Experiment 1

Group Members:

Pratham Rawat	201060019
Pranav Deshpande	201060022
Sarvesh Shirude	201060076
Shivansh Shetty	201060017

Aim: Fundamentals of Python Programming

Q1) Print the word [your hobby] in “My name is [your name] I like to [your hobby].

Q2) Print the date, month, and year. Find if the year is a leap or not.

Q3) Make a function that sums the list ‘[user i/p]’.

Q4) Given a list with pairs, sort on the first element.

[(3,6), (4,7), (5,9), (8,4), (3,1)] Now sort on the second element.

Theory:

1. Data Types: Python has several built-in data types such as integers, floating-point numbers, strings, Booleans, lists, and dictionaries. Understanding these data types is essential as they form the building blocks for more complex data structures.
2. Control Flow: Control flow refers to the order in which statements are executed in a program. Python provides various control flow statements such as if-else, for loops, and while loops. These statements help in making decisions and repeating actions based on certain conditions.
3. Functions: Functions are reusable blocks of code that can perform specific tasks. They are an essential part of Python programming as they allow us to break down complex problems into smaller, manageable parts.
4. Input and Output: Input and output operations are fundamental to any programming language. Python provides several built-in functions for input and output, such as print(),input(), and file operations.

5. Modules and Libraries: Python has a vast standard library, which provides several modules for various tasks. Additionally, there are many third-party libraries that can be installed to extend the functionality of Python.
6. Exceptions: Exceptions are events that occur during program execution that disrupt the normal flow of control. Python provides a built-in mechanism for handling exceptions, which helps in handling errors gracefully.

Conclusion:

1) Use of strings:
Here, we receive a string input from the user and combine the strings together to create a particular output.

2) Use of if-else structure:
Here we use the if-else statement to determine if a provided date occurs in a leap year or not.

3)**Use of for loop:**
Here, we use a for loop to repeat a certain task for n number of times.

4)**Use of sort function:**
In this context, we apply a sorting function to arrange the information in an array. By default, the sorting is performed based on the first element. In the second part, we've employed the "key" parameter within the sorting function to sort the information based on the second element instea**d.**

During our experiment, we have delved into several essential Python programming concepts, such as string concatenation, if-else statements, for loops, and sorting functions. We've come to understand that concatenating strings entails merging multiple strings into one, while if-else loops aid in regulating program flow based on specific conditions. For loops allow us to iterate over a sequence of elements, and sorting functions are useful for sorting lists of elements in either ascending or descending order. By comprehending these concepts, we can produce programs that are more legible, sustainable, and adaptable. Overall, our experiment has yielded invaluable insights into the fundamental principles of Python programming.

Name : Sarvesh Eknath Shirude, Reg No. : 201060076

```
yourname = input("Enter your name:")
hobby=input("Enter your hobby:")
print(f"your name is {yourname}\nI like to {hobby}")
```

```
Enter your name:Sarvesh
Enter your hobby:swim
your name is Sarvesh
I like to swim
```

```
import datetime as dt
```

```
print(str(dt.datetime.now()))
```

```
2023-03-08 05:00:39.820078
```

```
year=int(dt.datetime.now().year)
if((year%4==0 and year%100!=0)  or year%400==0):
    print(f"{year} is a leap year")
else:
    print (f"{year} is not a leap year")
```

```
2023 is not a leap year
```

```
def sum(lst):
    sum=0
    for i in lst:
        sum+=i
    return sum


itr=int(input("Enter the number of elements: "))
totalSum=[0]*itr
for j in range(itr):
    totalSum[j]=int (input(f"Enter the {j} element"))

sum(totalSum)
```

```
Enter the number of elements: 4
Enter the 0 element7
Enter the 1 element8
Enter the 2 element9
Enter the 3 element10
34
```

```
list1=[(3,6),(4,7),(5,9),(8,4),(3,1)]
list3=sorted(list1)
print(list3)

list1.sort(key=lambda x: x[1])
print(list1)
```

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

Experiment 2

Aim: Data Visualisation and Wrangling (ggplot)

Group Members:

Pratham Rawat	201060019
Pranav Deshpande	201060022
Sarvesh Shirude	201060076
Shivansh Shetty	201060017

Theory:

1. IMPORTING DATA : We store the contents of the csv file 'cartwheel data' in the 'url' string which helps in hosting our csv file Then we read the csv file using (.read_csv) function and store it as a pandas data frame

2. Viewing Data: Firstly we check the starting basic data of the dataframe using .head() function Similarly , we can see the last 5 rows and column values using .tail() method Then we see the entire data frame by simply typing the dataframe (df) Then we check the name of all columns in the dataframe using .columns method

3. Splicing Dataframe (a) .loc() :The .loc() method of a Pandas dataframe allows you to select specific rows and/or columns by specifying their names. On the other hand, the .iloc() method is used for integer-based slicing. You can check the data types of the values in the columns of a dataframe using the .dtypes property and find the unique values in a column using the .unique() method. The groupby() function is used for grouping the data based on the distinct values in a given column or columns and then calculating aggregated values for each group. It is a powerful tool for data analysis and is frequently used in Pandas.

4. SEABORN : Seaborn is a Python data visualization library that provides a high-level interface for creating attractive and informative statistical graphics. It offers a wide range of plot types, including scatter plots, line plots, bar plots, box plots, and heatmaps, among others. Seaborn also provides easy customization options, such as control over colors, styles, and fonts, allowing users to create professional-looking plots. One of the key advantages of Seaborn is its high-level interface, which simplifies the process of creating complex visualizations with minimal code. Seaborn integrates well with Pandas data structures and is built on top of Matplotlib, another popular data visualization library.

SEABORN-PLOTS USED IN THE EXPERIMENT:

(a) The line plot (lmplot) :

- It is one of the most basic plots. It shows a line on a 2 dimensional plane.
- It provides the basic details like which variable will be on the x-axis and y-axis.
- We can also use the hue parameter that determines which column in the data frame should be used for color encoding.

(b) Swarmplot :

- A swarm plot is a type of scatter plot that is used for representing categorical values.
- It is very similar to the strip plot, but it avoids the overlapping of points.
- We can use the seaborn. swarmplot() to create such graphs. It is not advisable to use this type of graph when the sample size is large.
- Syntax of swarm plot is same as the syntax of the lmplot , we specify the x-axis variable and y-axis variable from the dataset.

(c) BOXPLOT :

- A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.
- The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.
- Syntax : sns.boxplot(data=df, x = ‘variable_name’, y= ‘variable_name’, hue= ‘variable_name’)

(d) Distplot/Histogram :

- A Distplot or distribution plot, depicts the variation in the data distribution.
- Seaborn Distplot represents the overall distribution of continuous data variables.
- The seaborn.distplot() function accepts the data variable as an argument and returns the plot with the density distribution.
- Syntax: seaborn.distplot()

(e) Countplot:

- Seaborn literally counts the number of observations per category for a categorical variable, and displays the results as a bar chart.
- Syntax : sns.countplot(data= dataframe, x= ‘varaible_name’).

Conclusion: We worked with a dataset called 'cartwheel.csv' and used splicing techniques such as .loc and .iloc to select specific data that we wanted to visualize. To create the visualizations, we used various plotting functions provided by Seaborn, such as boxplot, line plot, swarm plot, and count plot. By using these different types of plots, we were able to explore and analyze the data in different ways. The splicing methods allowed us to extract

only the data we needed for each specific visualization, while Seaborn provided a range of tools to create professional-looking plots with minimal code.

Sarvesh Shirude , 201060076

Batch 2

Cartwheeldata.csv

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

```
url= "/content/Cartwheeldata.csv"
df = pd.read_csv(url)
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df.head()
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	Complete
0	1	56	F	1	Y	1	62.0	61.0	79	Y	
1	2	26	F	1	Y	1	62.0	60.0	70	Y	
2	3	33	F	1	Y	1	66.0	64.0	85	Y	
3	4	39	F	1	N	0	64.0	63.0	87	Y	
4	5	27	M	2	N	0	73.0	75.0	72	N	

```
df
```

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	Complet
0	1	56	F	1	Y	1	62.00	61.0	79	Y	
1	2	26	F	1	Y	1	62.00	60.0	70	Y	
2	3	33	F	1	Y	1	66.00	64.0	85	Y	
3	4	39	F	1	N	0	64.00	63.0	87	Y	
4	5	27	M	2	N	0	73.00	75.0	72	N	
5	6	24	M	2	N	0	75.00	71.0	81	N	
6	7	28	M	2	N	0	75.00	76.0	107	Y	
7	8	22	F	1	N	0	65.00	62.0	98	Y	
8	9	29	M	2	Y	1	74.00	73.0	106	N	
9	10	33	F	1	Y	1	63.00	60.0	65	Y	
10	11	30	M	2	Y	1	69.50	66.0	96	Y	
11	12	28	F	1	Y	1	62.75	58.0	79	Y	
12	13	25	F	1	Y	1	65.00	64.5	92	Y	
13	14	23	F	1	N	0	61.50	57.5	66	Y	
14	15	31	M	2	Y	1	73.00	74.0	72	Y	
15	16	26	M	2	Y	1	71.00	72.0	115	Y	
16	17	26	F	1	N	0	61.50	59.5	90	N	
17	18	27	M	2	N	0	66.00	66.0	74	Y	
18	19	23	M	2	Y	1	70.00	69.0	64	Y	
19	20	24	F	1	Y	1	68.00	66.0	85	Y	
20	21	23	M	2	Y	1	69.00	67.0	66	N	
21	22	29	M	2	N	0	71.00	70.0	101	Y	
22	23	25	M	2	N	0	70.00	68.0	82	Y	
23	24	26	M	2	N	0	69.00	71.0	63	Y	
24	25	23	F	1	Y	1	65.00	63.0	67	N	

```
df.columns
```

```
Index(['ID', 'Age', 'Gender', 'GenderGroup', 'Glasses', 'GlassesGroup',
       'Height', 'Wingspan', 'CWDistance', 'Complete', 'CompleteGroup',
       'Score'],
      dtype='object')
```

```
df.dtypes
```

```
ID                int64
Age               int64
Gender           object
GenderGroup       int64
Glasses          object
GlassesGroup      int64
Height          float64
Wingspan        float64
CWDistance        int64
Complete         object
CompleteGroup     int64
Score             int64
dtype: object
```

```
df.shape
```

```
(25, 12)
```

```
print(pd.isnull(df).sum())
```

```
ID              0
Age             0
Gender          0
GenderGroup     0
Glasses         0
GlassesGroup    0
Height          0
Wingspan        0
CWDistance      0
Complete        0
CompleteGroup   0
Score           0
dtype: int64
```

```
df.loc[2:5,["Height", "CWDistance"]]
```

	Height	CWDistance
2	66.0	85
3	64.0	87
4	73.0	72
5	75.0	81

```
df.iloc[2:5,[1, 2]]
```

	Age	Gender
2	33	F
3	39	F
4	27	M

Nhanes 2015-2016

```
url= "/content/nhanes_2015_2016.csv"
da = pd.read_csv(url)
type(da)
```

```
pandas.core.frame.DataFrame
```

```
da.head()
```

	SEQN	ALQ101	ALQ110	ALQ130	SMQ020	RIAGENDR	RIDAGEYR	RIDRETH1	DMDCITZN	DMDEDUC2	...	BPXSY2
0	83732	1.0	NaN	1.0	1	1	62	3	1.0	5.0	...	124.0
1	83733	1.0	NaN	6.0	1	1	53	3	2.0	3.0	...	140.0

