Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2023-24

Name: Krish Thakkar SAP ID: 60009230213

Course: Machine Learning -I Course Code: DJS22DSL402

Year: F-B.Tech / SE / TE / BE / M.Tech Sem: I / II / III / IV / V / VI / VII / VIII Batch: D2-2

Department: Computer Science and Engineering (Data Science)

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	Mini Project	A1	A2
Course Outcome	1, 2	1, 2	1, 2	2, 3	2	2	2	2	1, 2	3,4	1,2	3,4
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	3	3	3	3	3	3	3	3	3	3		
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	3	3	3	2	3	2	3	2	3	3		
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	3	3	3	2	3	2	3	2	3	3	–	–
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	3	3	3	2	3	3	3	3	3	3		
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–	3	–	–
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	3	3	3	3	3	3	3	3	3	3	–	–
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–	–	4	–	–
Total	15	15	15	13	15	13	15	13	15	22	6.5	9
Signature of the faculty member	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR	MR

Excellent (3), Good (2), Needs Improvement (1)

Laboratory marks Σ Avg. = 12.9	Mini Project marks Σ Avg. = 22.	Quiz marks Σ Avg. = 15.5	Total Term-work (25) = 22
Laboratory Scaled to (10) = 9.55	Mini Project Scaled to (05) = 4.4	Quiz Scaled to (10) = 7.75	Sign of the Student: Krish

Signature of the Faculty member: MR 6/5
Name of the Faculty member: Dr. Mrunal Rane

Signature of Head of the Department
Date:

Dr. Kriti Srivastava
Head of the Department
Computer Science and Engineering (Data Science)

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Name:	Krish Thakkar
SAP ID:	60009230213
ROLL NO :	D138
Batch:	D2-2
Course	ML-I

Mini Project (Task 1)

Problem Statement: Natural Language Processing with Disaster Tweets

1) Describe your problem in detail and discuss why it is a data science problem.

Problem:

- During disasters, a massive influx of information floods social media, particularly Twitter.
- This information includes a mix of real-time updates, requests for help, and even false information.
- Extracting critical details from this unstructured data (text) is crucial for emergency response teams to effectively allocate resources and aid victims. Here's where Natural Language Processing (NLP) comes in.

Data Science Connection: This problem is ideal for data science because:

Unstructured Data:

Tweets are short, informal messages with slang, emojis, and hashtags. NLP techniques like text cleaning, tokenization, and sentiment analysis are needed to understand the meaning.

Classification and Prediction:

We want to build models that can classify tweets as relevant to the disaster (e.g., reporting damage, requesting help) or not relevant (e.g., jokes, irrelevant news). This involves building machine learning models trained on labelled disaster tweet data.

Information Extraction:

Extracting key details like location, type of damage, and urgency from the tweets requires techniques like named entity recognition (NER) to identify crucial information.

2) Justify that the data chosen is appropriate to build a model to solve the problem.

Key Features of Disaster Tweet Dataset:

The key features of a disaster tweet dataset typically include:

id: A unique identifier for each tweet, helpful for tracking and managing individual data points.

text: The actual text content of the tweet, the core element for NLP analysis.

location: (may be blank) The location data associated with the tweet, crucial for understanding the geographical impact of the disaster.

keyword: (may be blank) A specific keyword extracted from the tweet, potentially indicating disaster relevance (e.g., "hurricane," "earthquake").

target: (in train.csv only) A label indicating whether the tweet is related to a real disaster (1) or not (0). This is critical for supervised machine learning tasks.

Justification for Data Appropriateness:

The chosen data is appropriate for building a disaster tweet classification model for several reasons:

Relevance: The "text" feature directly addresses the core problem of understanding the meaning within tweets. NLP techniques can be applied to analyze the text for keywords, sentiment, and urgency to classify disaster relevance.

Labeling (target): The presence of a "target" label in the training data (train.csv) is essential for supervised learning. The model learns to map the text features to the corresponding disaster relevance labels, enabling it to classify new unseen tweets.

Contextual Information: The "location" feature provides valuable context for the tweets. While it might be missing for some tweets, location data can significantly improve the model's accuracy in pinpointing disaster-affected areas and understanding the geographical spread of the event.

Keyword Guidance: The "keyword" feature, although potentially blank for some tweets, can offer additional clues for disaster relevance. These keywords might be pre-identified disaster terms or extracted during data processing. They can serve as starting points for the NLP model to focus its analysis on disaster-related vocabulary.

Conclusion:

In conclusion, the disaster tweet dataset with text, location (if available), keyword (if available), and target labels provides a solid foundation for building an NLP model to classify tweets regarding real-world disasters. This can empower various disaster response efforts and save lives during critical times.

Dataset Selected Link:

https://www.kaggle.com/competitions/nlp-getting-started/data

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Name:	Krish Thakkar
SAP ID:	60009230213
ROLL NO :	D138
Batch:	D2-2
Course	ML-I

Mini Project (Task 2)

Problem Statement: Natural Language Processing with Disaster Tweets

Perform Exploratory data analysis and do preprocessing on your dataset.

Reading & Cleaning, Preprocessing Dataset

Importing necessary Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
```

Importing dataset

```
# Reading the dataset
df = pd.read_csv('/content/disaster_tweets.csv')

# Displaying the first few rows of the dataset
print(f"Total records : {df.shape[0]} | Total columns : {df.shape[1]}\n")
df.head()
```

Total records : 11370 | Total columns : 5

	id	keyword	location	text	target
0	0	ablaze	NaN	Communal violence in Bhainsa, Telangana. "Ston...	1
1	1	ablaze	NaN	Telangana: Section 144 has been imposed in Bha...	1
2	2	ablaze	New York City	Arsonist sets cars ablaze at dealership https:...	1
3	3	ablaze	Morgantown, WV	Arsonist sets cars ablaze at dealership https:...	1
4	4	ablaze	NaN	"Lord Jesus, your love brings freedom and pard...	0

The dataset consists of the following features/columns/fields/variables:

id: A unique identifier for each tweet.

keyword: A keyword associated with the tweet.

location: The location from where the tweet was sent (it may have missing values).

text: The text content of the tweet.

target: This is our target variable. A value of 1 indicates that the tweet is about a real disaster, while a value of 0 indicates that it's not.

Cleaning Steps

- Check for any missing values in the dataset.
- Handle any duplicates if they exist.
- Clean the text data (remove URLs, special characters, numbers, etc.) to make it suitable for analysis.

