DATA VISUALIZATION
IMPORTANT QUESTIONS
UNIT-I

1. What is statistics? Explain descriptive and inferential

statistics in detail.

A: 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)

1.**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 is equal to the sum of all the values in the data set divided by the number of values in the data set.

$$Mean = \sum_{i=0}^{n} \frac{x_i}{n}$$

Median:

The median is the middle score for a set of data that has been arranged in order of magnitude.

Mode:

The mode is the most frequent score in our data set.

2 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, variance and standard deviation.

Range: Range of the set is the difference between the largest (max()) and smallest (min()) values.

Variance: The variance is the average difference of each value in the sample from the mean

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

standard deviation: The standard deviation is simply the square root of the variance.

$$\sigma = \sqrt{\frac{\Sigma\,(x - \mu)^2}{n - 1}}$$

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.

There are two main areas of inferential statistics:

1. Estimating parameters:

This means taking a statistic from the sample data and using it to infer about a population parameter.

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 of random

ii) Consistent

A consistent estimator is one that as the sample size increased, the probability that estimate has a value close to the parameter also increased.

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 approximate calculation.

2. Hypothesis tests:

Z TEST

• A z-test is a statistical test used to determine whether two population means are different when the variances are known and the sample size is large.

T Test

• The T-test is used to compare the mean of two given groups.

ANOVA

• An ANOVA test is a way to find out if survey or experiment results are significant. In other words, they help you to figure out if you need to reject the null hypothesis or accept the alternate hypothesis

2. Explain different types of sampling methods in detail.

Sampling: Sampling is the act, process, or technique of selecting a suitable sample

Sample: a representative part of a population for the purpose of determining parameters or characteristics of the whole population.

MAINLY TWO METHODS:

- PROBABILITY SAMPLING
- NON-PROBABILITY SAMPLING

PROBABILITY SAMPLING: The subjects of the population get an equal opportunity to be selected as a representative sample.
Unbiased, Objective, Tested.

- Simple random sampling
- Systematic Random Sampling
- Stratified Random Sampling
- Multistage Sampling.
- Multiphasic sampling
- Cluster Sampling

Simple random sampling: Each unit of the population has an equal chance of inclusion in the sample. Applicable when population is small, homogeneous and readily available.
Ex: Lottery method, Table of random number.

Systematic Random Sampling: Applicable when the population is large, scattered and not homogeneous. There is an equal interval between each selected unit**.** The time and labour is relatively small.

Stratified Random Sampling: Applicable when population is not homogeneous. The population is first divided into homogeneous groups (age, sex, area, classes) called Strata. It gives greater accuracy.

Multistage Sampling: The selection is done in stages. First stage, random number of districts chosen in all states. Followed by random number of talukas, villages. Then third stage units will be houses. All ultimate units (houses) selected at last step are surveyed.
Cluster sampling is an example of multistage sampling

Multiphasic sampling: Part of the information is collected from whole population and part from sub samples.
Ex: 20 fever cases clinical examination & blood test - high ESR - Widal, MP etc. – Those found negative again subjected to other tests.

Cluster Sampling: A cluster is a randomly selected group. Applicable when units of population are natural groups or clusters (villages, wards, blocks, factories, schools, etc.).

NON-PROBABILITY SAMPLING: Subjects chosen by researcher. Hence, who will be selected cannot be predicted.
Biased, Subjective, Generated

- Convenience/Purposive Sampling
- Judgment Sampling
- Quota Sampling

- Snowball sampling

Convenience/Purposive Sampling: Subjects selected because it is easy to access them. This method is also called CHUNK. Generally used for making pilot studies.
Results obtained are unsatisfactory in terms of drawing conclusions.

Judgment Sampling: The choice of study units depends exclusively on the judgment of the investigators. based on some character of sample members to serve a purpose.

Quota Sampling: In this, the population is divided into quotas according to some specific characteristics (age, sex, religion, etc.) represented to the exact extent that the investigator desires No. of study units selected within each quota depend on personal judgment.

Snowball sampling: Initial respondents chosen by probability or non-probability methods. Then additional respondents are obtained by information provided by the initial respondent When the target population is hidden and or hard to reach such as commercial sex workers, drug addicts or a patient of a rare disease or happenings in a controversial event.

3. Explain the data structures of R programming language with an example.

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.
The most essential data structures used in R include:

- Vectors
- Lists
- Dataframes
- Matrices
- Arrays
- Factors

Vectors:

A vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.
Example:

```
X = c(1, 3, 5, 7, 8)
  print(X)
```

Output:

```
[1] 1 3 5 7 8
```

Lists

A list is a generic object consisting of an ordered collection of objects. Lists are heterogeneous data structures. These are also one-dimensional data structures. A list can be a list of vectors, list of matrices, a list of characters and a list of functions and so on.

Example:

```
empName = c("Debi", "Sandeep", "Subham", "Shiba")
print(empList)
```

Output:

```
[1] "Debi"    "Sandeep" "Subham"  "Shiba"
```

Dataframes:

Dataframes are generic data objects of R which are used to store the tabular data. Dataframes are the foremost popular data objects in R programming because we are comfortable in seeing the data within the tabular form. They are two-dimensional, heterogeneous data structures. These are lists of vectors of equal lengths.

Example:

```
Name = c("Amiya", "Raj", "Asish")
Language = c("R", "Python", "Java")
Age = c(22, 25, 45)
df = data.frame(Name, Language, Age)
print(df)
```

Output:

```
Name Language Age
1 Amiya        R  22
2    Raj   Python  25
3 Asish     Java  45
```

Matrices

A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures.

Example:

```
A=matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,ncol=3,byrow=TRUE)
print(A)
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Arrays:

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures.

Example:

```
A = array(c(1, 2, 3, 4, 5, 6, 7, 8),dim = c(2, 2, 2))
print(A)
```

Output:

```
, , 1

     [,1] [,2]
[1,]    1    3
[2,]    2    4

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Factors:

Factors are the data objects which are used to categorize the data and store it as levels. They are useful for storing categorical data. They can store both strings and integers. They are useful to categorize unique values in columns like “TRUE” or “FALSE”, or “MALE” or “FEMALE”, etc..

Example:

```
fac = factor(c("Male", "Female", "Male",
                    "Male", "Female", "Male", "Female"))
print(fac)
```

Output:

```
[1] Male      Female Male      Male      Female Male      Female
Levels: Female Male
```

4. Explain any 6 Commands of R programming language with an example.

6 Commands of R programming language:

builtins():list of all built-in functions.

Example:

```
> builtins()
```

Outputs:

```
 [1] "zapsmall"                  "xzfile"
 [3] "xtfrm.POSIXlt"             "xtfrm.POSIXct"
 [5] "xtfrm.numeric_version"     "xtfrm.factor"
 [7] "xtfrm.difftime"            "xtfrm.default"
 [9] "xtfrm.Date"                "xtfrm.AsIs"
```

help(): to get information about objects
Example:
```
> help("library")
```
Output:
Description
library and require load and attach add-on packages.

length(): Return no.of elements in vector x
Example:
```
> X = c(1, 3, 5, 7, 8)
> length(X)
```
Output:
[1] 5

vec(): create a vector.
Example:
```
X = vec(1, 3, 5, 7, 8)
  print(X)
```
Output:
[1] 1 3 5 7 8

range(): returns minimum and maximum of x
Example:
```
X = c(1, 3, 5, 7, 8)
  > range(X)
```
Output:
[1] 1 8

rep(m,n): repeat m n times
Example:
```
rep(1,5)
```
Example:
[1] 1 1 1 1 1

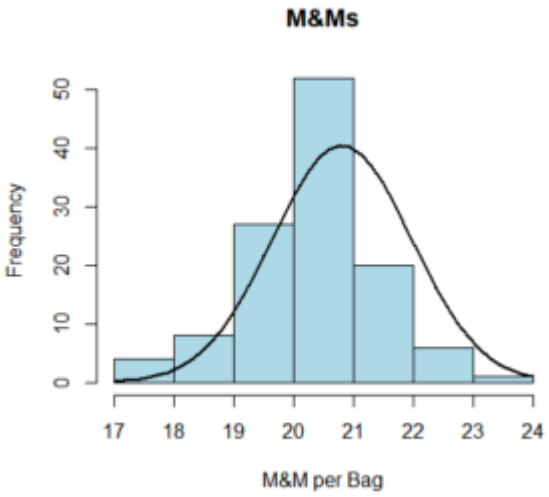
5) Explain the sample statistics and sampling Distributions.

A: Sample Statistics: Same as Second one

Sampling Distributions

A **sampling distribution** represents the distribution of the statistics for a particular sample.

For example, a sampling distribution of the mean indicates the frequency with which specific occur. This means that the frequency of values is mapped out. You can also create distributions of other statistics, like the variance. Below is an example of a sampling distribution for the mean



The shape of the curve allows you to compare the *empirical* distribution of value to a *theoretical* distribution of values. A theoretical distribution is a distribution that is based on equations instead of empirical data. Two common theoretical distributions are Student's t and the F-distribution.
The benefit of creating distributions is that the empirical ones can be compared to theoretical ones to identify differences or goodness of fit for the model. That is the ultimate goal of statistics, to create an empirical model that explains patterns in the data that differ significantly from the theoretical model.

6) Explain about descriptive data analysis using R? How to Describe basic functions used to describe data in R.

A: 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(mydata, 6)

summary() function:

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.

```
g=summary(emp.data, digit=1)
print(g)
```

Output:

```
 Min.   :1   Length:3         Min.   :20
 1st Qu.:2   Class :character   1st Qu.:20
 Median :2   Mode  :character   Median :20
 Mean   :2                     Mean   :30
 3rd Qu.:2                     3rd Qu.:35
 Max.   :3                     Max.   :50
```

Sapply functions():

It Compute the mean of each column.

```
a=sapply(emp.data[, -5], mean)
print(a)
```

Output:

```
emp_id emp_name   salary
     2         NA         30
```

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

UNIT-2

1. **Explain the verbs of Dplyr package with an example.**

ANS: The dplyr package in R is a structure of data manipulation that provides a uniform set of verbs, helping to resolve the most frequent data manipulation hurdles.

The dplyr package contains five key data manipulation functions, also called verbs:

- select()
- filter()
- arrange()
- mutate()
- summarise()

select() :

- which returns a subset of the columns.
- For choosing variables

Syntax:

Select(data,var1,var2,……)

Example:

```
d <- data.frame( name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),
                 age = c(7, 5, 9, 16),
                 ht = c(46, NA, NA, 69),
                 school = c("yes", "yes", "no", "no") )
d
select(d, starts_with("ht"))
```

output:

```
  name       age      ht school
1 Abhi         7      46 yes
```

2 Bhavesh 5 NA yes

3 Chaman 9 NA no

4 Dimri 16 69 no

ht

1 46

2 NA

3 NA

4 69

filter() :

- that is able to return a subset of the rows.
- For choosing cases and using their values as a base for doing so.

Example:

```
d %>% filter(is.na(ht))
```

output:

name age ht school

1 Bhavesh 5 NA yes

2 Chaman 9 NA no

arrange():

- that reorders the rows according to single or multiple variables
- For reordering of the cases.

Example:

```
d.name<- arrange(d, age)

print(d.name)
```

output:

name age ht school

1 Bhavesh 5 NA yes

2 Abhi 7 46 yes

3 Chaman 9 NA no

4 Dimri 16 69 no

mutate():

- used to add columns from existing data
- Addition of new variables which are the functions of prevailing variables.

Example:

```
mutate(d, x3 = ht + age)
```

output:

name age ht school x3

1 Abhi 7 46 yes 53

2 Bhavesh 5 NA yes NA

3 Chaman 9 NA no NA

4 Dimri 16 69 no 85

summarise():

- which reduces each group to a single row by calculating aggregate measures.
- Condensing various values to one value.

Example:

```
summarise(d, mean = mean(age))
```

output:

mean

1 9.25

2. **Explain the process of plotting a graph using different layers of ggplot2 in detail.**

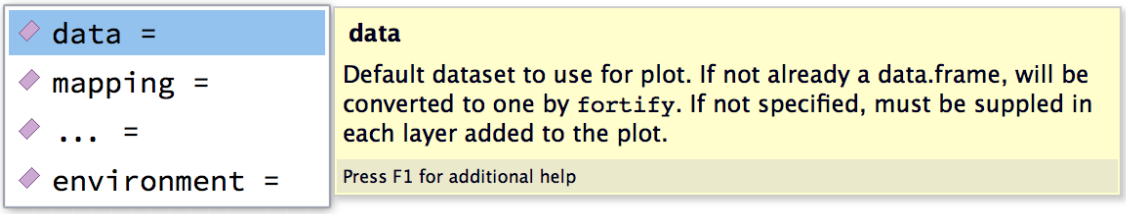
ANS: ggplot2 is a R package dedicated to data visualization. It can greatly improve the quality and aesthetics of your graphics, and will make you much more efficient in

creating them. ... ggplot2 builds charts through layers using geom_ functions.