```
da
```

	SEQN	ALQ101	ALQ110	ALQ130	SMQ020	RIAGENDR	RIDAGEYR	RIDRETH1	DMDCITZ
0	83732	1.0	NaN	1.0	1	1	62	3	1.(
1	83733	1.0	NaN	6.0	1	1	53	3	2.(
2	83734	1.0	NaN	NaN	1	1	78	3	1.(
3	83735	2.0	1.0	1.0	2	2	56	3	1.(
4	83736	2.0	1.0	1.0	2	2	42	4	1.(
...	...	...	...	...	...	...	...	...	..
5730	93695	2.0	2.0	NaN	1	2	76	3	1.(
5731	93696	2.0	2.0	NaN	2	1	26	3	1.(
5732	93697	1.0	NaN	1.0	1	2	80	3	1.(
5733	93700	NaN	NaN	NaN	1	1	35	3	2.(
5734	93702	1.0	NaN	2.0	2	2	24	3	1.(

5735 rows × 28 columns

+ Code + Text

```
da.columns
```

```
Index(['SEQN', 'ALQ101', 'ALQ110', 'ALQ130', 'SMQ020', 'RIAGENDR', 'RIDAGEYR',
       'RIDRETH1', 'DMDCITZN', 'DMDEDUC2', 'DMDMARTL', 'DMDHHSIZ', 'WTINT2YR',
       'SDMVPSU', 'SDMVSTRA', 'INDFMPIR', 'BPXSY1', 'BPXDI1', 'BPXSY2',
       'BPXDI2', 'BMXWT', 'BMXHT', 'BMXBMI', 'BMXLEG', 'BMXARML', 'BMXARMC',
       'BMXWAIST', 'HIQ210'],
      dtype='object')
```

```
da.dtypes
```

```
SEQN          int64
ALQ101      float64
ALQ110      float64
ALQ130      float64
SMQ020        int64
RIAGENDR      int64
RIDAGEYR      int64
RIDRETH1      int64
DMDCITZN    float64
DMDEDUC2    float64
DMDMARTL    float64
DMDHHSIZ      int64
WTINT2YR    float64
SDMVPSU       int64
SDMVSTRA      int64
INDFMPIR    float64
BPXSY1      float64
BPXDI1      float64
BPXSY2      float64
BPXDI2      float64
BMXWT       float64
BMXHT       float64
BMXBMI      float64
BMXLEG      float64
BMXARML     float64
BMXARMC     float64
BMXWAIST    float64
HIQ210      float64
dtype: object
```

```
da.shape
```

```
(5735, 28)
```

```
da.loc[:,["ALQ101","ALQ130"]]
```

	ALQ101	ALQ130
0	1.0	1.0
1	1.0	6.0
2	1.0	NaN
3	2.0	1.0
4	2.0	1.0
...	...	...
5730	2.0	NaN
5731	2.0	NaN
5732	1.0	1.0
5733	NaN	NaN

```
da.iloc[:,[1,3]]
```

	ALQ101	ALQ130
0	1.0	1.0
1	1.0	6.0
2	1.0	NaN
3	2.0	1.0
4	2.0	1.0
...	...	...
5730	2.0	NaN
5731	2.0	NaN
5732	1.0	1.0
5733	NaN	NaN
5734	1.0	2.0

5735 rows × 2 columns

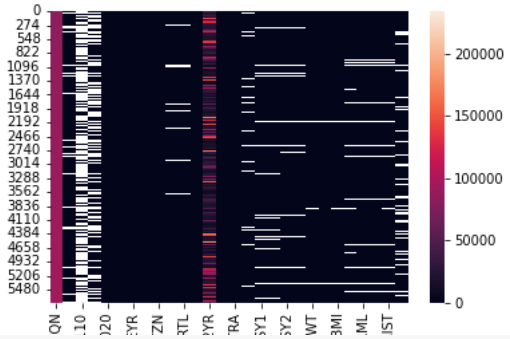
```
print(pd.isnull(da).sum())
```

```
SEQN           0
ALQ101       527
ALQ110      4004
ALQ130      2356
SMQ020         0
RIAGENDR       0
RIDAGEYR       0
RIDRETH1       0
DMDCITZN       1
DMDEDUC2     261
DMDMARTL     261
DMDHHSIZ       0
WTINT2YR       0
SDMVPSU        0
SDMVSTRA       0
INDFMPIR     601
BPXSY1       334
BPXDI1       334
BPXSY2       200
BPXDI2       200
BMXWT         69
BMXHT         62
BMXBMI        73
BMXLEG       390
BMXARML      308
BMXARMC      308
BMXWAIST     367
HIQ210      1003
dtype: int64
```

```
import seaborn as snc
```

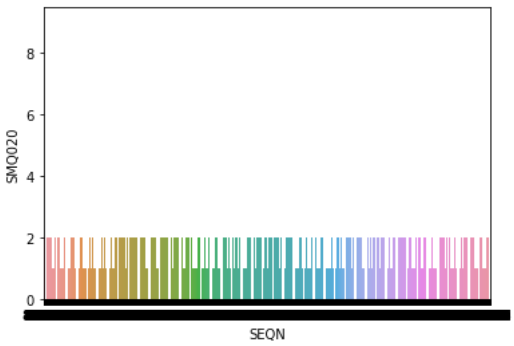
```
snc.heatmap(da)
```

```
<AxesSubplot:>
```



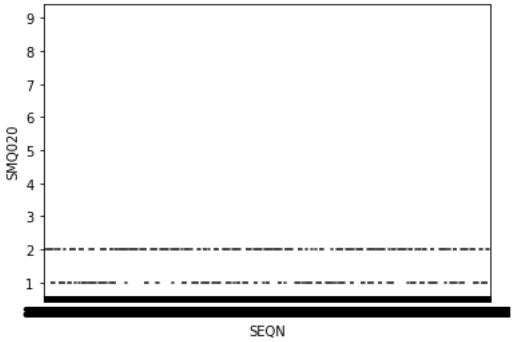

```
snc.barplot(x="SEQN",y="SMQ020",data=da)
```

```
<AxesSubplot:xlabel='SEQN', ylabel='SMQ020'>
```




```
snc.boxplot(x="SEQN",y="SMQ020",data=da)
```

```
<AxesSubplot:xlabel='SEQN', ylabel='SMQ020'>
```



Colab paid products - Cancel contracts here

✓ 2m 2s completed at 10:50 AM

Experiment 3

Aim: To implement linear regression with different regularization techniques.

Group Members:

Pratham Rawat	201060019
Pranav Deshpande	201060022
Sarvesh Shirude	201060076
Shivansh Shetty	201060017

Theory:

Regression:

A regression model is a tool that helps us understand how one or more independent variables are related to a dependent or response variable. By analyzing the data, we can create a function that describes this relationship. For instance, we can use a linear regression model to describe how someone's weight may be affected by their height. By examining the data and constructing a regression model, we can gain insights into how different variables may be related to each other and make predictions about future outcomes.

1)Simple Linear Regression: Simple linear regression is a regression model that estimates the relationship between one independent variable and one dependent variable using a straight line. Both variables should be quantitative.

2)Bias:

When creating a model to make predictions, we analyze our data to identify patterns and relationships that can help us make accurate predictions. These patterns and relationships are learned by the model during training and are used to predict outcomes for new data.
Bias refers to the difference between the actual values and the predictions made by our model. It arises from the assumptions that our model makes about the data, which can affect its accuracy. These assumptions are based on the patterns and relationships we observe in the data during training and may not always be accurate for new, unseen data. Thus, bias is a measure of how well our model can fit the data, and it is an important consideration when evaluating the performance of a model.

When the Bias is high, assumptions made by our model are too basic, the model can't capture
the important features of our data. This means that our model hasn't captured patterns in the
training data and hence cannot perform well on the testing data too. If this is the case, our
model cannot perform on new data and cannot be sent into production.
This instance, where the model cannot find patterns in our training set and hence fails for
both seen and unseen data, is called Underfitting.
The below figure shows an example of Underfitting. As we can see, the model has found no
patterns in our data and the line of best fit is a straight line that does not pass through any of
the data points. The model has failed to train properly on the data given and cannot predict new data either.

3)Variance:

In terms of linear regression, variance is a measure of how far observed values differ from the average of predicted values, i.e., their difference from the predicted value means. The goal is to have a value that is low.We can define variance as the model's sensitivity to fluctuations in the data. Our model may learn from noise. This will cause our model to consider trivial features as important.In the above figure, we can see that our model has learned extremely well for our training data, which has taught it to identify cats. But when given new data, such as the picture of a fox, our model predicts it as a cat, as that is what it has learned. This happens when the Variance is high, our model will capture all the features of the data given to it, including the noise, will tune itself to the data, and predict it very well but when given new data, it cannot predict on it as it is too specific to training data. Hence, our model will perform really well on testing data and get high accuracy but will fail to perform on new, unseen data. New data may not have the exact same features and the model won't be able to predict it very well. This is called Overfitting.

4)Bias - Variance Tradeoff:

For any model, we have to find the perfect balance between Bias and Variance. This just ensures that we capture the essential patterns in our model while ignoring the noise present it in. This is called Bias-Variance Tradeoff. It helps optimize the error in our model and keeps it as low as possible. An optimized model will be sensitive to the patterns in our data, but at the same time will be able to generalize to new data. In this, both the bias and variance should be low so as to prevent overfitting and underfitting.

5) Regularization:
Regularization is a method used in machine learning to adjust models and minimize the adjusted loss function to prevent overfitting or underfitting. Overfitting occurs when the model performs well with the training data but fails to perform well with the test data. This is because the model cannot predict the output when it deals with unseen data, leading to noise in the output. Regularization techniques reduce the magnitude of variables, allowing all variables or features to be maintained in the model while maintaining accuracy and generalization of the model.

1) Ridge Regression:
Ridge regression is a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated. It has been used in many fields including econometrics, chemistry, and engineering.Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called L2 regularization.

2) Lasso Regression:
Lasso regression is a regularization method used to improve the accuracy of regression models. The technique uses shrinkage, where data values are pulled closer to a central point, usually the mean. Lasso regression encourages simpler and sparser models. It is similar to Ridge Regression, but with a penalty term that contains only absolute weights instead of squared weights. Because Lasso Regression uses absolute values, it can shrink the slope to 0, whereas Ridge Regression can only reduce it near to 0.

3) Polynomial regression:
Polynomial regression is a statistical technique that establishes the relationship between a dependent variable (y) and an independent variable (x) by modeling their relationship as an nth degree polynomial. Polynomial regression is a variation of linear regression that accounts for the non-linear relationship between the dependent and independent variables by introducing polynomial terms. For example, if X is the independent data and Y is the dependent data, we convert the input variables into polynomial terms using a certain degree before feeding the data into the model during the preprocessing stage.

4) Logistic Regression :
It is a popular algorithm in supervised learning used for predicting the categorical dependent variable based on a given set of independent variables. It is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome.

The outcome of the dependent variable is binary, meaning it can only take on two possible values, such as Yes/No or True/False.

Logistic Regression generates probabilities of the output variable being in a certain category, rather than providing an exact value. The probabilities range from 0 to 1, and are used to classify the dependent variable into one of two categories. The model fits the data using a sigmoid function, which is a type of S-shaped curve that maps any input value to a value between 0 and 1. This curve is used to calculate the probability of an event occurring, given a set of inputs.

Logistic Regression is similar to Linear Regression, with the main difference being the type of problem it is used for. Linear Regression is used for predicting continuous values, while Logistic Regression is used for classification problems. In Linear Regression, the output variable is continuous, and the model fits the data using a line. In contrast, Logistic Regression models the relationship between the input variables and the probability of the output variable being in a particular category.

Conclusion:

In this experiment, we explored three types of regression techniques: linear regression, polynomial regression, and logistic regression. Linear regression is primarily used to handle regression problems, while logistic regression is designed to handle classification problems. Polynomial regression is a powerful method that provides an excellent approximation of the relationship between the dependent and independent variables.Linear regression predicts the value of a dependent variable based on one or more independent variables. Polynomial regression is an extension of linear regression, where the relationship between the dependent and independent variables is modeled as an nth degree polynomial. Logistic regression, on the other hand, is a popular algorithm that predicts the probability of an event occurring, given a set of input variables.

Experiment 3

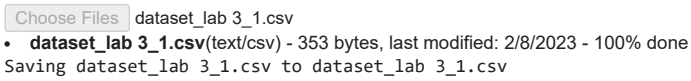
Group 9

- Pratham Rawat
- Pranav Deshpande
- Sarvesh Shirude
- Shivansh Shetty

Part 1

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