```
# Checking for missing values
missing_values = df.isnull().sum()

# Checking for duplicates
duplicates = df.duplicated().sum()

missing_values, duplicates
```

```
(id             0
 keyword        0
 location    3418
 text           0
 target         0
 dtype: int64,
 0)
```

Observations:

- The location column has 3418 missing values.
  Given the large number of missing values and the fact that location data can be very diverse , we might consider dropping this column.
- There are no duplicate entries in the dataset.
- Dropping the location column in below code.
- Clean the text column by:
  - ✓ Removing URLs.
  - ✓ Removing special characters and numbers.
  - ✓ Converting all text to lowercase.

```
# Dropping the 'location' column
df = df.drop(columns=['location'])

# Function to clean the text data
def clean_text(text):
    # Removing URLs
    text = re.sub(r'http\S+', '', text)
    # Removing special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Converting to lowercase
    text = text.lower()
    return text.strip()

# Applying the function to the 'text' column
df['text'] = df['text'].apply(clean_text)

# Displaying the first few cleaned rows
df.head()
```

	id	keyword	text	target
0	0	ablaze	communal violence in bhainsa telangana stones ...	1
1	1	ablaze	telangana section has been imposed in bhainsa...	1
2	2	ablaze	arsonist sets cars ablaze at dealership	1
3	3	ablaze	arsonist sets cars ablaze at dealership	1
4	4	ablaze	lord jesus your love brings freedom and pardon...	0

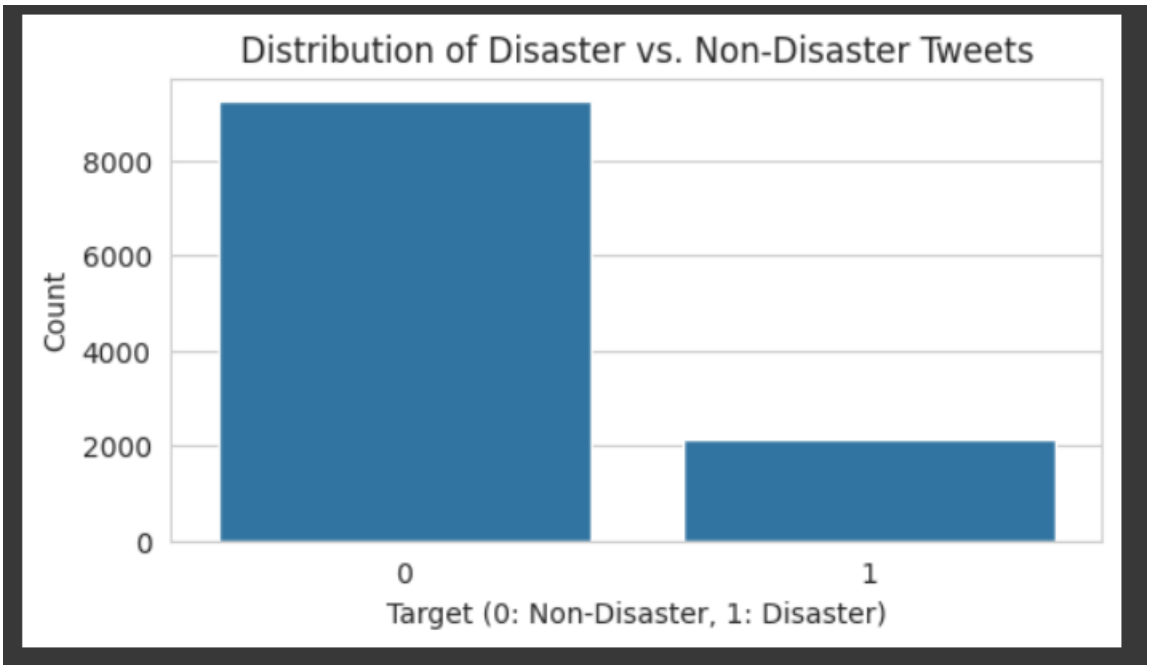
- The location column has been dropped.
- The text column has been cleaned of URLs, special characters, numbers, and all text has been converted to lowercase.

Exploratory Data Analysis (EDA)

Procedure for performing EDA

- Examine the distribution of disaster vs. non-disaster tweets.
- Visualize the top keywords associated with disaster tweets.
- Creating a word-cloud.
- Explore the length of tweets and its potential correlation with disaster classification.

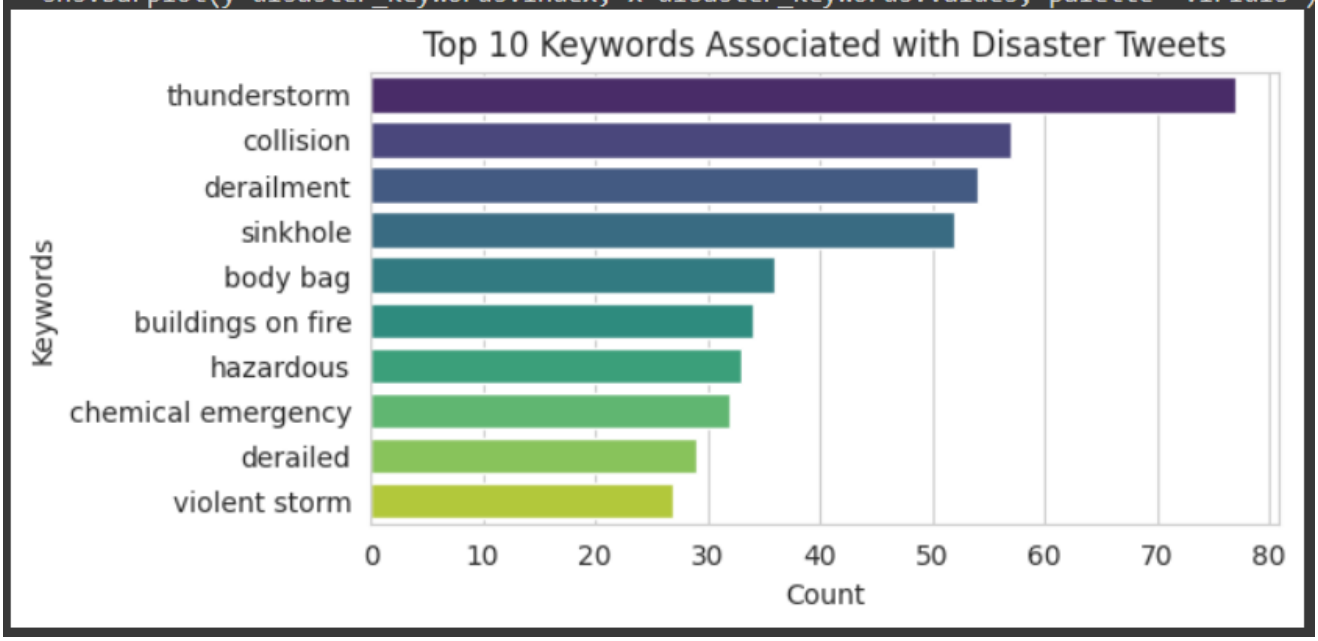
```
# Setting the style for the plots
sns.set_style("whitegrid")