Step 1: Creating a plot object

We start with a data set and create a plot object with the function ggplot(). This function has a data frame as the first argument. This means that we can use the pipe operator:

```
>
> ggplot()
```



We have two options. We prefer the pipe notation here, but it is also possible to specify the data frame as an argument within the function. Furthermore, we assign the object to a variable and call it p.

```
# 1st option
p <- ggplot(data = stress)

# 2nd option
p <- stress %>%
  ggplot()
```

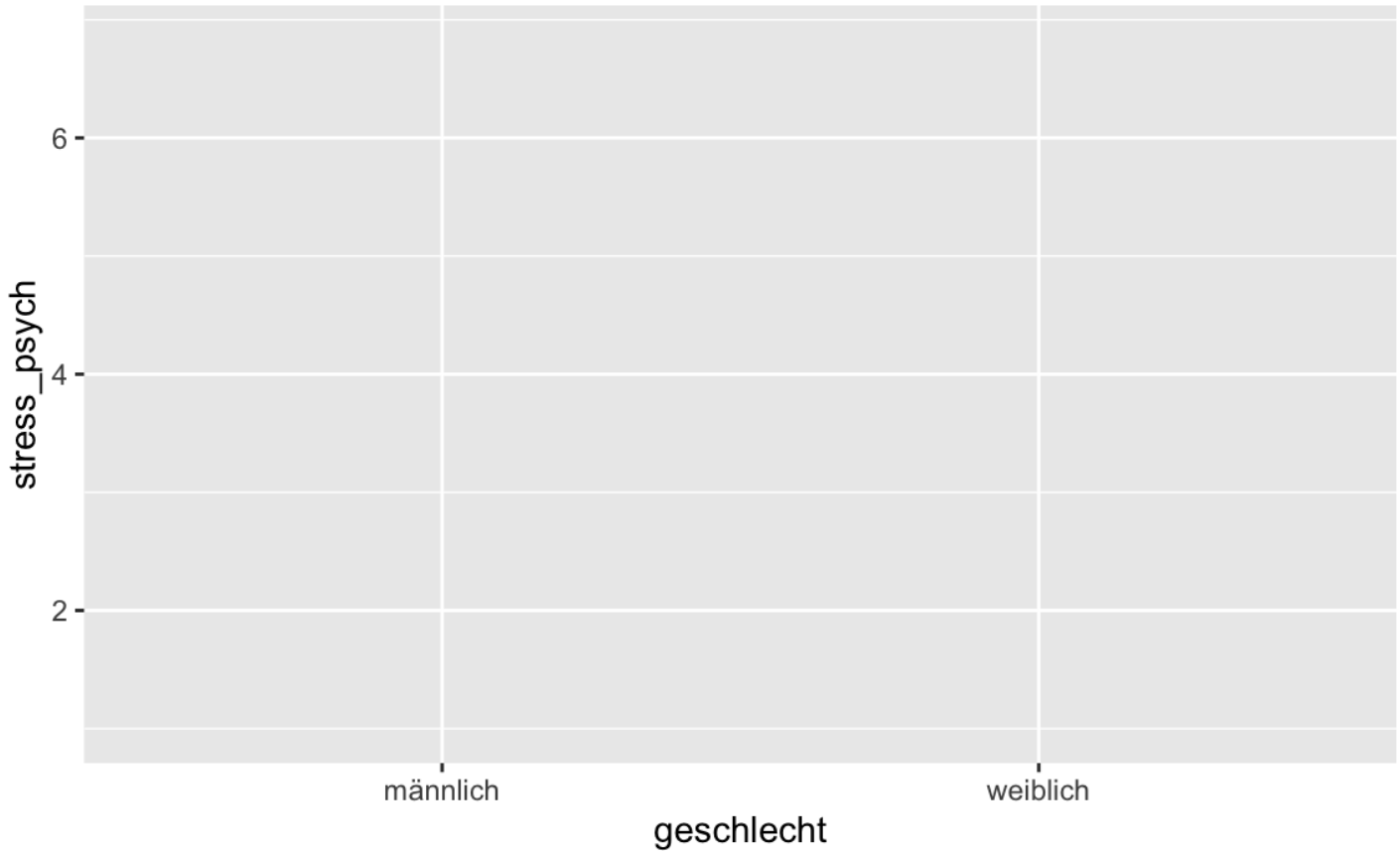
Step 2: Aesthetic mappings

With the second argument mapping we now define the “aesthetic mappings”. These determine how the variables are used to represent the data and are defined using the aes() function. We want to represent the grouping variable gender on the X-axis and stress_psychisch should be displayed on the Y-axis. In addition, aes() can have additional arguments: fill,color, shape,linetype, group. These are used to assign different colors, shapes, lines, etc. to the levels of grouping variables.

```
p <- stress %>%
  ggplot(mapping = aes(x = geschlecht,
                       y = stress_psychisch,
                       color = geschlecht,
                       fill = geschlecht))
```

p is now an ‘empty’ plot object. We can look at it, but nothing is displayed yet, as it does not yet contain “layers”. A ggplot2 object is displayed by having the object sent to the console, either with or without print ().

```
p

# or print(p)
```



We can see that `ggplot2` has already labeled the axes for us using the variable names.

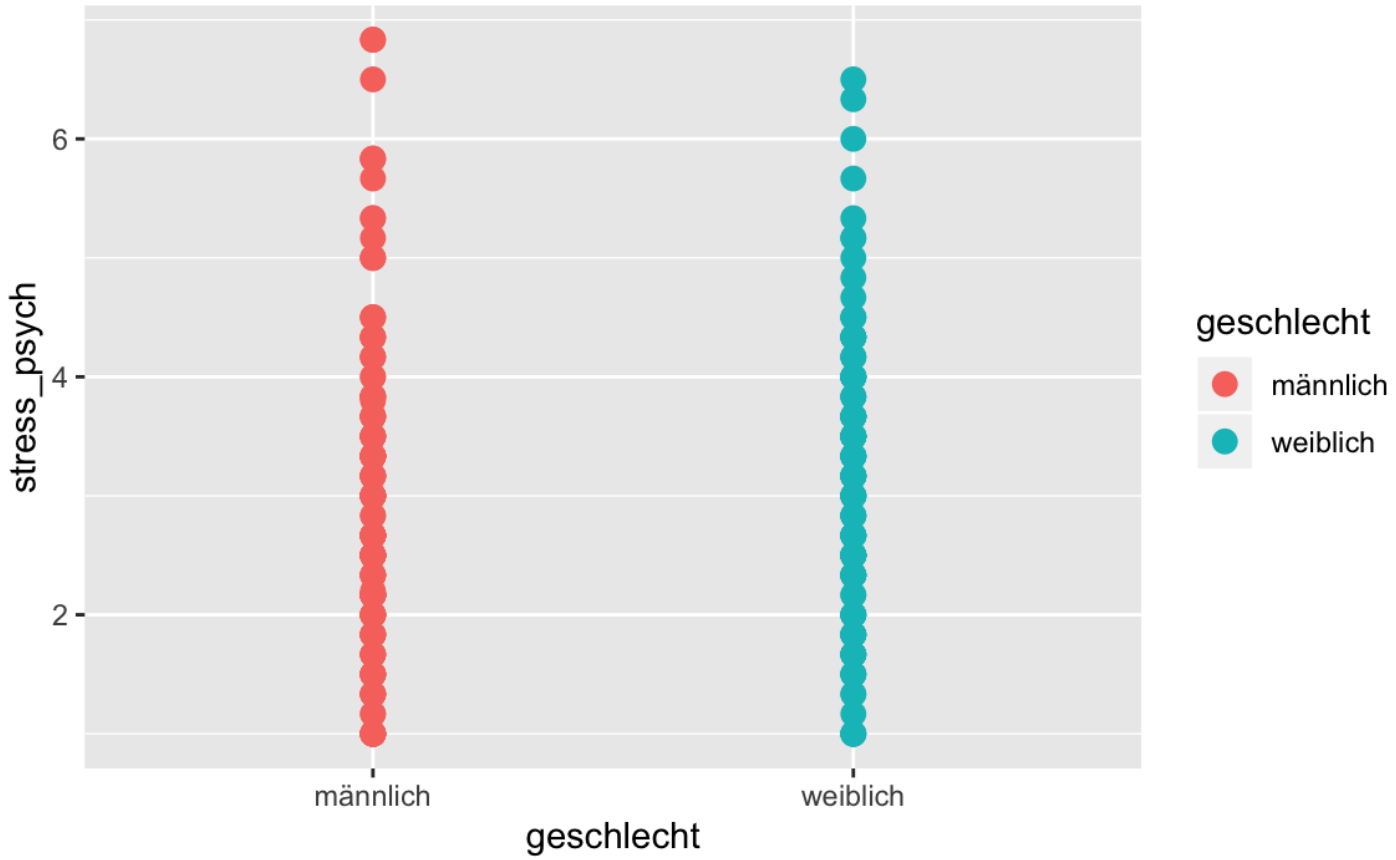
Step 3: Add geoms

Using the `geom_` functions we can now add "layers" to the plot object `p`. The syntax works like this: We "add" (`+`) a `geom` to the plot object `p`: `p + geom_`.

Example: Scatter plot

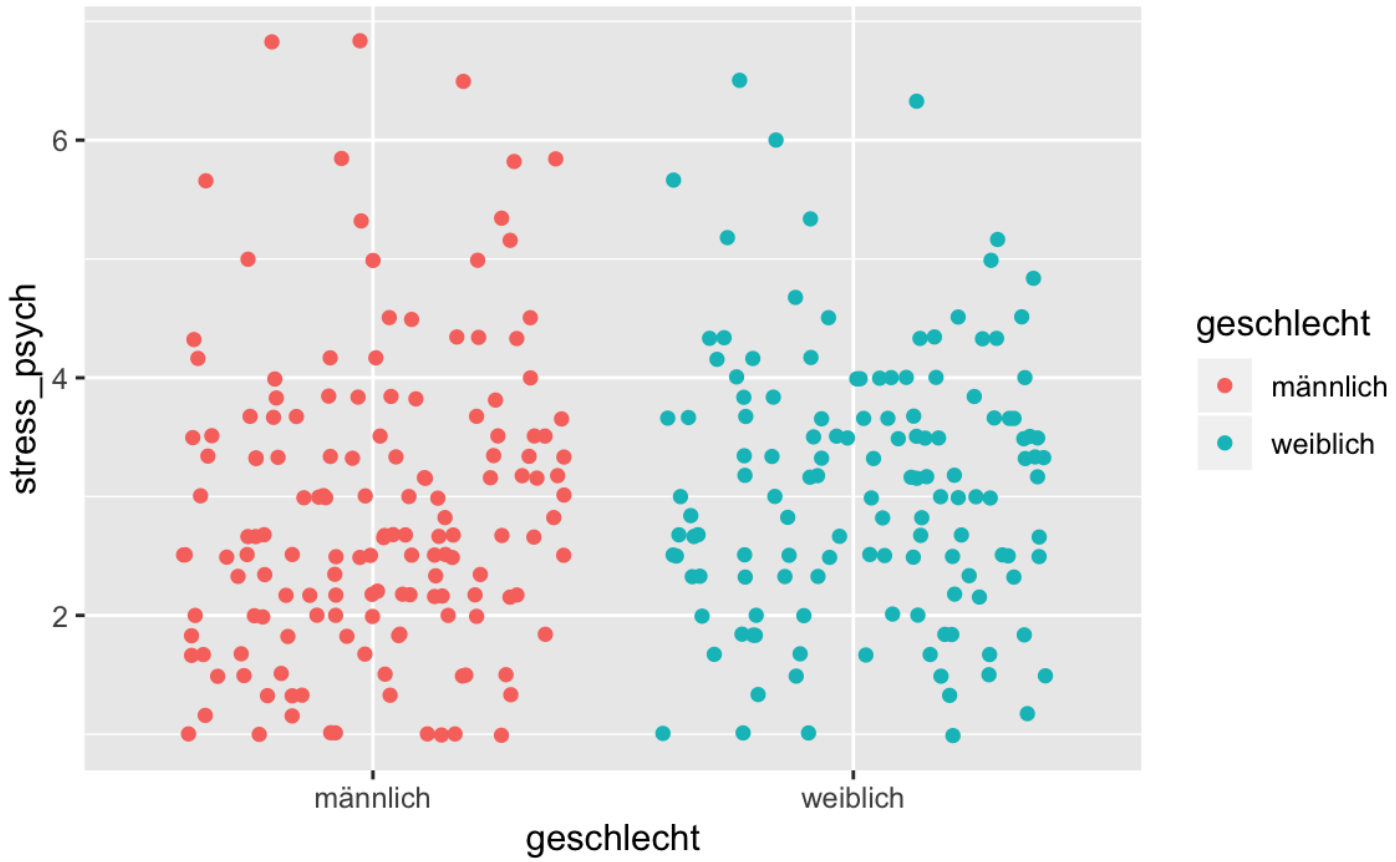
We first try to represent the observations as points:

```
# the geom_point() function has a size argument
p + geom_point(size = 3)
```



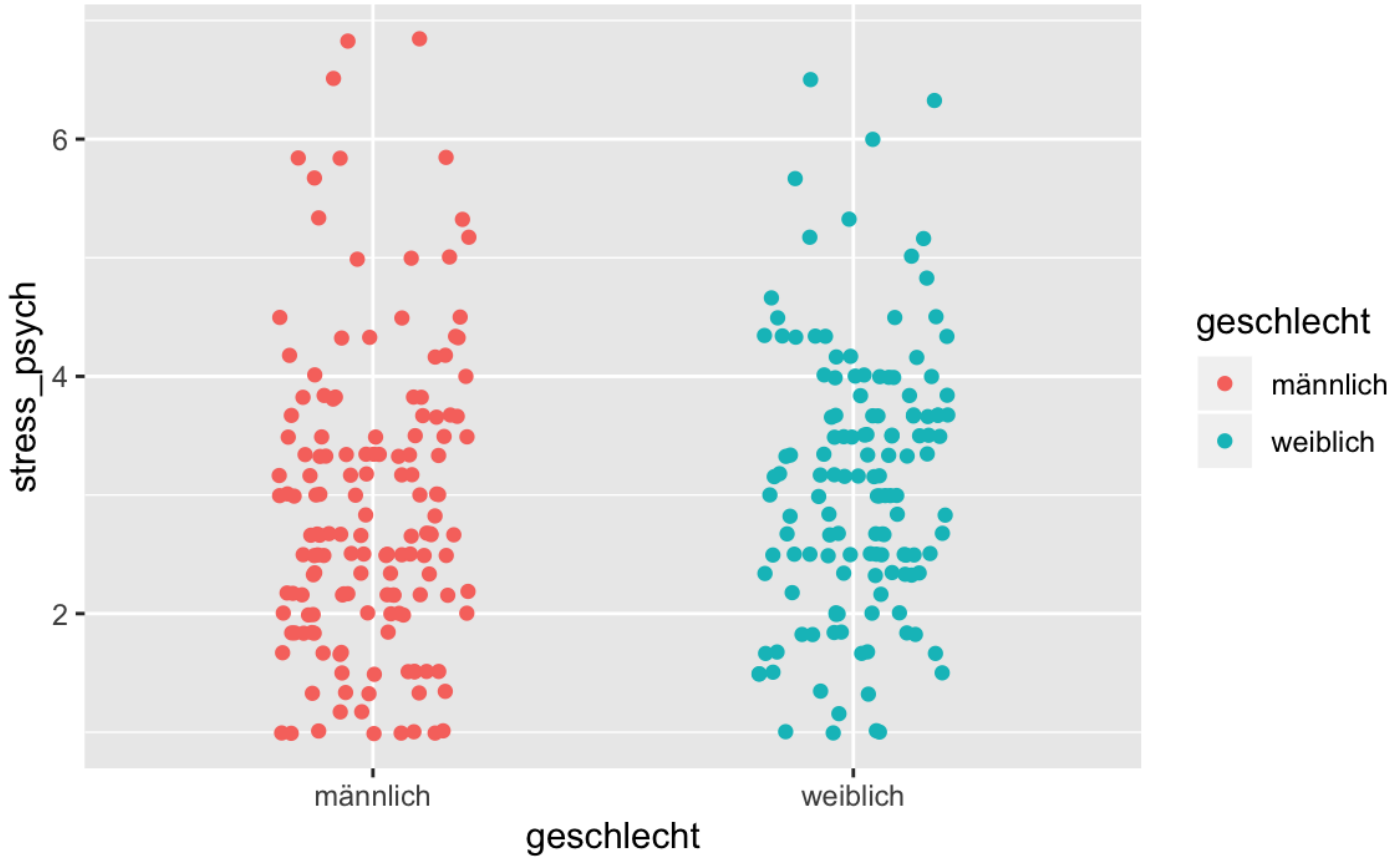
The points are now displayed in different colors, but within a gender points may be plotted on top of each other if they have the same value (overplotting). In this case, there is the function `geom_jitter ()`, which draws points with a jittering side by side:

```
p + geom_jitter()
```