```
from google.colab import files
uploaded = files.upload()
```



- **dataset_lab 3_1.csv**(text/csv) - 353 bytes, last modified: 2/8/2023 - 100% done

```
Saving dataset_lab 3_1.csv to dataset_lab 3_1.csv
```

```
dataset = pd.read_csv('dataset_lab 3_1.csv')
dataset
```

```
   YearsExperience  Salary
```

```
from scipy.sparse.construct import random
x = dataset.iloc[:,:-1].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 0)
```

```
<ipython-input-8-6420f8d3ef75>:1: DeprecationWarning: Please use `random` from the `scipy.sparse` namespace, the `scipy.sparse.
  from scipy.sparse.construct import random
```

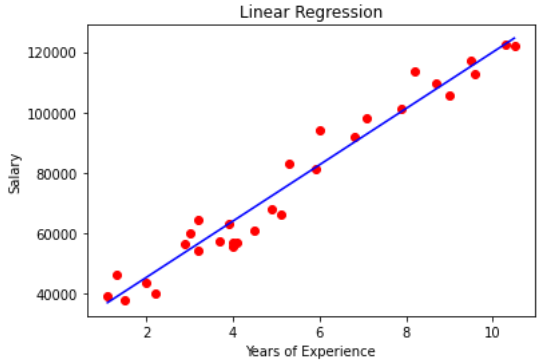
```
6                3.0   60150
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
11               4.0   55794
```

```
plt.scatter(x, y, color = 'red')
plt.plot(x, lin_reg.predict(x), color = 'blue')
plt.title('Linear Regression')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```




```
20                9.0  105582
```

```
lin_reg.score(x_train,y_train)
```

```
0.9411949620562126
```

```
28               10.3  122391
```

```
lin_reg.score(x_test,y_test)
```

```
0.988169515729126
```

```
lin_reg.predict([[6.5]])
```

```
array([87311.83747437])
```

Part 2

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files dataset_lab_3_2.csv
• dataset_lab_3_2.csv(text/csv) - 260 bytes, last modified: 2/8/2023 - 100% done
Saving dataset_lab_3_2.csv to dataset_lab_3_2 (1).csv
```

```
dataset = pd.read_csv('dataset_lab_3_2.csv')
dataset
```

	Position	Level(X-variable)	Salary(Y-variable)
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
6	Partner	7	200000

```
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None)
```

▾ LinearRegression
LinearRegression()

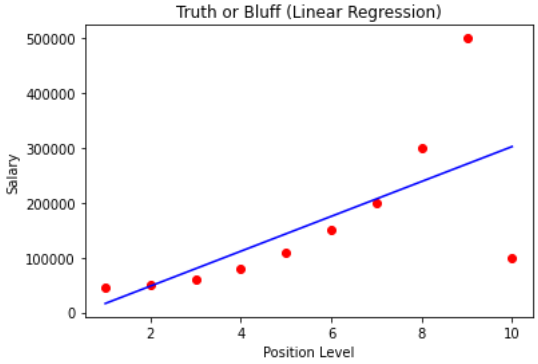
```
from sklearn.preprocessing import PolynomialFeatures
poly_regr = PolynomialFeatures(degree = 4)
X_poly = poly_regr.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

▾ LinearRegression
LinearRegression()

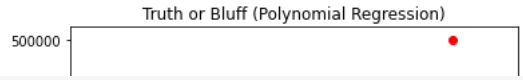
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None)
```

▾ LinearRegression
LinearRegression()

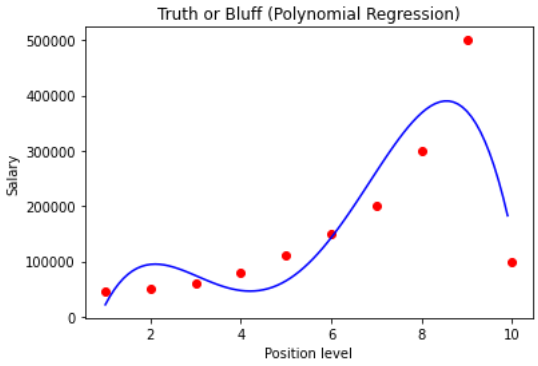
```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```




```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_regr.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```




```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, lin_reg_2.predict(poly_regr.fit_transform(X_grid)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```




```
lin_reg.predict([[6.5]])
```

```
array([191287.87878788])
```

```
lin_reg_2.predict(poly_regr.fit_transform([[6.5]]))
```

```
array([200410.74810596])
```

Part 3

```
from sklearn.model_selection import train_test_split
from math import exp
plt.rcParams["figure.figsize"] = (10, 6)
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files dataset_lab_3_3.csv

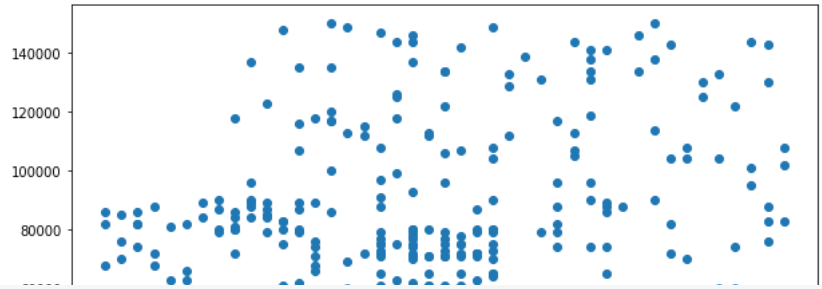
- **dataset_lab_3_3.csv**(text/csv) - 10926 bytes, last modified: 2/9/2023 - 100% done

Saving dataset_lab_3_3.csv to dataset_lab_3_3.csv

```
data = pd.read_csv('dataset_lab_3_3.csv')
data.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
plt.scatter(data['Age'], data['EstimatedSalary'])
plt.show()
x = data.iloc[:,[2,3]].values
y = data.iloc[:,4].values
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```




```
from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler()
X_train=sc_x.fit_transform(X_train)
X_test=sc_x.fit_transform(X_test)
print(X_train[0:10,:])
```

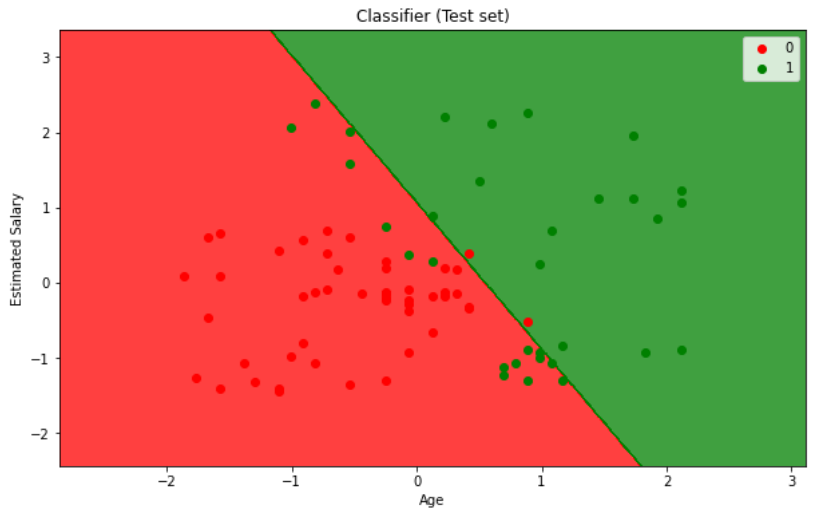
```
[[ 0.31985524  0.03258407]
 [-1.69083439  0.32880284]
 [ 1.660315    1.75065298]
 [-0.06313326  2.16535926]
 [-0.92485739 -0.79682851]
 [ 0.89433799 -0.61909725]
 [-0.15888039  1.39519044]
 [-0.54186889  1.36556856]
 [ 1.94755637  0.89161852]
 [-1.11635164 -1.59661921]]
```

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
y_pred_sk = lr_model.predict(X_test)
plt.clf()
print(f"Accuracy = {lr_model.score(X_test, y_test)}")
```

```
Accuracy = 0.775
<Figure size 720x432 with 0 Axes>
```

```
from matplotlib.colors import ListedColormap
x_set,y_set=X_test,y_test
x1,x2=np.meshgrid(np.arange(start=x_set[:,0].min()-1,stop=x_set[:,0].max()+1,step=0.01),
                  np.arange(start=x_set[:,1].min()-1,stop=x_set[:,1].max()+1,step=0.01))
plt.contourf(x1,x2, lr_model.predict(np.array([x1.ravel(),x2.ravel()]).T).reshape(x1.shape), alpha=0.75, cmap=ListedColormap(('red',
plt.xlim(x1.min(),x1.max())
plt.ylim(x2.min(),x2.max())