# Plotting the distribution of disaster vs. non-disaster tweets
plt.figure(figsize=(6, 3))
sns.countplot(x='target', data=df)
plt.title("Distribution of Disaster vs. Non-Disaster Tweets")
plt.xlabel("Target (0: Non-Disaster, 1: Disaster)")
plt.ylabel("Count")
plt.show()
```



- Above plot showcases the distribution of disaster vs non-disaster tweets. It's evident that the dataset contains a relatively balanced number of both types of tweets.
- Next, let's visualize the top keywords associated with disaster tweets to understand which terms are commonly associated with genuine disaster situations.

```
# Cleaning the 'keyword' column
df['keyword'] = df['keyword'].str.replace('%20', ' ')

# Filtering the dataset for disaster tweets
disaster_keywords = df[df['target'] == 1]['keyword'].value_counts().head(10)

# Plotting the top keywords associated with disaster tweets
plt.figure(figsize=(6, 3))
sns.barplot(y=disaster_keywords.index, x=disaster_keywords.values,
palette="viridis")
plt.title("Top 10 Keywords Associated with Disaster Tweets")
plt.xlabel("Count")
plt.ylabel("Keywords")
plt.show()
```

```
<ipython-input-7-64a21c161088>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.barplot(y=disaster_keywords.index, x=disaster_keywords.values, palette="viridis")
```

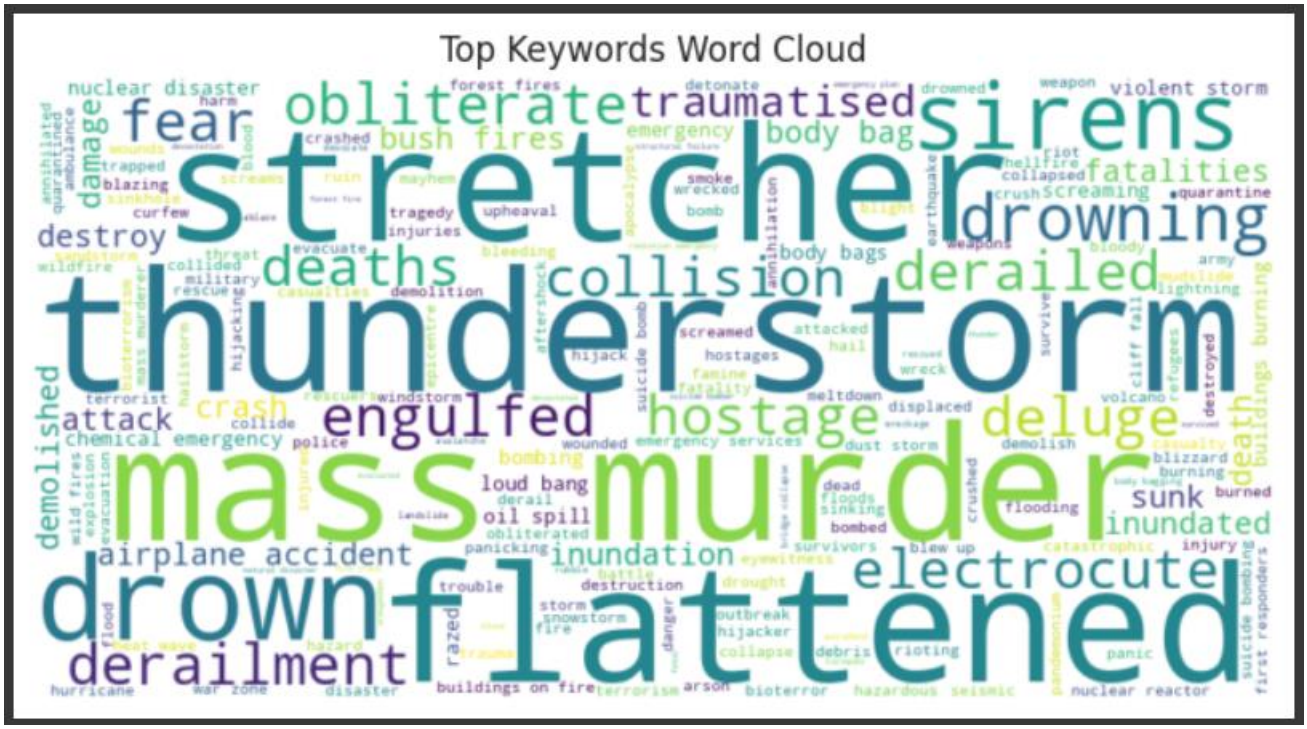


- Above bar plot above showcases the top 10 keywords associated with disaster tweets. These keywords provide insights into the common themes or incidents that are frequently labelled as real disasters in the dataset.
- Next, let's explore the length of tweets and its potential correlation with disaster classification. We'll create a new column for tweet length and visualize its distribution for both disaster and non-disaster tweets.

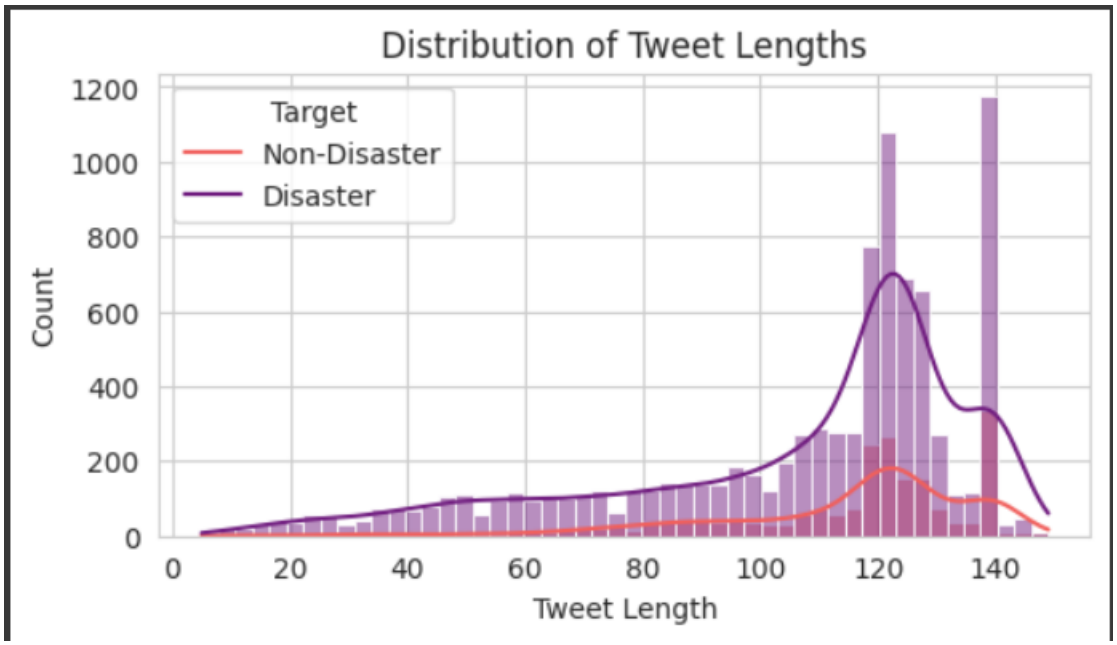
Word Cloud to get TOP words:

```
from wordcloud import WordCloud

# Getting the frequency distribution of the keywords
keywords_freq = df['keyword'].value_counts().to_dict()

wordcloud = WordCloud(width=800, height=400,
background_color='white').generate_from_frequencies(keywords_freq)
```

```
# Plotting the word cloud
plt.figure(figsize=(8, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Top Keywords Word Cloud')
plt.show()
```




```
# Creating a new column for tweet length
df['tweet_length'] = df['text'].apply(len)

# Plotting the distribution of tweet lengths for disaster and non-disaster
tweets
plt.figure(figsize=(6, 3))
sns.histplot(df, x='tweet_length', hue='target', bins=50, kde=True,
palette="magma")
plt.title("Distribution of Tweet Lengths")
plt.xlabel("Tweet Length")
plt.ylabel("Count")
plt.legend(title='Target', labels=['Non-Disaster', 'Disaster'])
plt.show()
```



Sentiment Analysis

Sentiment analysis can provide insights into the general sentiment or emotion behind the tweets. While our main target is to classify tweets as disaster or non-disaster, understanding sentiment can offer additional context.

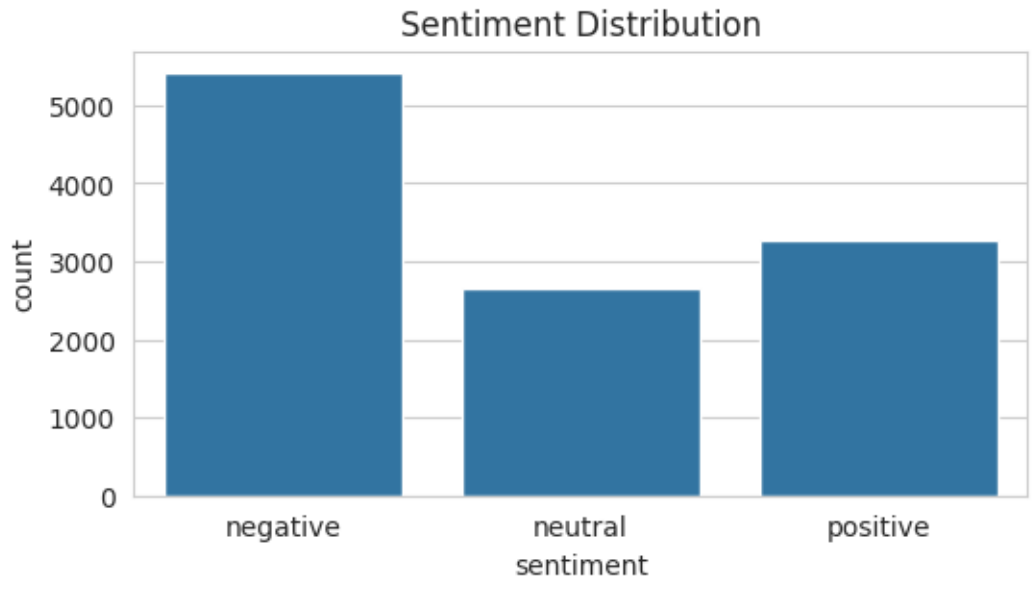
Using the spaCy model.