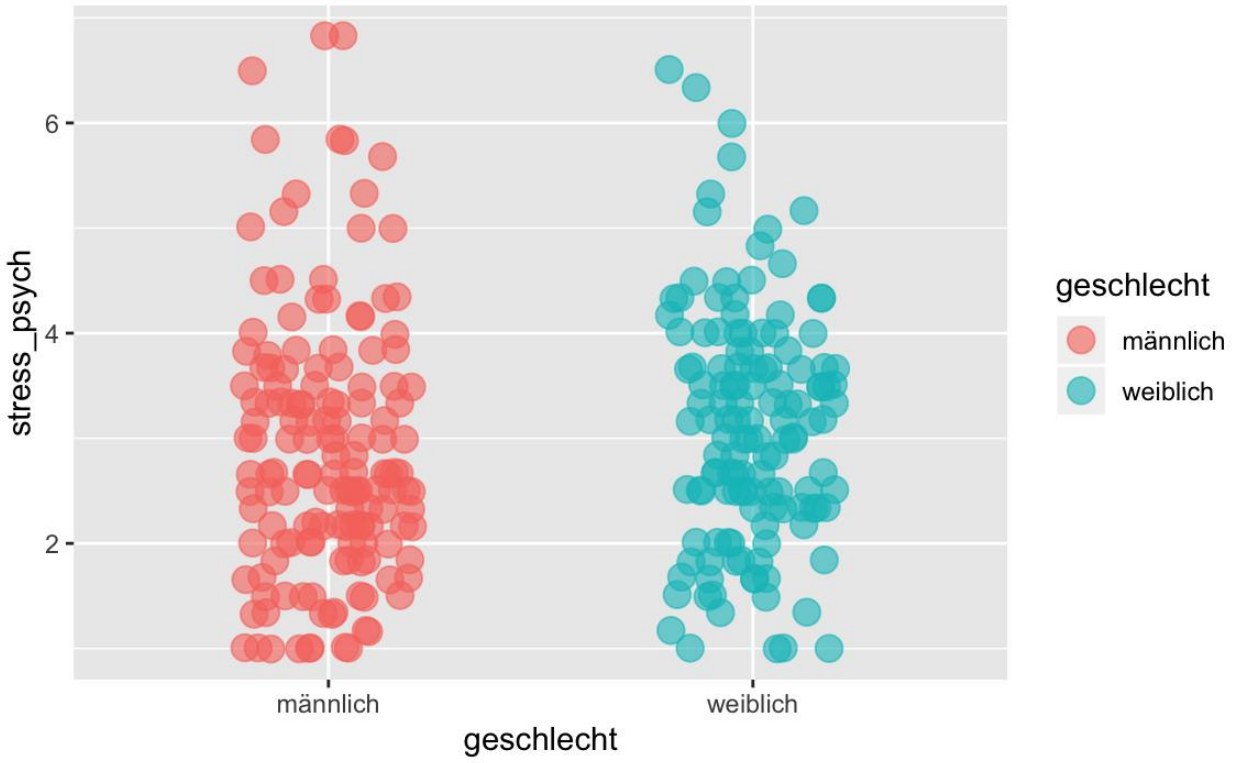
geom_jitter () has an argument width, with which we can define how widely the points are being jittered.

```
p + geom_jitter(width = 0.2)
```



geom_jitter() has further arguments: size determines the diameter of the points, alpha determines their transparency.

```
p + geom_jitter(width = 0.2, size = 4, alpha = 0.6)
```



3. Explain the below plots with visualization

a) bar plot b) jitter plot c) box plot

ANS:

(a) bar plot:- A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function barplot() to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax:- The basic syntax to create a bar-chart in R is –

barplot(H, xlab, ylab, main, names.arg, col)

Following is the description of the parameters used –

- H is a vector or matrix containing numeric values used in bar chart.
- xlab is the label for x axis.
- ylab is the label for y axis.
- main is the title of the bar chart.
- names.arg is a vector of names appearing under each bar.
- col is used to give colors to the bars in the graph.

Example:-

A simple bar chart is created using just the input vector and the name of

each bar.

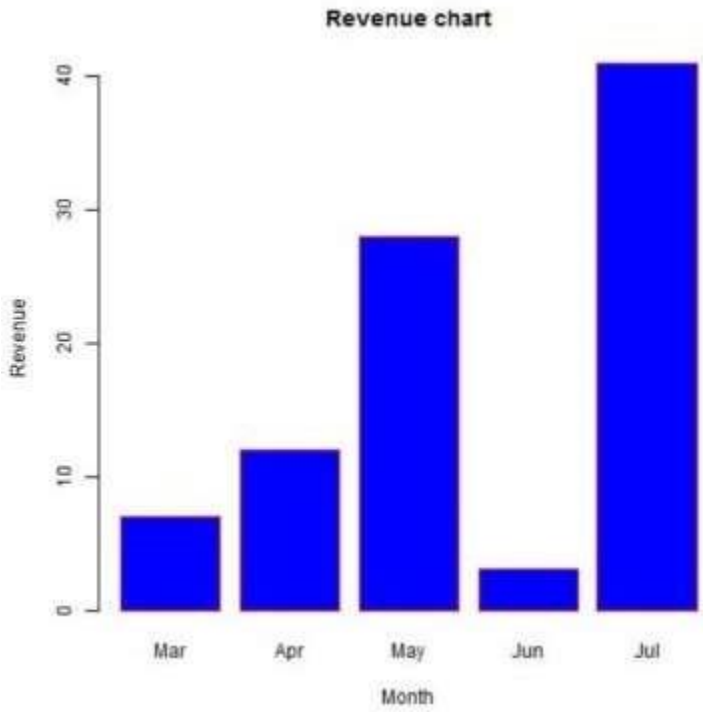
The below script will create and save the bar chart in the current R

working directory.

Live Demo

```
# Create the data for the chart
H <- c(7,12,28,3,41)
# Give the chart file a name
png(file = "barchart.png")
# Plot the bar chart
barplot(H)
```

```
# Save the file

dev.off()
```



(b) jitter plot (or) scatter plot:-

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis. The simple scatterplot is created using the plot() function.

Syntax:-

The basic syntax for creating scatterplot in R is −

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

Following is the description of the parameters used −

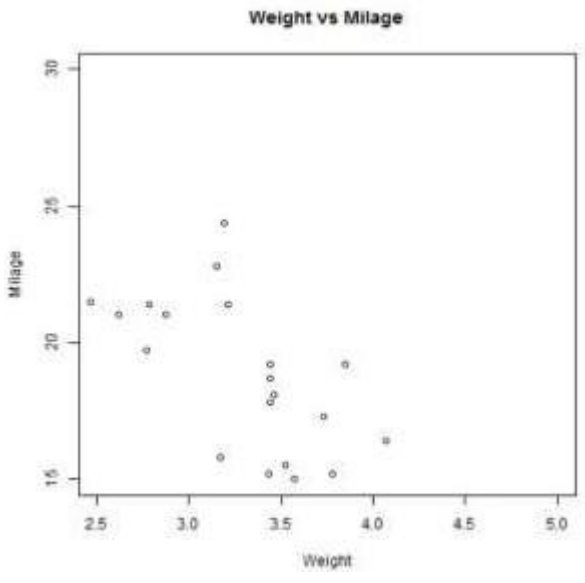
- x is the data set whose values are the horizontal coordinates.
- y is the data set whose values are the vertical coordinates.
- main is the tile of the graph.
- xlab is the label in the horizontal axis.
- ylab is the label in the vertical axis.
- xlim is the limits of the values of x used for plotting.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

Example:-

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

Live Demo

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



(c) box plot:-

Box plots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing box plots for each of them.

Box plots are created in R by using the box plot() function.

Syntax:-

The basic syntax to create a boxplot in R is −

boxplot(x, data, notch, varwidth, names, main)

Following is the description of the parameters used −

- x is a vector or a formula.
- data is the data frame.
- notch is a logical value. Set as TRUE to draw a notch.
- varwidth is a logical value. Set as true to draw width of the box proportionate to the sample size.
- names are the group labels which will be printed under each boxplot.
- main is used to give a title to the graph.

Example:-

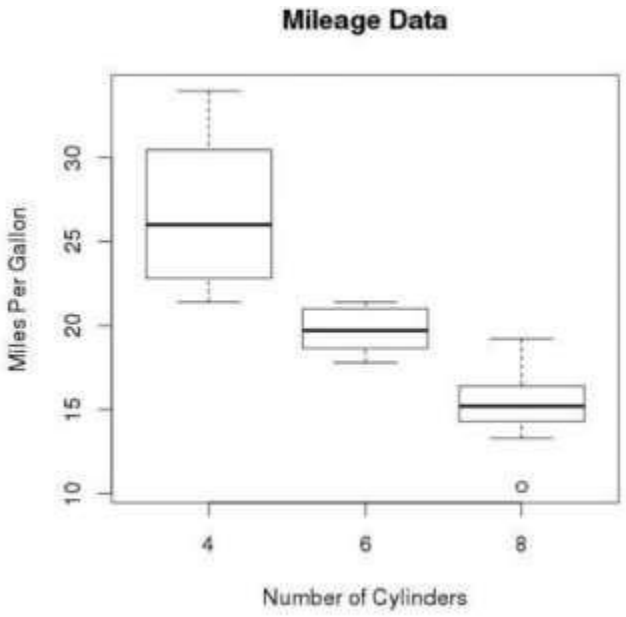
The below script will create a boxplot graph for the relation

between mpg (miles per gallon) and cyl (number of cylinders).

Live Demo

```
# Give the chart file a name.
png(file = "boxplot.png")
# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of
Cylinders",
  ylab = "Miles Per Gallon", main = "Mileage Data")
# Save the file.
```

```
dev.off()
```



4. Explain the following commands of Tidyr Package with an example

a) gather () b) spread () c) unite () d) separate ()

ANS: The sole purpose of the tidyr package is to simplify the process of creating tidy data. Tidy data describes a standard way of storing data that is used wherever possible throughout the tidyverse. If you once make sure that your data is tidy, you'll spend less time punching with the tools and more time working on your analysis. # load the tidyr package
library(tidyr)
code:

```
n = 3
tidy_dataframe = data.frame(
                                S.No = c(1:n),
                              Group.1 = c(23, 345, 76),
                              Group.2 = c(117, 89, 66),
                              Group.3 = c(29, 101, 239))
tidy_dataframe
S.No Group.1 Group.2 Group.3
1       1        23       117         29
2       2       345        89        101
3       3        76        66        239
```

a) gather: It takes multiple columns and gathers them into key-value pairs. Basically it makes "wide" data longer. The gather() function will take multiple columns and collapse them into key-value pairs, duplicating all other columns as needed.
Syntax:

```
gather(data, key = "key", value = "value", …, na.rm = FALSE, convert = FALSE, factor_key =
```

```
FALSE)
 long <- tidy_dataframe %>%
+         gather(Group, Frequency,
+                      Group.1:Group.3)
> long
   S.No      Group Frequency
1       1 Group.1            23
2       2 Group.1           345
3       3 Group.1            76
4       1 Group.2           117
5       2 Group.2            89
6       3 Group.2            66
7       1 Group.3            29
8       2 Group.3           101
9       3 Group.3           239
```

b) spread (): It helps in reshaping a longer format to a wider format. The spread() function spreads a key-value pair across multiple columns.

Syntax:

spread(data, key, value, fill = NA, convert = FALSE)

```
back_to_wide <- long %>%
+         spread(Group, Frequency)
> back_to_wide
   S.No Group.1 Group.2 Group.3
1       1          23        117          29
2       2         345          89         101
3       3          76          66         239
```

c)separate(): It converts longer data to a wider format. The separate() function turns a single character column into multiple columns.

Syntax:

separate(data, col, into, sep = ” “, remove = TRUE, convert = FALSE)

```
separate_data <- long %>%
+         separate(Group, c("Allotment",
+                                    "Number"))
> separate_data
   S.No Allotment Number Frequency
1       1         Group         1            23
2       2         Group         1           345
3       3         Group         1            76
4       1         Group         2           117
5       2         Group         2            89
6       3         Group         2            66
7       1         Group         3            29
```

```
8       2        Group          3          101
9       3        Group          3          239
```

d)unite(): It merges two columns into one column. The unite() function is a convenience function to paste together multiple variable values into one. In essence, it combines two variables of a single observation into one variable.

Syntax:

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

```
unite_data <- separate_data %>%
+         unite(Group, Allotment,
+                   Number, sep = ".")
> unite_data
   S.No      Group Frequency
1     1 Group.1            23
2     2 Group.1           345
3     3 Group.1            76
4     1 Group.2           117
5     2 Group.2            89
6     3 Group.2            66
7     1 Group.3            29
8     2 Group.3           101
9     3 Group.3           239
```

5) Explain about a) Reshape2 b) Tidyr c) lubridate d) dplyr with examples?

ANS: a) Reshape()

This package is useful in reshaping data. The data come in many forms. Hence, we are required to shape it according to our need.

- Using the reshape2 package, we can combine features that have unique values like 'Aggregation' in SQL.
- It has 2 functions namely melt() and cast() functions.

Melt():

It converts data from wide format to long format. It is a form of restructuring where multiple categorical columns are „melted" into unique rows. Let us understand it using the code below.

```
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)
print(thisdata)
```

Output:

	ID	NAMES	DATE OF BIRTH	SUBJECT
1	1	JOSEPH	1993	MATHS
2	2	MATRIN	1992	BIOLOGY
3	3	JOSEPH	1993	SCIENCE
4	4	JAMES	1994	PYSCOLOGY
5	5	MATRIN	1992	PHYSICS

b) tidyr:

This package can make the data look 'tidy'(nice). It has 4 major functions to accomplish this task such as

1) gather() – it 'gathers' multiple columns. Then, it converts them into key:value pairs. This function will transform wide from of data to long form. You can use it as in alternative to 'melt()' in reshape2 package.

2)spread() – It does reverse of gather. It takes a key:value pair and converts it into separate columns.

3)separate() – It splits a column into multiple columns.

4)unite() – It does reverse of separate. It unites multiple columns into single column

Program:

```
Instrall.packages("tidyr")
Library(tidyr)
Names<-c('A','B','C','D','E','A','B')
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')
tdata<-data.frame(names,age,weight,class)
```

Output:

	names	age	weight	Class
1	A	21	55	Maths
2	B	20	49	Science
3	C	25	76	Social
4	D	29	71	Physics
5	E	33	65	Biology
6	A	32	44	Economics
7	B	38	34	Accounts

C) lubridate:

Time series analysis in R requires a specific format of date objects. Lubridate is a package useful to create those objects. Lubridate is an R package that makes it easier to work with dates and times.

Program:

```
require(lubridate)
datestr <- c("090101", "90102")
parsed <- parse_date_time(datestr, orders=c("ymd"))
parsed_date <- as.Date(parsed)
```

Output:

"2009-01-01" "1990-10-02"

D) dplyr:

1)This package is created and maintained by Hadley Wickham. dplyr is a powerful R-package which transforms and summarizes tabular datawithrows and columns.

2)It is best known for data exploration and transformation.

3) Its chaining syntax makes it highly adaptive to use.