for i,j in enumerate (np.unique (y_set)):
    plt.scatter(x_set[y_set==j, 0],x_set[y_set==j,1],c=ListedColormap(('red', 'green')) (i), label=j)
plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGB
WARNING:matplotlib.axes._axes:*c* argument looks like a single numeric RGB or RGB
```



DS EXPERIMENT - 4

GROUP MEMBERS
Pranav Deshpande - 201060022
Sarvesh Shirude - 201060076
Shivansh Shetty - 201060017
Pratham Rawat - 201060019

AIM: To implement different Clustering methods.

SOFTWARE USED: Google Colab

THEORY:
Cluster analysis is a machine learning technique that involves grouping an unlabelled dataset. This technique involves dividing the data points into various clusters based on their similarities, and ensuring that objects with the possible similarities remain in the same group. The clusters are separated based on similar patterns in the unlabelled dataset, such as size, shape, color, and behavior. As it is an unsupervised learning method, it operates on unlabeled datasets without any supervision provided to the algorithm. Once the clustering technique has been applied, each cluster is assigned a unique cluster-ID, which can be used by ML systems to simplify the processing of large and complex datasets.

Types of Clustering Algorithms:
There are various types of clustering algorithms, each with its unique approach to grouping data points. One such algorithm is the mean-shift algorithm, which searches for dense areas in the smooth density of data points. It belongs to the centroid-based model and updates the centroid candidates to be the center of the points in a given region.

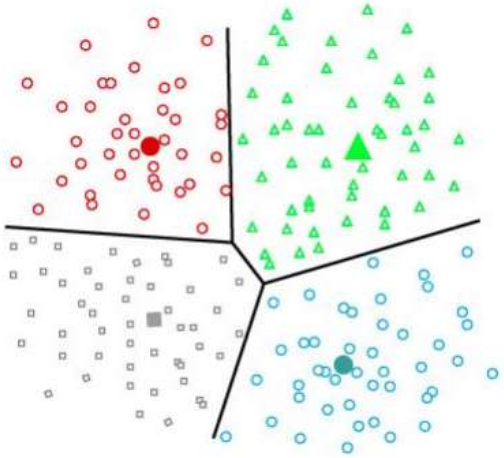
Another clustering algorithm is the DBSCAN algorithm, which stands for Density-Based Spatial Clustering of Applications with Noise. It is similar to the mean-shift algorithm but offers remarkable advantages. This density-based model separates areas of high density from those of low density, enabling the detection of clusters in any arbitrary shape.

The Expectation-Maximization clustering algorithm using Gaussian Mixture Models (GMM) is an alternative to the K-means algorithm, suitable for

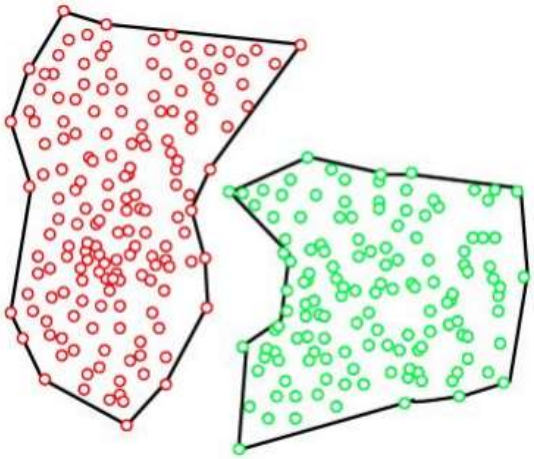
cases where K-means fails. The algorithm assumes that data points are Gaussian distributed.

Lastly, the Agglomerative Hierarchical algorithm adopts a bottom-up hierarchical clustering approach. It begins by treating each data point as a single cluster and then merging them successively. The cluster hierarchy is represented in the form of a tree structure.

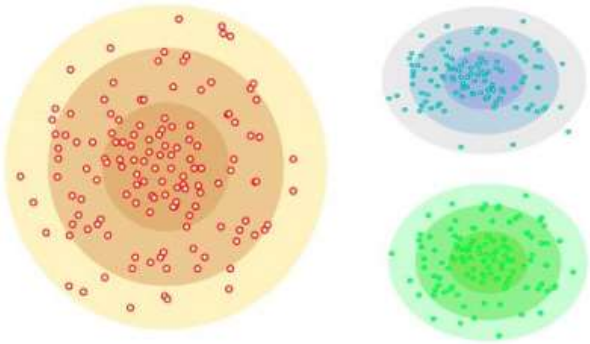
Partitioning Clustering (centroid method clustering):
It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the **K-Means Clustering algorithm**. In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



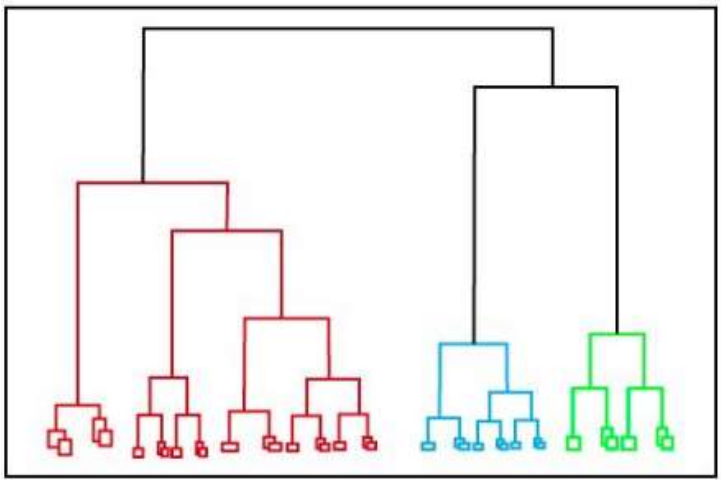
Density-Based Clustering:
The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected. This algorithm does it by identifying different clusters in the dataset and connects the areas of high densities into clusters. The dense areas in data space are divided from each other by sparser areas. These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering:
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions, commonly **Gaussian Distribution.** The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



Hierarchical Clustering:
Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram.** The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm.**



Conclusion:

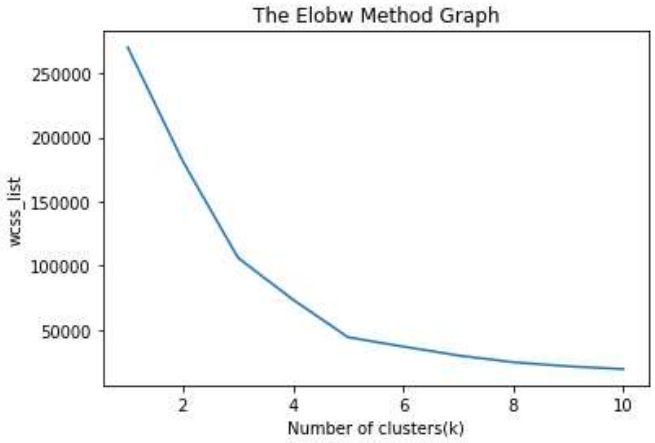
```
[ ]  import numpy as nm
     import matplotlib.pyplot as mtp
     import pandas as pd
```

```
[ ]  df=pd.read_csv("Mall_Customers.csv")
     x=df.iloc[:,[3,4]].values
```

```
print(x)
```

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]
 [ 20  13]
 [ 20  79]
 [ 21  35]
 [ 21  66]
 [ 23  29]
 [ 23  98]
 [ 24  35]
 [ 24  73]
 [ 25   5]
 [ 25  73]
 [ 28  14]
 [ 28  82]
```

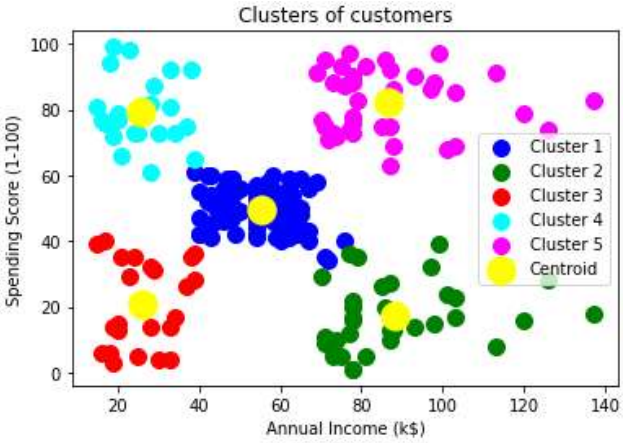
```
#@title
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= [] #Initializing the list for the values of WCSS
#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',
random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elbow Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```




```
[ ]  #@title
     #training the K-means model on a dataset
     kmeans = KMeans(n_clusters=5, init='k-means++',
     random_state= 42)
     y_predict= kmeans.fit_predict(x)
     y_predict
```

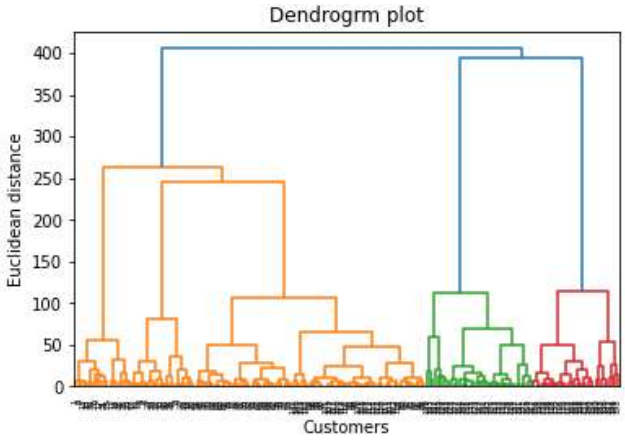
```
array([2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 0,
       2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 4, 0, 4, 1, 4, 1, 4,
       0, 4, 1, 4, 1, 4, 1, 4, 1, 4, 0, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4], dtype=int32)
```

```
#@title
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s =
100, c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s =
100, c = 'green', label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s =
100, c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s =
100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s =
100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label =
'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```



Hierarchial clustering

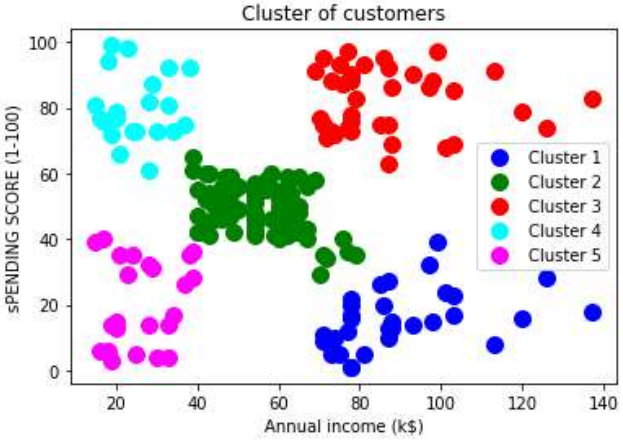
```
[ ] #Finding the optimal number of cluster using the dendrogram
    import scipy.cluster.hierarchy as shc
    dendro=shc.dendrogram(shc.linkage(x,method="ward"))
    mtp.title("Dendrogrm plot")
    mtp.ylabel("Euclidean distance")
    mtp.xlabel("Customers")
    mtp.show()
```




```
[ ] #step 3
    #training the hierarchial model on dataset
    from sklearn.cluster import AgglomerativeClustering
    hc=AgglomerativeClustering(n_clusters=5,affinity="euclidean",linkage="ward")
    y_pred=hc.fit_predict(x)
    y_pred
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

```
[ ] #Step 4
    #visualizing the cluster
    mtp.scatter(x[y_pred==0,0],x[y_pred==0,1],s=100,c="blue",label="Cluster 1")
    mtp.scatter(x[y_pred==1,0],x[y_pred==1,1],s=100,c="green",label="Cluster 2")
    mtp.scatter(x[y_pred==2,0],x[y_pred==2,1],s=100,c="red",label="Cluster 3")
    mtp.scatter(x[y_pred==3,0],x[y_pred==3,1],s=100,c="cyan",label="Cluster 4")
    mtp.scatter(x[y_pred==4,0],x[y_pred==4,1],s=100,c="magenta",label="Cluster 5")
    mtp.title("Cluster of customers")
    mtp.xlabel("Annual income (k$)")
    mtp.ylabel("sPENDING SCORE (1-100)")
    mtp.legend()
    mtp.show()
```



EXPERIMENT 5

GROUP MEMBERS:

Pranav Deshpande - 201060022
Sarvesh Shirude - 201060076
Shivansh Shetty - 201060017
Pratham Rawat – 201060019

AIM: Implementing Different types of Naive Bayes Classifier models

Software Used: Google Colab

THEORY:

The Naive Bayes algorithm is a type of supervised learning that utilizes Bayes' theorem to solve classification problems. It is particularly useful in text classification tasks that involve large training datasets with high dimensionality. The Naive Bayes Classifier is a straightforward yet highly effective classification algorithm that enables the creation of fast machine learning models capable of making rapid predictions.

There are three types of Naive Bayes Model, which are given below:

Gaussian Naive Bayes: Assumes that the features have a Gaussian (normal) distribution. It is commonly used for continuous data and is suitable for regression and classification problems.

Multinomial Naive Bayes: Typically used for discrete data, such as text classification, where the frequency of occurrence of each feature matters. It is commonly used in natural language processing and document classification tasks.

Bernoulli Naive Bayes: Similar to the Multinomial Naive Bayes, but assumes that the features are binary (either present or absent). It is commonly used in text classification tasks where the presence or absence of a word is important.

Each type of Naive Bayes model has its own assumptions and is suitable for different types of data. The choice of which model to use depends on the nature of the problem and the type of data available.

The Naive Bayes Classifier has several advantages:

- It is a fast and straightforward machine learning algorithm for predicting the class of datasets.
- It can be used for binary as well as multi-class classifications.
- It performs well in multi-class predictions compared to other algorithms.
- It is the most popular choice for text classification problems.

To implement the Naive Bayes algorithm in Python, we can use the "user_data" dataset, which we have used in other classification models. This will allow us to compare the performance of the Naive Bayes model with other models.

Steps to implement:

- Data Preprocessing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)

CONCLUSION:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
import seaborn as sns
%matplotlib inline
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
iris = pd.read_csv('Iris.csv')
```

```
iris.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris['Species'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
iris.drop('Id', inplace=True, axis=1)
```

```
iris
```