```
import spacy
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment import SentimentIntensityAnalyzer

# Load spaCy model and VADER sentiment intensity analyzer
nlp = spacy.load("en_core_web_sm")
sid = SentimentIntensityAnalyzer()

# Tokenize the text using spaCy
df['tokenized_text'] = df['text'].apply(lambda x: " ".join([token.text for
token in nlp(x)]))
```

```
# Get sentiment scores using VADER
df['sentiment_score'] = df['tokenized_text'].apply(lambda x:
sid.polarity_scores(x)['compound'])

# Categorize the sentiment based on score
df['sentiment'] = df['sentiment_score'].apply(lambda x: 'positive' if x > 0.05
else ('neutral' if -0.05 <= x <= 0.05 else 'negative'))
print(df['sentiment'].value_counts())
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
sentiment
negative    5457
positive    3297
neutral     2616
Name: count, dtype: int64
```

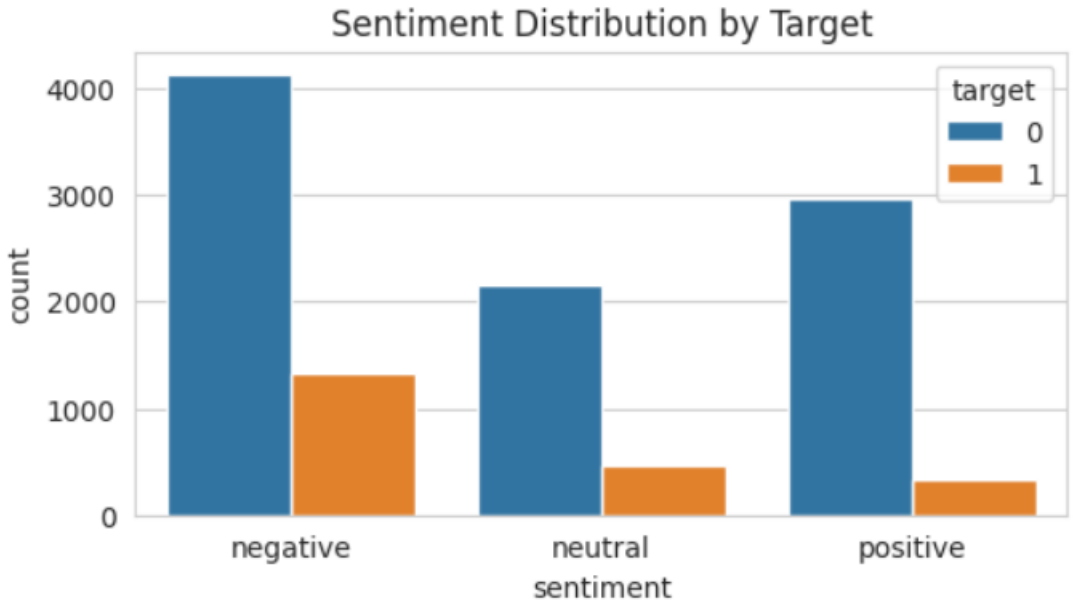
```
# Visualize the sentiment distribution
plt.figure(figsize=(6, 3))
sns.countplot(data=df, x='sentiment')
plt.title("Sentiment Distribution")
plt.show()
```



Insights for above bar-graph:

- Which sentiment is predominant in the dataset? -> Negative
- Are most tweets neutral, or do they lean towards positive/negative? -> Lean towwards negative & then positive

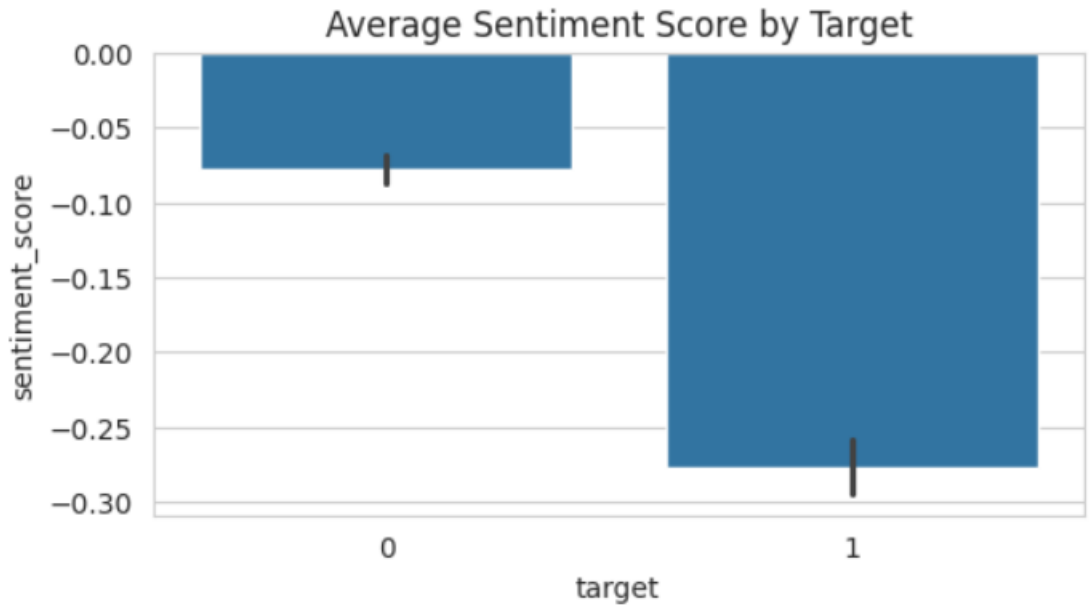
```
# Visualize sentiment distribution by target
plt.figure(figsize=(6, 3))
sns.countplot(data=df, x='sentiment', hue='target')
plt.title("Sentiment Distribution by Target")
plt.show()
```



Insights for above bar-graph:

- Do disaster tweets tend to be more negative compared to non-disaster tweets? -> Non-disaster tweets
- Are non-disaster tweets predominantly positive or neutral? -> Non-disaster negative tweets

