and **median**() can be used to compute the mean and the median, respectively;
- The function **mfv**() [in the **modeest** R package] can be used to compute the mode of a variable.

The R code below computes the mean, median and the mode of the variable *Sepal.Length* [in *my_data* data set]:

```
# Compute the mean value
mean(my_data$Sepal.Length)
[1] 5.843333

# Compute the median value
median(my_data$Sepal.Length)
[1] 5.8

# Compute the mode
# install.packages("modeest")
require(modeest)
mfv(my_data$Sepal.Length)
[1] 5
```

Measure of variability

Measures of variability gives how "spread out" the data are.

Range: minimum & maximum

- **Range** corresponds to biggest value minus the smallest value. It gives you the full spread of the data.

```
# Compute the minimum value
min(my_data$Sepal.Length)
[1] 4.3

# Compute the maximum value
max(my_data$Sepal.Length)
[1] 7.9

# Range
range(my_data$Sepal.Length)
[1] 4.3 7.9
```

Interquartile range

The **interquartile range** (IQR) - corresponding to the difference between the first and third quartiles - is sometimes used as a robust alternative to the standard deviation.

- R function:

```
quantile(x, probs=seq(0, 1, 0.25))
```

- **x**: numeric vector whose sample quantiles are wanted.
- **probs**: numeric vector of probabilities with values in [0,1].
- Example:

```
quantile(my_data$Sepal.Length)
0% 25% 50% 75% 100%
4.3 5.1 5.8 6.4  7.9
```

To compute deciles (0.1, 0.2, 0.3, …., 0.9), use this:

```
quantile(my_data$Sepal.Length, seq(0, 1, 0.1))
```

To compute the interquartile range, type this:

```
IQR(my_data$Sepal.Length)
[1] 1.3
```

Variance and standard deviation

The variance represents the average squared deviation from the mean. The standard deviation is the square root of the variance. It measures the average deviation of the values, in the data, from the mean value.

```
# Compute the variance
var(my_data$Sepal.Length)

# Compute the standard deviation =
# square root of th variance
sd(my_data$Sepal.Length)
```

Computing an overall summary of a variable and an entire data frame

summary() function

- **Summary of a single variable**. Five values are returned: the mean, median, 25th and 75th quartiles, min and max in one single line call:

```
summary(my_data$Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

- **Summary of a data frame**. In this case, the function **summary**() is automatically applied to each column. The format of the result depends on the type of the data contained in the column. For example:
  - If the column is a numeric variable, mean, median, min, max and quartiles are returned.
  - If the column is a factor variable, the number of observations in each group is returned.

```
summary(my_data, digits=1)
 Sepal.Length Sepal.Width Petal.Length  Petal.Width  Species
 Min.   :4    Min.   :2   Min.   :1    Min.   :0.1   setosa    :50
 1st Qu.:5    1st Qu.:3   1st Qu.:2    1st Qu.:0.3   versicolor:50
 Median :6    Median :3   Median :4    Median :1.3   virginica :50
 Mean   :6    Mean   :3   Mean   :4    Mean   :1.2
 3rd Qu.:6    3rd Qu.:3   3rd Qu.:5    3rd Qu.:1.8
 Max.   :8    Max.   :4   Max.   :7    Max.   :2.5
```

sapply() function

```
# Compute the mean of each column
sapply(my_data[, -5], mean)
Sepal.Length Sepal.Width Petal.Length  Petal.Width
```

```
                    5.843333     3.057333     3.758000     1.199333
# Compute quartiles
sapply(my_data[, -5], quantile)
                Sepal.Length Sepal.Width Petal.Length Petal.Width
 0%          4.3         2.0         1.00        0.1
 25%         5.1         2.8         1.60    0.3
 50%         5.8         3.0         4.35    1.3
 75%         6.4         3.3         5.10    1.8
100%         7.9         4.4         6.90    2.5
```

Descriptive Data Analysis using R > Description of Basic Functions used to Describe Data in R

builtins()	# List all built-in functions
help() or ? or ??	#i.e. help(boxplot)
getwd() and setwd()	# working with a file directory
q()	#To close R
ls()	#Lists all user defined objects.
rm()	#Removes objects from an environment.
demo()	#Lists the demonstrations in the packages that are loaded.
demo(package = .packages(all.available = TRUE))	#Lists the demonstrations in all installed packages.
?NA	# Help page on handling of missing data values
abs(x)	# The absolute value of "x"
append()	# Add elements to a vector
cat(x)	# Prints the arguments
cbind()	# Combine vectors by row/column (cf. "paste" in Unix)
grep()	# Pattern matching
identical()	# Test if 2 objects are *exactly* equal
length(x)	# Return no. of elements in vector x
ls()	# List objects in current environment
mat.or.vec()	# Create a matrix or vector
paste(x)	# Concatenate vectors after converting to character
range(x)	# Returns the minimum and maximum of x
rep(1,5)	# Repeat the number 1 five times
rev(x)	# List the elements of "x" in reverse order
seq(1,10,0.4)	# Generate a sequence (1 -> 10, spaced by 0.4)
sequence()	# Create a vector of sequences
sign(x)	# Returns the signs of the elements of x
sort(x)	# Sort the vector x
order(x)	# list sorted element numbers of x
tolower(),toupper()	# Convert string to lower/upper case letters
unique(x)	# Remove duplicate entries from vector
vector()	# Produces a vector of given length and mode
formatC(x)	# Format x using 'C' style formatting specifications
floor(x), ceiling(x), round(x),	# rounding functions

signif(x), trunc(x)	
Sys.time()	# Return system time
Sys.Date()	# Return system date
getwd()	# Return working directory
setwd()	# Set working directory

Inferential statistics using R

Simple linear regression analysis

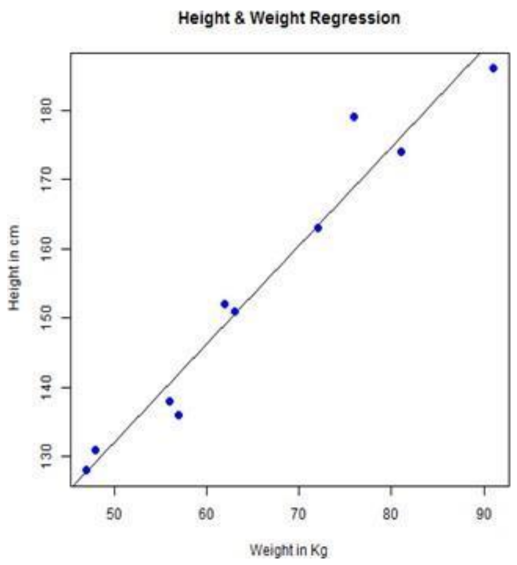
- Regression analysis is a very widely used statistical tool to establish a relationship model between two variables
- One of these variable is called predictor variable
- The other variable is called response variable
- The general mathematical equation for a linear regression is y = mx + b

	Register_no	Name	Dept	CGPA	Height	Weight
1	18N312001	JOHN	IT	8.5	151	63
2	18N312005	SIM	CSE	9.2	174	81
3	18N312011	TIM	IT	9.5	138	56
4	18N312061	LILLY	IT	9.34	186	91
5	18N312099	CARL	MECH	8.12	128	47

- lm() Function
- This function creates the relationship model between the predictor and the response variable.
- The basic syntax for lm() function in linear regression is −
- lm(formula,data)
- # Apply the lm() function.
- relation <- lm(stud.data$weight~ stud.data$height)
- print(relation)

Output

```
Coefficients:
        (Intercept (m))        x
          -38.4551          0.6746
```



UNIT – II

When it comes to Predictive Modeling, data Manipulation is an important and unavoidable phase. Machine learning algorithms are just not sufficient to build a robust predictive model. The approach must be to understand the business problem, the data, performing data manipulations, and then extracting business insights. Majority of time is spent in understanding the data and manipulating data as required. In this chapter we shall look at the details of Data Manipulation.

Data Manipulation is also called as Data Exploration (also known as Data Wrangling or Data Cleaning). Data Manipulation is done to improve data accuracy and precision. Data Manipulation is a mandatory step when it comes to predictive modeling because of the many faults in data collection process, because of many uncontrollable factors involved in data collection.

In reality, there is no right or wrong way to do Data Manipulation. However, one has to take necessary steps to improve the accuracy. Following are some of the points to be considered for the process of Data Manipulation:

- Inbuilt functions in R can be used for data manipulation. Though it is a good step to start with initially, it is not very efficient, because the process must be repeated for number of times and it is also time consuming.
- Packages in CRAN (Comprehensive R Archived Network) can be used for data manipulation which is more efficient. Using the CRAN packages is the most widely accepted industry way of doing Data Manipulation.
- Machine Learning algorithms can also be used. For example, tree based boosting algorithms take care of missing data and outliers. Though time-efficient but requires thorough understanding of data.

dplyr Package

dplyr is a powerful R-package which transforms and summarizes tabular data with rows and columns. It is best known for data exploration and transformation. Its chaining syntax makes it highly adaptive to use. It includes 5 major data manipulation commands:

- filter – filters the data based on a condition
- select –used to select columns of interest from a data set
- arrange –used to arrange data set values on ascending or descending order

- mutate – used to create new variables from existing variables
- summarise (with group_by) – used to perform analysis by commonly used operations such as min, max, mean count etc.

Filter rows with filter()

filter() command can be used to select a subset of rows in a data frame. The first argument for filter() is the data frame. The subsequent arguments refer to variables within that data frame, selecting rows where the expression is TRUE.

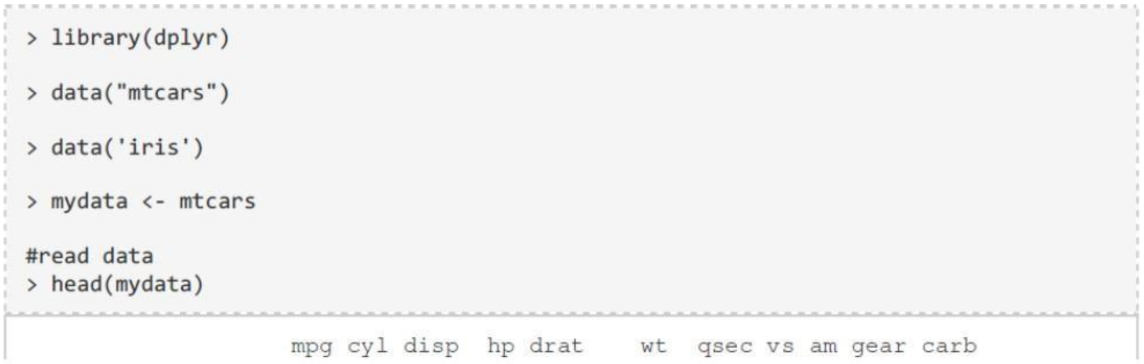
In this example, we will be using dataset named Iris, this dataset is a public data set. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

To install dplyr, use the below command

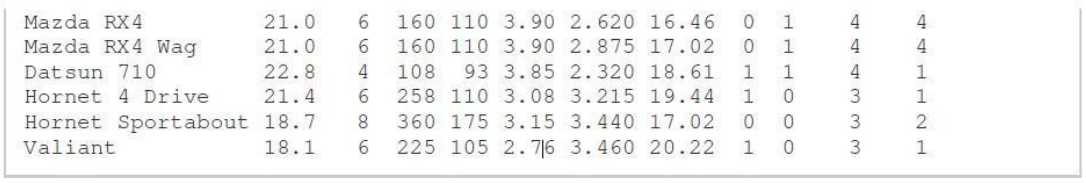
install.packages("dplyr")

To load dplyr, use the below command

library(dplyr)



```
> library(dplyr)

> data("mtcars")

> data('iris')

> mydata <- mtcars

#read data
> head(mydata)

                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
```



```
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
#creating a local dataframe. Local data frame are easier to read
> mynewdata <- tbl_df(mydata)

> myirisdata <- tbl_df(iris)

#now data will be in tabular structure
> mynewdata
```

```
 # A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# ... with 22 more rows
```

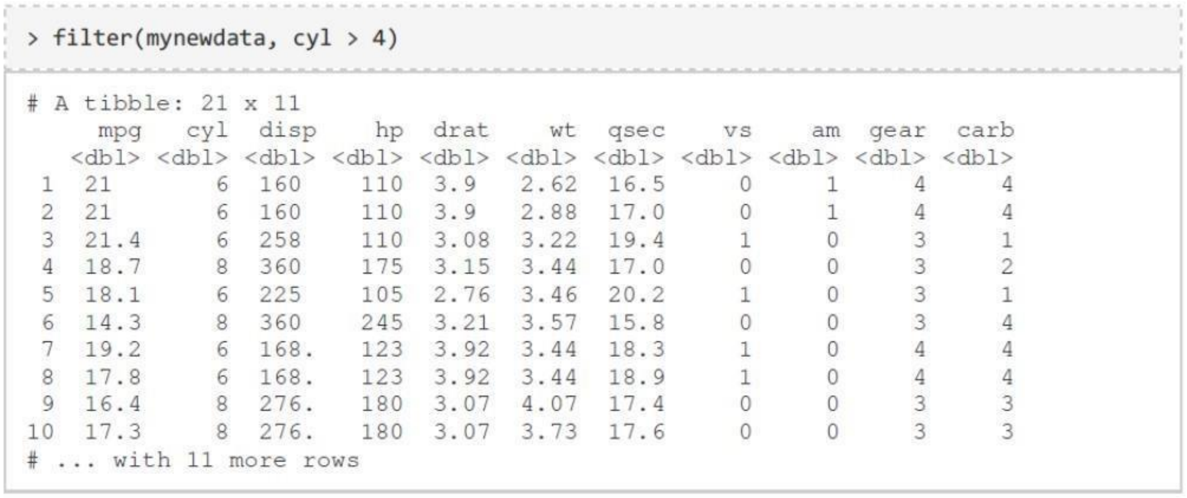
```
> myirisdata
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
```

```
# ... with 140 more rows
```

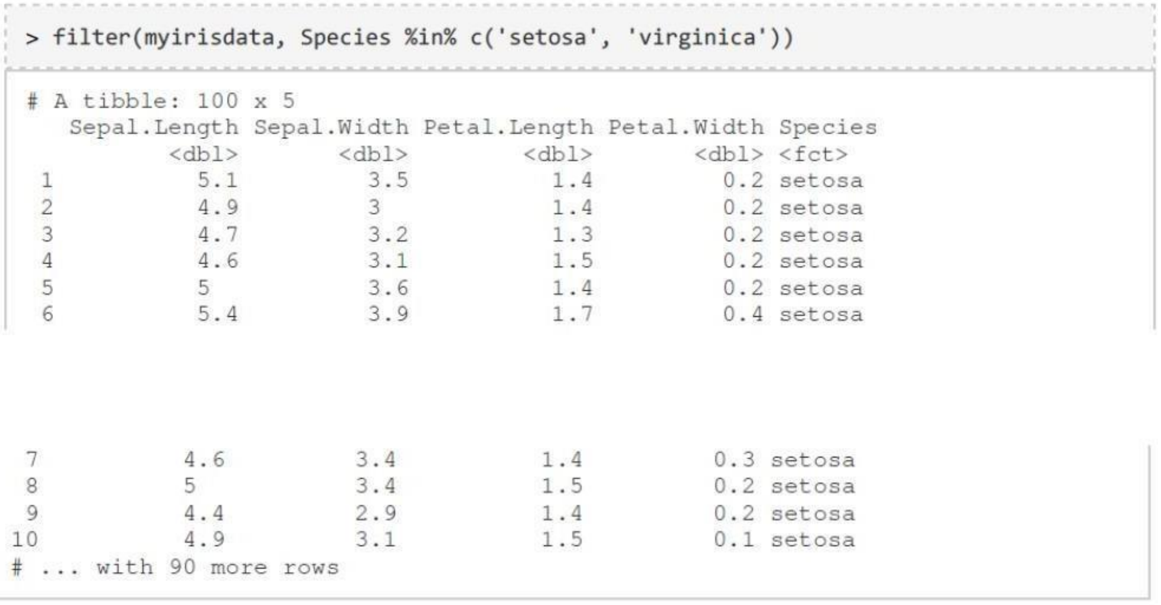
```
#use filter to filter data with required condition
> filter(mynewdata, cyl > 4 & gear > 4 )
```

```
# A tibble: 3 x 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  15.8     8   351   264  4.22  3.17  14.5     0     1     5     4
2  19.7     6   145   175  3.62  2.77  15.5     0     1     5     6
3  15       8   301   335  3.54  3.57  14.6     0     1     5     8
```



```
> filter(mynewdata, cyl > 4)
```

```
# A tibble: 21 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 4  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 5  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 6  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 7  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
 8  17.8     6  168.   123  3.92  3.44  18.9     1     0     4     4
 9  16.4     8  276.   180  3.07  4.07  17.4     0     0     3     3
10  17.3     8  276.   180  3.07  3.73  17.6     0     0     3     3
# ... with 11 more rows
```



```
> filter(myirisdata, Species %in% c('setosa', 'virginica'))
```

```
# A tibble: 100 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 90 more rows
```

<u>Select columns with select()</u>

select() allows one to rapidly zoom in on a useful subset using operations that usually only work on numeric variable positions:

```
# use select to pick columns by name
> select(mynewdata, cyl,mpg,hp)
```

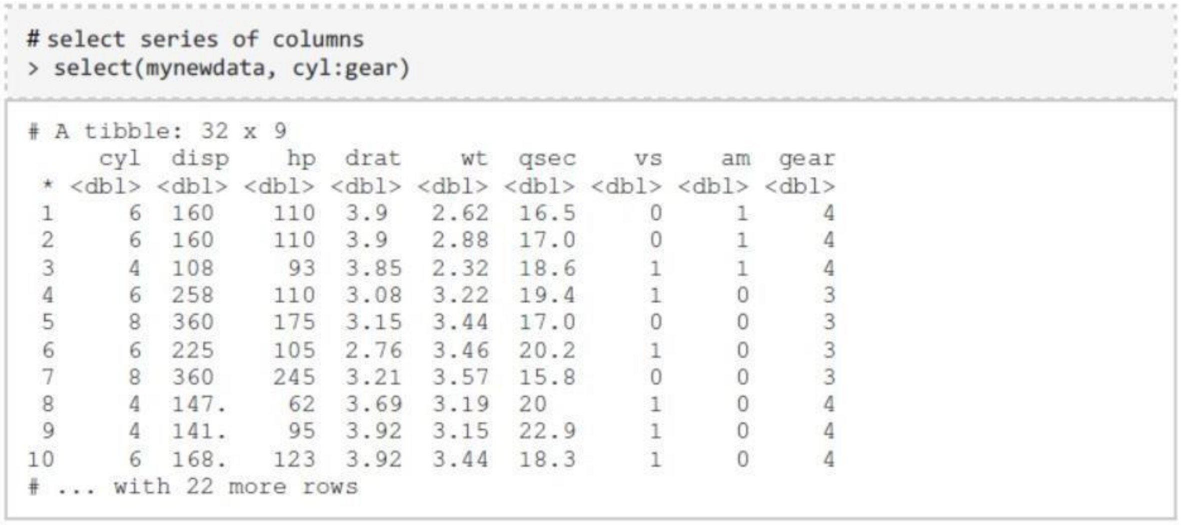
```
# A tibble: 32 x 3
     cyl   mpg    hp
 * <dbl> <dbl> <dbl>
 1     6  21     110
 2     6  21     110
 3     4  22.8    93
 4     6  21.4   110
 5     8  18.7   175
 6     6  18.1   105
 7     8  14.3   245
 8     4  24.4    62
 9     4  22.8    95
10     6  19.2   123
# ... with 22 more rows
```

```
# here you can use (-) to hide columns
> select(mynewdata, -cyl, -mpg )
```

```
# A tibble: 32 x 9
    disp    hp  drat    wt  qsec    vs    am  gear  carb
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  160    110  3.9   2.62  16.5     0     1     4     4
 2  160    110  3.9   2.88  17.0     0     1     4     4
 3  108     93  3.85  2.32  18.6     1     1     4     1
 4  258    110  3.08  3.22  19.4     1     0     3     1
 5  360    175  3.15  3.44  17.0     0     0     3     2
 6  225    105  2.76  3.46  20.2     1     0     3     1
 7  360    245  3.21  3.57  15.8     0     0     3     4
 8  147.    62  3.69  3.19  20       1     0     4     2
 9  141.    95  3.92  3.15  22.9     1     0     4     2
10  168.   123  3.92  3.44  18.3     1     0     4     4
# ... with 22 more rows
```

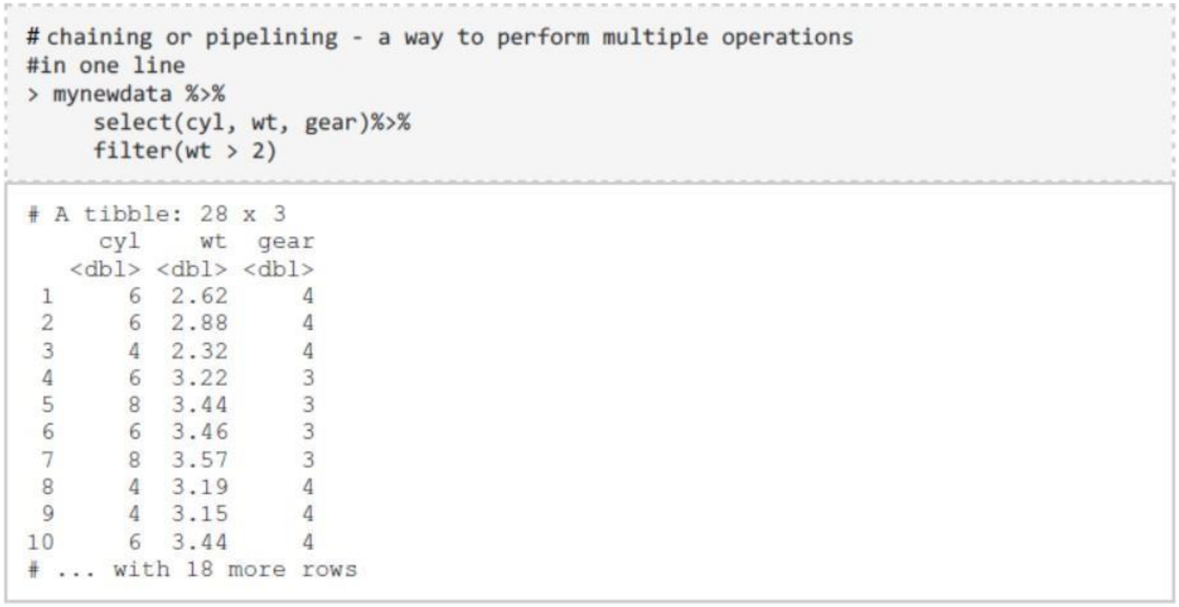
```
# hide a range of columns
> select(mynewdata, -c(cyl,mpg))
```

```
# A tibble: 32 x 9
    disp    hp  drat    wt  qsec    vs    am  gear  carb
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  160    110  3.9   2.62  16.5     0     1     4     4
 2  160    110  3.9   2.88  17.0     0     1     4     4
 3  108     93  3.85  2.32  18.6     1     1     4     1
 4  258    110  3.08  3.22  19.4     1     0     3     1
 5  360    175  3.15  3.44  17.0     0     0     3     2
 6  225    105  2.76  3.46  20.2     1     0     3     1
 7  360    245  3.21  3.57  15.8     0     0     3     4
 8  147.    62  3.69  3.19  20       1     0     4     2
 9  141.    95  3.92  3.15  22.9     1     0     4     2
10  168.   123  3.92  3.44  18.3     1     0     4     4
# ... with 22 more rows
```



```
# select series of columns
> select(mynewdata, cyl:gear)
```

```
# A tibble: 32 x 9
     cyl  disp    hp  drat    wt  qsec    vs    am  gear
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1     6  160    110  3.9   2.62  16.5     0     1     4
 2     6  160    110  3.9   2.88  17.0     0     1     4
 3     4  108     93  3.85  2.32  18.6     1     1     4
 4     6  258    110  3.08  3.22  19.4     1     0     3
 5     8  360    175  3.15  3.44  17.0     0     0     3
 6     6  225    105  2.76  3.46  20.2     1     0     3
 7     8  360    245  3.21  3.57  15.8     0     0     3
 8     4  147.    62  3.69  3.19  20       1     0     4
 9     4  141.    95  3.92  3.15  22.9     1     0     4
10     6  168.   123  3.92  3.44  18.3     1     0     4
# ... with 22 more rows
```

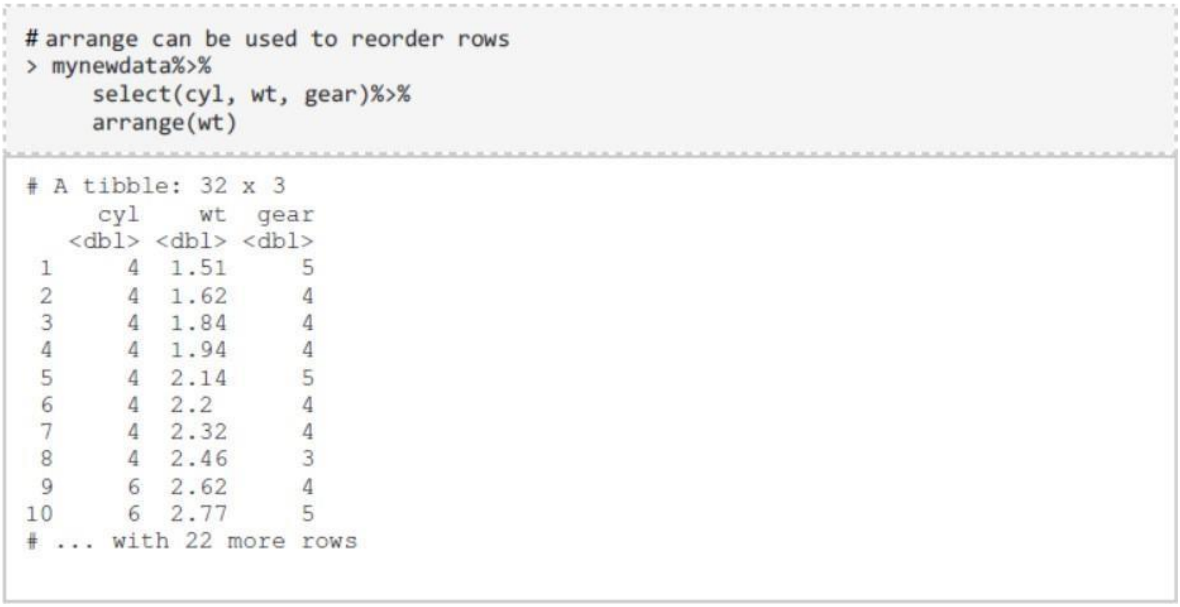


```
# chaining or pipelining - a way to perform multiple operations
#in one line
> mynewdata %>%
     select(cyl, wt, gear)%>%
     filter(wt > 2)
```

```
# A tibble: 28 x 3
     cyl    wt  gear
   <dbl> <dbl> <dbl>
 1     6  2.62     4
 2     6  2.88     4
 3     4  2.32     4
 4     6  3.22     3
 5     8  3.44     3
 6     6  3.46     3
 7     8  3.57     3
 8     4  3.19     4
 9     4  3.15     4
10     6  3.44     4
# ... with 18 more rows
```

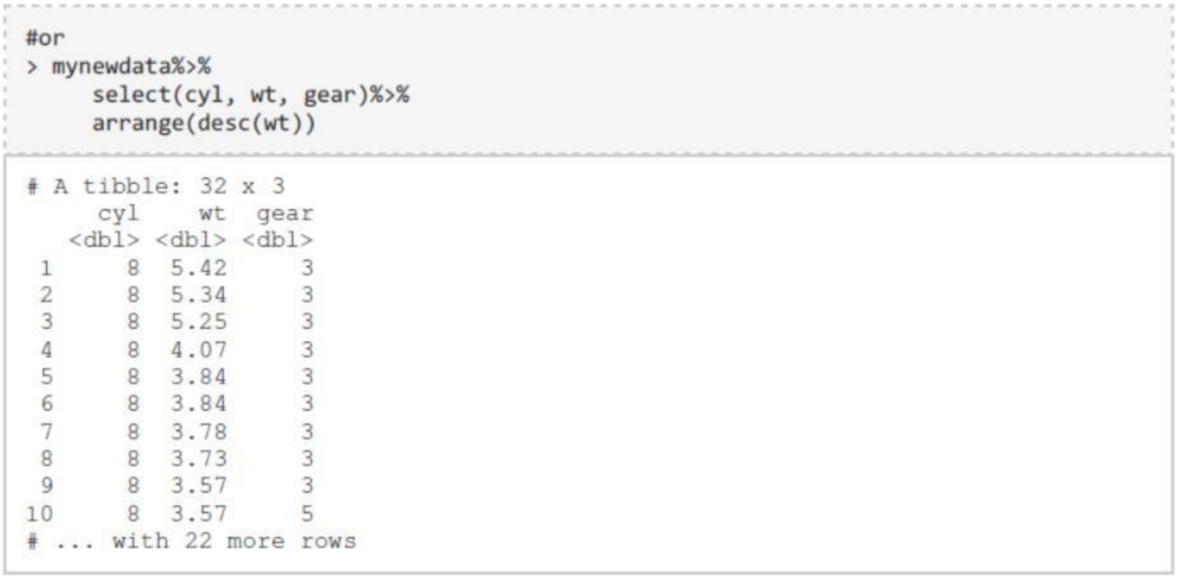
Arrange rows with arrange()

arrange() re-orders the rows. It takes a data frame, and a set of column names (or more complicated expressions) to order the rows. If more than one column name is given, each additional column is used to break ties in the values of preceding columns.



```
# arrange can be used to reorder rows
> mynewdata%>%
    select(cyl, wt, gear)%>%
    arrange(wt)
```

```
# A tibble: 32 x 3
     cyl    wt  gear
   <dbl> <dbl> <dbl>
 1     4  1.51     5
 2     4  1.62     4
 3     4  1.84     4
 4     4  1.94     4
 5     4  2.14     5
 6     4  2.2      4
 7     4  2.32     4
 8     4  2.46     3
 9     6  2.62     4
10     6  2.77     5
# ... with 22 more rows
```

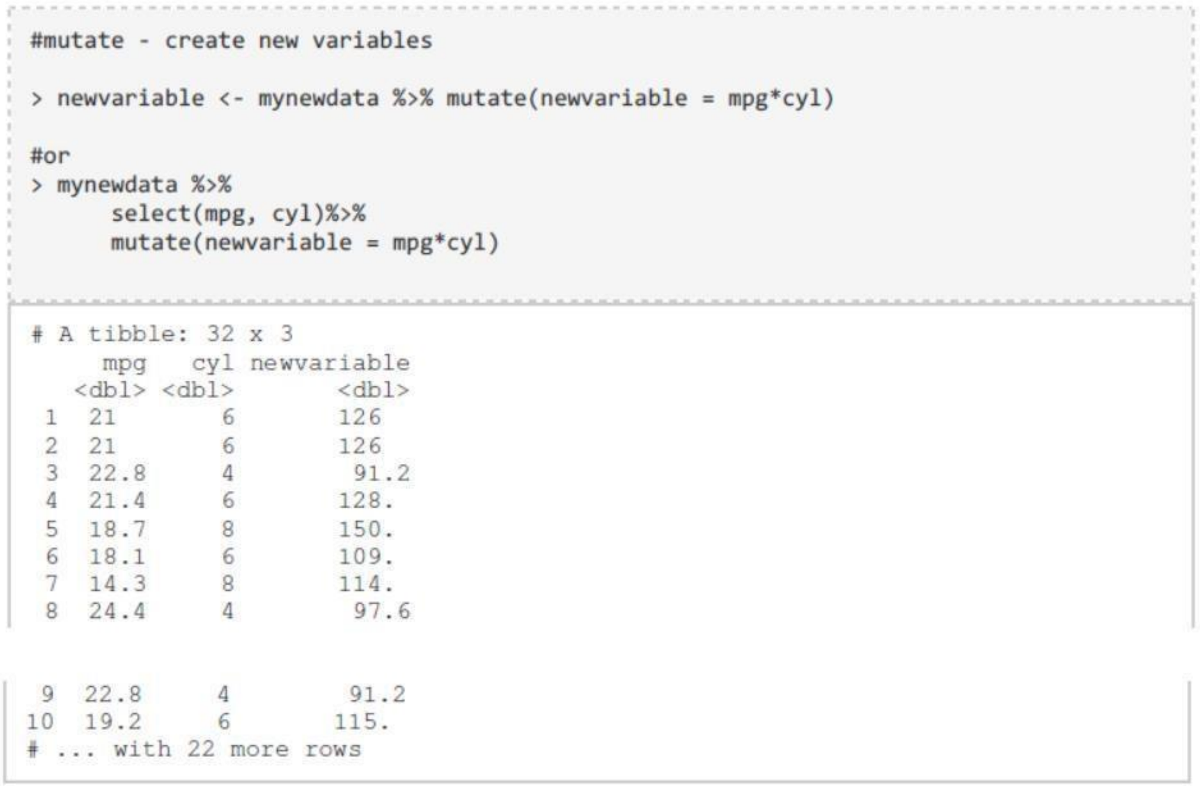


```
#or
> mynewdata%>%
    select(cyl, wt, gear)%>%
    arrange(desc(wt))
```

```
# A tibble: 32 x 3
     cyl    wt  gear
   <dbl> <dbl> <dbl>
 1     8  5.42     3
 2     8  5.34     3
 3     8  5.25     3
 4     8  4.07     3
 5     8  3.84     3
 6     8  3.84     3
 7     8  3.78     3
 8     8  3.73     3
 9     8  3.57     3
10     8  3.57     5
# ... with 22 more rows
```

Add new columns with mutate()

This function, mutate() adds new variables while preserving the existing ones. Mutate() is used to select sets of existing columns and add new columns that are functions of existing columns. Following is the example:

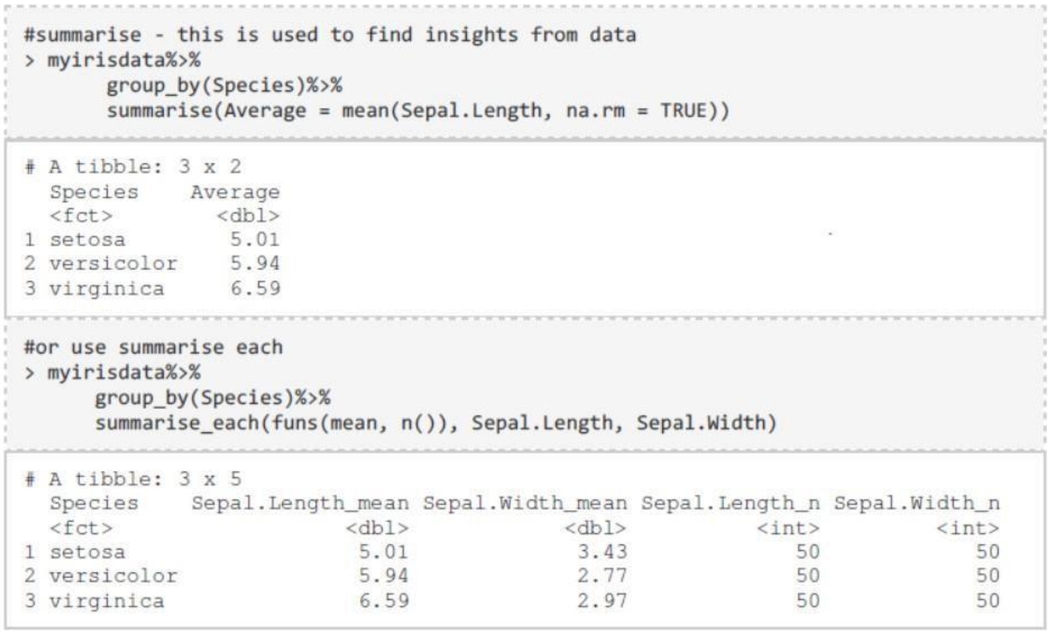