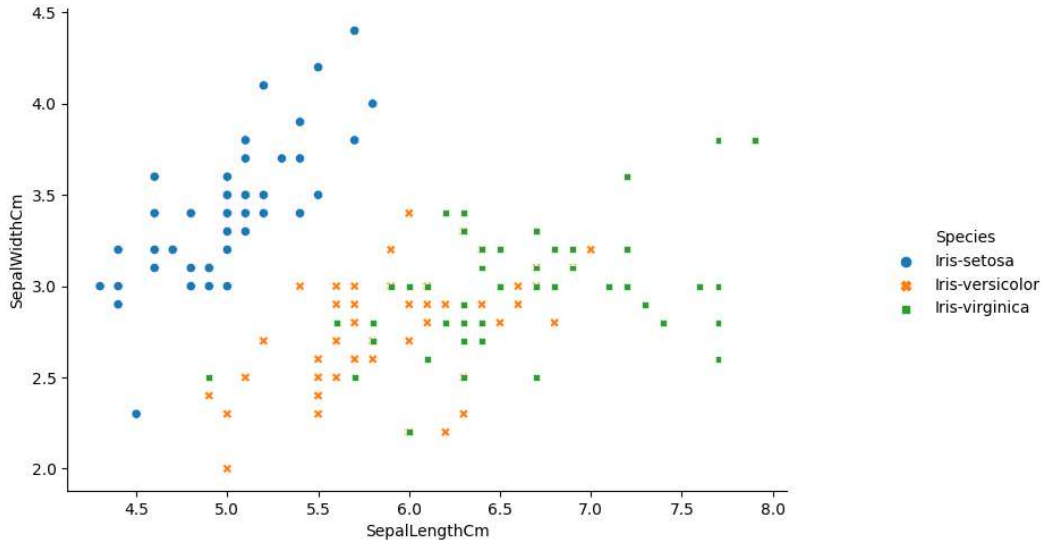
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

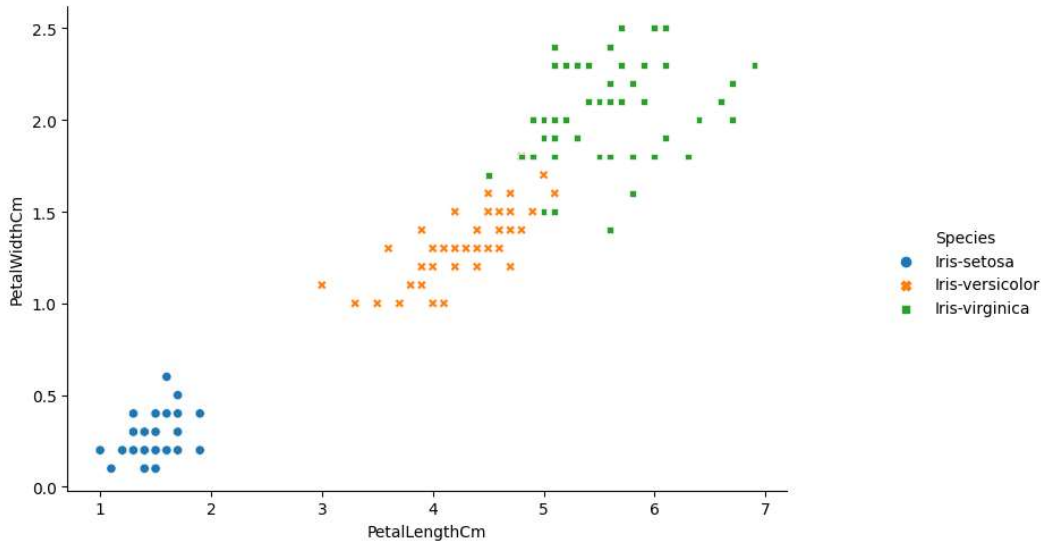
```
iris.isnull().sum()
```

```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
```

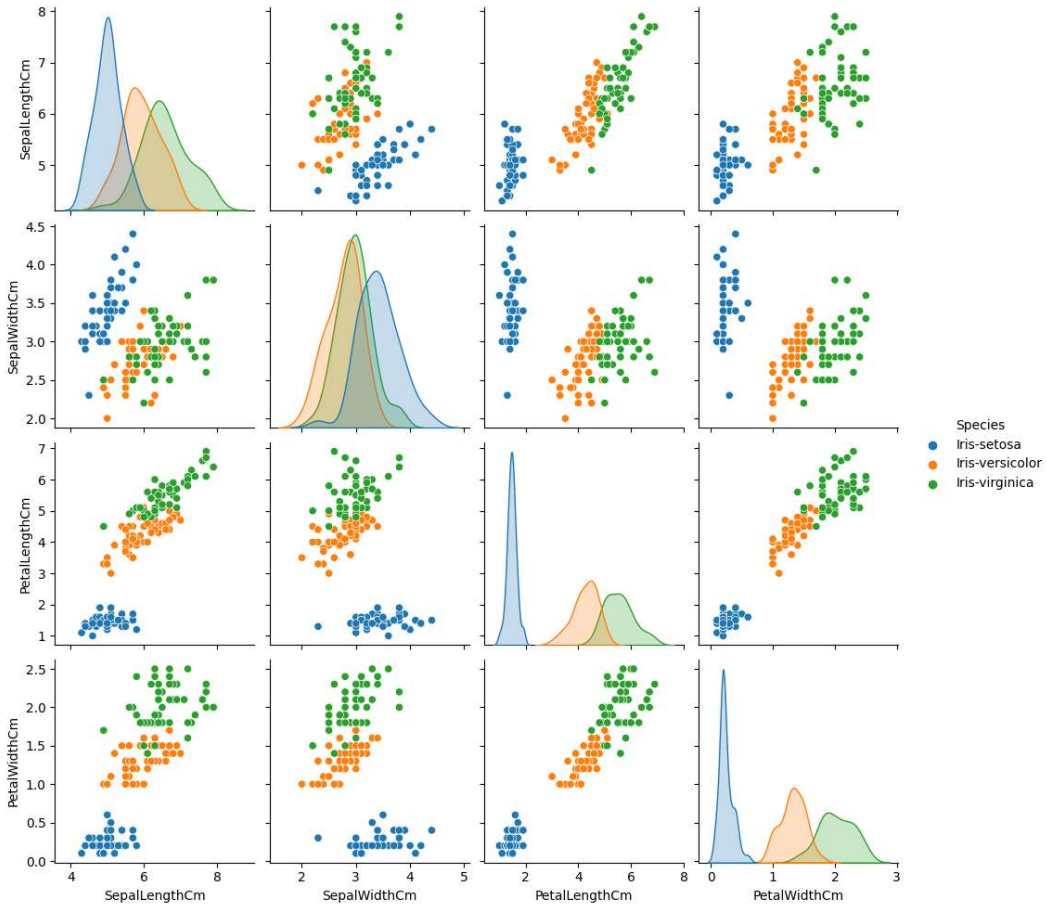
```
g=sns.relplot(x='SepalLengthCm',y='SepalWidthCm',data=iris,hue='Species',style='Species')
g.fig.set_size_inches(10,5)
plt.show()
```



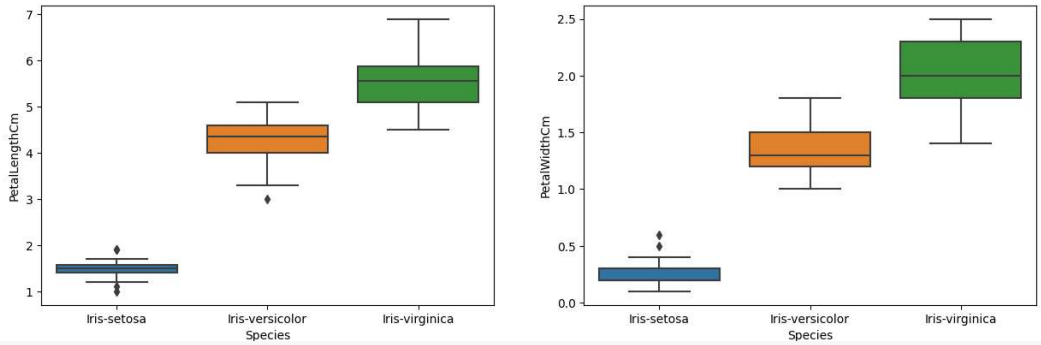

```
g=sns.relplot(x='PetalLengthCm',y='PetalWidthCm',data=iris,hue='Species',style='Species')
g.fig.set_size_inches(10,5)
plt.show()
```



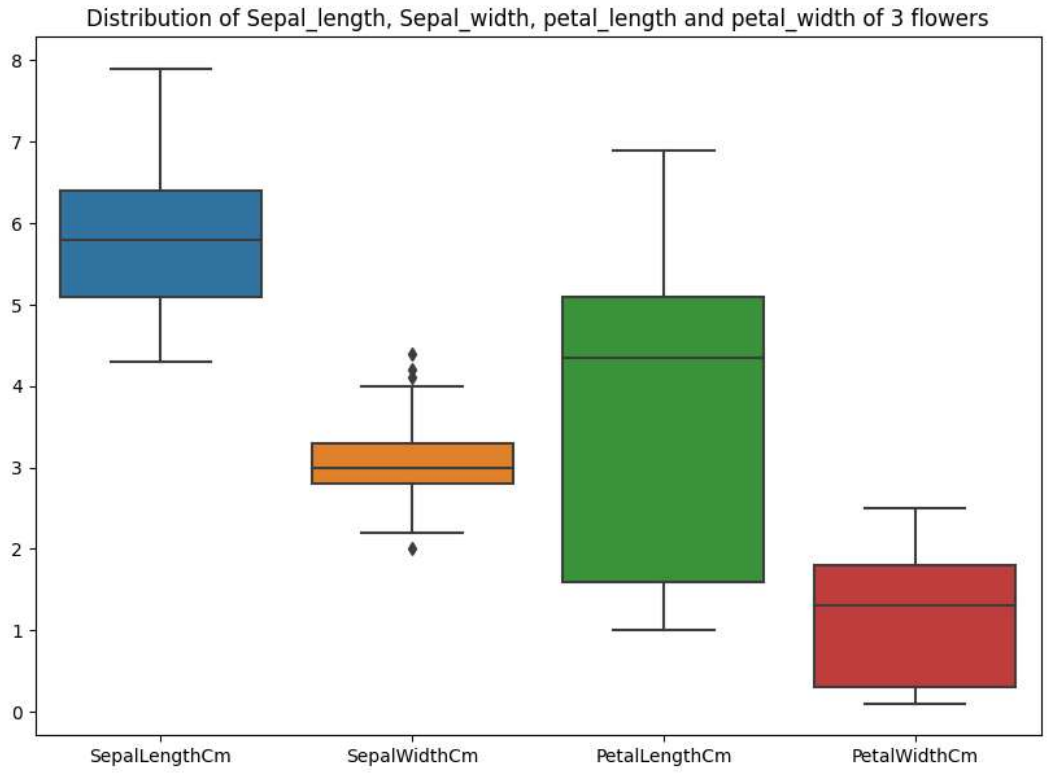

```
sns.pairplot(iris,hue="Species")
plt.show()
```