```
# Average sentiment score by target
plt.figure(figsize=(6, 3))
sns.barplot(x='target', y='sentiment_score', data=df)
plt.title("Average Sentiment Score by Target")
plt.show()
```

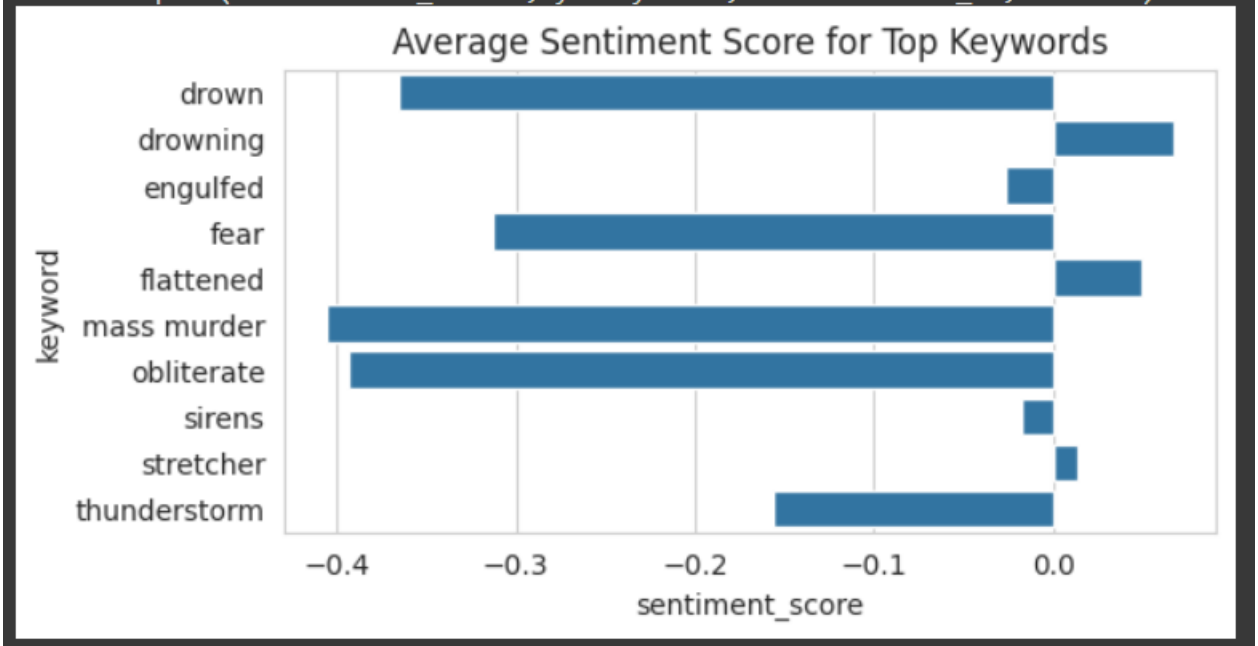


Insights for above bar-graph:

- Is the average sentiment score for disaster tweets lower (more negative) than for non-disaster tweets? -> No

```
# Average sentiment score for top keywords
top_keywords = df['keyword'].value_counts().head(10).index
filtered_df = df[df['keyword'].isin(top_keywords)]

plt.figure(figsize=(6, 3))
sns.barplot(x='sentiment_score', y='keyword', data=filtered_df, ci=None)
plt.title("Average Sentiment Score for Top Keywords")
plt.show()
```

```
<ipython-input-36-e6e793ca7c0d>:6: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='sentiment_score', y='keyword', data=filtered_df, ci=None)
```



Predictive Modelling and preprocessing the data using TF-IDF.

we'll need to preprocess the textual data for machine learning purposes. This involves converting our text data into numerical format. We'll use TF-IDF (Term Frequency-Inverse Document Frequency) for this purpose.

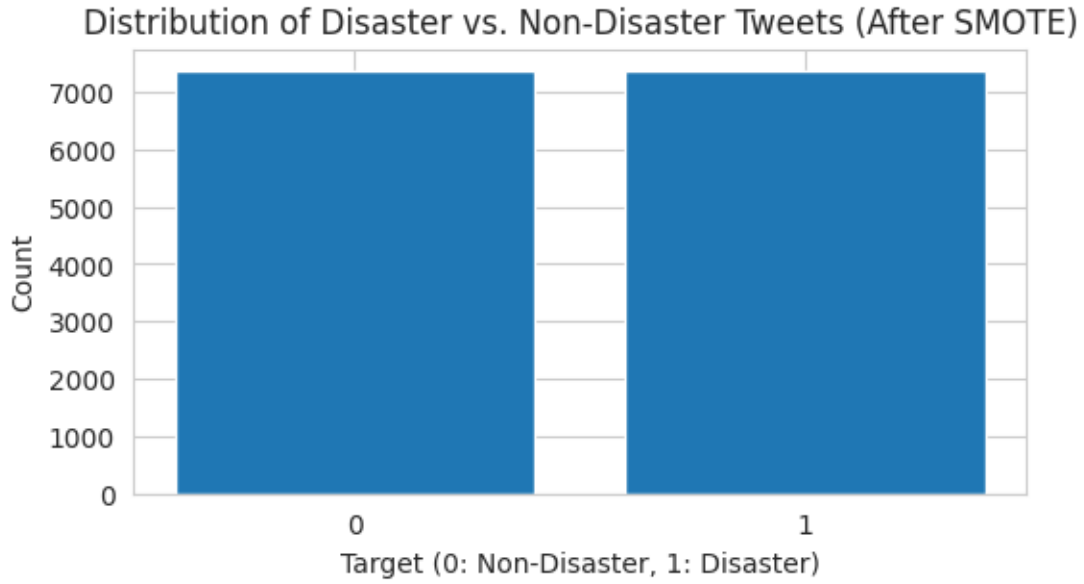
Once the data is pre-processed, we'll:

- Split the data into training and testing sets.
- Apply SMOTE to balance the classes.
- Apply Random Forest, Neural Network & LSTM models.
- Evaluate the performance of both models.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import SMOTE
# Initializing the TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')

# Fitting and transforming the vectorizer on our text data
X = tfidf_vectorizer.fit_transform(df['text'])
y = df['target']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Apply SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

X_train.shape, X_test.shape
```

```
((9096, 5000), (2274, 5000))
```

```
# Checking if class is balanced after using SMOTE
plt.figure(figsize=(6, 3))
plt.bar(['0', '1'], y_train_resampled.value_counts())
plt.title("Distribution of Disaster vs. Non-Disaster Tweets (After SMOTE)")
plt.xlabel("Target (0: Non-Disaster, 1: Disaster)")
plt.ylabel("Count")
plt.show()
```



We can see in above bar-graph that our classes are balanced now & we removed the bias from the data.