```
#mutate - create new variables

> newvariable <- mynewdata %>% mutate(newvariable = mpg*cyl)

#or
> mynewdata %>%
      select(mpg, cyl)%>%
      mutate(newvariable = mpg*cyl)
```

```
# A tibble: 32 x 3
     mpg   cyl newvariable
   <dbl> <dbl>       <dbl>
 1  21       6       126
 2  21       6       126
 3  22.8     4        91.2
 4  21.4     6       128.
 5  18.7     8       150.
 6  18.1     6       109.
 7  14.3     8       114.
 8  24.4     4        97.6
 9  22.8     4        91.2
10  19.2     6       115.
# ... with 22 more rows
```

Summarise values with summarise()

The summarise() collapses a data frame to a single row.

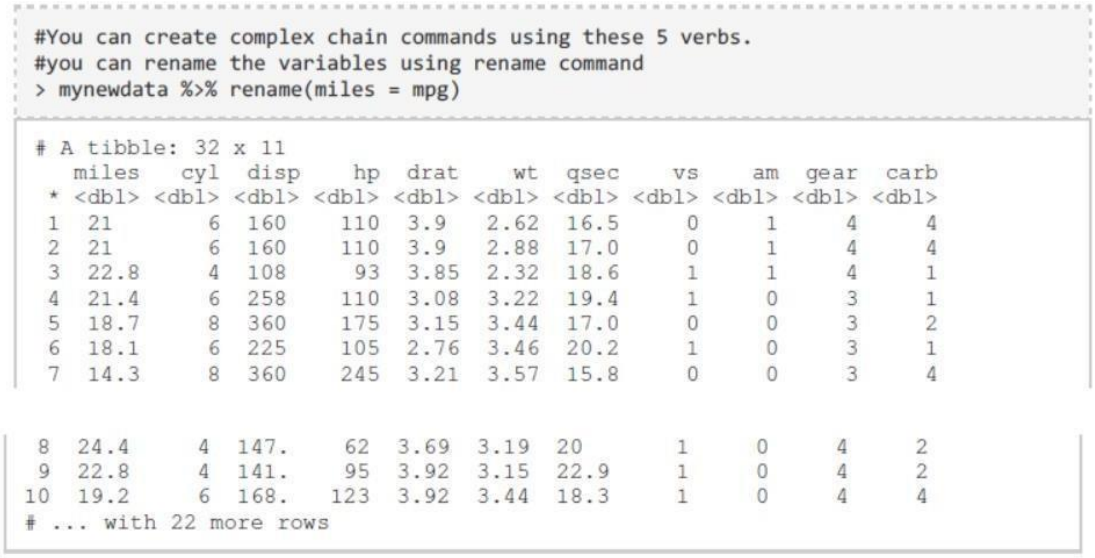


```
#summarise - this is used to find insights from data
> myirisdata%>%
       group_by(Species)%>%
       summarise(Average = mean(Sepal.Length, na.rm = TRUE))
```

```
# A tibble: 3 x 2
  Species    Average
  <fct>        <dbl>
1 setosa        5.01
2 versicolor    5.94
3 virginica     6.59
```

```
#or use summarise each
> myirisdata%>%
      group_by(Species)%>%
      summarise_each(funs(mean, n()), Sepal.Length, Sepal.Width)
```

```
# A tibble: 3 x 5
  Species    Sepal.Length_mean Sepal.Width_mean Sepal.Length_n Sepal.Width_n
  <fct>                  <dbl>            <dbl>          <int>         <int>
1 setosa                  5.01             3.43             50            50
2 versicolor              5.94             2.77             50            50
3 virginica               6.59             2.97             50            50
```



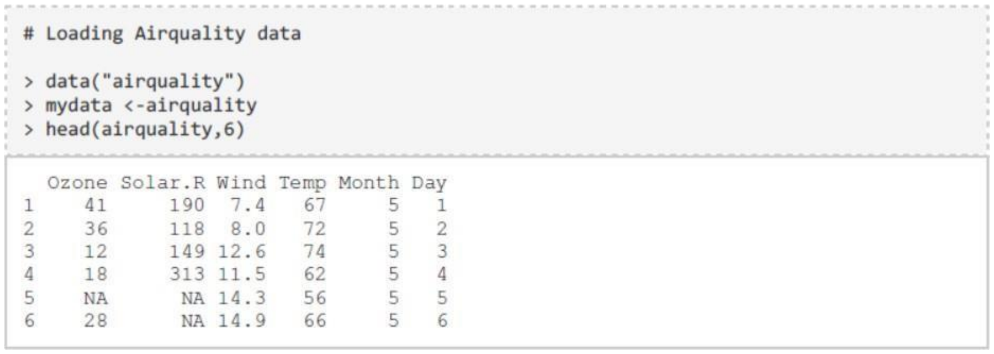
```
#You can create complex chain commands using these 5 verbs.
#you can rename the variables using rename command
> mynewdata %>% rename(miles = mpg)
```

```
# A tibble: 32 x 11
   miles   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
 * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# ... with 22 more rows
```

data.table Package

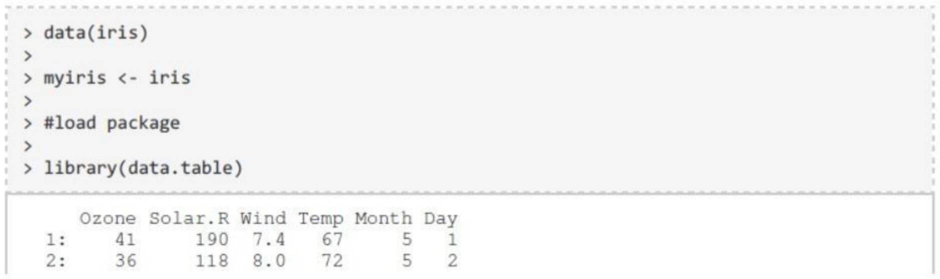
A data table is nothing but a group of related facts arranged in rows and columns and is used to record information. data.table can be used to perform faster manipulation in a data set. Using data.table reduces computing time when compared to data.frame. A data table has 3 parts namely DT[i,j,by].

Here, we are instructing R to subset the rows using 'i', to calculate 'j' which is grouped by 'by'. Most of the times, 'by' relates to categorical variable. In the code below, we have used 2 data sets (airquality and iris).



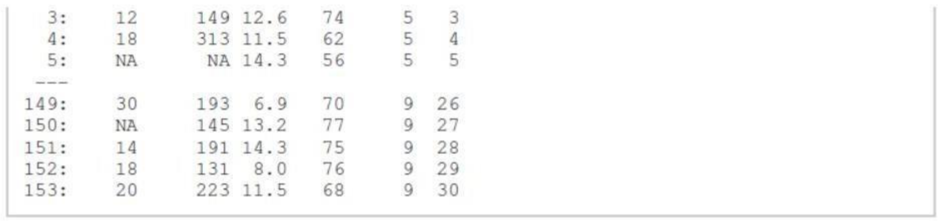
```
# Loading Airquality data

> data("airquality")
> mydata <-airquality
> head(airquality,6)
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

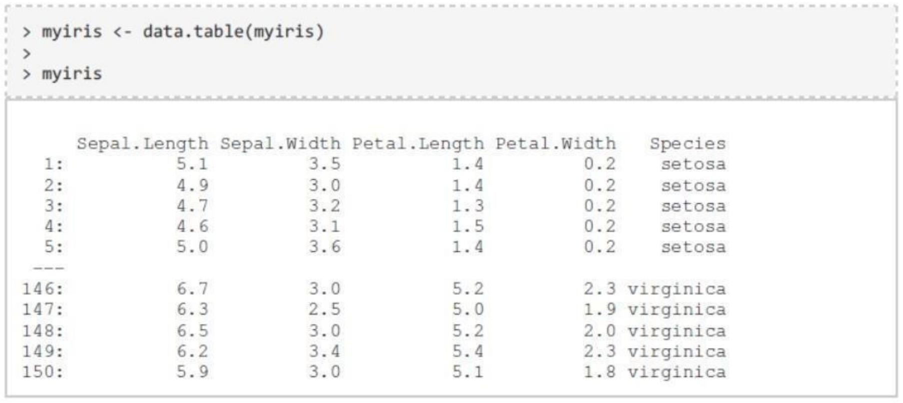


```
> data(iris)
>
> myiris <- iris
>
> #load package
>
> library(data.table)
```

```
     Ozone Solar.R Wind Temp Month Day
  1:    41     190  7.4   67     5   1
  2:    36     118  8.0   72     5   2
```

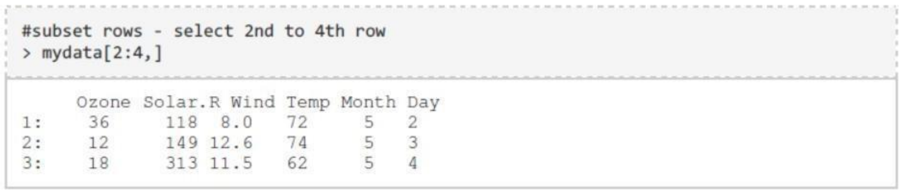


```
  3:    12     149 12.6   74     5   3
  4:    18     313 11.5   62     5   4
  5:    NA      NA 14.3   56     5   5
 ---
149:    30     193  6.9   70     9  26
150:    NA     145 13.2   77     9  27
151:    14     191 14.3   75     9  28
152:    18     131  8.0   76     9  29
153:    20     223 11.5   68     9  30
```



```
> myiris <- data.table(myiris)
>
> myiris
```

```
     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  1:          5.1         3.5          1.4         0.2    setosa
  2:          4.9         3.0          1.4         0.2    setosa
  3:          4.7         3.2          1.3         0.2    setosa
  4:          4.6         3.1          1.5         0.2    setosa
  5:          5.0         3.6          1.4         0.2    setosa
 ---
146:          6.7         3.0          5.2         2.3 virginica
147:          6.3         2.5          5.0         1.9 virginica
148:          6.5         3.0          5.2         2.0 virginica
149:          6.2         3.4          5.4         2.3 virginica
150:          5.9         3.0          5.1         1.8 virginica
```



```
#subset rows - select 2nd to 4th row
> mydata[2:4,]
```

```
   Ozone Solar.R Wind Temp Month Day
1:    36     118  8.0   72     5   2
2:    12     149 12.6   74     5   3
3:    18     313 11.5   62     5   4
```

```
# select columns with particular values
```

```
> myiris[Species == 'setosa']
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
 1:          5.1         3.5          1.4         0.2  setosa
 2:          4.9         3.0          1.4         0.2  setosa
 3:          4.7         3.2          1.3         0.2  setosa
 4:          4.6         3.1          1.5         0.2  setosa
 5:          5.0         3.6          1.4         0.2  setosa
 6:          5.4         3.9          1.7         0.4  setosa
 7:          4.6         3.4          1.4         0.3  setosa
 8:          5.0         3.4          1.5         0.2  setosa
```

```
# select columns with multiple values. This will give you columns with Setosa
#and virginica species

> myiris[Species %in% c('setosa', 'virginica')]
```

```
 47:          5.1         3.8          1.6         0.2    setosa
 48:          4.6         3.2          1.4         0.2    setosa
 49:          5.3         3.7          1.5         0.2    setosa
 50:          5.0         3.3          1.4         0.2    setosa
 51:          6.3         3.3          6.0         2.5 virginica
 52:          5.8         2.7          5.1         1.9 virginica
 53:          7.1         3.0          5.9         2.1 virginica
 54:          6.3         2.9          5.6         1.8 virginica
 55:          6.5         3.0          5.8         2.2 virginica
```

```
# select columns. Returns a vector
> mydata[,Temp]
```

```
  [1] 67 72 74 62 56 66 65 59 61 69 74 69 66 68 58 64 66 57 68 62 59 73 61 61 57 58 57 67
 [29] 81 79 76 78 74 67 84 85 79 82 87 90 87 93 92 82 80 79 77 72 65 73 76 77 76 76 76 75
 [57] 78 73 80 77 83 84 85 81 84 83 83 88 92 92 89 82 73 81 91 80 81 82 84 87 85 74 81 82
 [85] 86 85 82 86 88 86 83 81 81 81 82 86 85 87 89 90 90 92 86 86 82 80 79 77 79 76 78 78
[113] 77 72 75 79 81 86 88 97 94 96 94 91 92 93 93 87 84 80 78 75 73 81 76 77 71 71 78 67
[141] 76 68 82 64 71 81 69 63 70 77 75 76 68
```

```
> mydata[,.(Temp,Month)]
```

```
     Temp Month
```

```
  1:   67     5
  2:   72     5
  3:   74     5
  4:   62     5
  5:   56     5
 ---
149:   70     9
150:   77     9
151:   75     9
152:   76     9
153:   68     9
```

```
# returns sum of selected column
> mydata[,sum(Ozone, na.rm = TRUE)]
```

```
 [1] 4887
```

```
#returns sum and standard deviation
> mydata[,.(sum(Ozone, na.rm = TRUE), sd(Ozone, na.rm = TRUE))]
```

```
     V1       V2
1: 4887 32.98788
```

```
#grouping by a variable
> myiris[,.(sepalsum = sum(Sepal.Length)), by=Species]
```

```
      Species sepalsum
1:     setosa    250.3
2: versicolor    296.8
3:  virginica    329.4
```

```
#grouping by a variable
> myiris[,.(sepalsum = sum(Sepal.Length)), by=Species]
```

```
      Species sepalsum
1:     setosa    250.3
2: versicolor    296.8
3:  virginica    329.4
```

```
#select a column for computation, hence need to set the key on column
> setkey(myiris, Species)

#selects all the rows associated with this data point
```

```
> myiris['setosa']
> myiris[c('setosa', 'virginica')]
```

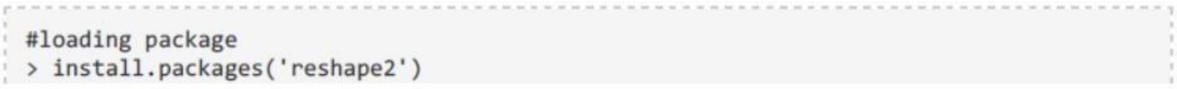
reshape2 Package

reshape2 is an R package, was written by Hadley Wickham which makes it easy to transform data between wide and long formats. reshape2 package is used to reshape data. Using the reshape2 package, we can combine features that have unique values. It has 2 functions namely melt and cast.