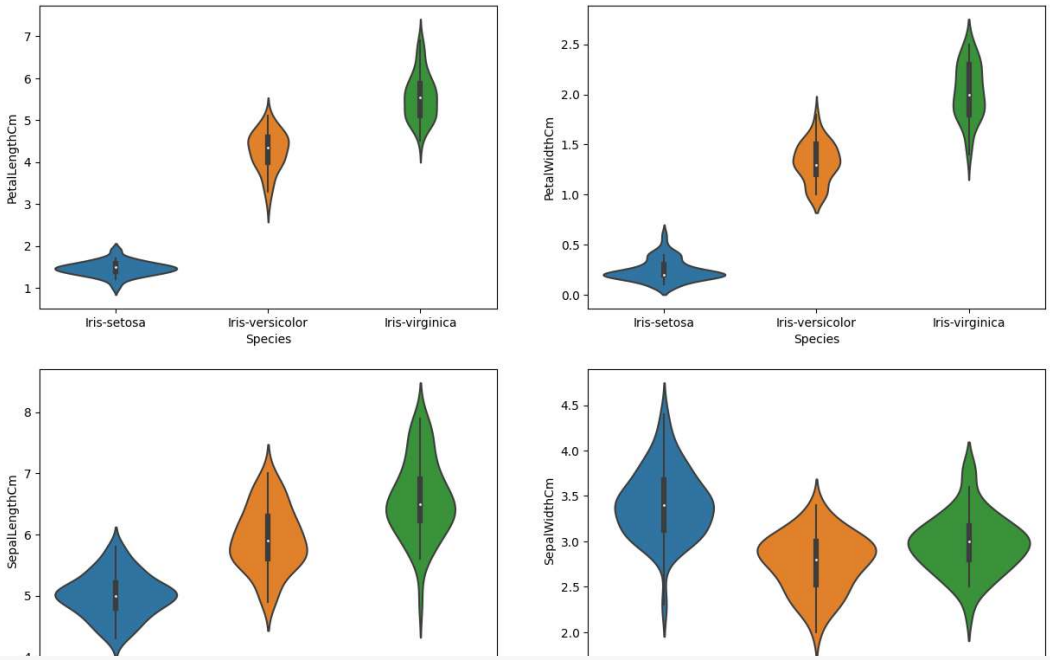

```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.boxplot(x='Species',y='PetalLengthCm',data=iris)
plt.subplot(2,2,2)
sns.boxplot(x='Species',y='PetalWidthCm',data=iris)
plt.subplot(2,2,3)
sns.boxplot(x='Species',y='SepalLengthCm',data=iris)
plt.subplot(2,2,4)
sns.boxplot(x='Species',y='SepalWidthCm',data=iris)
plt.show()
```



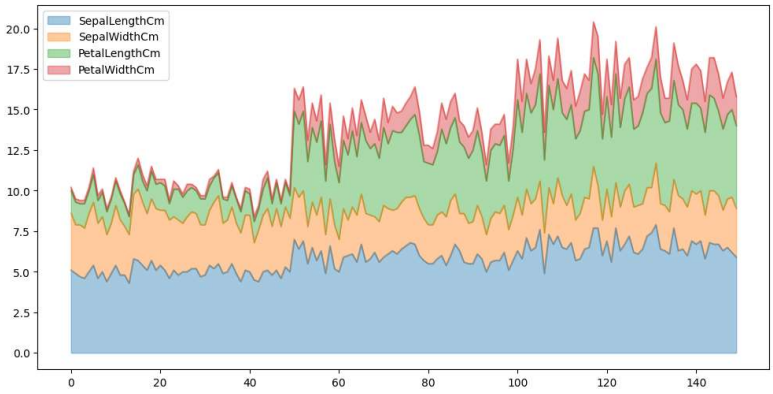

```
plt.subplots(figsize=(10,7))
sns.boxplot(data=iris).set_title("Distribution of Sepal_length, Sepal_width, petal_length and petal_width of 3 flowers")
plt.show()
```




```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='Species',y='PetalLengthCm',data=iris)
plt.subplot(2,2,2)
sns.violinplot(x='Species',y='PetalWidthCm',data=iris)
plt.subplot(2,2,3)
sns.violinplot(x='Species',y='SepalLengthCm',data=iris)
plt.subplot(2,2,4)
sns.violinplot(x='Species',y='SepalWidthCm',data=iris)
plt.show()
```




```
iris.plot.area(y=['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm'],alpha=0.4,figsize=(12, 6));
```



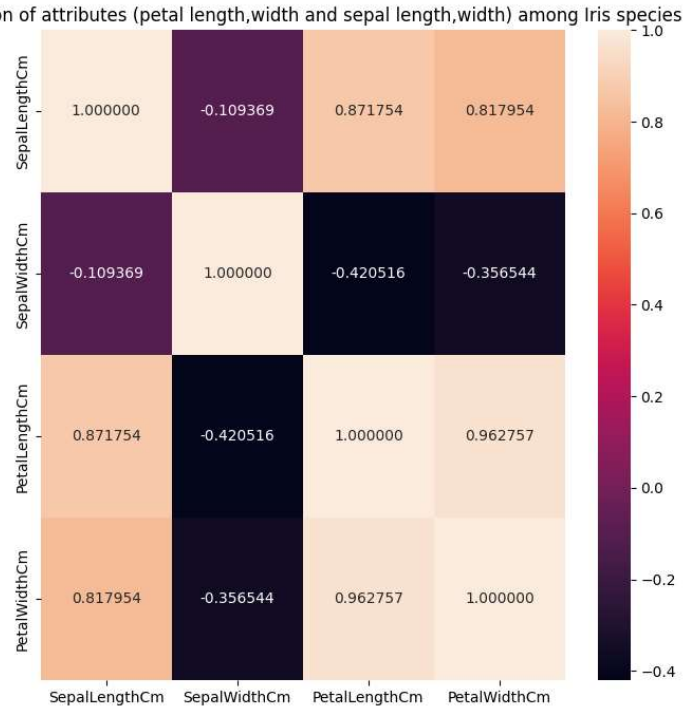

```
iris.corr()
```

```
<ipython-input-99-156dd03bc859>:1: FutureWarning: The default value of numer
  iris.corr()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

```
plt.subplots(figsize = (8,8))
sns.heatmap(iris.corr(),annot=True,fmt="f").set_title("Corelation of attributes (petal length,width and sepal length,width)
plt.show()
```

```
<ipython-input-100-22a60b3c6422>:2: FutureWarning: The default value of nume
  sns.heatmap(iris.corr(),annot=True,fmt="f").set_title("Corelation of attri
```




```
X=iris.iloc[:,0:4].values
y=iris.iloc[:,4].values
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

```
from sklearn.metrics import make_scorer
from sklearn.metrics import confusion_matrix

#Model Select
from sklearn.model_selection import KFold,train_test_split,cross_val_score
from sklearn import linear_model
from sklearn.naive_bayes import GaussianNB
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
Y_pred = gaussian.predict(X_test)
accuracy_nb=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_gaussian = round(gaussian.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for Naive Bayes\n',cm)
print('accuracy_Naive Bayes: %.3f' %accuracy)
print('precision_Naive Bayes: %.3f' %precision)
print('recall_Naive Bayes: %.3f' %recall)
print('f1-score_Naive Bayes : %.3f' %f1)
```

```
Confusion matrix for Naive Bayes
 [[16  0  0]
 [ 0 18  0]
 [ 0  0 11]]
accuracy_Naive Bayes: 1.000
precision_Naive Bayes: 1.000
recall_Naive Bayes: 1.000
f1-score_Naive Bayes : 1.000
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
multinomial = MultinomialNB()
multinomial.fit(X_train, y_train)
Y_pred = multinomial.predict(X_test)
accuracy_nb=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_gaussian = round(multinomial.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for Multinomial_ Naive Bayes\n',cm)
print('accuracy_Multinomial_Naive_Bayes: %.3f' %accuracy)
print('precision_Multinomial_Naive Bayes: %.3f' %precision)
print('recall_Multinomial_Naive Bayes: %.3f' %recall)
print('f1-score_Multinomial_Naive Bayes : %.3f' %f1)
```

```
Confusion matrix for Multinomial_ Naive Bayes
 [[16  0  0]
 [ 0  0 18]
 [ 0  0 11]]
accuracy_Multinomial_Naive_Bayes: 0.600
precision_Multinomial_Naive Bayes: 0.600
recall_Multinomial_Naive Bayes: 0.600
f1-score_Multinomial_Naive Bayes : 0.600
```

```
from sklearn.naive_bayes import BernoulliNB
```

```
bernoulli = BernoulliNB()
bernoulli.fit(X_train, y_train)
Y_pred = bernoulli.predict(X_test)
accuracy_nb=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_gaussian = round(bernoulli.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for bernoulli_Naive Bayes\n',cm)
print('accuracy_bernoulli_Naive_Bayes: %.3f' %accuracy)
print('precision_bernoulli_Naive Bayes: %.3f' %precision)
print('recall_bernoulli_Naive Bayes: %.3f' %recall)
print('f1-score_bernoulli_Naive Bayes : %.3f' %f1)
```

```
Confusion matrix for bernoulli_Naive Bayes
 [[ 0  0 16]
 [ 0  0 18]
 [ 0  0 11]]
accuracy_bernoulli_Naive_Bayes: 0.244
precision_bernoulli_Naive Bayes: 0.244
recall_bernoulli_Naive Bayes: 0.244
f1-score_bernoulli_Naive Bayes : 0.244
```

Colab paid products - Cancel contracts here

✓ 0s completed at 11:01 AM

EXPERIMENT 6

GROUP MEMBERS

Pranav Deshpande - 201060022
Sarvesh Shirude - 201060076
Shivansh Shetty - 201060017
Pratham Rawat - 201060019

Aim: To implement SVM Regression.

Software used: Google Colab

Theory:

Support Vector Machine (SVR)

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let's spend a few minutes understanding the idea behind SVR.

Support Vector Regression

The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample. So let's now dive deep and understand how SVR works actually.

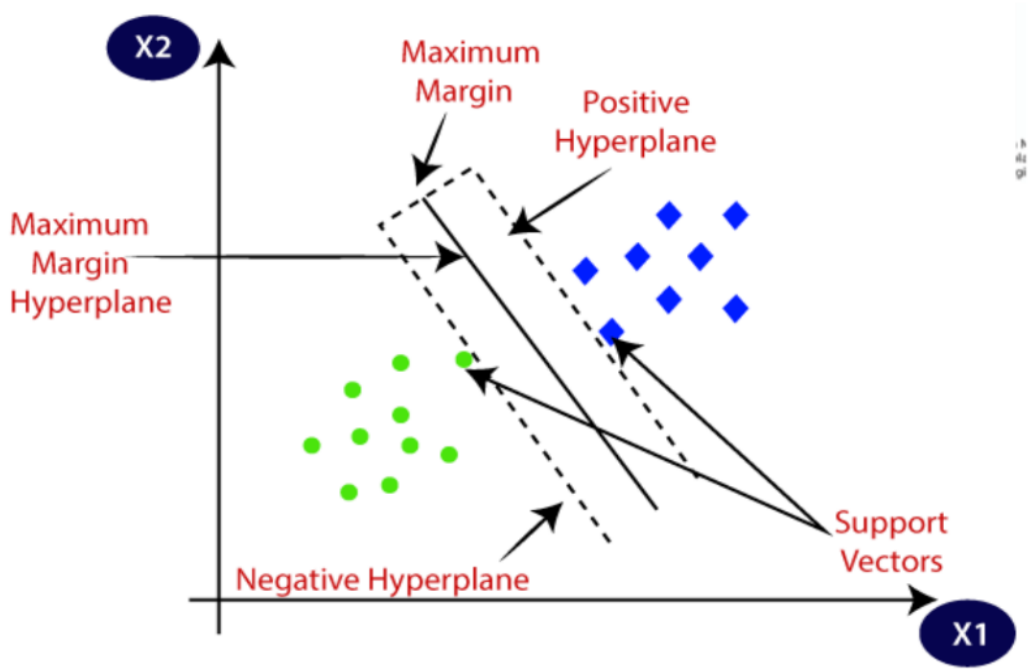
Support Vector Machine

A support vector machine (SVM) is a type of deep learning algorithm that performs supervised learning for classification or regression of data groups. In AI and machine learning, supervised learning systems provide both input and desired output data, which are labeled for classification.