Google Collab Link:

https://colab.research.google.com/drive/1iBo3yW6yx4b7rGC8Fxjfqw2sm4NIWxW4?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 1

(Regression)

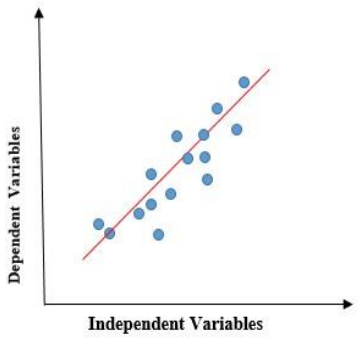
Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: Implement Linear Regression on the given Dataset and apply Regularization to overcome overfitting in the model.

Theory:

- **Linear Regression**: Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. *If there is a single input variable (x), such linear regression is called* ***simple linear regression****. And if there is more than one input variable, such linear regression is called* ***multiple linear regression*****.** The linear regression model gives a sloped straight line describing the relationship within the variables.



The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (**independent variable**) increases, the value of y (**dependent variable**) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best. *To calculate best-fit line linear regression uses a traditional slope-intercept form.*

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

y= Dependent Variable; x= Independent Variable; a0= intercept; a1 = Linear regression coefficient.

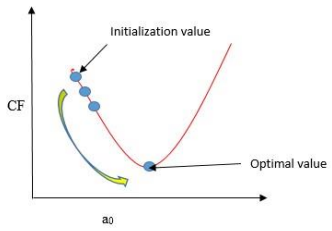
- **Cost function:** The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to find the accuracy of the **mapping function** that maps the input variable to the output variable. This mapping function is also known as **the Hypothesis function**. In Linear Regression, **Mean Squared Error (MSE)** cost function is used, which is the average of squared error that occurred between the predicted values and actual values. *By simple linear equation y=mx+b we can calculate MSE as: Let's y = actual values, y_i = predicted values*

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Model parameters ***xi, b (a_0, a_1)*** can be manipulated to minimize the cost function. These parameters can be determined using the gradient descent method so that the cost function value is minimum.

- **Gradient descent:** Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE). A regression model uses gradient descent to update the coefficients of the line

(a0, a1 => xi, b) by reducing the cost function by a random selection of coefficient values and then iteratively update the values to reach the minimum cost function.



To update a_0 and a_1, we take gradients from the cost function. To find these gradients, we take partial derivatives for a_0 and a_1.

$$J = \frac{1}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)^2$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(a_0 + a_1 \cdot x_i - y_i) \cdot x_i$$

$$\frac{\partial J}{\partial a_0} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$\frac{\partial J}{\partial a_1} = \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

$$a_0 = a_0 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i)$$

$$a_1 = a_1 - \alpha \cdot \frac{2}{n}\sum_{i=1}^{n}(pred_i - y_i) \cdot x_i$$

- **Regularization:** When linear regression is underfitting there is no other way (given you can't add more data) then to increase complexity of the model making it polynomial regression (cubic,

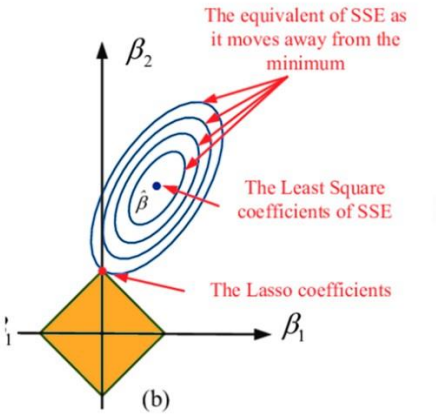
quadratic, etc…) or using other complex model to capture data that linear regression cannot capture due to its simplicity. When linear regression is overfitting, number of columns(independent variables) approach number of observations there are two ways to mitigate it

1. Add more observations
2. Regularization

Since adding more observations is time consuming and often not provided we will use regularization technique to mitigate overfitting. There are multiple regularization techniques, all share the same concept of **adding constraints on weights** of independent variables(except theta_0) however they differ in way of constraining. We will go through three most popular regularization techniques: Ridge regression (L2) and Lasso regression (L1)

- **Lasso Regression**

The word "LASSO" denotes Least Absolute Shrinkage and Selection Operator. Lasso regression follows the regularization technique to create prediction. It is given more priority over the other regression methods because it gives an accurate prediction. Lasso regression model uses shrinkage technique. In this technique, the data values are shrunk towards a central point similar to the concept of mean. The lasso regression algorithm suggests a simple, sparse models (i.e. models with fewer parameters), which is well-suited for models or data showing high levels of multicollinearity or when we would like to automate certain parts of model selection, like variable selection or parameter elimination using feature engineering. Lasso Regression algorithm utilises L1 regularization technique It is taken into consideration when there are more number of features because it automatically performs feature selection.

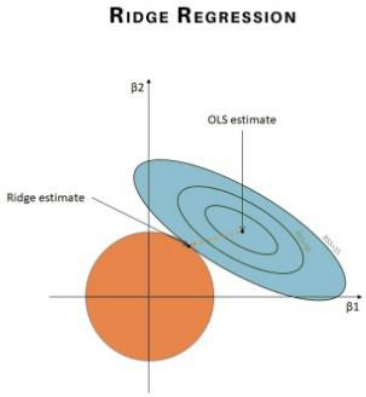


Residual Sum of Squares + λ * (Sum of the absolute value of the coefficients) The equation looks like:

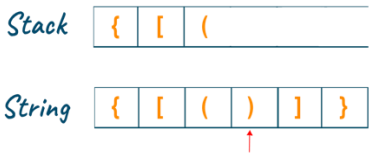
$$\sum_{i=1}^{n}(y_i - \sum_{j} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **Ridge Regression**

Ridge Regression is another type of regression algorithm in data science and is usually considered when there is a high correlation between the independent variables or model parameters. As the value of correlation increases the least square estimates evaluates unbiased values. But if the collinearity in the dataset is very high, there can be some bias value. Therefore, we create a bias matrix in the equation of Ridge Regression algorithm. It is a useful regression method in which the model is less susceptible to overfitting and hence the model works well even if the dataset is very small.