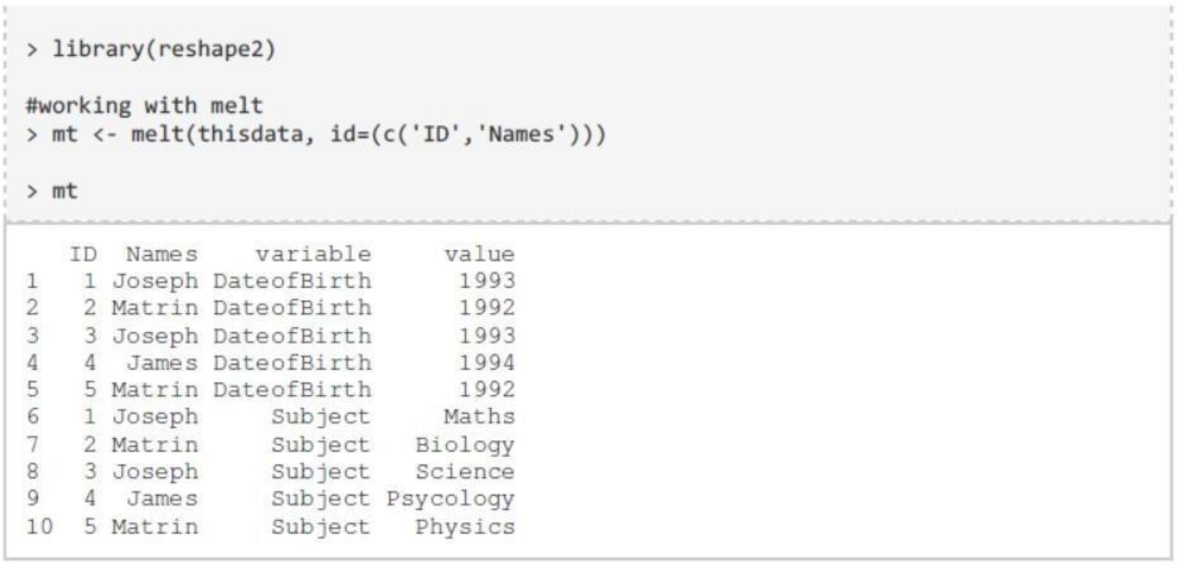
- **melt:** Converts data from wide format to long format. It is a form of restructuring where multiple categorical columns are 'melted' into unique rows. Example is given below:

```
#create a new data
> ID <- c(1,2,3,4,5)

> Names <- c('Joseph','Matrin','Joseph','James','Matrin')

> DateofBirth <- c(1993,1992,1993,1994,1992)

> Subject<- c('Maths','Biology','Science','Psycology','Physics')

> thisdata <- data.frame(ID, Names, DateofBirth, Subject)

> data.table(thisdata)
```

```
   ID  Names DateofBirth   Subject
1:  1 Joseph        1993     Maths
2:  2 Matrin        1992   Biology
3:  3 Joseph        1993   Science
4:  4  James        1994 Psycology
5:  5 Matrin        1992   Physics
```



```
#loading package
> install.packages('reshape2')
```

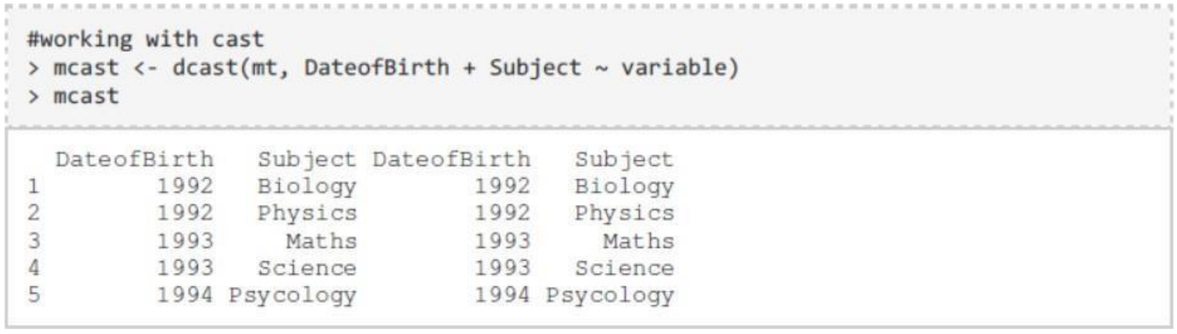


```
> library(reshape2)

#working with melt
> mt <- melt(thisdata, id=(c('ID','Names')))

> mt
```

```
   ID  Names    variable     value
1   1 Joseph DateofBirth      1993
2   2 Matrin DateofBirth      1992
3   3 Joseph DateofBirth      1993
4   4  James DateofBirth      1994
5   5 Matrin DateofBirth      1992
6   1 Joseph     Subject     Maths
7   2 Matrin     Subject   Biology
8   3 Joseph     Subject   Science
9   4  James     Subject Psycology
10  5 Matrin     Subject   Physics
```

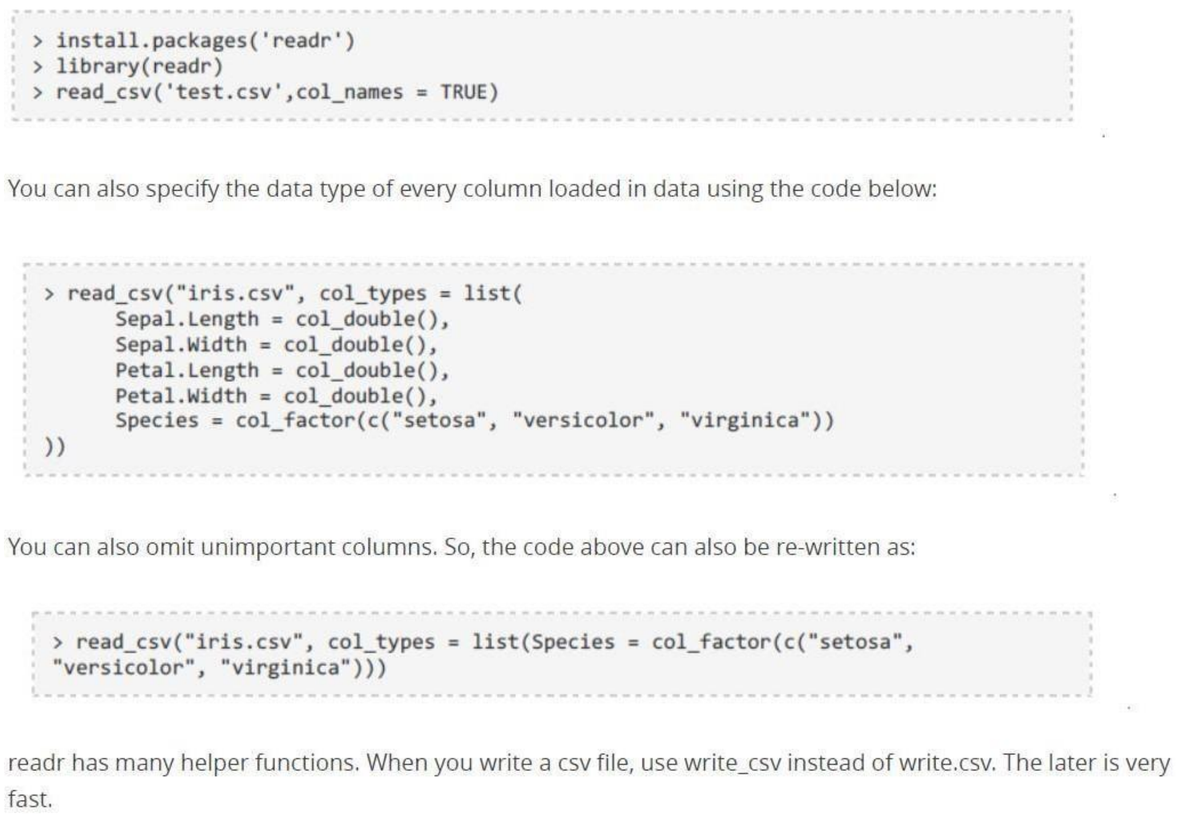
- **cast:** converts data from long format to wide format. It starts with melted data and reshapes into long format. It's the reverse of melt function. It has two functions namely, **dcast** and **acast**.
  - dcast returns a data frame as output.
  - acast returns a vector/matrix/array as the output.

  Let's understand it using the code below.



```
#working with cast
> mcast <- dcast(mt, DateofBirth + Subject ~ variable)
> mcast
```

```
  DateofBirth   Subject DateofBirth   Subject
1        1992   Biology        1992   Biology
2        1992   Physics        1992   Physics
3        1993     Maths        1993     Maths
4        1993   Science        1993   Science
5        1994 Psycology        1994 Psycology
```



```
> install.packages('readr')
> library(readr)
> read_csv('test.csv',col_names = TRUE)
```

You can also specify the data type of every column loaded in data using the code below:

```
> read_csv("iris.csv", col_types = list(
      Sepal.Length = col_double(),
      Sepal.Width = col_double(),
      Petal.Length = col_double(),
      Petal.Width = col_double(),
      Species = col_factor(c("setosa", "versicolor", "virginica"))
))
```

You can also omit unimportant columns. So, the code above can also be re-written as:

```
> read_csv("iris.csv", col_types = list(Species = col_factor(c("setosa",
"versicolor", "virginica")))
```

readr has many helper functions. When you write a csv file, use write_csv instead of write.csv. The later is very fast.

tidyr Package

tidyr is a package which was developed by Hadley Wickham which makes it easy to tidy the data. To make the data look neat and tidy, use the tidyr package. The package has 4 major functions. You can use these functions if you are stuck in the data exploration phase, along with dplyr.

- gather() – 'gathers' multiple columns and converts them into key:value pairs. This function transforms wide form of data to long form. It can be used as an alternative to 'melt' in reshape package.
- spread() – Does reverse of gather. It accepts a key:value pair and converts it into separate columns.
- separate() – Splits a column into multiple columns.
- unite() – Does reverse of separate. It unites multiple columns into single column

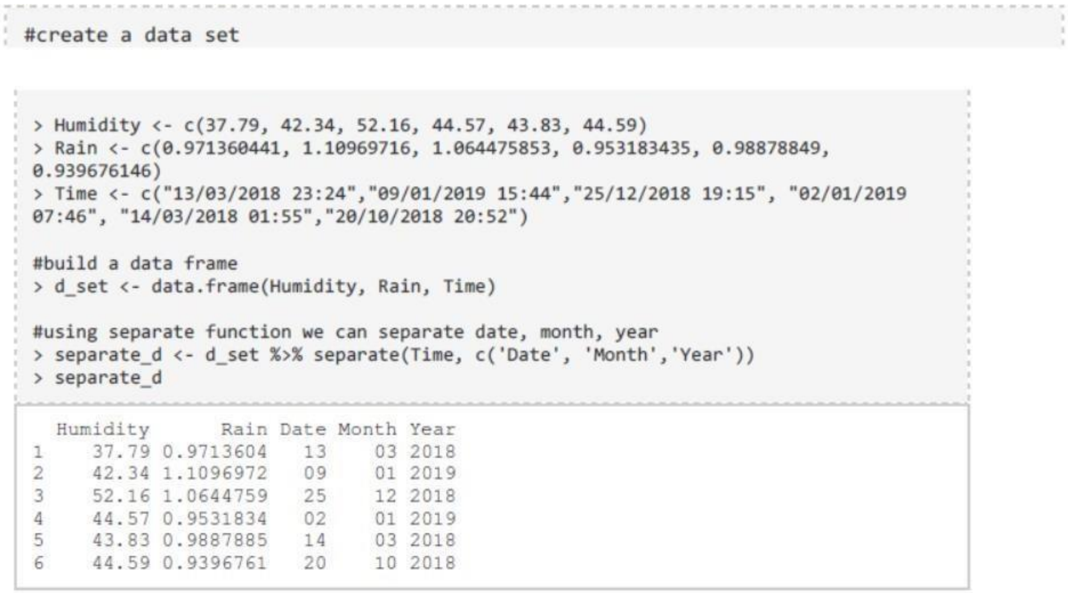
```
# Loading the package
```

```
> library(tidyr)

#create a dummy data set
> names <- c('Akanksh','Bhanu','Susheel','Vinay','Varun','Prashanth','Manohar')
> weight <- c(55,49,76,71,65,44,34)
> age <- c(21,20,25,29,33,32,38)
> Class <- c('Maths','Science','Social','Physics','Biology','Economics','Accounts')

#create data frame
> tdata <- data.frame(names, age, weight, Class)
> tdata
```

```
      names age weight     Class
1   Akanksh  21     55     Maths
2     Bhanu  20     49   Science
3   Susheel  25     76    Social
4     Vinay  29     71   Physics
5     Varun  33     65   Biology
6 Prashanth  32     44 Economics
7   Manohar  38     34  Accounts
```

```
#using gather function
> long_t <- tdata %>% gather(Key, Value, weight:Class)
> long_t
```

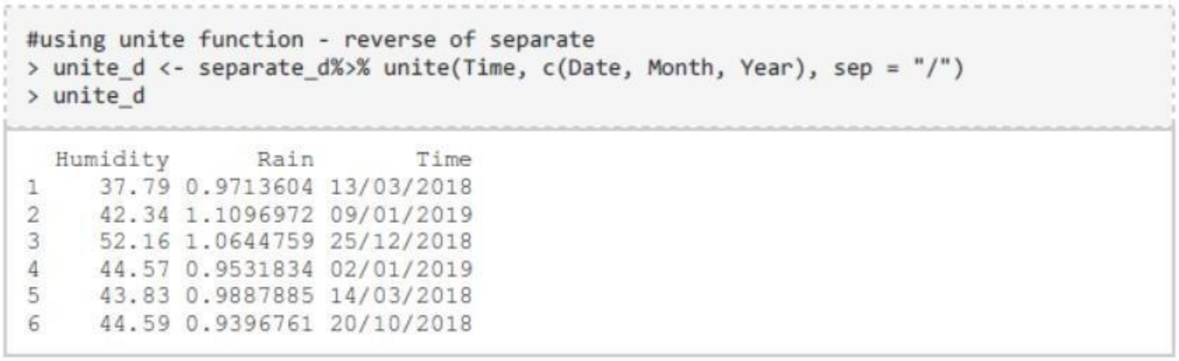
```
       names age    Key     Value
1    Akanksh  21 weight        55
2      Bhanu  20 weight        49
3    Susheel  25 weight        76
4      Vinay  29 weight        71
5      Varun  33 weight        65
6  Prashanth  32 weight        44
7    Manohar  38 weight        34
8    Akanksh  21  Class     Maths
9      Bhanu  20  Class   Science
10   Susheel  25  Class    Social
11     Vinay  29  Class   Physics
12     Varun  33  Class   Biology
13 Prashanth  32  Class Economics
14   Manohar  38  Class  Accounts
```



```
#create a data set
```

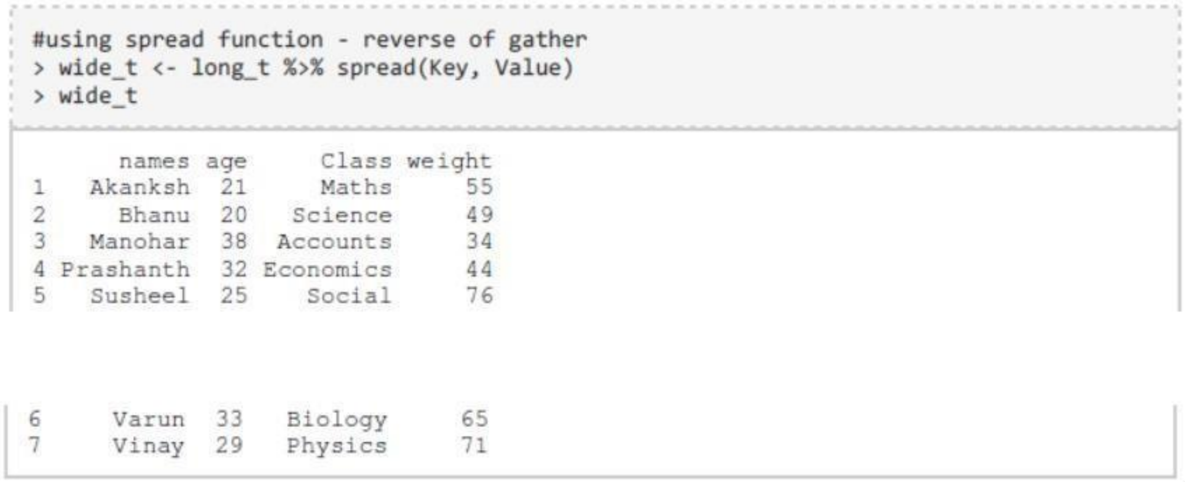
```
> Humidity <- c(37.79, 42.34, 52.16, 44.57, 43.83, 44.59)
> Rain <- c(0.971360441, 1.10969716, 1.064475853, 0.953183435, 0.98878849,
0.939676146)
> Time <- c("13/03/2018 23:24","09/01/2019 15:44","25/12/2018 19:15", "02/01/2019
07:46", "14/03/2018 01:55","20/10/2018 20:52")

#build a data frame
> d_set <- data.frame(Humidity, Rain, Time)

#using separate function we can separate date, month, year
> separate_d <- d_set %>% separate(Time, c('Date', 'Month','Year'))
> separate_d
```

```
  Humidity      Rain Date Month Year
1    37.79 0.9713604   13    03 2018
2    42.34 1.1096972   09    01 2019
3    52.16 1.0644759   25    12 2018
4    44.57 0.9531834   02    01 2019
5    43.83 0.9887885   14    03 2018
6    44.59 0.9396761   20    10 2018
```



```
#using unite function - reverse of separate
> unite_d <- separate_d%>% unite(Time, c(Date, Month, Year), sep = "/")
> unite_d
```

```
  Humidity      Rain       Time
1    37.79 0.9713604 13/03/2018
2    42.34 1.1096972 09/01/2019
3    52.16 1.0644759 25/12/2018
4    44.57 0.9531834 02/01/2019
5    43.83 0.9887885 14/03/2018
6    44.59 0.9396761 20/10/2018
```

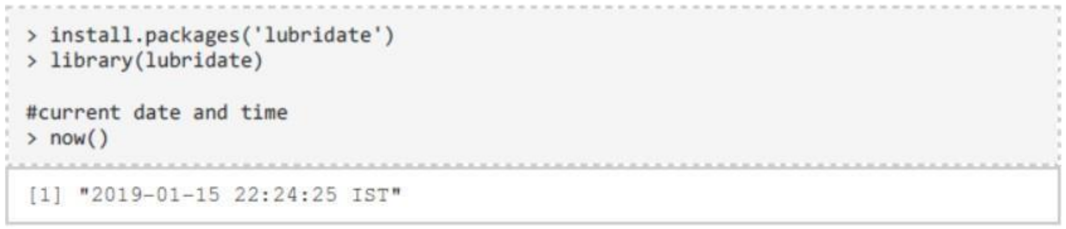


```
#using spread function - reverse of gather
> wide_t <- long_t %>% spread(Key, Value)
> wide_t
```

```
      names age     Class weight
1   Akanksh  21     Maths     55
2     Bhanu  20   Science     49
3   Manohar  38  Accounts     34
4 Prashanth  32 Economics     44
5   Susheel  25    Social     76
6     Varun  33   Biology     65
7     Vinay  29   Physics     71
```

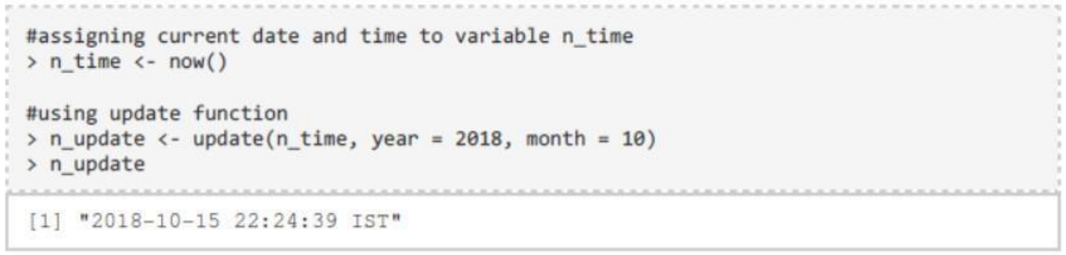
Lubridate Package

Lubridatepackage, makes it easier to work with dates and times. Use the Lubridate package to reduce the issues related to working of data time variable in R. The inbuilt function of this package helps in easy parsing in dates and times. Lubridate is used with data comprising of timely data. Following are three basic tasks that are accomplished using Lubridate – The update, duration function, and data extraction functions.