SUPPORT VECTOR MACHINE ALGORITHM:

The Support Vector Machine (SVM) is a popular Supervised Learning algorithm used for both Classification and Regression problems, with primary use for Classification in Machine Learning. The main objective of the SVM algorithm is to create a decision boundary or hyperplane that can segregate an n-dimensional space into classes, enabling accurate categorization of new data points in the future. SVM achieves this by selecting the extreme points or vectors that aid in generating the hyperplane. These extreme cases are referred to as support vectors, thus giving the algorithm its name.

Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithm can be used for Face detection, image classification, text categorization, etc.

Types of SVM

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

SVM KERNEL FUNCTIONS:

SVM kernel functions are mathematical functions used in support vector machines to transform the input data into the required form for processing. The kernel function allows for the manipulation of the data using a set of mathematical functions, resulting in a non-linear decision surface that can be transformed into a linear equation in a higher

number of dimension spaces. In essence, the kernel function returns the inner product between two points in a standard feature dimension.

Different SVM algorithms use different types of kernel functions, including linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. These functions can be applied to various types of data, such as sequence data, graphs, text, images, and vectors. Among these kernel functions, RBF is the most widely used because of its localized and finite response along the entire x-axis.

Popular SVM Kernel Functions are:

Linear Kernel:

It is the most basic type of kernel, usually one dimensional in nature. It proves to be the best function when there are lots of features.
The linear kernel is mostly preferred for text-classification problems as most of these kinds of classification problems can be linearly separated.
Linear Kernel Formula: F(x, xj) = sum(x.xj)
Here, x, xj represents the data you're trying to classify.

Polynomial Kernel:

It is a more generalized representation of the linear kernel. It is not as preferred as other kernel functions as it is less efficient and accurate.
Polynomial Kernel Formula: F(x, xj) = (x.xj+1)^d
Here '.' shows the dot product of both the values, and d denotes the degree.
F(x, xj) representing the decision boundary to separate the given classes.

Gaussian Radial Basis Function (RBF) :

It is a commonly used kernel function in support vector machines (SVMs), especially for non-linear data. It can facilitate proper separation when there is no prior knowledge of the data. The Gaussian Radial Basis Formula is represented as
F(x, xj) = exp(-gamma * ||x - xj||^2),
where the value of gamma ranges from 0 to 1, and its value needs to be manually specified in the code. The recommended value for gamma is typically 0.1. Additionally, there are other kernel examples, such as the Sigmoid Kernel, Gaussian Kernel, Bessel function kernel, among others.

CONCLUSION:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets,svm
from sklearn.model_selection import train_test_split
from __future__ import division ,print_function
```

```
df=pd.read_csv('Hawks.csv')
print (df)
```

```
     Unnamed: 0  Month  Day  Year CaptureTime ReleaseTime  BandNumber Species  \
0             1      9   19  1992       13:30         NaN   877-76317      RT
1             2      9   22  1992       10:30               877-76318      RT
2             3      9   23  1992       12:45               877-76319      RT
3             4      9   23  1992       10:50               745-49508      CH
4             5      9   27  1992       11:15              1253-98801      SS
..          ...    ...  ...   ...         ...         ...         ...     ...
903         904     11   18  2003       14:44              1177-04777      RT
904         905     11   19  2003       10:18               803-05985      SS
905         906     11   19  2003       12:02              1807-53145      RT
906         907     11   20  2003        9:56              1177-04778      RT
907         908     11   20  2003       13:30              1207-53145      RT

    Age  Sex   Wing  Weight  Culmen  Hallux  Tail  StandardTail  Tarsus  \
0     I  NaN  385.0   920.0    25.7    30.1   219           NaN     NaN
1     I  NaN  376.0   930.0     NaN     NaN   221           NaN     NaN
2     I  NaN  381.0   990.0    26.7    31.3   235           NaN     NaN
3     I    F  265.0   470.0    18.7    23.5   220           NaN     NaN
4     I    F  205.0   170.0    12.5    14.3   157           NaN     NaN
..   ..  ...    ...     ...     ...     ...   ...           ...     ...
903   I  NaN  380.0  1525.0    26.0    27.6   224         227.0     NaN
904   I    F  190.0   175.0    12.7    15.4   150         153.0     NaN
905   I  NaN  360.0   790.0    21.9    27.6   211         215.0     NaN
906   I  NaN  369.0   860.0    25.2    28.0   207         210.0     NaN
907   A  NaN  199.0  1290.0    28.7    32.1   222         226.0     NaN

     WingPitFat  KeelFat  Crop
0           NaN      NaN   NaN
1           NaN      NaN   NaN
2           NaN      NaN   NaN
3           NaN      NaN   NaN
4           NaN      NaN   NaN
..          ...      ...   ...
903         NaN      3.0   0.0
904         NaN      4.0   0.0
905         NaN      2.0   0.0
906         NaN      2.0   0.0
907         NaN      1.0   0.0

[908 rows x 20 columns]
```

```
[ ] df.columns
```

```
Index(['Unnamed: 0', 'Month', 'Day', 'Year', 'CaptureTime', 'ReleaseTime',
       'BandNumber', 'Species', 'Age', 'Sex', 'Wing', 'Weight', 'Culmen',
       'Hallux', 'Tail', 'StandardTail', 'Tarsus', 'WingPitFat', 'KeelFat',
       'Crop'],
      dtype='object')
```

```
used_coloumns = ['Species','Wing','Weight','Tail']
df = df[used_coloumns]
df
```

	Species	Wing	Weight	Tail
0	RT	385.0	920.0	219
1	RT	376.0	930.0	221
2	RT	381.0	990.0	235
3	CH	265.0	470.0	220
4	SS	205.0	170.0	157
...	...	...	...	...
903	RT	380.0	1525.0	224
904	SS	190.0	175.0	150
905	RT	360.0	790.0	211
906	RT	369.0	860.0	207
907	RT	199.0	1290.0	222

908 rows × 4 columns

```
dups=df.duplicated()
print(dups.any())
print(df[dups])
```

```
True
    Species   Wing  Weight  Tail
332      RT  368.0  1244.0   220
430      SS  202.0   180.0   160
```

```
[ ] df.drop_duplicates(inplace=True)
```

```
<ipython-input-12-16cdb8520be8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop_duplicates(inplace=True)
```

```
[ ] print(df.isnull().sum())
```

```
Species     0
Wing        1
Weight     10
Tail        0
dtype: int64
```

```
[ ] df=df.dropna()
```

```
print(df)
print(df.isnull().sum())
```

```
    Species   Wing  Weight  Tail
0        RT  385.0   920.0   219
1        RT  376.0   930.0   221
2        RT  381.0   990.0   235
3        CH  265.0   470.0   220
4        SS  205.0   170.0   157
..      ...    ...     ...   ...
903      RT  380.0  1525.0   224
904      SS  190.0   175.0   150
905      RT  360.0   790.0   211
906      RT  369.0   860.0   207
907      RT  199.0  1290.0   222

[895 rows x 4 columns]
Species    0
Wing       0
Weight     0
Tail       0
dtype: int64
```

```
[ ] x=df.iloc[:,[1,3]].values
    y_code=df.iloc[:,0].values
    y=[]
    for i in y_code:
      if i == 'RT':
        y.append(0)
      elif i=='SS':
        y.append(1)
      else:
        y.append(2)
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=25)
```

```
[ ] x_reg = df.iloc[:, [1]].values
    y_reg = np.reshape(y, (-1,1))

    x_train_reg,x_test_reg,y_train_reg,y_test_reg = train_test_split(x_reg,y_reg,test_size=0.2,random_state=25)
```

```
[ ] from sklearn.preprocessing import StandardScaler
    StdS_X = StandardScaler()
    StdS_Y = StandardScaler()
    x_l = StdS_X.fit_transform(x_reg)
    y_p = StdS_Y.fit_transform(y_reg)
```

```
[ ] from sklearn.svm import SVR
```

```
    regressor_linear = SVR(kernel = 'linear')
    regressor_linear.fit(x_train_reg, y_train_reg)
    print("Kernel = linear")
    print("Accuracy obtained :" , regressor_linear.score(x_test_reg,y_test_reg), "\n")

    regressor_poly = SVR(kernel = 'poly')
    regressor_poly.fit(x_train_reg, y_train_reg)
    print("Kernel = poly")
    print("Accuracy obtained :" , regressor_poly.score(x_test_reg,y_test_reg), "\n")

    regressor_rbf = SVR(kernel = 'linear')
    regressor_rbf.fit(x_train_reg, y_train_reg)
    print("Kernel = rbf")
    print("Accuracy obtained :" , regressor_rbf.score(x_test_reg,y_test_reg), "\n")
```

```
    print(regressor_linear.support_vectors_)
```

```
[[412. ]
 [366. ]
 [209. ]
 [205. ]
 [366. ]
 [435. ]
 [377. ]
 [213. ]
 [369. ]
 [208. ]
 [253. ]
 [345. ]
 [355. ]
 [362. ]
 [361. ]
```

```
[412. ]
[369. ]
[358. ]
[161. ]
[370. ]
[205. ]
[417. ]
[365. ]
[158. ]
[369. ]
[258. ]
[413. ]
[362. ]
[230. ]
[252. ]
[359. ]
[415. ]
[313. ]
[420. ]
[415. ]
[199. ]
[370. ]
[272. ]
[205. ]
[370. ]
[412. ]
[208. ]
[427. ]
[213. ]
[161. ]
[365. ]
[478. ]
```

```
[ ] len(regressor_linear.support_vectors_)
```

```
266
```

```
print(regressor_poly.support_vectors_)
```

```
[225. ]
[366. ]
[415. ]
[410. ]
[240. ]
[372. ]
[268. ]
[252. ]
[369. ]
[375. ]
[402. ]
[372. ]
[418. ]
[359. ]
[375. ]
[363. ]
[403. ]
[408. ]
[220. ]
[375. ]
[223. ]
[362. ]
[372. ]
```

```
[354. ]
[362. ]
[213. ]
[374. ]
[422. ]
[422. ]
[412. ]
[260. ]
[254. ]
[223. ]
[412. ]
[371. ]
[418. ]
[374. ]
[369. ]
[416. ]
[271. ]
[408. ]
[351. ]
[259. ]
[404. ]
[215. ]
[425. ]
[225. ]
[366. ]
```

```
[ ] len(regressor_poly.support_vectors_)
```

```
326
```

```
print(regressor_poly.support_vectors_)
```

```
[[412. ]
 [405. ]
 [410. ]
 [366. ]
 [366. ]
 [375. ]
 [435. ]
 [377. ]
 [213. ]
 [369. ]
 [409. ]
 [372. ]
 [253. ]
 [345. ]
 [372. ]
 [355. ]
 [362. ]
 [361. ]
 [480. ]
 [410. ]
```