To install dplyr, use the below command

Install.packages("dplyr")

To load dplyr, use the below command

library(dplyr)

❖ It includes 5 major data manipulation commands (or Verbs of dplyr package), which are

- select( ) –used to select columns of interest from a data set
- filter( ) – used to select the row's by filters the data based on a condition
- arrange( ) –used to arrange data set values on ascending or descending order
- mutate( ) – used to create new variables or columns, its values are based on existing columns.
- summarise( ) – used to perform analysis by commonly used operations such as min,max, mean count etc.
- Code:

```
>d<-data.frame(name=c("abhi","Bhavesh","chaman","dimri"),
                    Age=c(7,5,9,16)
                    Ht=c(46,NA,NA,69)
                    School = c("yes","yes","no","no"))
>select(d,name,age)
```

Output:

	Name	age
1	Abhi	7
2	bhavesh	5
3	chaman	9
4	bimri	16

6) explain below plots with visualization
a) line plot b) scatter plot & Histogram
c) Pie chart and subplots.

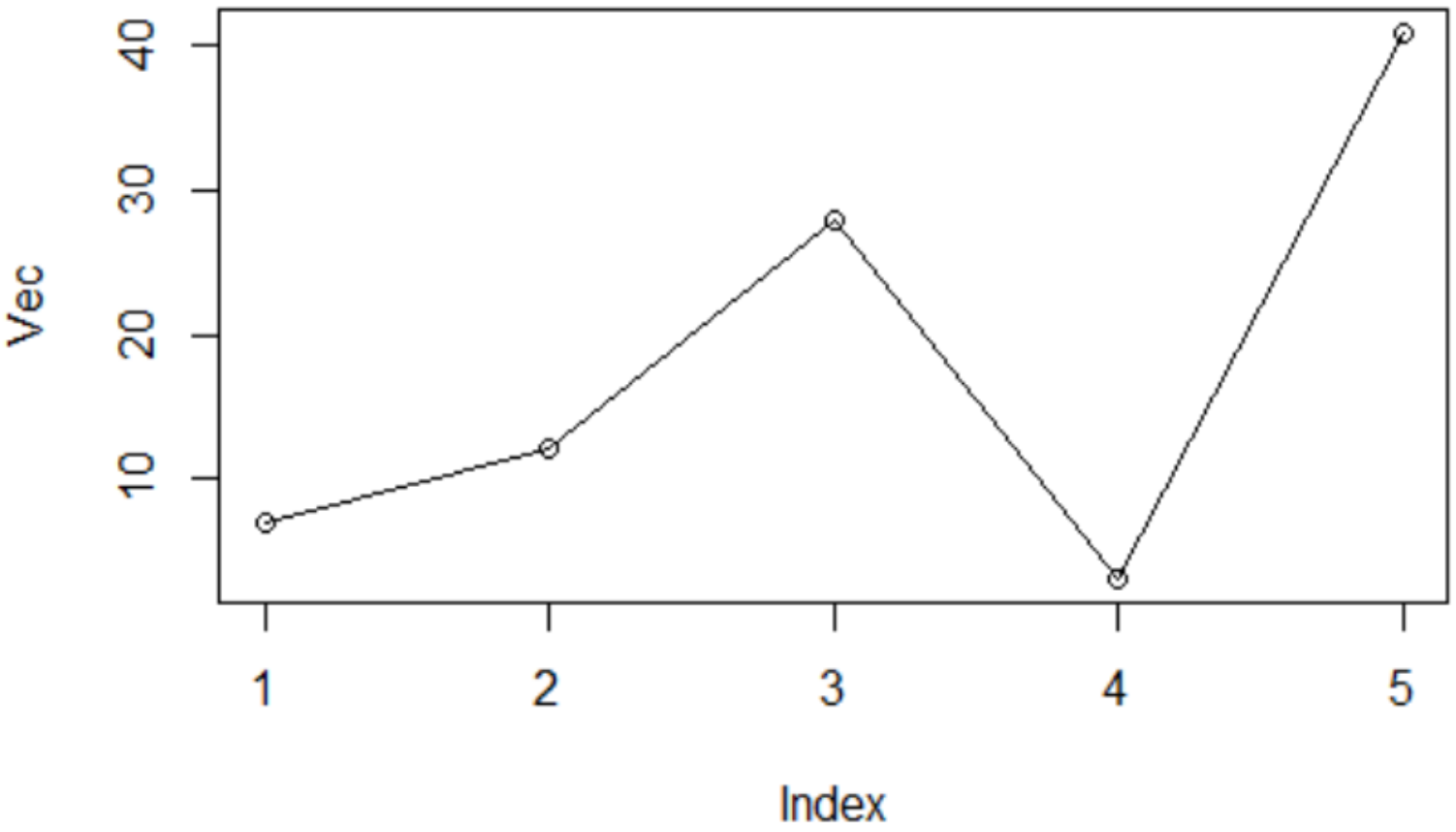
ANS: a) Line Plot:

Line Plot Definition: A line plot (or line graph; line chart) visualizes values along a sequence (e.g. over time). Line plots consist of an x-axis and a y-axis. The x-axis usually displays the sequence and the y-axis the values corresponding to each point of the sequence.

Code:

```
Vec <- c(7,12,28,3,41) #Create the data for the chart
plot(Vec,type = "o")  # Plot the bar chart.
```

Output:

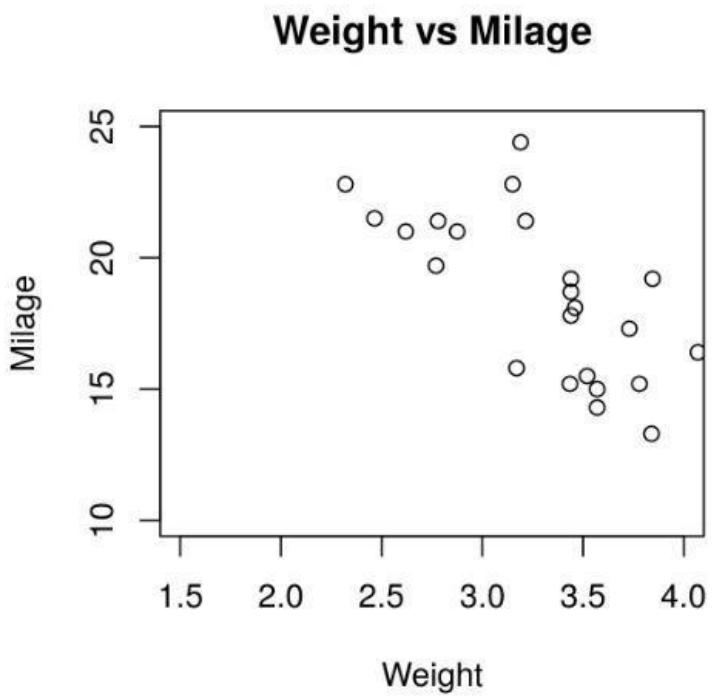


b) scatter plot & histogram():

A scatterplot (or scatter plot; scatter graph; scatter chart; scattergram; scatter diagram) displays two numerical variables with points, whereby each point represents the value of one variable on the x-axis and the value of the other variable on the y-axis

Code:

```
input <- mtcars[, c('wt', 'mpg')]
plot(x = input$wt, y = input$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(1.5, 4),
     ylim = c(10, 25),
     main = "Weight vs Milage"
)
```

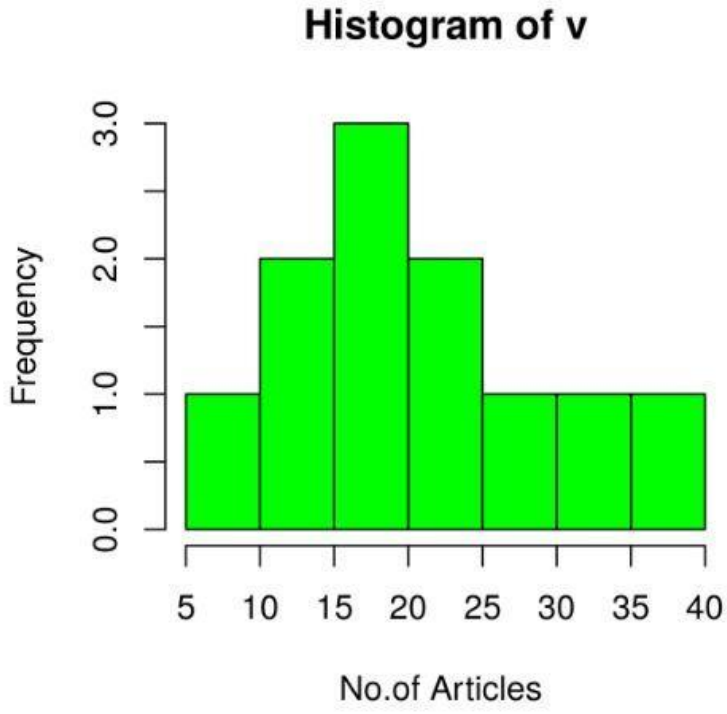


Histogram: A histogram groups continuous data into ranges and plots this data as bars. The height of each bar shows the amount of observations within each range.

Code:

```
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)
hist(v, xlab = "No.of Articles ", col = "green", border = "black")
```

OUTPUT:



C) Pie chart & Subplots:

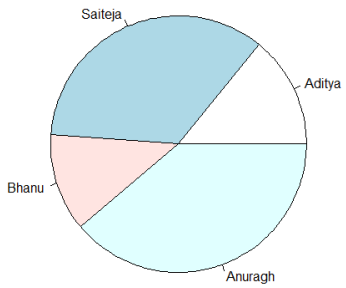
Pie chart:

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportions. It depicts a special chart that uses "pie slices", where each sector shows the relative sizes of data

Code:

```
Age<- c(23, 56, 20, 63)
Names <- c("Aditya", "Saiteja", "Bhanu", "Anuragh")
pie(Age, Names)
```

Output:



7) Explain about various types Data manipulation with R packages with examples.

ANS:

Data Manipulation is the process of manipulating or changing information to make it more organized and readable.

We use DML to accomplish this. DML stands for "Data Manipulation Language capable of adding, removing, and altering databases, i.e. changing the information to something that what we need.

Data Manipulation is also called as Data Exploration (also known as Data Cleaning).

1) Data Manipulation is done to improve data accuracy and precision.
2) Data Manipulation is a mandatory because many faults in data collection process.

Different ways to manipulate:

1. Use the inbuilt functions in R to manipulate data. This is the first step, but is often repetitive and time consuming. Hence, it is a less efficient way to solve the problem.
2. Use the packages available in CRAN to manipulate data. CRAN has more than 8000 packages available today. These packages are a collection of pre-written commonly used pieces of codes. They helps to perform the repetitive tasks fasts, reduce errors in coding and take help of code written by experts (across the open source eco-system for R) to make code more efficient. This is usually the most common way of performing data manipulation.
3. Use Machine Learning (ML) algorithms to manipulate data. ML algorithms like tree based boosting algorithms to take care of missing data & outliers. These algorithms are less time consuming.

Data manipulation packages in R:

1.dplyr.

2.ggplot2.

3.readr.

4.tidyr.

1.dplyr:

This package is created and maintained by Hadley Wickham. dplyr is a powerful R-package which transforms and summarizes tabular data with rows and columns.

Code:

```
>d<-data.frame(name=c("abhi","Bhavesh","chaman","dimri"),
                Age=c(7,5,9,16)
                Ht=c(46,NA,NA,69)
                School = c("yes","yes","no","no"))
>select(d,name,age)
```

Output:

	Name	age
1	**Abhi**	**7**
2	**bhavesh**	**5**

3 chaman 9

4 bimri 16

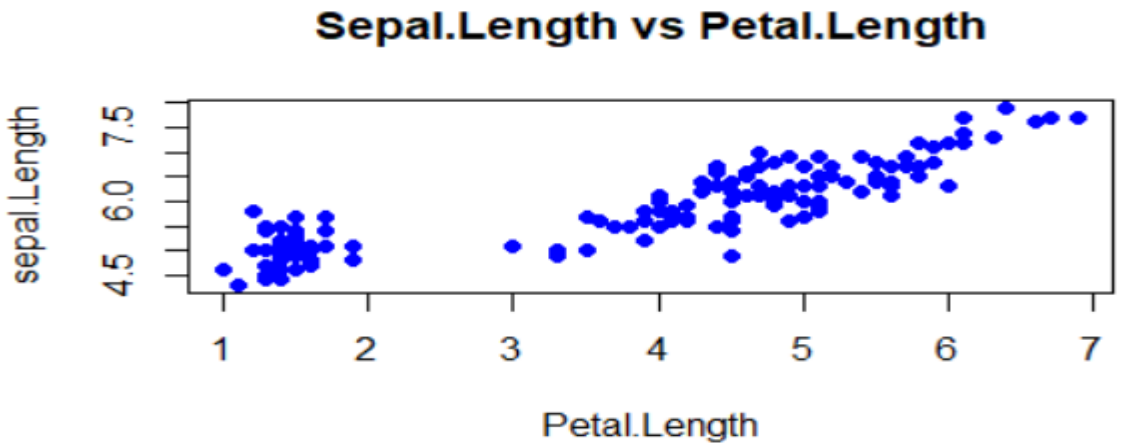
2.ggplot2:

ggplot2 is a R package dedicated to data visualization. It can greatly improve the quality and aesthetics of your graphics, and will make you much more efficient in creating them. ... ggplot2 builds charts through layers using geom_ functions.

EXAMPLE:

```
install.packages("ggplot2")
library(ggplot2)
a<-iris
print(a)
c=plot(a$Sepal.Length~a$Petal.Length,
ylab = "sepal.Length”, xlab="Petal.Length",
    main="Sepal.Length vs Petal.Length", col="blue", pch=16)
print(c)
```

Output:



3.readr: The ‘readr’ package helps to read various forms of data into R. The main goal of '*readr*' is to provide a fast and friendly way to read rectangular data (like 'csv', 'tsv', and 'fwf').

- A csv (comma-separated values) file is a text file that has a specific format which allows data to be saved in a table structured format.
- A tsv (tab-separated values) file is a simple text format for storing data in a tabular structure, e.g., database table.
- A fwf (Fixed-width format) file is a text file where data is arranged in columns, but instead of those columns being delimited by a certain character (as they are in CSV) every row is the exact same length.

  EXAMPLE:

  read_csv(readr_example("mtcars.csv"))

  OUTPUT:

  Mpg=col_double(),

  Cyl=col_double(),

  Wt=col_double(),

  Gear=col_double(),

  Hp=col_double())

4.tidyr:

This package can make the data look 'tidy'(nice). It has 4 major functions to accomplish this task such as

1) gather() – it 'gathers' multiple columns. Then, it converts them into key:value pairs. This function will transform wide from of data to long form. You can use it as in alternative to 'melt()' in reshape2 package.

2)spread() – It does reverse of gather. It takes a key:value pair and converts it into separate columns.

3)separate() – It splits a column into multiple columns.