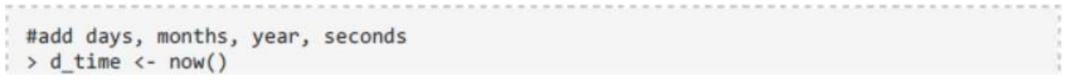
```
> install.packages('lubridate')
> library(lubridate)

#current date and time
> now()
```

```
[1] "2019-01-15 22:24:25 IST"
```



```
#assigning current date and time to variable n_time
> n_time <- now()

#using update function
> n_update <- update(n_time, year = 2018, month = 10)
> n_update
```

```
[1] "2018-10-15 22:24:39 IST"
```



```
#add days, months, year, seconds
> d_time <- now()
> d_time + ddays(10)
```

```
 [1] "2019-01-25 22:25:15 IST"
```

```
> d_time + dweeks(2)
```

```
 [1] "2019-01-29 22:25:15 IST"
```

```
> d_time + dhours(2)
```

```
 [1] "2019-01-16 00:25:15 IST"
```

```
#extract date,time
> n_time$hour <- hour(now())
> n_time$minute <- minute(now())
> n_time$second <- second(now())
> n_time$month <- month(now())
> n_time$year <- year(now())

#check the extracted dates in separate columns
> new_data <- data.frame(n_time$hour, n_time$minute, n_time$second, n_time$month,
n_time$year)

> new_data
```

```
  n_time.hour n_time.minute n_time.second n_time.month n_time.year
1          22            26      33.22655            1        2019
```

Working with Base R Graphics (Scatter Plot, Bar Plot, and Histogram)

ggplot2 Package

ggplot2 offers a wide range of colors and patterns. To understand what is necessary to get started, follow the codes below. You must be proficient with plotting at least 3 graphs – Scatter Plot, Bar Plot, and Histogram.

Scatter Plot :

A Scatter Plot is a graph in which the values of two variables are plotted along two axes, the pattern of the resulting points revealing any correlation present.

With scatter plots we can explain how the variables relate to each other. Which is defined as correlation. Positive, Negative, and None (no correlation) are the three types of correlation.

Limitations of a Scatter Diagram

Below are the few limitations of a scatter diagram:

- With Scatter diagrams we cannot get the exact extent of correlation.
- Quantitative measure of the relationship between the variable cannot be viewed. Onlyshows the quantitative expression.
- The relationship can only show for two variables.

Advantages of a Scatter Diagram

Below are the few advantages of a scatter diagram:

- Relationship between two variables can be viewed.
- For non-linear pattern, this is the best method.
- Maximum and minimum value, can be easily determined.
- Observation and reading is easy to understand
- Plotting the diagram is very simple.

Bar Plot

Abarplot (or barchart) is one of the most common type of graphic. It shows the relationship between a numeric variable and a categoric variable.

Bar Plot are classified into four types of graphs - bar graph or bar chart, line graph, pie chart, and diagram.

Limitations of Bar Plot:

When we try to display changes in speeds such as acceleration, Bar graphs wont help us.

Advantages of Bar plot:

- Bar charts are easy to understand and interpret.
- Relationship between size and value helps for in easy comparison.
- They're simple to create.
- They can help in presenting very large or very small values easily.

Histogram

A histogram represents the frequency distribution of continuous variables. while, a bar graph is a diagrammatic comparison of discrete variables.

Histogram presents numerical data whereas bar graph shows categorical data.

The histogram is drawn in such a way that there is no gap between the bars.

Limitations of Histogram:

A histogram can present data that is misleading as it has many bars.
Only two sets of data are used, but to analyze certain types of statistical data, more than two sets of data are necessary

Advantages of Histogram:

Histogram helps to identify different data, the frequency of the data occurring in the dataset and categories which are difficult to interpret in a tabular form. It helps to visualize the distribution of the data.

lements of ggplot2

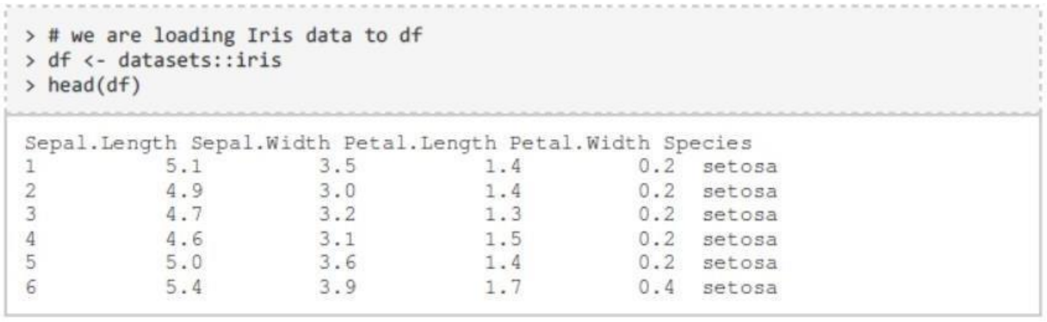
Data: The data-set for which we would want to plot a graph.

Aesthetics: The metrics onto which we plot our data, we can map xaxis, yaxis, fill, col, shape, size.

Geometry: Visual Elements to plot the data.

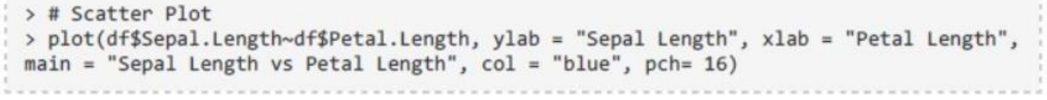
Facet: Groups by which we divide the data.

Working with Base R Graphics

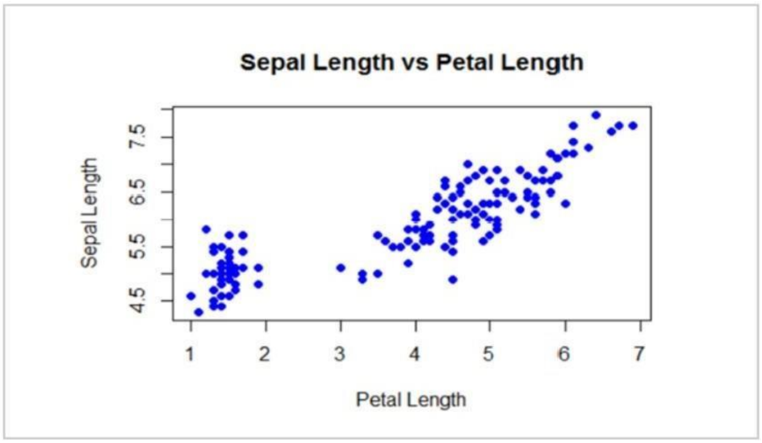


```
> # we are loading Iris data to df
> df <- datasets::iris
> head(df)
```

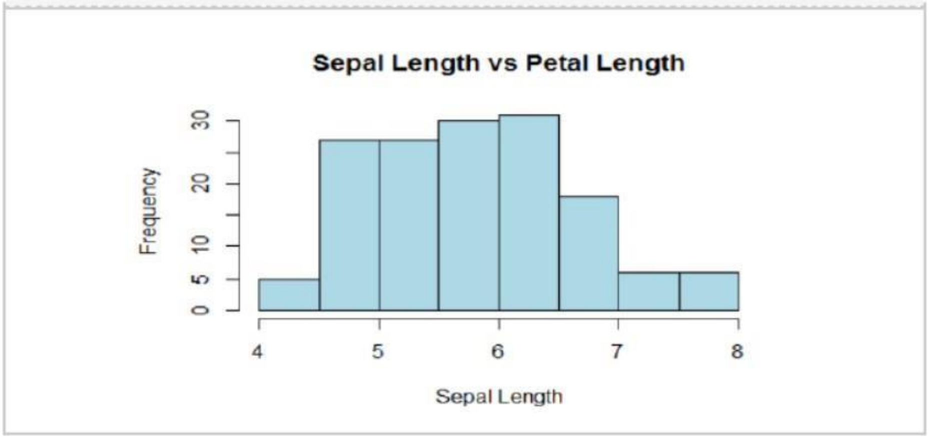
```
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```



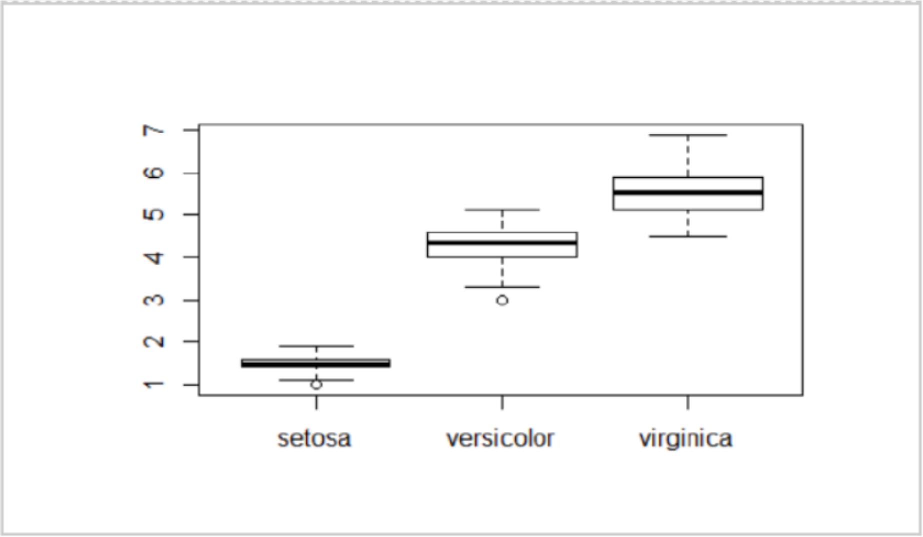
```
> # Scatter Plot
> plot(df$Sepal.Length~df$Petal.Length, ylab = "Sepal Length", xlab = "Petal Length",
main = "Sepal Length vs Petal Length", col = "blue", pch= 16)
```



```
# Histogram
> hist(df$Sepal.Length,xlab = "Sepal Length", main = "Sepal Length vs Petal Length",
col = "light blue", pch= 16)
```



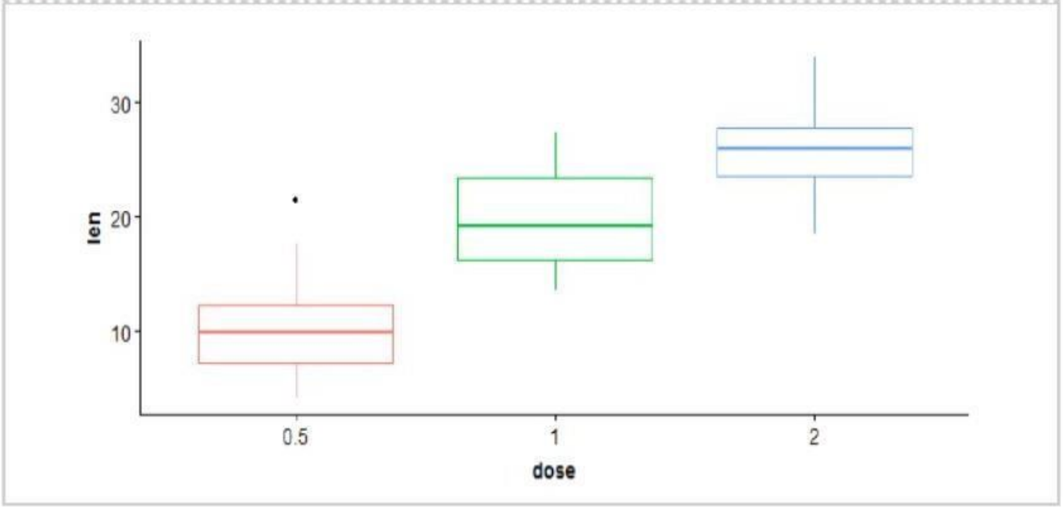
```
# Boxplot
> boxplot(df$Petal.Length~df$Species)
```



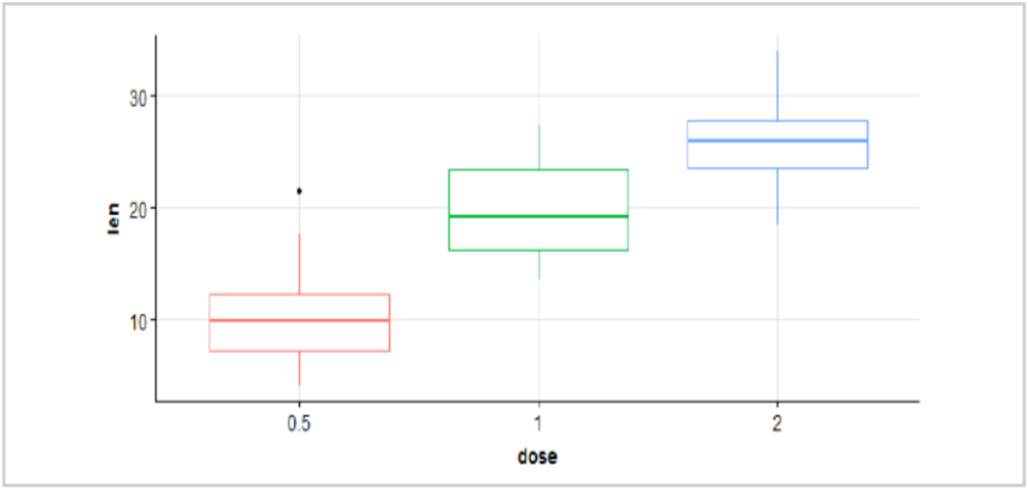
```
> library(ggplot2) # Loading ggplot2 package
> library(grid)

# Loading Tooth Growth data
> df <- datasets::ToothGrowth
> df$dose <- as.factor(df$dose)
> head(df)
```

```
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
```

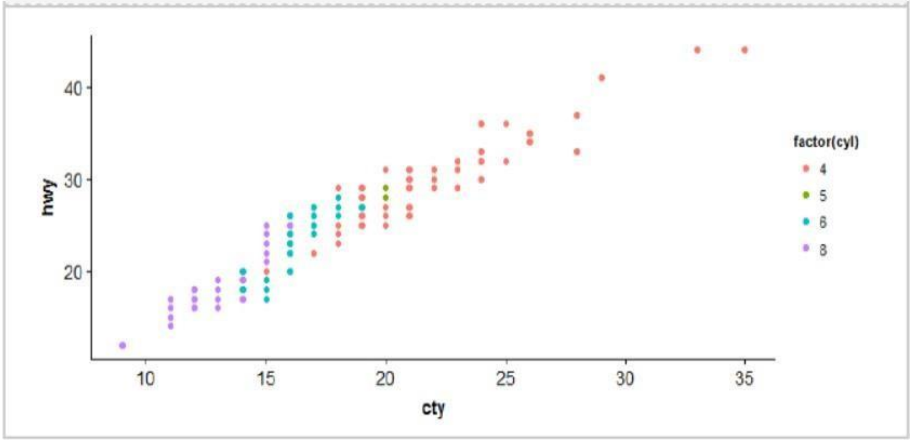
```
# Boxplot
> bp <- ggplot(df, aes(x = dose, y = len, color = dose)) + geom_boxplot() + theme(leg
end.position = 'none')