```
[ ] len(regressor_rbf.support_vectors_)
```

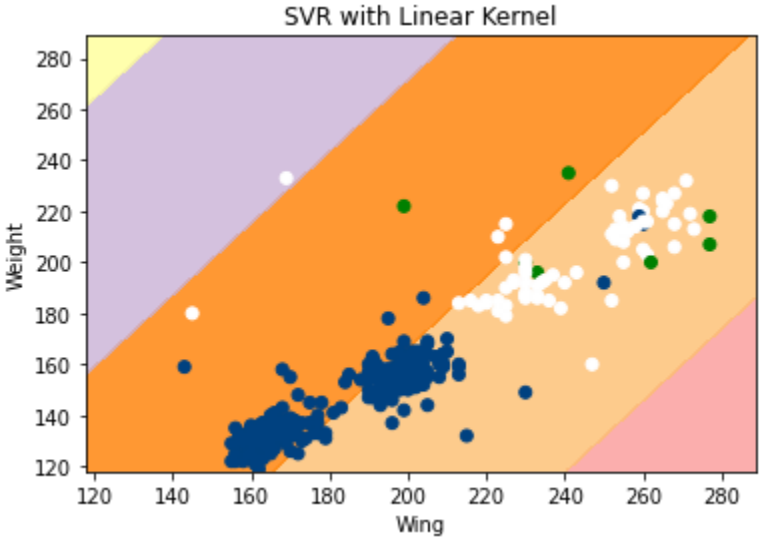
```
266
```

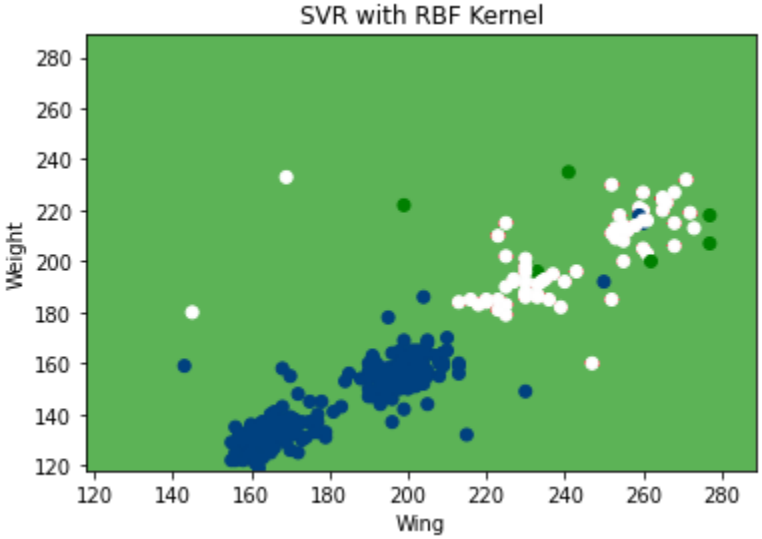
```
[ ] #visualization of various kernels:
    from sklearn.svm import SVR
    regressor_linear = SVR (kernel ='linear').fit(x_train, y_train)
    regressor_rbf= SVR(kernel ='rbf', gamma= 0.7).fit(x_train, y_train)
    regressor_poly= SVR(kernel= 'poly', degree= 3).fit(x_train, y_train)
    #creating a mesh plot:
```

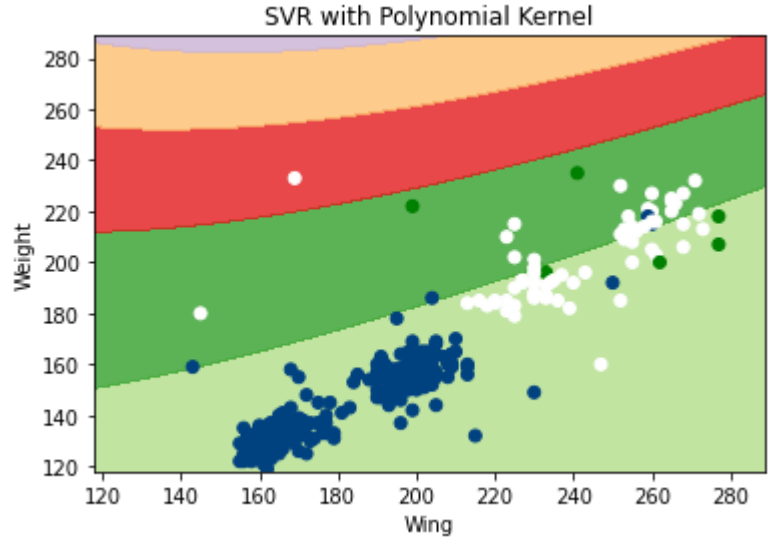
```
[ ] h = 0.2;
    x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
    y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
    xx, yy= np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
[ ] titles = ['SVR with Linear Kernel', 'SVR with RBF Kernel', 'SVR with Polynomial Kernel']
```

```
for i, clf in enumerate((regressor_linear, regressor_rbf,regressor_poly)):
  plt.figure(i)
  z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
  z = z.reshape(xx.shape)
  plt.contourf(xx, yy, z, cmap= plt.cm.Paired, alpha=0.8)

  plt.scatter(x[:,0], x[:, 1], c=y, cmap = plt.cm.ocean)
  plt.xlabel('Wing')
  plt.ylabel('Weight')
  plt.xlim(xx.min(), xx.max())
  plt.xlim(yy.min(), yy.max())
  #plt.xticks(())
  #plt.yticks(())
  plt.title(titles[i])
plt.show()
```


SVR with Linear Kernel
Weight
Wing
120
140
160
180
200
220
240
260
280


SVR with RBF Kernel
Weight
Wing
120
140
160
180
200
220
240
260
280


SVR with Polynomial Kernel
Weight
Wing
120
140
160
180
200
220
240
260
280

DS EXPERIMENT - 7

GROUP MEMBERS:

Pranav Deshpande - 201060022

Sarvesh Shirude - 201060076

Shivansh Shetty - 201060017

Pratham Rawat – 201060019

AIM: To study random walks and monte-carlo simulations.

THEORY:

Random walk

In mathematics, a random walk is a random process that describes a path that consists of a succession of random steps on some mathematical space.

An elementary example of a random walk is the random walk on the integer number line which starts at 0, and at each step moves +1 or −1 with equal probability.

Other examples include the path traced by a molecule as it travels in a liquid or a gas (see Brownian motion), the search path of a foraging animal, or the price of a fluctuating stock and the financial status of a gambler.

Monte Carlo Simulations

A Monte Carlo simulation is a type of computational algorithm that estimates the probability of occurrence of an undeterminable event due to the involvement of random variables.

The algorithm relies on repeated random sampling in an attempt to determine the probability.

This means simulating an event with random inputs a large number of times to obtain your estimation.

Monte Carlo simulations can be utilized in a broad range of fields spanning from economics, engineering, energy, gambling etc.

PART 1:

This Python code defines a function random_walk(n) that simulates a random walk of n steps in two dimensions. The function starts at the point (0, 0) and then iterates n times, taking a random step in one of four directions - up, down, left, or right - at each iteration. The function returns the final position (x, y) after n steps.

The code then sets the number of trials to 10,000 and initializes an empty list called distances. It then simulates a random walk with 30 steps 10,000 times by calling the random_walk function and recording the distance travelled for each walk.

For each of the 10,000 trials, the code calculates the distance travelled from the starting point using the Pythagorean theorem, distance = math.sqrt(x*2 + y*2), where (x, y) is the final position after 30 steps. The code then appends the distance value to the distances list.

After simulating the random walk 10,000 times and recording the distance travelled for each walk in the distances list, the code calculates the expected distance by taking the sum of all distances and dividing by the number of trials, expected_distance = sum(distances) / num_trials.

Finally, the code prints out the expected distance, which represents the average distance travelled from the starting point after 30 steps over all 10,000 trials

PART 2:

Performing the calculation of the ratio of the area of a circle to the area of a square. The code does this by randomly generating a large number of points within the square, and counting how many of those points fall within the inscribed circle. The ratio of the count of points inside the circle to the total number of points gives an approximation of the ratio of the area of the circle to the area of the square.

CONCLUSION:

Code:-

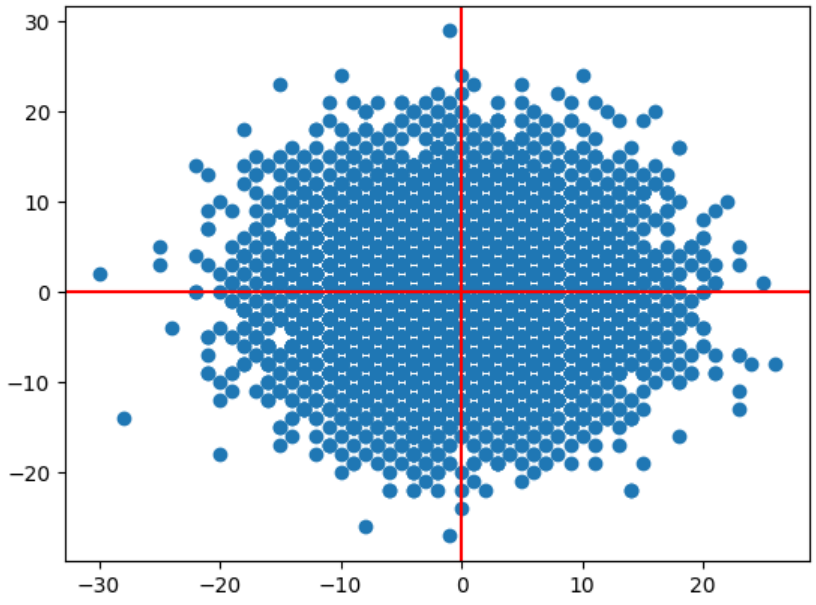
Random Walk

```
import random
import math
import matplotlib.pyplot as plt


def random_walk(n):

    x, y = 0, 0
    for i in range(n):

        dx, dy = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)])
        x += dx
        y += dy
    return x, y


num_trials = 10000
distances = []
X = []
Y = []
for i in range(num_trials):
    x, y = random_walk(100)
    X.append(x)
    Y.append(y)
    distance = math.sqrt(x**2 + y**2)
    distances.append(distance)


expected_distance = sum(distances)/num_trials
print(expected_distance)
```

```
8.862983838545384
```

```
plt.scatter(X,Y)
plt.axhline(y=0,color ='red')
plt.axvline(x=0,color ='red')
```

```
<matplotlib.lines.Line2D at 0x7f20250a5db0>
```



Monte Carlo

```
import random
import math
import matplotlib.pyplot as plt
```

```
side = 2
```

```
area_square = side**2
```

```
incenter = (side / 2, side / 2)
radius = side / 2
```

```
num_points = 100000

count_inside_circle = 0
X1,X2,Y1,Y2 = [],[],[],[]
for i in range(num_points):
    x = random.uniform(0, side)
    y = random.uniform(0, side)
    if math.sqrt((x - incenter[0])**2 + (y - incenter[1])**2) <= radius:
        count_inside_circle += 1
        X1.append(x)
        Y1.append(y)
    else:
      X2.append(x)
      Y2.append(y)
```

```
ratio_m = count_inside_circle / num_points
print(ratio_m)
```

```
0.78707
```

```
print("the value of pi",ratio_m*4)
```

```
the value of pi 3.14828
```

```
plt.scatter(X1,Y1,color = 'blue')
plt.scatter(X2,Y2, color = 'orange')
```

```
<matplotlib.collections.PathCollection at 0x7f8401cd3400>
```