4)unite() – It does reverse of separate. It unites multiple columns into single column

Program:

```
Instrall.packages("tidyr")
Library(tidyr)
Names<-c('A','B','C','D','E','A','B')
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')

tdata<-data.frame(names,age,weight,class)

Output:

	names	age	weight	Class
1	A	21	55	Maths
2	B	20	49	Science
3	C	25	76	Social
4	D	29	71	Physics
5	E	33	65	Biology
6	A	32	44	Economics
7	B	38	34	Accounts

UNIT-3

1. Explain the mutable and immutable objects of python programming language.

ANS:

Mutable and immutable objects are handled differently in python. Immutable objects are quicker to access and are expensive to change because it involves the creation of a copy. Whereas mutable objects are easy to change. Use of mutable objects is recommended when there is a need to change the size or content of the object.

MUTABLE	IMMUTABLE
Object can be changed after created.	Object cannot be changed once created.
list dictionary set	int, float, long, complex string tuple bool

MUTABLE

list: Lists are collections of items (strings, integers, or even other lists).

- Lists are enclosed in [ ]

list = [10, -20 , 15.5, 'vijay', "mary"]

dictionary: A dictionary is a collection which is unordered, changeable and indexed.

- In Python dictionaries are written with curly brackets, and they have keys and values.

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
   }
```

set: A set object is an unordered collection of distinct hashable objects.

```
thisset = {"apple", "banana", "cherry"}
```

IMMUTABLE

int: An integer is a whole number, negative, positive or zero.

```
>>> a = 5

>>> type(a)

<class 'int'>
```

float: Floats are decimals, positive, negative and zero.

```
>>> c = 6.2

>>> type(c)

<class 'float'>
```

complex: A complex number is defined in Python using a real component + an imaginary component j.

```
>>> comp = 4 + 2j

>>> type(comp)

<class 'complex'>
```

string: In python, str represents string datatype.

- A string is represented by a group of characters.
- Strings are enclosed in single quotes or double quotes.

```
Str = “welcome”

Str = ‘welcome’
```

tuple: A tuple is a collection which is ordered and unchangeable.

- In Python tuples are written with round brackets ( ).

thistuple = ("apple", "banana", "cherry")

bool: Python boolean type is one of the built-in data types provided by Python, which are defined by the True or False keywords.

b =False

type(b)

<class 'bool'>

2. Explain the commands of Numpy library related to multidimensional arrays.

ANS: The multidimensional list can be easily converted to Numpy arrays as below:

```
import numpy as nmp

arr = nmp.array( [ [1, 0], [6, 4] ] )

print(arr)
```

Creating a Numpy Array

```
import numpy as nmp
X = nmp.array( [ [ 1, 6, 7], [ 5, 9, 2] ] )
print(X)                                                    #Array of
integers
```

output:

[[1 6 7] [5 9 2]]

Accessing Numpy Matrix Elements, Rows and Columns

Each element of the Numpy array can be accessed in the same way as of Multidimensional List,

Example:

```
import numpy as nmp
X = nmp.array( [ [ 1, 6, 7],
[ 5, 9, 2],
[ 3, 8, 4] ] )
print(X[1][2]) # element at the given index i.e. 2
```

Output: 2

to declare a random array:

```
x3=np.random.random((3,3))
```

to print an array:

```
print(x3)
```

```
[[0.34652592 0.92320518 0.38518527]
[0.68624424 0.24758342 0.47053378]
[0.26924799 0.23444484 0.75890992]]
```

to print shape of array:

```
print(x3.shape)
```

```
(3, 3)
```

to print dimensions of array:

```
print(x3.ndim)
```

```
2
```

to print datatype of array:

```
print(x3.dtype)
```

```
float64
```

to print size of an array:

```
print(x3.size)
```

```
9
```

to print length of array:

```
len(x3)
```

```
3
```

3. **Explain how to read and write CSV and EXCEL files using Pandas library.**

ANS: **Reading CSV file with íead_csv() in Pandas**

For reading CSV file, we use pandas **read_csv** function. This function basically helps in fetching the contents of CSV file into a dataframe. Now, let us look at the syntax of this pandas function.

Syntax

pandas.read_csv(filepath_or_buffer)

```
df = pd.read_csv('test.csv')

df
```

Unnamed: 0	Quantity Grown (in Kg)	State in which grown	
0	Apple	150	Jammu & Kashmir
1	Mango	225	Madhya Pradesh
2	Banana	450	Karnataka
3	Watermelon	75	Maharashtra
4	Pineapple	175	Uttar Pradesh

Wíiting CSV file in Pandas : to_csv()

Using pandas to_csv function we can store the data in CSV file format. Now we will learn about its syntax and also look at the examples.

Syntax

pandas.to_csv(filepath_or_buffer)

```
df = pd.DataFrame({'fruits': ['Mango',
'Guava','Pineapple','Apple','Watermelon','Strawberry','Orange'],

'states_grown': ['Madhya Pradesh', 'Maharashtra', 'Karnataka', 'Jammu & Kashmir',
'Gujarat', 'Himachal Pradesh', 'Kerala'],

'quantity (in Kgs)': ['150', '200', '390','545','273','250','175']})

Df
```

fruits	states_grown	quantity (in Kgs)	
0	Mango	Madhya Pradesh	150
1	Guava	Maharashtra	200
2	Pineapple	Karnataka	390
3	Apple	Jammu & Kashmir	545
4	Watermelon	Gujarat	273
5	Strawberry	Himachal Pradesh	250
6	Orange	Kerala	175

Reading Excel file in Pandas : íead_excel()

By using the pandas **read_excel()** function, we can fetch the excel file into pandas dataframe. read_excel function gives the liberty to fetch data from a single sheet or multiple excel sheets.

Syntax

pandas.read_excel(io,sheet_name=0,kwds)

```
df = pd.read_excel('test.xlsx', index_col=0)

df
```

	Subjects Liked	Number of Hours studied
Student Name		
Rohit	Mathematics	12
Virat	English	15
Shikhar	Hindi	3
Shreyas	Physics	9
Manish	Chemistry	7

Wíiting to Excel file in Pandas: to_excel()

For writing the object to excel file, to_excel() function of pandas is used. Let's look at the syntax of this function.

Syntax

dataframe.to_excel(excel_writer,sheet_name='Sheet1')

```
df.to_excel("output.xlsx",sheet_name='Sheet2')

df1 = pd.DataFrame([['75', '97'], ['25', '99']],

                           index=['Rakesh', 'Suresh'],

                            columns=['Marks in English', 'Marks in Mathematics'])

df1
```

	Marks in English	Marks in Mathematics
Rakesh	75	97

	Marks in English	Marks in Mathematics
Suresh	25	99

3) Explain the looping and conditional statements?

ANS: In programming, loops are a sequence of instructions that does a specific set of instructions or tasks based on some conditions and continue the tasks until it reaches certain conditions.

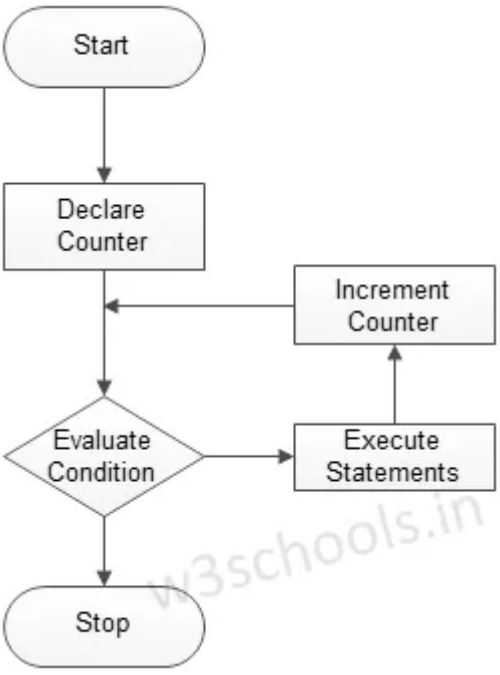
Types of Loops in Python:

Loop	Description
for Loop	This is traditionally used when programmers had a piece of code and wanted to repeat that 'n' number of times.
while Loop	The loop gets repeated until the specific Boolean condition is met.

For loop:

The graphical representation of the logic behind for looping is shown below:

Figure - for loop Flowchart:



Syntax:

```
for iterating_var in sequence:

   #execute your code
```

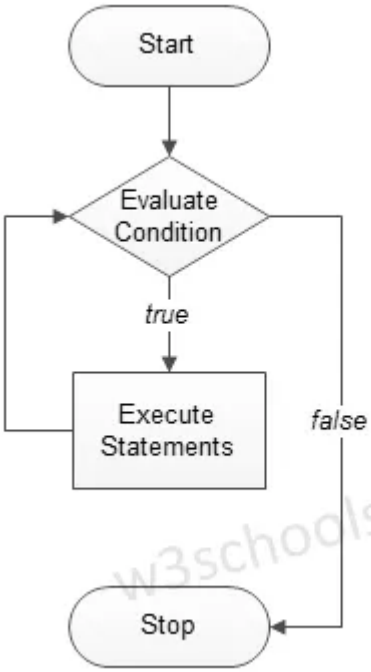
Example:

```
for x in range (0,3) :
     print ('Loop execution %d' % (x))
 Output:
Loop execution 0

Loop execution 1

Loop execution 2
```

While loop:

The graphical representation of the logic behind while looping is shown below:

Figure - while loop Flowchart:



Syntax:

```
while expression:

   #execute your code
```

Example:

```
#initialize count variable to 1
count =1

while count < 6 :
    print (count)
    count+=1
#the above line means count = count + 1
Output:
1

2

3
```

4

5

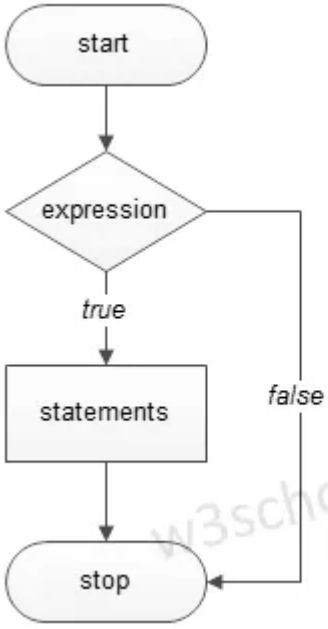
Conditional statements: It is the prediction of conditions that occur while executing a program to specify actions. Multiple expressions get evaluated with an outcome of either TRUE or FALSE. These are logical decisions, and Python also provides decision-making statements that to make decisions within a program for an application based on the user requirement.

Python Conditional Statements

Statement	Description
if Statements	It consists of a Boolean expression which results are either TRUE or FALSE, followe more statements.
if else Statements	It also contains a Boolean expression. The if the statement is followed by an optiona & if the expression results in FALSE, then else statement gets executed. It is also ca execution in which there are two possibilities of the condition determined in which an will get executed.

If statements:

The decision-making structures can be recognized and understood using flowcharts.



Syntax:

```
if expression:
   #execute your code
```

Example:

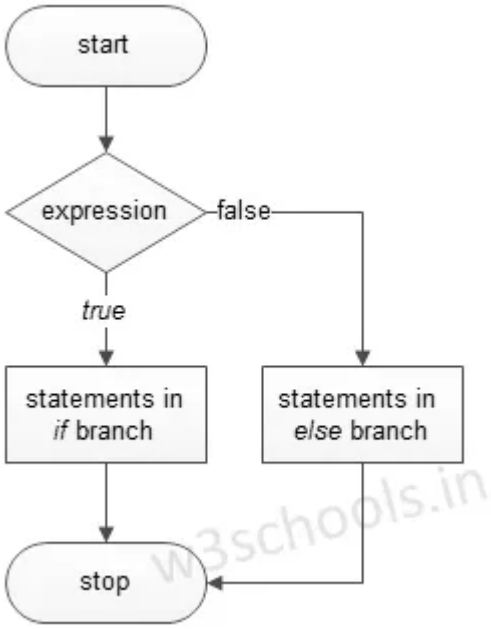
```
a = 15
```

```
if a > 10:
   print("a is greater")
```

Output:

```
a is greater
```

if else statements:



Syntax:’

```
if expression:
   #execute your code
else:
   #execute your code
```

Example:

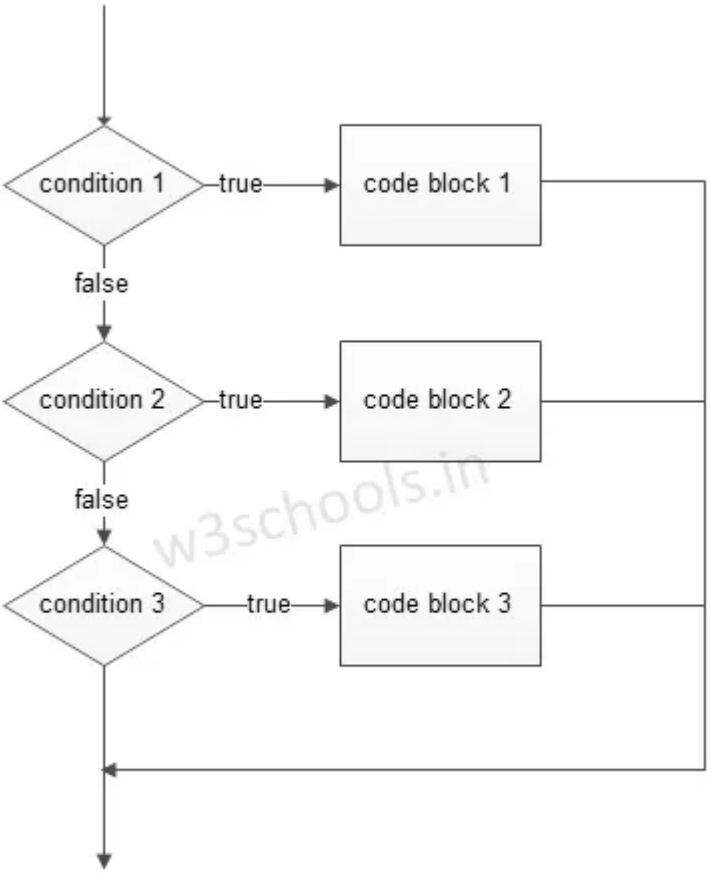
```
a = 15
b = 20

if a > b:
   print("a is greater")
else:
   print("b is greater")
```

Output:

```
b is greater
```

elif statements:

elif - is a keyword used in Python replacement of else if to place another condition in the program. This is called chained conditional.



Syntax:

```
if expression:
   #execute your code
elif expression:
   #execute your code
else:
   #execute your code
```

Example:

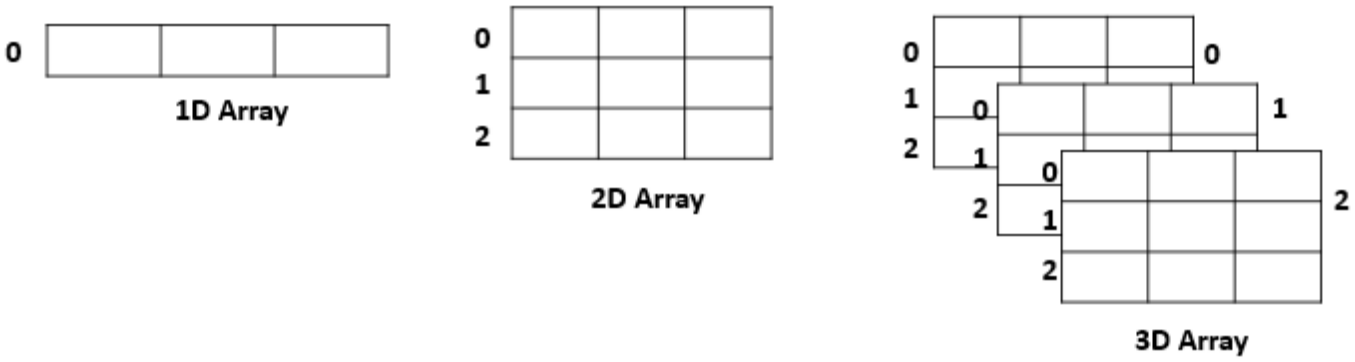
```
a = 15
b = 15

if a > b:
   print("a is greater")
elif a == b:
   print("both are equal")
else:
   print("b is greater")
```

Output:both are equal

4) What is Numpy? Explain about creating ndarrray with example?