> bp
```



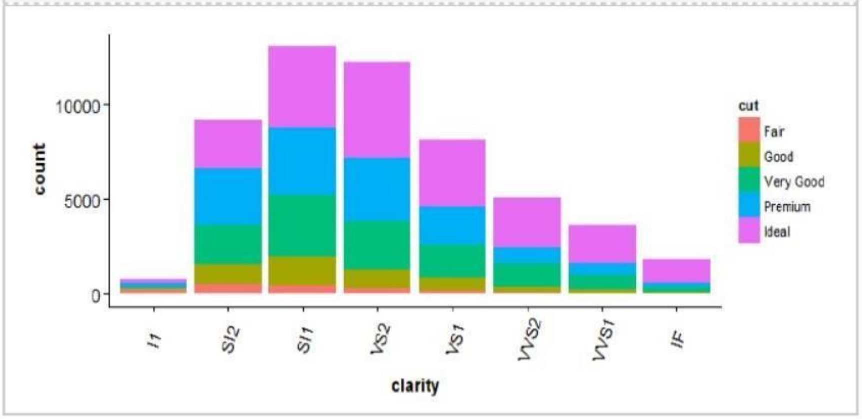
```
#add gridlines
> bp + background_grid(major = "xy", minor = 'none')
```



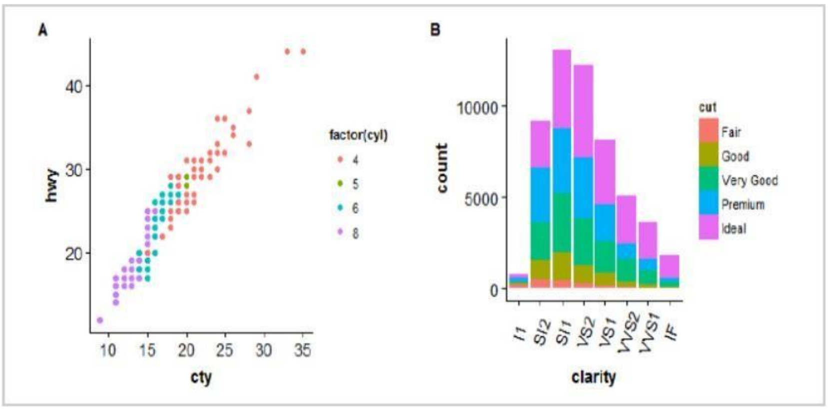
```
# Scatterplot
> sp <- ggplot(mpg, aes(x = cty, y = hwy, color = factor(cyl)))+geom_point(size = 2.5
)
> sp
```

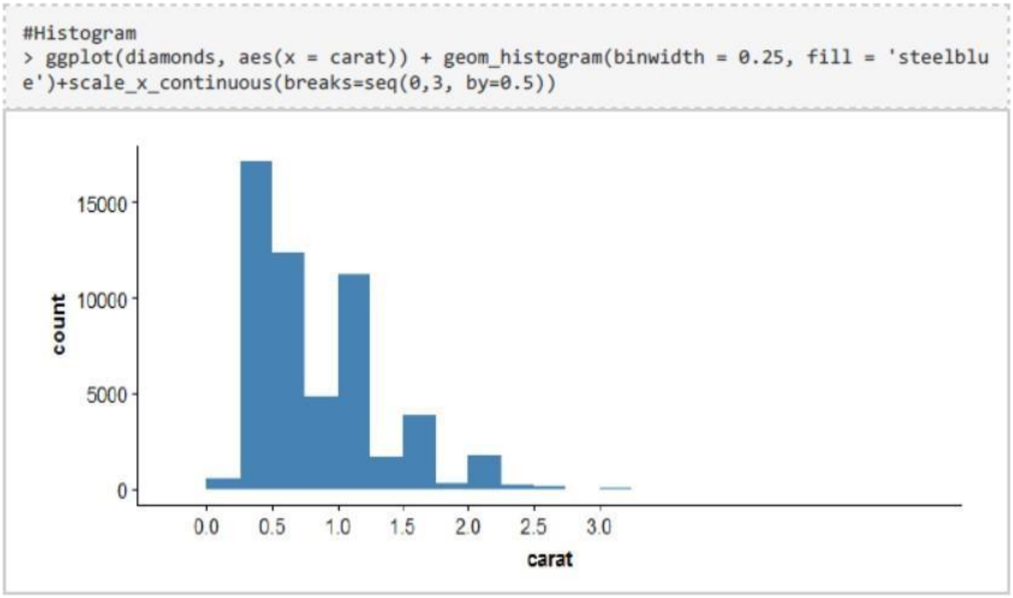


```
#Barplot
> bp <- ggplot(diamonds, aes(clarity, fill = cut)) + geom_bar() +theme(axis.text.x =
element_text(angle = 70, vjust = 0.5))

> bp
```



```
# Compare two plots
> plot_grid(sp, bp, labels = c("A","B"), ncol = 2, nrow = 1)
```



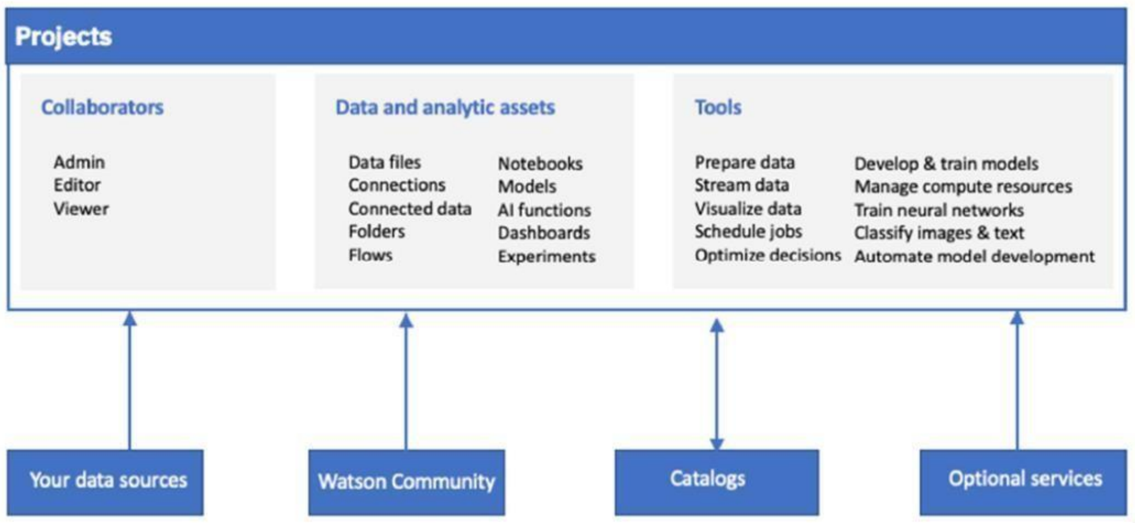