A: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an opensource project and you can use it freely. NumPy stands for Numerical Python**.**

Ndarray is one of the most important classes in the NumPy python library. It is basically a multidimensional or n-dimensional array of fixed size with homogeneous elements(i.e. data type of all the elements in the array is the same)



In Numpy, the number of dimensions of the array is given by Rank. In the above example, the ranks of the array of 1D, 2D, and 3D arrays are 1, 2 and 3 respectively.

Syntax:

```
np.ndarray()
```

An array can be created using the following functions :

- **np.ndarray(shape, type):** Creates an array of the given shape with random numbers.
- **np.array(array_object):** Creates an array of the given shape from the list or tuple.
- **np.zeros(shape):** Creates an array of the given shape with all zeros.
- **np.ones(shape):** Creates an array of the given shape with all ones.
- **np.full(shape,array_object, dtype):** Creates an array of the given shape with complex numbers.
- **np.arange(range):** Creates an array with the specified range.

Example:

```
import numpy as np
#creating an array to understand its attributes
A = np.array([[1,2,3],[1,2,3],[1,2,3]])
```

```
print("Array A is:\n",A)
#type of array
print("Type:", type(A))
#Shape of array
print("Shape:", A.shape)
#no. of dimensions
print("Rank:", A.ndim)
#size of array
print("Size:", A.size)
#type of each element in the array
print("Element type:", A.dtype)
```

Output:

```
Array A is:
 [[1 2 3]
 [1 2 3]
 [1 2 3]]
Type: <class 'numpy.ndarray'>
Shape: (3, 3)
Rank: 2
Size: 9
Element type: int32
```

5) Explain about data types, array attributes in Numpy?

ANS:

1	**bool_** Boolean (True or False) stored as a byte
2	**int_** Default integer type (same as C long; normally either int64 or int32)
3	**intc** Identical to C int (normally int32 or int64)
4	**intp** Integer used for indexing (same as C ssize_t; normally either int32 or int64)
5	**int8** Byte (-128 to 127)

a) ndarray.ndim :

- This attribute will return minimum dimensions of the array created, for e.g. :

```
import numpy as np
a = np.array([[10,20,30],[40,50,60]])
a.ndim
```

Copy

Output :

```
2
```

Copy

b) ndarray.shape :

- This attribute will return a tuple that contains the dimensions of the array, for e.g., in above code we will get the output as (2,3) ie 2 rows and 3 columns.

```
#Initializing numpy array
import numpy as np
arr = np.arr([[20,40,60], [10,30,50]])
print(arr)

#To find the shape of array (product of rows and columns)
arr.shape
```

Copy

Output :

```
[[20,40,60],
 [10,30,50]]

(2,3)
```

Copy

c) ndarray.size :

- This attribute will return the size of array ie total number of elements present in it.

```
#Initializing numpy array
import numpy as np
```

```
arr = np.arr([[20,40,60], [10,30,50]])
print(arr)

#To find the size of array
print(arr.size)
```

Copy

Output :

```
[[20,40,60],
 [10,30,50]]

6
```

Copy

d) ndarray.itemsize :

- It will return the length of eaach element in the array irrespective of data type, for e.g., :

```
import numpy as np
arr_1 = np.array([1.5, 2.1, 3.2, 4.5], dtype = np.float16)
print(arr_1.itmesize)
```

Copy

- **Note :** We can change the dtype of the object.

Output : Here the size of float16 is of 2 bytes.

2

6) Explain in detail about indexing , slicing?

A:

.Indexing in Python means referring to an element of an iterable by its position within the iterable

- Each character can be accessed using their index number.
- To access characters in a string we have two ways:
  - **Positive index number**
  - **Negative index number**

Positive indexing example in Python:

In **Python Positive indexing**, we pass a positive index that we want to access in square brackets. The index number starts from **0** which denotes the **first character** of a string.

Example:

my_str = "Python Guides"

print(my_str[0])

Output:

P

Negative indexing example in Python:

In **negative indexing in Python**, we pass the negative index which we want to access in square brackets. Here, the index number starts from index number **-1** which denotes the **last character** of a string.

Example:

```
my_str = "Python Guides"
print(my_str[-1])
```

OUTPUT:

S

Slicing in python is used for accessing parts of a sequence. The slice object is used to slice a given sequence or any object. We use slicing when we require a part of a string and not the complete string.

Syntax:

```
string[start : end : step]
```

Example:

```
my_str = "Python Guides"
print(my_str[-1 : -10 : -2])
```

Output:

sdu o

Python get substring using slice object:

To get the substring using slice object in python, we will use **"slice(5)"**. Here, start **"0"** and the end **"5"**, so it takes value one less than the end value.

Example:

```
my_str = 'Guides'
s_obj = slice(5)
print(my_str[s_obj])
```

OUTPUT:

Guide

Python get substring using negative index:

To **get substring using negative index in python**, we will use **"slice(-1, -4, -1)"**. Here, the start is **"-1"**, the end **"-4"**, and the step is negative **"-1"**.

Example:

```
my_str = 'Guides'
s_obj = slice(-1, -4, -1)
print(my_str[s_obj])
```

OUTPUT:

sed

DATA VISUALIZATION IMPORTANT QUESTIONS FOR MID-2

UNIT-4

1) **Explain the following functions in pandas ----------10M**

a) adding columns
b) Aggregations
c) handling missing data
d) groupby and merging

Ans: a) **Adding columns:** We can use a Python dictionary to add a new column in pandas DataFrame. Use an existing column as the key values and their respective values will be the values for new column.
Code:

```
import pandas as pd
data = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Height': [5.1, 6.2, 5.1, 5.2],
        'Qualification': ['Msc', 'MA', 'Msc', 'Msc']}
address = {'Delhi': 'Jai', 'Bangalore': 'Princi',
        'Patna': 'Gaurav', 'Chennai': 'Anuj'}
df = pd.DataFrame(data)
df['Address'] = address
df
```

Output:

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Patna
3	Anuj	5.2	Msc	Chennai

b) Aggregations: Pandas has a number of aggregating functions that reduce the dimension of the grouped object. In this post will examples of using 13 aggregating function after performing Pandas groupby operation.
Here are the 13 aggregating functions available in Pandas and quick summary of what it does.

1) mean(): Compute mean of groups
2) sum(): Compute sum of group values
3) size(): Compute group sizes
4) count(): Compute count of group
5) std(): Standard deviation of groups
6) var(): Compute variance of groups
7) sem(): Standard error of the mean of groups
8) describe(): Generates descriptive statistics
9) first(): Compute first of group values
10) last(): Compute last of group values
11) nth() : Take nth value, or a subset if n is a list
12) min(): Compute min of group values
13) max(): Compute max of group values

Code:

```
import pandas as pd
data = {
  "x": [50, 40, 30],
  "y": [300, 1112, 42]
}
df = pd.DataFrame(data)

print(df.aggregate(["mean"]))
print(df.aggregate(["sum"]))
print(df.aggregate(["size"]))
print(df.aggregate(["count"]))
print(df.aggregate(["std"]))
print(df.aggregate(["var"]))
print(df.aggregate(["sem"]))
print(df.aggregate(["describe"]))
print(df.aggregate(["min"]))
print(df.aggregate(["max"]))
```

Output:

```
          x         y
mean  40.0  484.666667
       x    y
sum  120  1454
     x  y
size  3  3
        x  y
count  3  3
     x         y
std  10.0  558.391738
      x           y
var  100.0  311801.333333
        x         y
sem  5.773503  322.38762
         x          y
    describe    describe
count     3.0    3.000000
mean     40.0  484.666667
std      10.0  558.391738
min      30.0   42.000000
25%      35.0  171.000000
50%      40.0  300.000000
75%      45.0  706.000000
max      50.0  1112.000000
    x  y
min  30  42
    x    y
max  50  1112
```

c)handling missing data: Handling missing values is an essential part of data cleaning and preparation process because almost all data in real life comes with some missing values.

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame({'column_a':[1,2,4,4,np.nan,np.nan,6],
                   'column_b':[1.2,1.4,np.nan,6.2,None,1.1,4.3],
                   'column_c':['a','?','c','d','--',np.nan,'d'],
                   'column_d':[True,True,np.nan,None,False,True,False]})
```

```
df.replace({"?": np.nan, "--": np.nan}, inplace=True)
```

```
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1
1	2.0	1.4	NaN	True	2
2	4.0	NaN	c	NaN	NaN
3	4.0	6.2	d	None	4
4	NaN	NaN	NaN	False	NaN
5	NaN	1.1	NaN	True	5
6	6.0	4.3	d	False	NaN

d) Groupby: The groupby() method allows you to group your data and execute functions on these groups.
Syntax
dataframe.groupby( )

Code:

```
import pandas as pd
data = {
  'co2': [95, 90, 99, 104, 105, 94, 99, 104],
  'model': ['zen', 'Eon', 'Baleno', 'venue', 'Breeza', 'grand nios', 's-cross', 'i20'],
  'car': ['Suzuki', 'Hyundai', 'Suzuki', 'Hyundai', 'Suzuki', 'Hyundai', 'Suzuki', 'Hyundai']
}
df = pd.DataFrame(data)
print(df.groupby(["car"]).mean())
```

Output:

```
   car      co2
Hyundai  98.0
Suzuki   99.5
```

Merge: The merge() method updates the content of two DataFrame by merging them together, using the specified method(s).

Syntax: *dataframe*.merge( )
Example:

```
import pandas as pd
data1 = {"name": ["Sally", "Mary", "John"],
         "age": [50, 40, 30]}
data2 = {"name": ["Sally", "Peter", "Micky"],
         "age": [77, 44, 22]}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
newdf = df1.merge(df2, how='right')
print(newdf)
```

output:

```
   name  age
 0  Sally   77
```

1 Peter 44
2 Micky 22

2) How to read and write CSV and Excel files with example dataset.----------10M

Reading CSV file with read_csv() in Pandas

For reading CSV file, we use pandas read_csv function. This function basically helps in fetching the contents of CSV file into a dataframe. Now, let us look at the syntax of this pandas function.

Syntax

pandas.read_csv(filepath_or_buffer)

Code:

```
df = pd.read_csv('test.csv')
df
```

Output:

Unnamed:0	Quantity Grown (in kg)	State in which grown	
0	**Apple**	**150**	**Jammu & kashmir**
1	**Mango**	**225**	**Madhya Pradesh**
2	**Banana**	**450**	**Karnataka**
3	**Watermelon**	**75**	**Maharashtra**
4	**Pineapple**	**175**	**Uttarpradesh**

Writing CSV file in Pandas : to_csv()

Using pandas to_csv function we can store the data in CSV file format. Now we will learn about its syntax and also look at the examples.

Syntax

pandas.to_csv(filepath_or_buffer)

Code:

```
Df=pd.DataFRame({'fruits': ['Mango',
'guava','pineapple','Apple','WaterMelon','Strawberry','Orange'],

'States_grown' :['Madhya Pradesh','Maharashtra','Karnataka','Jammu &Kashmir','Gujarat','Himachal Pradesh','Kerala],

'quantity (in kgs)' : ['150','200','390','545','273','250','175']})

Df
```

Output:

Fruits	States_grown	Quantity(in kgs)	
0	Mango	Madhya Pradesh	150
1	Guava	Maharashtra	200
2	Pineapple	Karnataka	390
3	Apple	Jammu & Kashmir	545
4	Watermelon	Gujarat	273
5	Strawberry	Himachal Pradesh	250
6	Orange	Kerala	175

Reading Excel file in Pandas : read_excel()

By using the pandas read_excel() function, we can fetch the excel file into pandas dataframe. read_excel function gives the liberty to fetch data from a single sheet or multiple excel sheets.

Syntax

pandas.read_excel(io,sheet_name=0,kwds)

Code:

```
df = pd.read_excel('test.xlsx', index_col=0)
```

df

Output:

Student Name	Subjects Liked	Number of Hours Studied
Rohit	Mathematics	12
Virat	English	15
Shikhar	Hindi	3
Shreyas	Physics	9
Manish	Chemistry	7

Writing to Excel file in Pandas: to_excel()

For writing the object to excel file, to_excel() function of pandas is used. Let's look at the syntax of this function.

Syntax

dataframe.to_excel(excel_writer,sheet_name='Sheet1')

Code:

```
df.to_excel("output.xlsx",sheet_name='sheet2)
df1 = pd.DataFrame([{'75','97'], ['25','99']],
                                index = ['Rakesh', 'Suresh'],
                                columns = ['Marks in English','Marks in Mathematics])
```

df1

Output:

	Marks in English	Marks in Mathematics
Rakesh	75	97
Suresh	25	99

3) Explain how to summarise and group the data using pandas library with an example.

Summarize means to combine the data from different groups. It is also called as aggregation.
different functions used to summarize data contained in a pandas dataframe. Such as

Summarising Groups in the **DataFrame** There's further power put into your hands by mastering the **Pandas** "**groupby** ()" functionality. **Groupby** essentially splits the **data** into different **groups** depending on a variable of your choice. For example, the expression **data**.**groupby** ('month') will split our current **DataFrame** by month.

- min(): returns minimum value.
- max(): returns maximum value.
- mean(): returns mean value.
- median() : returns median value.
- describe():returns summary of data
- sum(): returns sum of data.

  count(): returns count of data.

```
import pandas as pd

data = {
  'co2': [95, 90, 99, 104, 105, 94, 99, 104],
  'model': ['Citigo', 'Fabia', 'Fiesta', 'Rapid', 'Focus', 'Mondeo', 'Octavia', 'B-Max'],
  'car': ['Skoda', 'Skoda', 'Ford', 'Skoda', 'Ford', 'Ford', 'Skoda', 'Ford']
}
```

```
df = pd.DataFrame(data)
print(df)
print("summarise and grouped data")
print(df.groupby(["car"]).mean())
```

output:

```
co2    model    car
0   95   Citigo  Skoda
1   90    Fabia  Skoda
2   99   Fiesta   Ford
3  104    Rapid  Skoda
4  105    Focus   Ford
5   94   Mondeo   Ford
6   99  Octavia  Skoda
7  104    B-Max   Ford
```

Summarise and grouped data

```
car        co2
Ford   100.5
Skoda   97.0
```

4) What is Matplotlib and its architecture? Explain about working axes and working with legends?

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy, the numerical mathematics extension of Python.

Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks. Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB

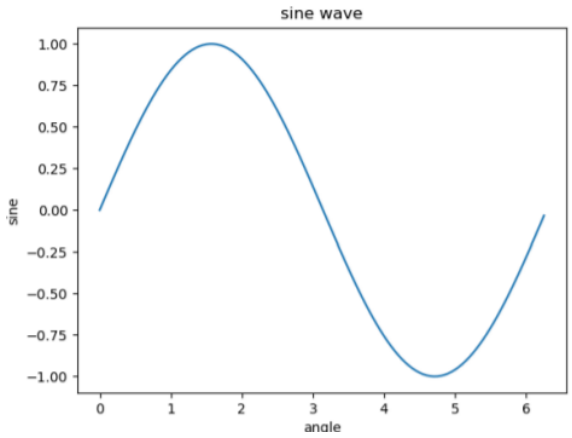
The Matplotlib architecture is composed of three main layers:

- **Backend Layer** —The backend layer handles all the heavy works via communicating to the toolkits like wxPython or drawing languages like PostScript in your machine. It is the most complex layer of the Matplotlib library**.**
- **Artist Layer** — Allows full control and fine-tuning of the Matplotlibfigure — the top-level container for all plot elements. This layer is comprised of one main object, Artist, that uses the Renderer to draw on the canvas. It allows you to do more customization compare to the **Scripting layer** and is more convenient for advanced plots.
- **Scripting Layer** — The lightest scripting interface among the three layers, designed to make Matplotlib work like MATLAB script.

Axis Functions

Function	Description
Axes	Add axes to the figure.
Text	Add text to the axes.
Title	Set a title of the current axes.
Xlabel	Set the x axis label of the current axis.
Xlim	Get or set the x limits of the current axes.
Xscale	Set the scaling of the x-axis.
Xticks	Get or set the x-limits of the current tick locations and labels.
Ylabel	Set the y axis label of the current axis.
Ylim	Get or set the y-limits of the current axes.
Yscale	Set the scaling of the y-axis.
Yticks	Get or set the y-limits of the current tick locations and labels.

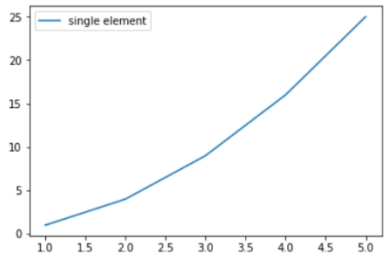
```
from matplotlib import pyplot as plt

import numpy as np

import math #needed for definition of pi

x=np.arange(0, math.pi*2, 0.05)

y=np.sin(x)

plt.plot(x,y)

plt.xlabel("angle")

plt.ylabel("sine")

plt.title('sine wave')

plt.show()
```



Legend(): A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called legend() which is used to Place a legend on the axes.

Ex:

```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.plot(x, y)
plt.legend(['single element'])
plt.show()
```



5) Briefly explain about following Basic plots using matplotlib ----------12M

a. **Area Plots**
b. **Line plot.**
c. **Bar Charts**
d. **Jitter plot.**
e. **Histograms**

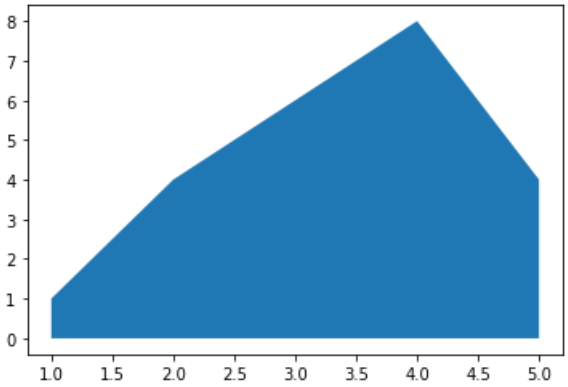
a. Area plots:

There are 2 main functions to draw a **basic area chart** using matplotlib: fill_between() and stackplot() functions. I advise the fill_between() function which allows easier customisation. The stackplot() function also works, but it is more adapted for stacked area charts. The inputs are 2 numerical variables(x and y values) for both functions.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
x=range(1,6)
y=[1,4,6,8,4]
plt.fill_between(x, y)
plt.show()
```

Output:

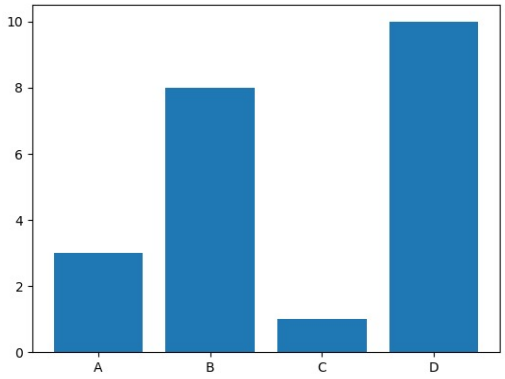


b. Bar Charts:

The bar() function takes arguments that describes the layout of the bars.
The categories and their values represented by the first and second argument as arrays.
Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

Output:

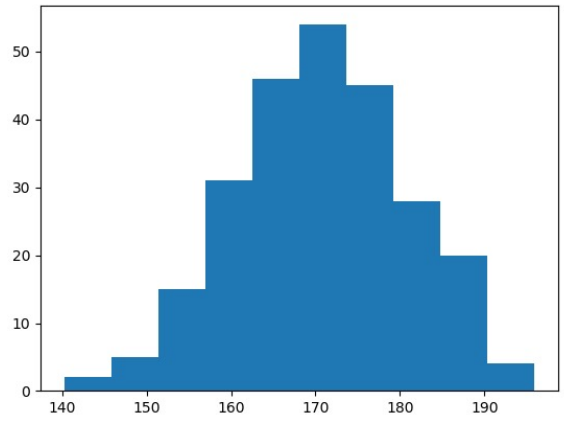


c. Histograms:
A histogram is a graph showing frequency distributions.
It is a graph showing the number of observations within each given interval.
The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument. The hist() function will read the array and produce a histogram.

Code:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

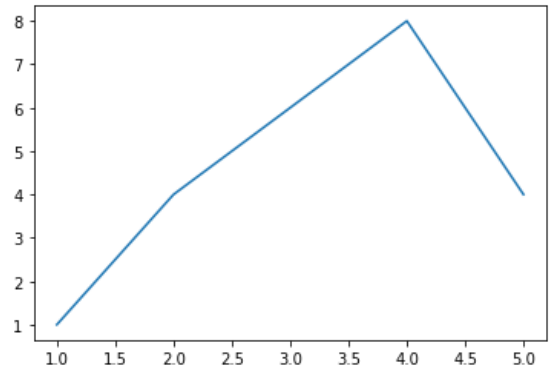
Output:



Line plot: First import Matplotlib.pyplot library for plotting functions. Also, import the Numpy library as per requirement. Then define data values x and y.
Ex:

```
import numpy as np
import matplotlib.pyplot as plt
x=range(1,6)
y=[1,4,6,8,4]
plt.plot(x, y)
plt.show()
```

output:



4) Explain the following Specialized Visualization plots using Matplotlib ----------10M

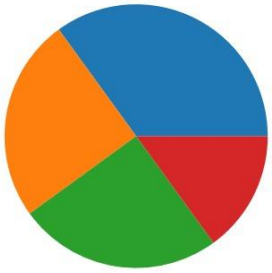
a. **Pie Charts**
b. **Box Plots**
c. **Scatter Plots**

Ans : a) Pie Charts : A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges**.**

Syntax:
matplotlib.pyplot.pie( )

code:
```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

Output:



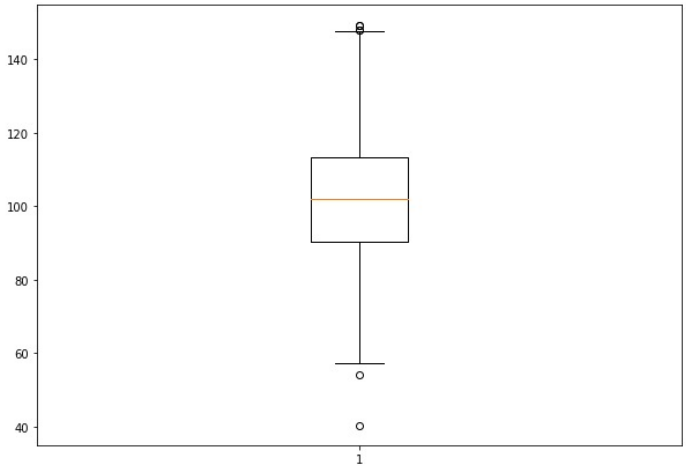
b) Box Plots: A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median.

Syntax:
matplotlib.pyplot.boxplot( )

Code:

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(10)
data = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
plt.boxplot(data)
plt.show()
```

Output:



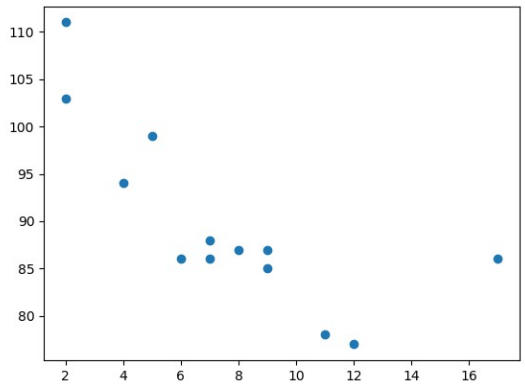
c) Scatter Plots: The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

Syntax: matplotlib.pyplot.scatter()

code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

Output:

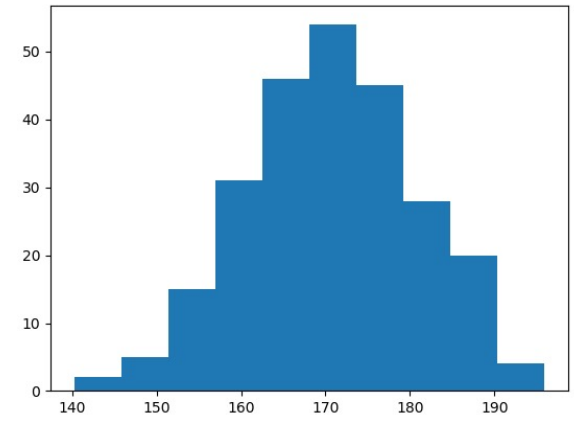


c. Histograms:
A histogram is a graph showing frequency distributions.
It is a graph showing the number of observations within each given interval.
The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument. The hist() function will read the array and produce a histogram.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

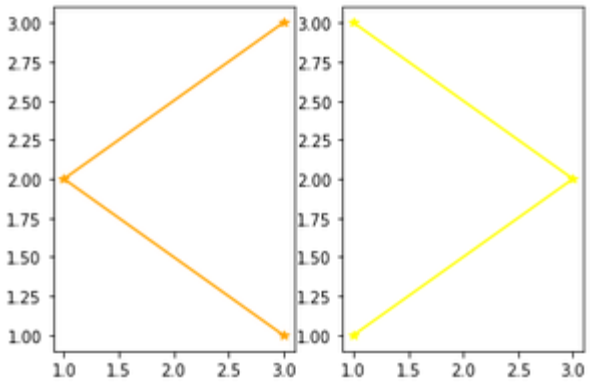
Output:



Subplot: subplot() function adds subplot to a current figure at the specified grid position. It is similar to the subplots() function however unlike subplots() it adds one subplot at a time.
Ex:

```
import matplotlib.pyplot as plt
x = [3, 1, 3]
y = [3, 2, 1]
z = [1, 3, 1]
plt.figure()
plt.subplot(121)
plt.plot(x, y, color="orange", marker="*")
plt.subplot(122)
plt.plot(z, y, color="yellow", marker="*")
```

output:



UNIT-5

1) Define seaborn? Explain functionalities and usages of seaborn with matplotlib --------10M

Ans: Seaborn: Seaborn is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive**.** Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data. One has to be familiar with Numpy and Matplotlib and Pandas to learn about Seaborn

Seaborn offers the functionalities:

1) Dataset oriented API to determine the relationship between variables.

2) Automatic estimation and plotting of linear regression plots.

3) It supports high level abstractions for multi-plot grids.

4) visualizing univariate and bivariate distribution.

These are only some of the functionalities offered by seaborn, there are many more of them, and we can explore all of them.

Code:

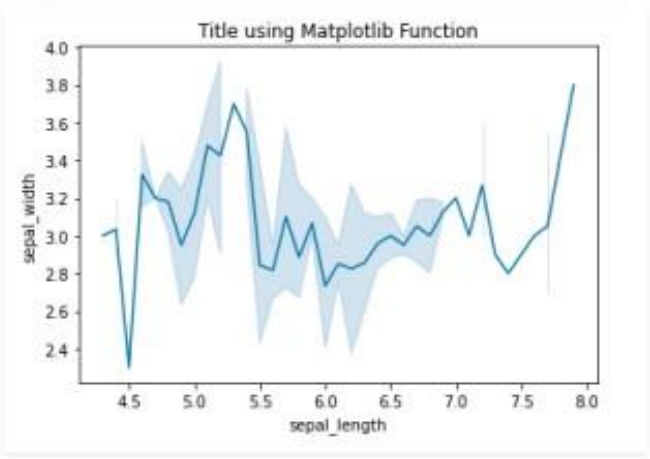
```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.lineplot(x="sepal_length", y="sepal_width", data=data)
plt.title('Title using Matplotlib Function')
```

plt.show()

Output:

9/1/2021

Output:



2) Explain the following Categorical Plots methods----------10M

a) barplot ()
b) countplot ()
c) boxplot ()
d) stripplot ()

Ans: a) barplot():A barplot is basically used to aggregate the categorical data according to some methods and by default its the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x axis and a numerical column for the y axis and we see that it creates a plot taking a mean per categorical column. It can be created using the barplot() method.

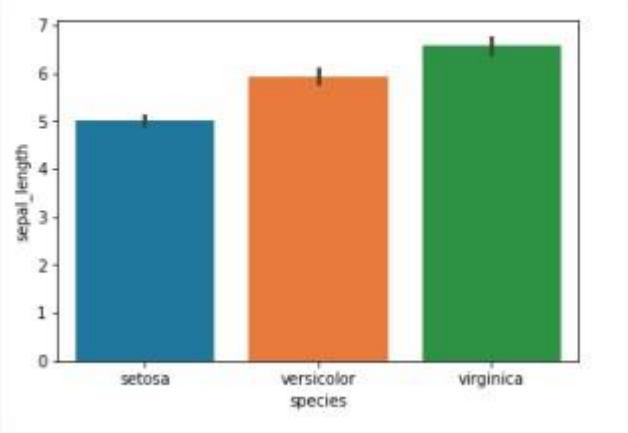
Syntax:

barplot([x, y, hue, data, order, hue_order, …])

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.barplot(x='species', y='sepal_length', data=data)
plt.show()
```

Output:



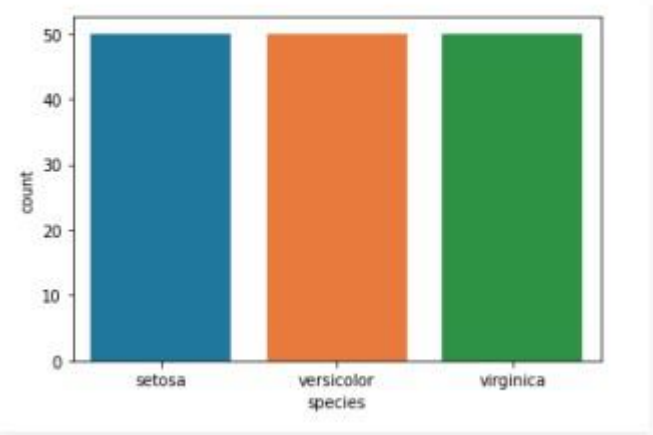
b) countplot ():
A countplot basically counts the categories and returns a count of their occurrences. It is one of the most simple plots provided by the seaborn library. It can be created using the countplot() method.

Syntax:
countplot([x, y, hue, data, order, …])

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.countplot(x='species', data=data)
plt.show()
```

Output:



c) boxplot ():
A boxplot is sometimes known as the box and whisker plot.It shows the distribution of the quantitative data that represents the comparisons between variables. boxplot shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution i.e. the dots indicating the presence of outliers. It is created using the boxplot() method.

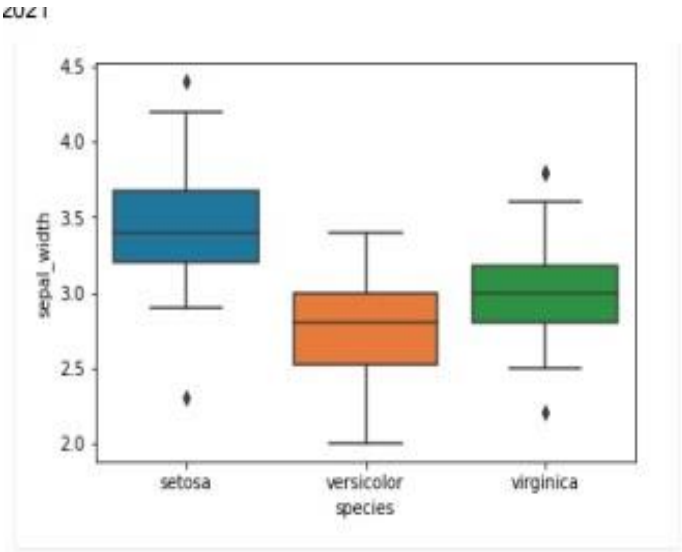
Syntax:

boxplot([x, y, hue, data, order, hue_order, …])

Code:

```
import seaborn as sns

import matplotlib.pyplot as plt

data = sns.load_dataset("iris")

sns.boxplot(x='species', y='sepal_width', data=data)

plt.show()
```

Output:



d) stripplot () :

It basically creates a scatter plot based on the category. It is created using the stripplot() method.

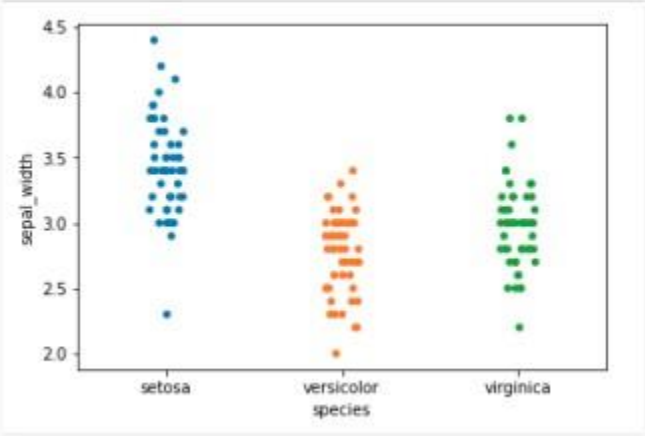
Syntax:
stripplot([x, y, hue, data, order, …])

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.stripplot(x='species', y='sepal_width', data=data)
plt.show()
```

Output:

Output:



3) Briefly explain the customizing seaborn plots?

ANS: **Customizing Seaborn Plots:**
Seaborn comes with some customized themes and a high-level interface for customizing the looks of the graphs.Consider the above example where the default of the Seaborn is used. It still looks nice and pretty but we can customize the graph according to our own needs. So let's see the styling of plots in detail.

Changing Figure Aesthetic:

set_style() method is used to set the aesthetic of the plot. It means it affects things like the color of the axes, whether the grid is active or not, or other aesthetic elements. There are five themes available in Seaborn.

1) darkgrid
2) whitegrid
3) dark
4) white
5) ticks

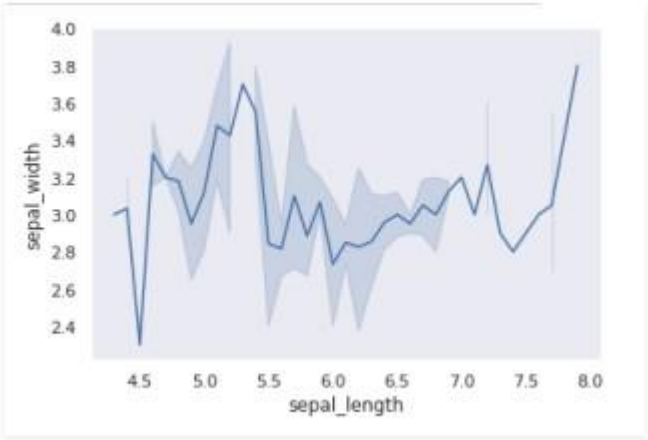
Syntax:

set_style(style=None, rc=None)

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
sns.lineplot(x="sepal_length", y="sepal_width", data=data)
sns.set_style("dark")
plt.show()
```

Output:

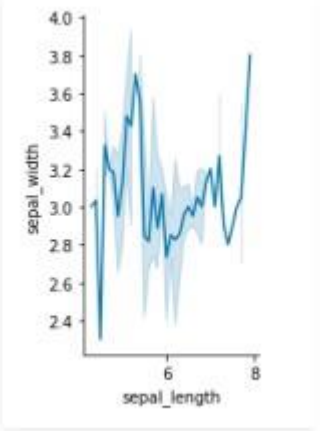


Changing the figure Size:
The figure size can be changed using the figure() method of Matplotlib. figure() method creates a new figure of the specified size passed in the **figsize** parameter.

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
plt.figure(figsize = (2, 4))
sns.lineplot(x="sepal_length", y="sepal_width", data=data)
sns.despine()
plt.show()
```

Output:



Scaling the plots:
It can be done using the set_context() method. It allows us to override default parameters. This affects things like the size of the labels, lines, and other elements of the plot, but not the overall style. The base context is “notebook”, and the other contexts are “paper”, “talk”, and “poster”. font_scale sets the font size.

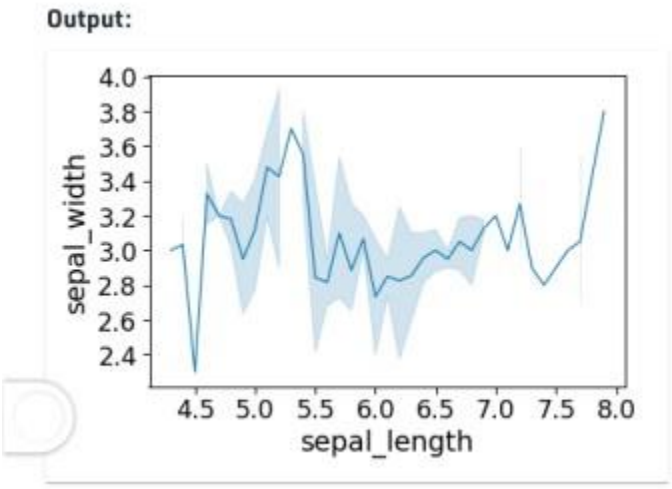
Syntax:
set_context(context=None, font_scale=1, rc=None)

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
```

```
sns.lineplot(x="sepal_length", y="sepal_width", data=data)
sns.set_context("paper")
plt.show()
```

Output:



Color palette:

Colormaps are used to visualize plots effectively and easily. One might use different sorts of colormaps for different kinds of plots. color_palette() method is used to give colors to the plot. Another function palplot() is used to deal with the color palettes and plots the color palette as a horizontal array.

Code:

```
import seaborn as sns

import matplotlib.pyplot as plt

data = sns.load_dataset("iris")

def plot():

sns.lineplot(x="sepal_length", y="sepal_width", data=data)

sns.set_palette('vlag')
```

plt.subplot(211)

plot()

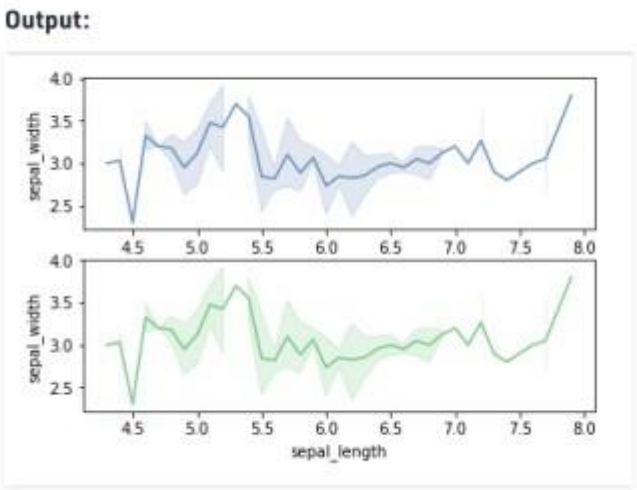
sns.set_palette('Accent')

plt.subplot(212)

plot()

plt.show()

Output:



4) Briefly explain the following multiple plots functions using seaborn ----------10M

a) FacetGrid ()

b) PairGrid ()

Ans: a) Facegrid():

1) FacetGrid class helps in visualizing distribution of one variable as well as the relationship between multiple variables separately within subsets of your dataset using multiple panels.

2) A FacetGrid can be drawn with up to three dimensions ? row, col, and hue. The first two have obvious correspondence with the resulting array of axes; think of the hue variable as a third dimension along a depth axis, where different levels are plotted with different colors.

3) FacetGrid object takes a dataframe as input and the names of the variables that will form the row, column, or hue dimensions of the grid. The variables should be categorical and the data at each level of the variable will be used for

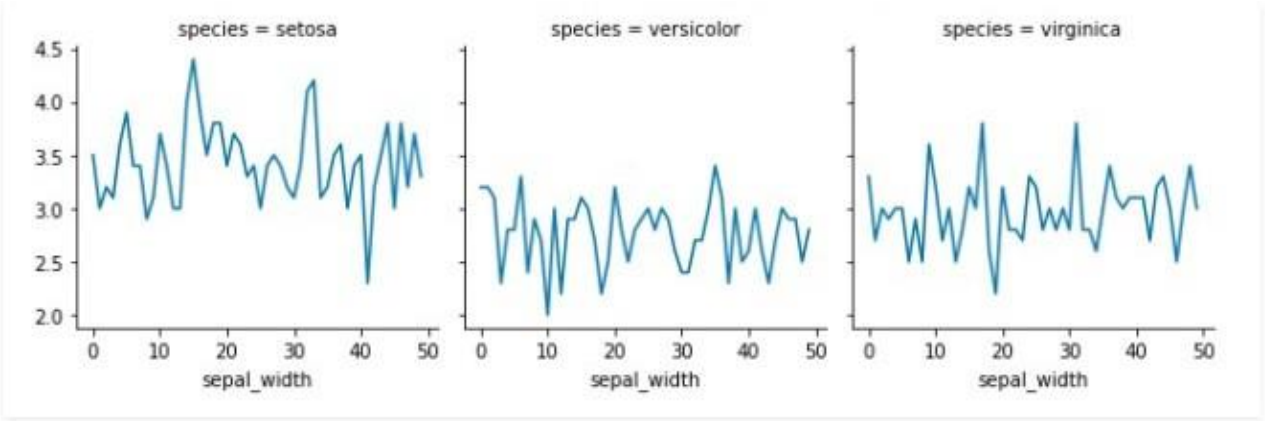
a facet along that axis.

Syntax: seaborn.FacetGrid(data, **kwargs)

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("iris")
plot = sns.FacetGrid(data, col="species")
plot.map(plt.plot, "sepal_width")
plt.show()
```

Output:



b) PairGrid ()

1)Subplot grid for plotting pairwise relationships in a dataset.

2)This class maps each variable in a dataset onto a column and row in a grid of multiple axes. Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the marginal distribution of each variable can be shown on the diagonal.

3)It can also represent an additional level of conventionalization with the hue parameter, which plots different subsets of data in different colors. This uses color to resolve elements on a third dimension, but only draws subsets on top of each other and will not tailor the hue parameter for the specific visualization the way that axes-level functions that accept hue will.

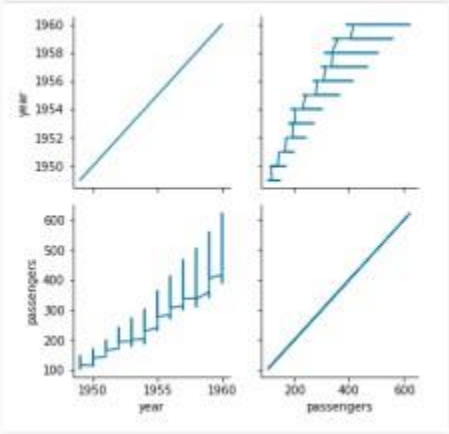
Syntax:

seaborn.PairGrid(data, **kwargs)

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
data = sns.load_dataset("flights")
plot = sns.PairGrid(data)
plot.map(plt.plot)
plt.show()
```

Output:



5) Briefly explain Data visualization in Watson Studio-Adding data to data refinery and how Visualize the Data on Watson Studio?

The IBM Watson Studio is one of the services on the IBM Cloud Pak for Data platform. Provision the integrated Lite versions of Watson Studio and Watson Machine Learning for free today as part of Cloud Pak for Data as a Service.

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

Bring the data into Data Refinery:

1. Download the airline-data.csv file (1.5 MB) .
2. Right-click the browser window, and then choose **Save Page As** or **Save As** depending on your browser. (desktop.) Make sure the downloaded file name is airline-data.csv.
3. Add the airline-data.csv file to your project:
    1. From your project's **Assets** page, click **Add to project > Data**.
    2. In the **Load** pane that opens, browse to the airline-data.csv file. Stay on the page until the load completes.
4. The airline-data.csv file is added to your project as a data asset.
5. Click the airline-data.csv data asset to preview its contents.
6. Click **Refine** to open a sample of the file in Data Refinery.

Refine the data

Refining data is a series of steps to build a *Data Refinery flow.* As you go through this tutorial, view the **Steps** pane to follow your progress. You can select a step to delete or edit it. If you make a mistake, you can also click the Undo icon .

1. Go back to the **Data** tab.
2. Select the *Year* column. Click the Actions menu () and choose **Sort descending**.
3. Focus on the delays for a specific airline:
4. This tutorial uses United Airlines (UA), but you can choose any airline.
    1. Click **Operation +**, and then choose the GUI operation **Filter**.
    2. Choose the *UniqueCarrier* column.
    3. For Operator, choose *Is equal to.*

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.

To visualize your data:

- From Data Refinery, click the Visualizations tab.

- ❑ Start with a chart or select columns.