```
#Histogram
> ggplot(diamonds, aes(x = carat)) + geom_histogram(binwidth = 0.25, fill = 'steelblu
e')+scale_x_continuous(breaks=seq(0,3, by=0.5))
```

WATSON STUDIO

Watson Studio provides you with the environment and tools to solve your business problems by collaboratively working with data. You can choose the tools you need to analyze and visualize data, to cleanse and shape data, to ingest streaming data, or to create and train machine learning models.

This illustration shows how the architecture of Watson Studio is centered around the project. A project is where you organize your resources and work with data.



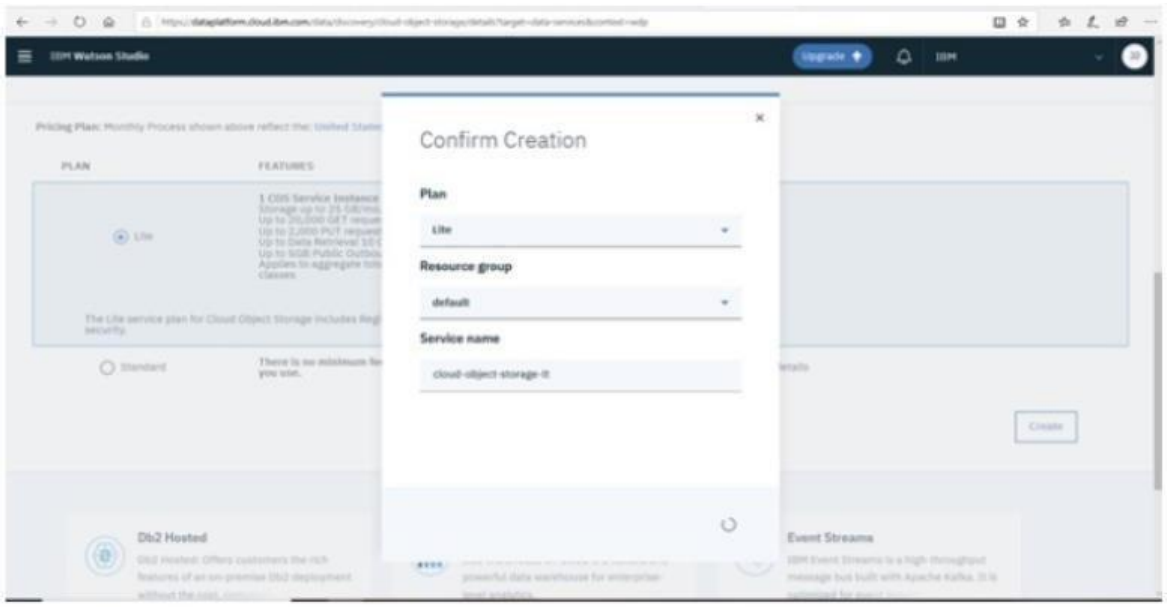
Visualizing information in graphical ways can give you insights into your data. By enabling you to look at and explore data from different perspectives, visualizations can help you identify patterns, connections, and relationships within that data as well as understand large amounts of information very quickly.

Create a project -

To create a project :

- Click New project on the Watson Studio home page or your My Projects page.
- Choose whether to create an empty project or to create a project based on an exported project file or a sample project.
- If you chose to create a project from a file or a sample, upload a project file or select a sample project. See Importing a project.
- On the New project screen, add a name and optional description for the project.

- Select the Restrict who can be a collaborator check box to restrict collaborators to members of your organization or integrate with a catalog. The check box is selected by default if you are a member of a catalog. You can't change this setting after you create the project.
- If prompted, choose or add any required services.
- Choose an existing object storage service instance or create a new one.
- Click Create. You can start adding resources if your project is empty or begin working with the resources you imported.


Confirm Creation

Plan

Lite

Resource group

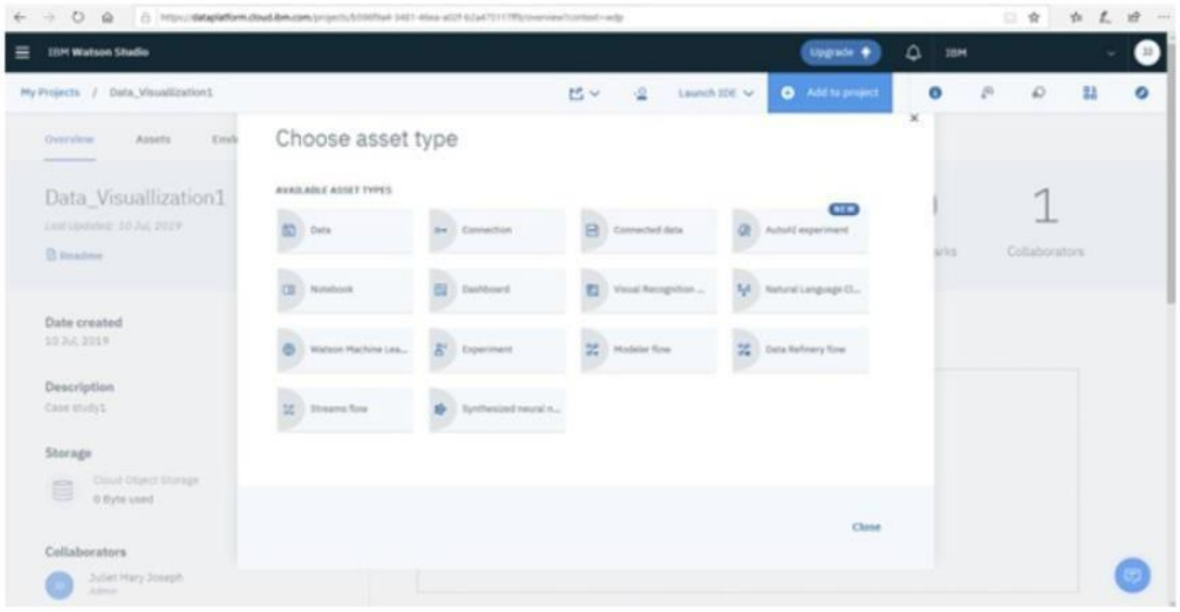
default

Service name

cloud-object-storage-lt

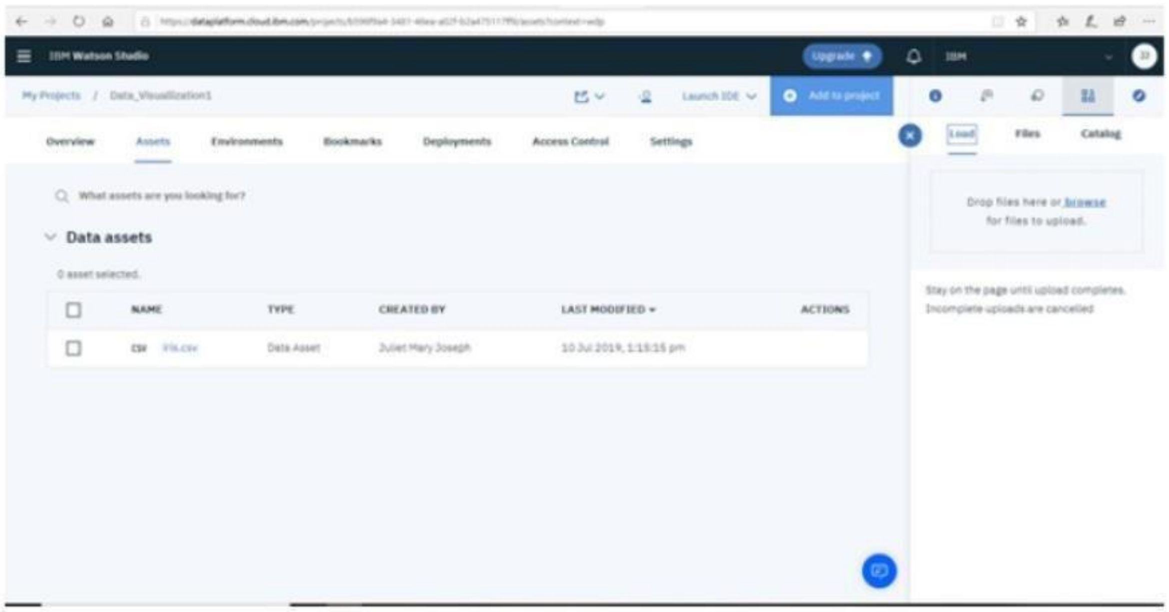
To add data files to a project:

- From your project’s Assets page, click Add to project > Data or click the Find and add data icon ().You can also click the Find and add data icon from within a notebook or canvas.
- In the Load pane that opens, browse for the files or drag them onto the pane. You must stay on the page until the load is complete. You can cancel an ongoing load process if you want to stop loading a file.



Case Study:

Let us take the Iris Data set to see how we can visualize the data in Watson studio.



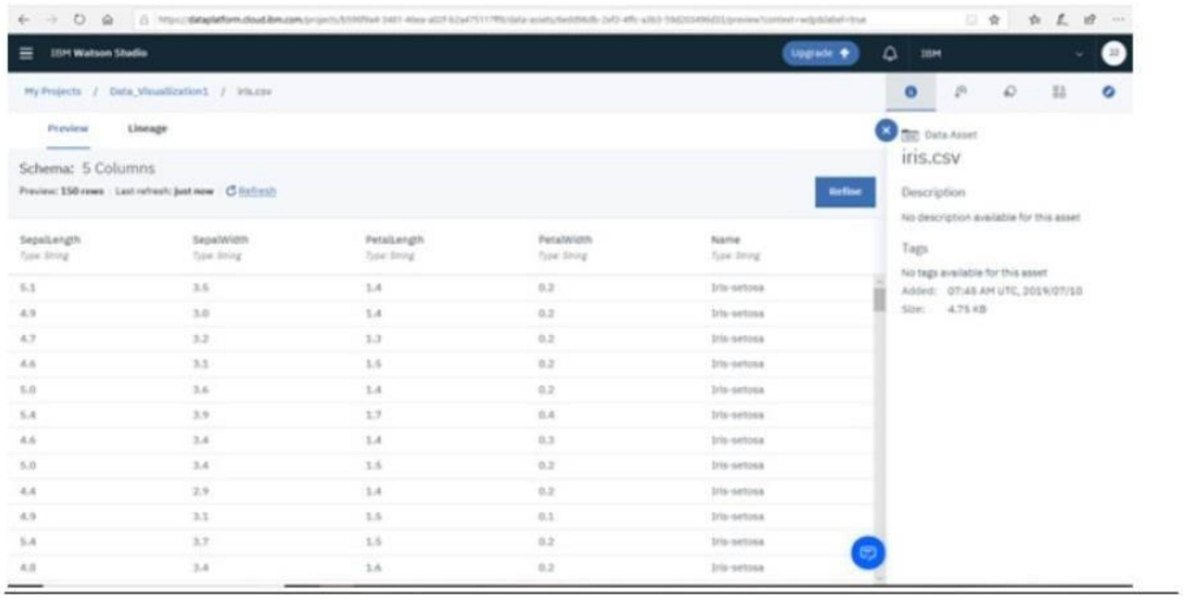
Data assets

0 asset selected.

	NAME	TYPE	CREATED BY	LAST MODIFIED	ACTIONS
	CSV iris.csv	Data Asset	Juliet Mary Joseph	10 Jul 2019, 1:15:15 pm	

Drop files here or browse for files to upload.

Stay on the page until upload completes. Incomplete uploads are cancelled.



Schema: 5 Columns

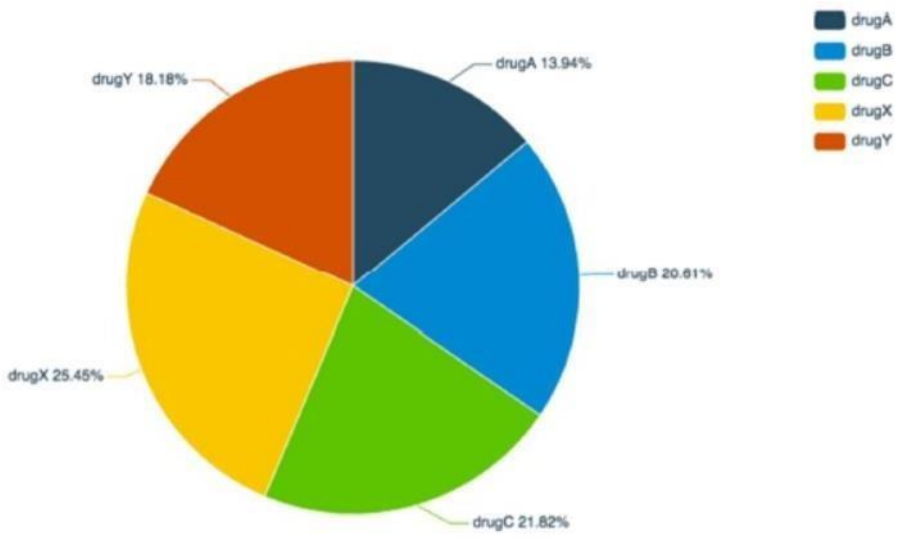
Preview: 150 rows | Last refresh: just now | Refresh

SepalLength Type: String	SepalWidth Type: String	PetalLength Type: String	PetalWidth Type: String	Name Type: String
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5.0	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa

Data Asset
iris.csv
Description
No description available for this asset
Tags
No tags available for this asset
Added: 07:45 AM UTC, 2019/07/10
Size: 4.75 KB

Adding Data to Data Refinery

Visualizing information in graphical ways can give you insights into your data. By enabling you to look at and explore data from different perspectives, visualizations can help you identify patterns, connections, and relationships within that data as well as understand large amounts of information very quickly. You can also visualize your data with these same charts in an SPSS Modeler flow. Right-click a node and select **Profile.**

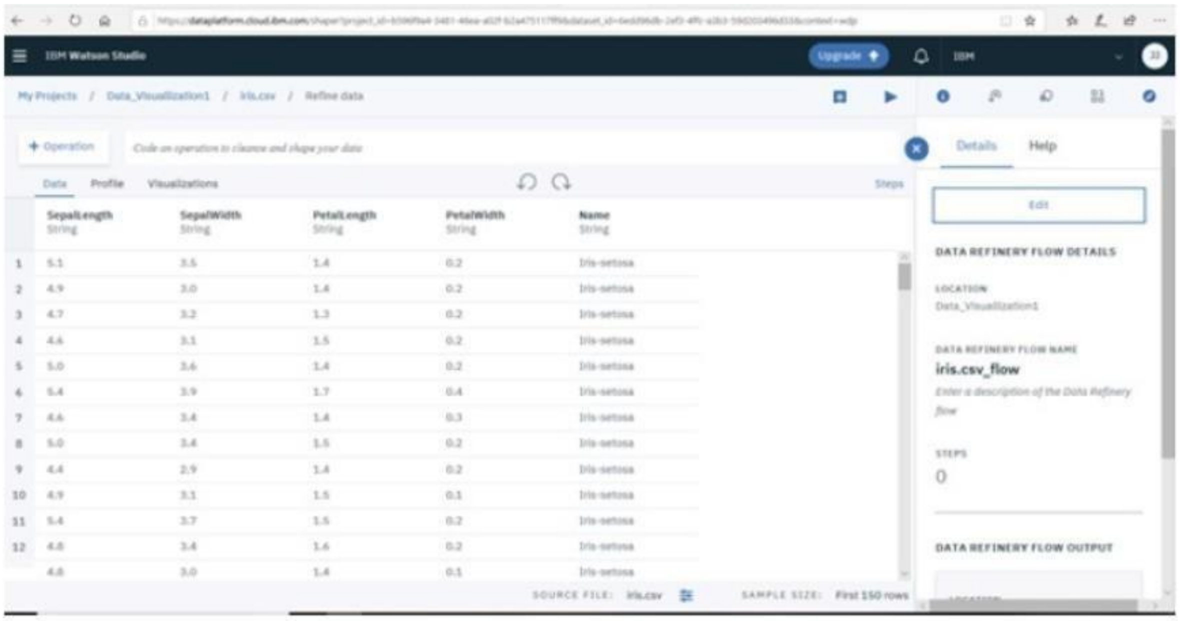


To visualize your data:

- From Data Refinery, click the **Visualizations** tab.
- Start with a chart or select columns.

1. Click any of the available charts. Then add columns in the **DETAILS** panel that opens on the left side of the page.
2. Select the columns that you want to work with. Suggested charts will be indicated with a dot next to the chart name. Click a chart to visualize your data.

Click on refine


IBM Watson Studio
Upgrade
IBM
My Projects / Data_Visualization1 / iris.csv / Refine data
+ Operation
Code an operation to cleanse and shape your data
Data
Profile
Visualizations
Steps
Details
Help
Edit

	SepalLength String	SepalWidth String	PetalLength String	PetalWidth String	Name String
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
	4.8	3.0	1.4	0.1	Iris-setosa

SOURCE FILE: iris.csv SAMPLE SIZE: First 150 rows

DATA REFINERY FLOW DETAILS

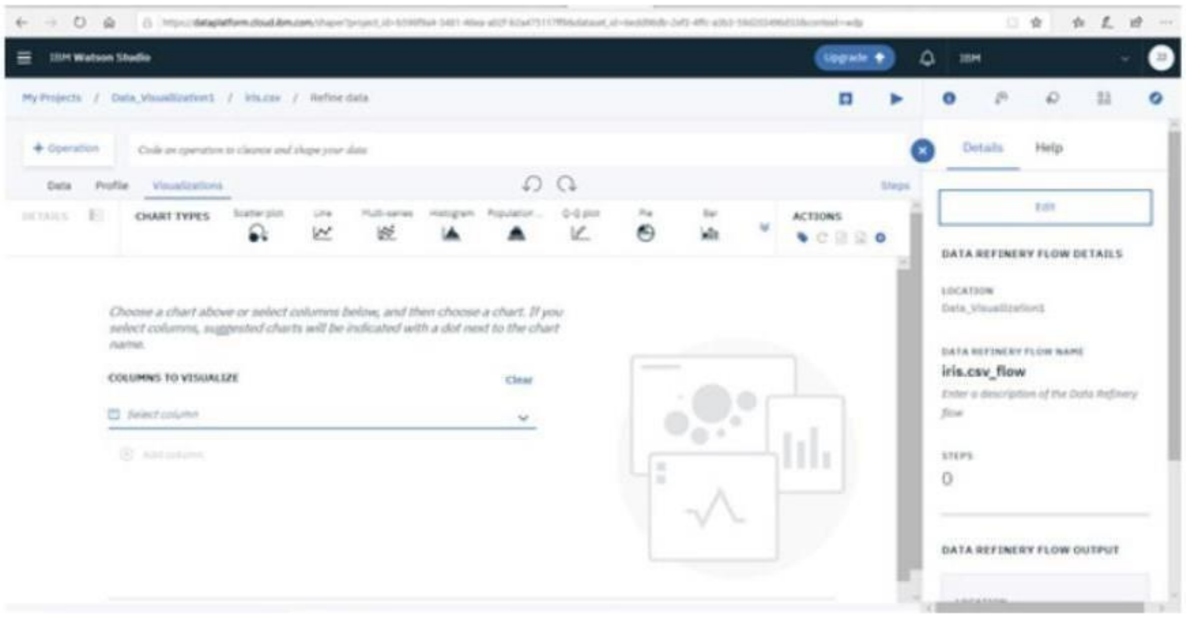
LOCATION
Data_Visualization1

DATA REFINERY FLOW NAME
iris.csv_flow
Enter a description of the Data Refinery flow

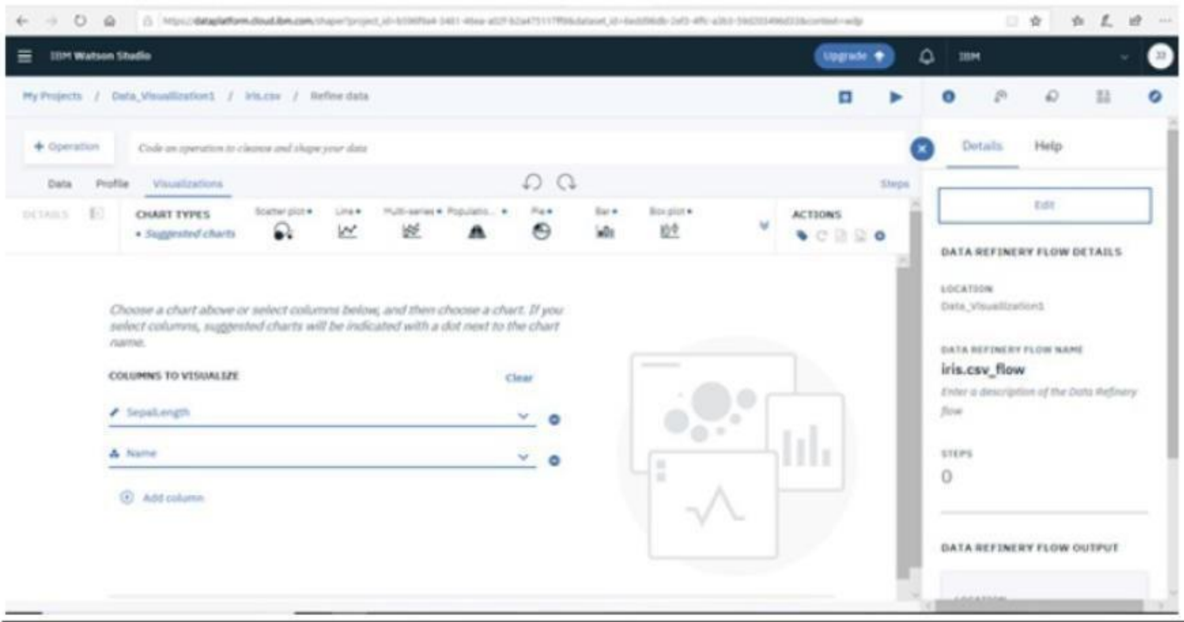
STEPS
0

DATA REFINERY FLOW OUTPUT

Click on Visualization tab:

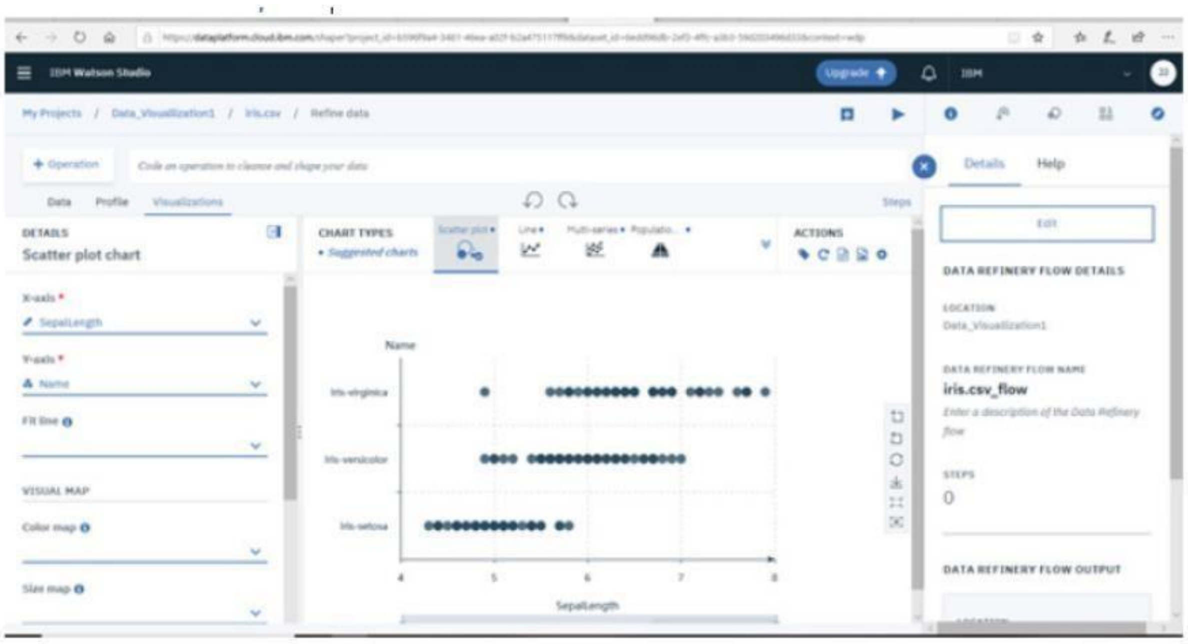


Add the columns by selecting.

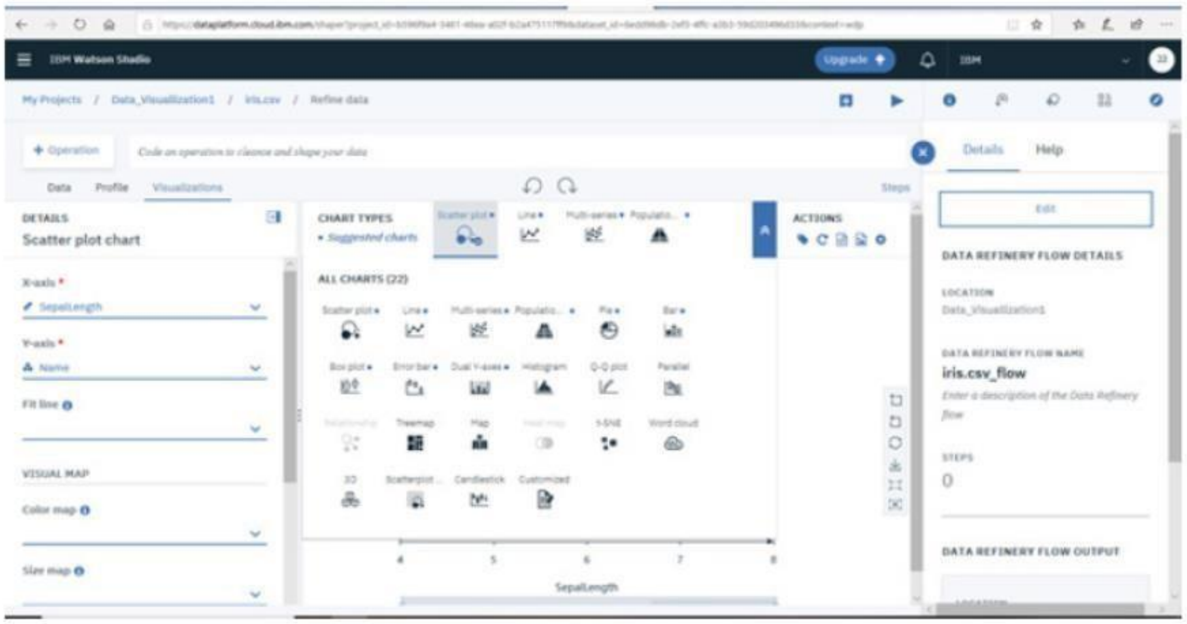


Visualization of Data on Watson Studio

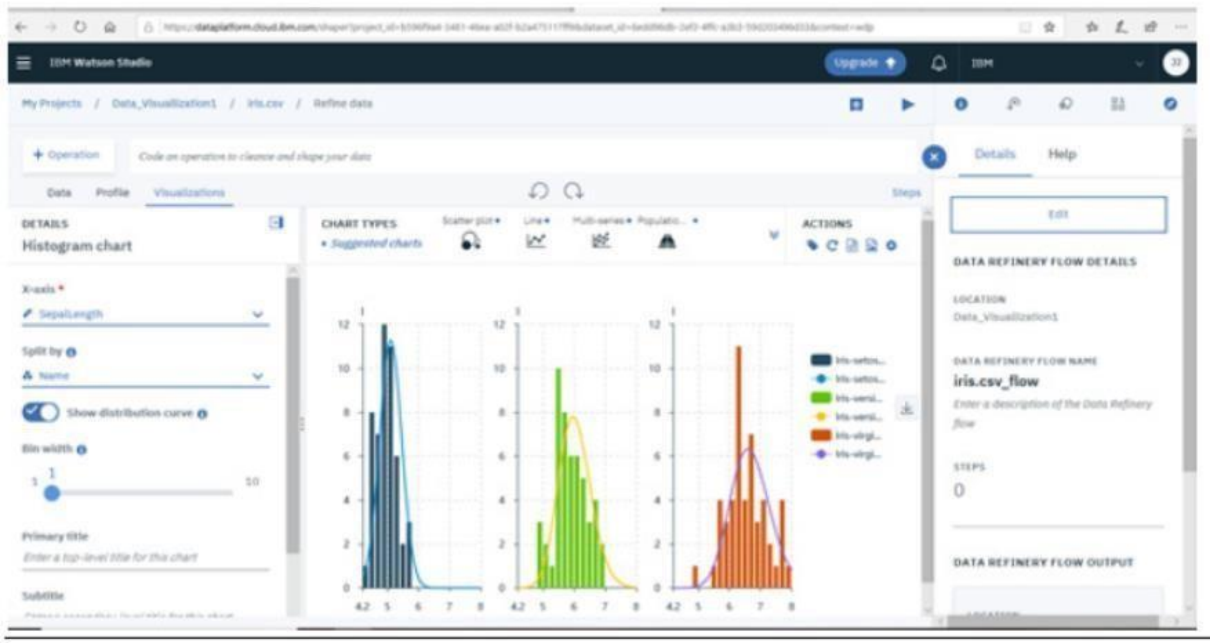
Select Scatter plot:



Various types of option to visualize the data:



Select Histogram and select the x axis and y axis :



IBM WATSON STUDIO

IBM Watson Studio is an integrated environment designed to make it easy to develop, train, manage models, and deploy AI-powered applications and is a SaaS solution delivered on the IBM Cloud. **It is evolving Data Science Experience on IBM Cloud with lot of new features to build AI applications.**

With Watson Studio provides the following facilities:

- Extends the capabilities we provide around deep learning, including TensorFlow scoring
- Allows one to access pre-trained models from the Watson Services, such as Watson Visual Recognition
- Enables you to bring in non-structured data
- Further automating and providing insight into model management
- Continues to provide you with a choice of best-in-breed data science/ML tools
- Strengthens the drag-and-drop interface to build analytics models using SPSS Modeler
- Enables one to visualize the insights with dynamic dashboards

Projects

A project is how you organize your resources to work with data. project resources can include:

- Data asset files, connections, connected data and flows
- Analytic assets that describe how to work with data
- Runtime environments
- Collaborators

- A readme file to document the project
- Access to the RStudio IDE
- Other project-wide settings

Overview page

The Overview page provides the following information about the project:

- A summary of storage usage.
- The number of assets, and collaborators.
- A list of recent notifications for the project.
- A readme file to document the project, at the bottom of the page. The readme file uses standard Markdown formatting.

Assets

If one have the Admin or Editor permissions on a project, he can add assets by choosing the asset type from the Add to project menu. The types of assets can be added are:

- Data assets from local files, catalogs, or the Gallery
- Connections to cloud, on-premises, and streaming data sources
- Connected data from an existing connection asset
- Folder assets to view the files within a folder in a file system
- Jupyter Notebooks to analyze data
- Modeler flows to automate the flow of data through a model
- Streams flows to ingest streaming data
- Models to analyze data
- Decision Optimization models to solve scenarios
- Experiments to train deep learning models
- Visual Recognition models to categorize images
- Dashboards to visualize data without code
- Data Refinery flows to refine data
- Natural Language Classification models

Environments

On the **Environments** page, one can define the hardware size and software configuration for the runtime environments.

A. <u>Creating a Project</u>

1. Click New project on the Watson Studio home page or your My Projects page.
2. Choose whether to create an empty project or to create a project based on an exported project file or a sample project.

3. If you chose to create a project from a file or a sample, upload a project file or select a sample project. See Importing a project.
4. On the New project screen, add a name and optional description for the project.
5. Select the Restrict who can be a collaborator checkbox to restrict collaborators to members of the organization or integrate with a catalog. The checkbox is selected by default if you are a member of a catalog. You can't change this setting after you create the project.
6. If prompted, choose or add any required services.
7. Choose an existing object storage service instance or create a new one.
8. Click Create. You can start adding resources if your project is empty, or begin working with the resources you imported.

After creating a project, the next step is to add data to the project and prepare the data for analysis. Data assets can be added to the project from the local system, from a catalog, from the Gallery, or from connections to data sources. The following types of data assets can be added to a project:

- Data assets from files from local system, including structured data, unstructured data, and images. The files are stored in the project's IBM Cloud Object Storage bucket.
- Connection assets that contain information for connecting to data sources.
- Connected data assets that specify a table, view, or file that is accessed through a connection to a data source.
- Folder assets that specify a path in IBM Cloud Object Storage.

B. <u>Preparing the Data:</u>

<u>Adding Data to Data Refinery</u>

After creating a project, connections or after adding data assets to the project, data can be added to Data Refinery and start prepping that data for analysis. Data can be added to Data Refinery in one of several ways:

- Select Refine from the menu of a data asset in the project
- Preview a data asset in the project and then choose to refine it
- Navigate to Data Refinery first and then add data to it

<u>Navigate to Data Refinery</u>

To access Data Refinery from within a project, Click Add to project > Data Refinery flow. If a Data Refinery flow is already available, go to the project's Assets tab and click New Data Refinery flow in the Data Refinery flows section.

Add data:

To add data after you navigate to Data Refinery, the following steps need to be followed:

1. Select the data you want to work with from Data assets or from Connections. Data Refinery supports Avro, CSV, JSON, parquet and text files.

   From Data assets:

   - Select a data file (the selection includes data files that have already been shaped with Data Refinery)
   - Select a connected data asset

   From Connections:

   - Select a connection and file
   - Select a connection, folder, and file
   - Select a connection, schema, and table or view

2. Click Add to load the data into Data Refinery.

Specifying the format for data in Data Refinery

When the data is read into Data Refinery, it should look like a well-formatted spreadsheet. If it doesn't display in tabular form or it doesn't look as expected, go to the Data tab. Scroll down to the SOURCE FILE information at the bottom of the page. Click the Specify data format icon. Modifying the default data format specification can help Data Refinery correctly read your data.

To specify the format of your data, the following steps need to be followed:

1. Indicate whether the first row of data contains column headers. If data doesn't contain column headers, Data Refinery will add them so that they can be used in cleansing and shaping operations.
2. Select the appropriate character encoding for data source, for example, CSV files are often UTF-8 encoded.
3. Identify the character that separates each field or column value from the next value, for example, CSV files are often comma-delimited.
4. Identify the character that encloses string values, for example, CSV files typically enclose strings in double quotation marks.
5. Identify the character that's used to escape other characters, for example, backslashes (\ ) are commonly used as escape characters. Escaping is a string technique that identifies characters (such as double quotation marks) as being part of a string value.
6. Click Apply to apply the format specification to the data and return to Data Refinery.

<u>Validating data in Data Refinery:</u>

Validation must be done at multiple points in the refinement process. To validate data, the following steps need to be followed:

1. From Data Refinery, click the **Profile** tab.
2. Review the metrics for each column.
3. Take appropriate actions, as described in the following sections, depending on what you learn.

The metrics are frequency and statistics. Frequency is the number of times that a value, or a value in a specified range, occurs. Each frequency distribution (bar) shows the count of unique values in a column. Review the frequency distribution to find anomalies in data. Simply remove the values for cleansing the data. Statistics are a collection of quantitative data. The statistics for each column show the minimum, maximum, mean, and number of unique values in that column. Depending on a column’s data type, the statistics for each column will vary slightly. For example, statistics for a column of data type integer have minimum, maximum, and mean values while statistics for a column of data type string have minimum length, maximum length, and mean length values.

Visualizing data in Data Refinery:

Visualizing information in graphical ways can give insights into the data. By enabling one to look at and explore data from different perspectives, visualizations can identify patterns, connections, and relationships within that data as well as understand large amounts of information very quickly.

To visualize your data:

1. From Data Refinery, click the Visualizations tab.
2. Start with a chart or select columns:
   - Click any of the available charts. Then add columns in the DETAILS panel that opens on the left side of the page.
   - Select the columns that you want to work with. Suggested charts will be indicated with a dot next to the chart name. Click a chart to visualize your data.

Examples for dplyr, reshape2, data.table packages:

```
df<-data.frame(id=101:110,
name=c("A","B","C","D","E","F","G","H","I","J"),sal=c(45,89,26,45,33,75,68,12,19,58),strings
AsFactors=FALSE)

> str(df)
'data.frame':      10 obs. of 3 variables:
```

```
$ id : int  101 102 103 104 105 106 107 108 109 110
$ name: Factor w/ 10 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10
$ sal : num 45 89 26 45 33 75 68 12 19 58
```

Dplyr package:

```
> install.packages("dplyr")

> library("dplyr")

> filter(df,df$id>=106)
```

```
  id   name  sal
1 106   F    75
2 107   G    68
3 108   H    12
4 109   I    19
5 110   J    58
```

```
> arrange(df, df$sal)
```

```
     id name  sal
1   108    H  12
2   109    I  19
3   103    C  26
4   105    E  33
5   101    A  45
6   104    D  45
7   110    J  58
8   107    G  68
9   106    F  75
10  102    B  89
```

```
> arrange(df, desc(df$sal))
     id name  sal
1   102    B  89
2   106    F  75
3   107    G  68
4   110    J  58
5   101    A  45
6   104    D  45
7   105    E  33
8   103    C  26
9   109    I  19
10  108    H  12
```

```
> select(df,name)
    name
1      A
2      B
3      C
4      D
5      E
6      F
7      G
```

```
8     H
9I
10    J
```

> select(df,-(sal))

```
    id name
1  101    A
2  102    B
3  103    C
4  104    D
5  105    E
6  106    F
7  107    G
8  108    H
9  109    I
10 110    J
```

> rename(df, salary=sal)

```
    id name salary
1  101    A     45
2  102    B     89
3  103    C     26
4  104    D     45
5  105    E     33
6  106    F     75
7  107    G     68
8  108    H     12
9  109    I     19
10 110    J     58
```

> mutate(df, inc=sal+15)

```
    id name sal inc
1  101    A  45  60
2  102    B  89 104
3  103    C  26  41
4  104    D  45  60
5  105    E  33  48
6  106    F  75  90
7  107    G  68  83
8  108    H  12  27
9  109    I  19  34
10 110    J  58  73
```

> transmute(df, inc=sal+15)

```
   inc
1   60
2  104
3   41
4   60
5   48
6   90
7   83
8   27
9   34
10  73
```

> summarise(df)

```
data frame with 0 columns and 1 row
```

> summarise(df,meansal=mean(df$sal))

```
 meansal
1      47
```

> summarise(df,meansal=mean(df$sal),nrows=n())

```
 meansal nrows
1      47    10
```

> df1<-data.frame(id=101:110, name=c("A","B","C","D","E","F","G","H","I","J"),sal=c(45,89,26,45,33,75,68,12,19,58), eadd=c("kphb","ecil","bhel","jntu","nzmpt","kphb","ecil","bhel","jntu","nzmpt"),edept=c("sales" ,"marketing","hr","sales","marketing","hr","sales","marketing","hr","hr"))

> df1%>%group_by(df1$eadd)

```
                          # A tibble: 10 x 6
                        # Groups:   df1$eadd [5]
                    id name     sal eadd  edept      `df1$eadd`
                  <int> <fct> <dbl> <fct> <fct>       <fct>
  1   101    A    45   kphb     sales   kphb
  2   102    B    89   ecil   marketing ecil
  3   103    C    26   bhel      hr     bhel
  4   104    D    45   jntu     sales   jntu
  5   105    E    33   nzmpt  marketing nzmpt
  6   106    F    75   kphb      hr     kphb
  7   107    G    68   ecil     sales   ecil
  8   108    H    12   bhel   marketing bhel
  9   109    I    19   jntu      hr     jntu
 10   110    J    58   nzmpt     hr     nzmpt
```

> sample_n(df1,4)

```
   id name sal  eadd     edept
1 103    C  26  bhel        hr
2 102    B  89  ecil marketing
3 110    J  58 nzmpt        hr
4 109    I  19  jntu        hr
```

> sample_n(df1,4)

```
   id name sal eadd     edept
1 109    I  19 jntu        hr
2 103    C  26 bhel        hr
3 107    G  68 ecil     sales
4 102    B  89 ecil marketing
```

> sample_n(df1,4)

```
   id name sal  eadd     edept
1 110    J  58 nzmpt        hr
2 101    A  45  kphb     sales
3 105    E  33 nzmpt marketing
4 109    I  19  jntu        hr
```

> sample_frac(df1,0.5)

```
   id name sal eadd      edept
1 106    F  75 kphb         hr
2 108    H  12 bhel marketing
3 101    A  45 kphb      sales
4 104    D  45 jntu      sales
5 107    G  68 ecil      sales
```

> sample_frac(df1,0.6)

```
   id name sal  eadd     edept
1 110    J  58 nzmpt        hr
2 101    A  45  kphb     sales
3 107    G  68  ecil     sales
4 106    F  75  kphb        hr
5 104    D  45  jntu     sales
6 105    E  33 nzmpt marketing
```

> sample_frac(df1,0.6)

```
   id name sal  eadd     edept
1 101    A  45  kphb     sales
2 103    C  26  bhel        hr
3 107    G  68  ecil     sales
4 104    D  45  jntu     sales
5 102    B  89  ecil marketing
6 110    J  58 nzmpt        hr
```

Data.table package:

> emp<-data.table(eid=101:110, ename=c("A","B","C","D","E","F","G","H","I","J"),
esal=c(85,89,78,63,26,45,31,95,81,62),
eloc=c("kphb","nzmpt","mp","jntu","kkp","lkpl","lbngr","dsnr","nzmpt","mp"))

> emp

```
    eid ename esal  eloc
 1:  101   A    85   kphb
 2:  102   B    89  nzmpt
 3:  103   C    78    mp
 4:  104   D    63   jntu
 5:  105   E    26    kkp
 6:  106   F    45   lkpl
 7:  107   G    31  lbngr
 8:  108   H    95   dsnr
 9:  109   I    81  nzmpt
10:  110   J    62    mp
```

> emp[eid==106]

```
   eid ename esal eloc
1: 106     F   45 lkpl
```

> emp[eid==106&esal==78]

```
Empty data.table (0 rows and 4 cols): eid,ename,esal,eloc
```

> emp[eid==106&esal==45]

```
   eid ename esal eloc
1: 106     F   45 lkpl
```

> emp[order(esal)]

```
    eid ename esal  eloc
 1: 105   E    26   kkp
 2: 107   G    31  lbngr
 3: 106   F    45   lkpl
 4: 110   J    62    mp
 5: 104   D    63   jntu
 6: 103   C    78    mp
 7: 109   I    81  nzmpt
 8: 101   A    85   kphb
 9: 102   B    89  nzmpt
10: 108   H    95   dsnr
```

> emp[order(-esal)]

```
    eid ename esal  eloc
 1: 108   H    95   dsnr
 2: 102   B    89  nzmpt
 3: 101   A    85   kphb
 4: 109   I    81  nzmpt
 5: 103   C    78    mp
 6: 104   D    63   jntu
 7: 110   J    62    mp
 8: 106   F    45   lkpl
 9: 107   G    31  lbngr
10: 105   E    26   kkp
```

> emp[,ename]

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

> emp[,list(ename)]

```
    ename
 1:     A
 2:     B
 3:     C
 4:     D
 5:     E
 6:     F
 7:     G
 8:     H
 9:     I
10:     J
```

> emp[,.(ename)]

```
    ename
 1:     A
 2:     B
 3:     C
 4:     D
 5:     E
 6:     F
 7:     G
 8:     H
 9:     I
10:     J
```

```
> emp[,.(eid,ename)]
    eid ename
 1: 101     A
 2: 102     B
 3: 103     C
 4: 104     D
 5: 105     E
 6: 106     F
 7: 107     G
 8: 108     H
 9: 109     I
10: 110     J

> emp[,.(eid,esal)]
    eid esal
 1: 101   85
 2: 102   89
 3: 103   78
 4: 104   63
 5: 105   26
 6: 106   45
 7: 107   31
 8: 108   95
 9: 109   81
10: 110   62

> emp[,.(id=eid,nm=ename)]
     id nm
 1: 101  A
 2: 102  B
 3: 103  C
 4: 104  D
 5: 105  E
 6: 106  F
 7: 107  G
 8: 108  H
 9: 109  I
10: 110  J

> emp[,.(eid,eloc=="mp")]
    eid    V2
 1: 101 FALSE
 2: 102 FALSE
 3: 103  TRUE
 4: 104 FALSE
 5: 105 FALSE
 6: 106 FALSE
 7: 107 FALSE
 8: 108 FALSE
 9: 109 FALSE
10: 110  TRUE

> emp[eid==106,.(eid,eloc=="mp")]
   eid    V2
1: 106 FALSE
```

```
> emp[eid==110,.(eid,eloc=="mp")]
   eid   V2
1: 110 TRUE

> emp[,eid,by="eloc"]
        eloc eid
 1:    kphb 101
 2:   nzmpt 102
 3:   nzmpt 109
 4:      mp 103
 5:      mp 110
 6:    jntu 104
 7:     kkp 105
 8:    lkpl 106
 9:   lbngr 107
10:    dsnr 108

> emp[eloc=="nzmpt",eloc:="nizampet"]

> emp
    eid ename esal     eloc
 1: 101     A   85     kphb
 2: 102     B   89 nizampet
 3: 103     C   78       mp
 4: 104     D   63     jntu
 5: 105     E   26      kkp
 6: 106     F   45     lkpl
 7: 107     G   31    lbngr
 8: 108     H   95     dsnr
 9: 109     I   81 nizampet
10: 110     J   62       mp
```

Reshape2 package:

```
> install.packages("reshape2")

> library("reshape2")

> molten<-melt(emp,id=c("eid","ename"),measured=c("esal","eloc"))
```

	eid	ename	variable	value
1	101	A	esal	85
2	102	B	esal	89
3	103	C	esal	78
4	104	D	esal	63
5	105	E	esal	26
6	106	F	esal	45

	eid	ename	variable	value
7	107	G	esal	31
8	108	H	esal	95
9	109	I	esal	81
10	110	J	esal	62
11	101	A	eloc	kphb
12	102	B	eloc	nizampet
13	103	C	eloc	mp
14	104	D	eloc	jntu
15	105	E	eloc	kkp
16	106	F	eloc	lkpl
17	107	G	eloc	lbngr
18	108	H	eloc	dsnr
19	109	I	eloc	nizampet
20	110	J	eloc	mp

```
> t<-dcast(molten,eid+ename~variable)

> t
   eid ename esal     eloc
1  101     A   85     kphb
2  102     B 89 nizampet
3  103     C   78       mp
4  104     D   63     jntu
5  105     E   26      kkp
6  106     F   45     lkpl
7  107     G   31    lbngr
8  108     H   95     dsnr
9  109     I 81 nizampet
10 110     J   62       mp

> t1<-acast(molten,eid+ename~variable)

> t1
      esal eloc
101_A "85" "kphb"
102_B "89" "nizampet"
103_C "78" "mp"
104_D "63" "jntu"
105_E "26" "kkp"
106_F "45" "lkpl"
```

```
107_G "31" "lbngr"
108_H "95" "dsnr"
109_I "81" "nizampet"
110_J "62" "mp"

> class(t)
[1] "data.frame"

> class(t1)
[1] "matrix"

> t2<-dcast(molten,variable~eid+ename)

> t2
  variable 101_A    102_B 103_C 104_D 105_E 106_F 107_G 108_H    109_I 110_J
1     esal    85       89    78    63    26    45    31    95       81    62
2     eloc  kphb nizampet    mp  jntu   kkp  lkpl lbngr  dsnr nizampet    mp
```