The cost function for ridge regression algorithm is:



Where λ is the penalty variable. λ given here is denoted by an alpha parameter in the ridge function. Hence, by changing the values of alpha, we are controlling the penalty term. Greater the values of alpha, the higher is the penalty and therefore the magnitude of the coefficients is reduced.We can conclude that it shrinks the parameters. Therefore, it is used to prevent multicollinearity, it also reduces the model complexity by shrinking the coefficient.

Department of Computer Science and Engineering (Data Science)

Lab Assignments to complete in this session

Use the given dataset and perform the following tasks:

Dataset 1: Simulate a sine curve between 60° and 300° with some random noise.

Dataset 2: food_truck_data.csv

Dataset 3: home_data.csv

1. **Perform Linear Regression on Dataset 1 and Dataset 2 by computing cost function and gradient descent from scratch.**

Ans.

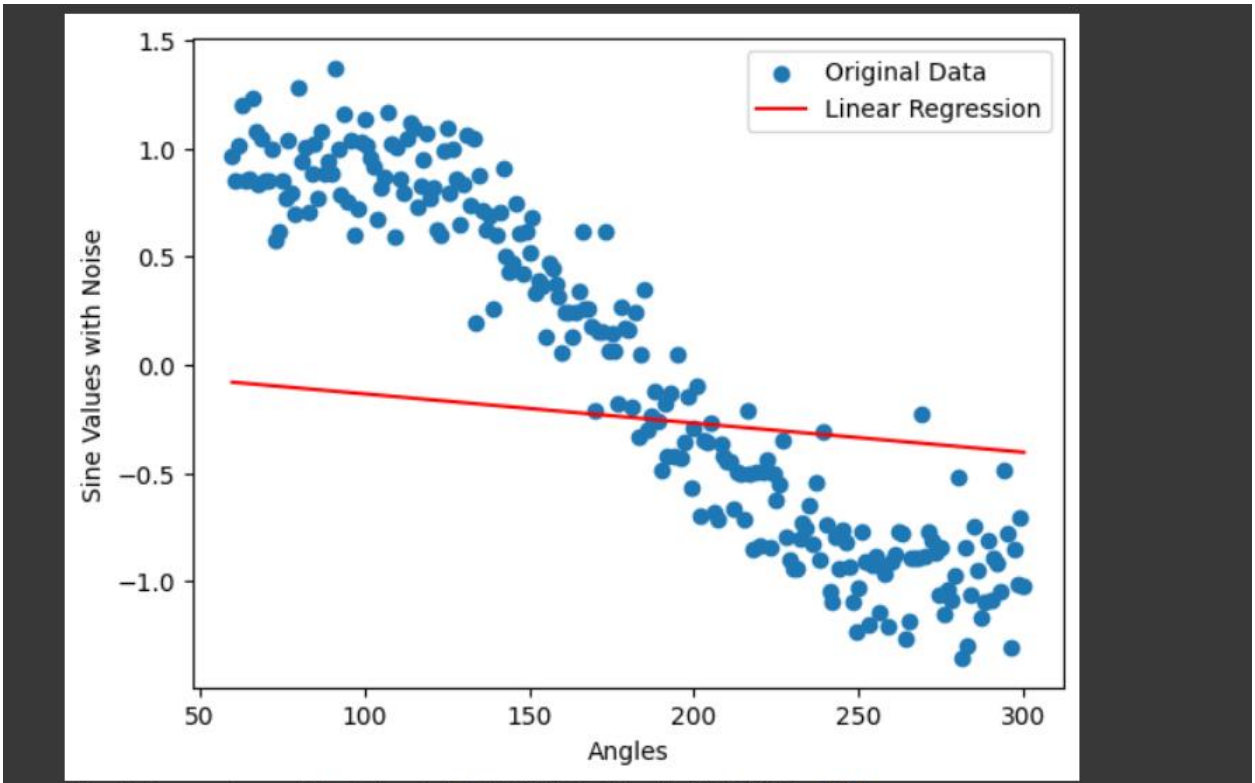
```
import pandas as pd
import numpy as np
```

```
import pandas as pd
import numpy as np
np.random.seed(42)
angles = np.arange(60, 301, 1)
sine = np.sin(np.radians(angles))
noise = np.random.normal(0, 0.2, len(angles))
sine_curve = sine+noise
dataset = pd.DataFrame({'Angle': angles, 'Sine_Value': sine_curve})
dataset.to_csv('sine_curve_dataset.csv', index=False)
```

```
import matplotlib.pyplot as plt


# Linear Regression with Gradient Descent
def compute_cost(X, y, theta):
    m = len(y)
    predictions = X.dot(theta)
    cost = (1/(2*m)) * np.sum(np.square(predictions - y))
    return cost
```

```
def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)

    for i in range(iterations):
        predictions = X.dot(theta)
        errors = predictions - y
        theta = theta - (1/m) * learning_rate * (X.T.dot(errors))

    return theta

# Prepare data for linear regression
X = np.vstack((np.ones(len(angles)), angles)).T
y = sine_curve

# Initialize theta (parameters)
theta_initial = np.zeros(2)

# Set hyperparameters
learning_rate = 0.000000001
iterations = 100000

# Perform gradient descent
theta_final = gradient_descent(X, y, theta_initial, learning_rate,
iterations)

# Plot the original data and the regression line
plt.scatter(angles, sine_curve, label='Original Data')
plt.plot(angles, X.dot(theta_final), color='red', label='Linear
Regression')
plt.xlabel('Angles')
plt.ylabel('Sine Values with Noise')
plt.legend()
plt.show()

# Print the final parameters
print('Final Parameters (theta):', theta_final)
```

Output:

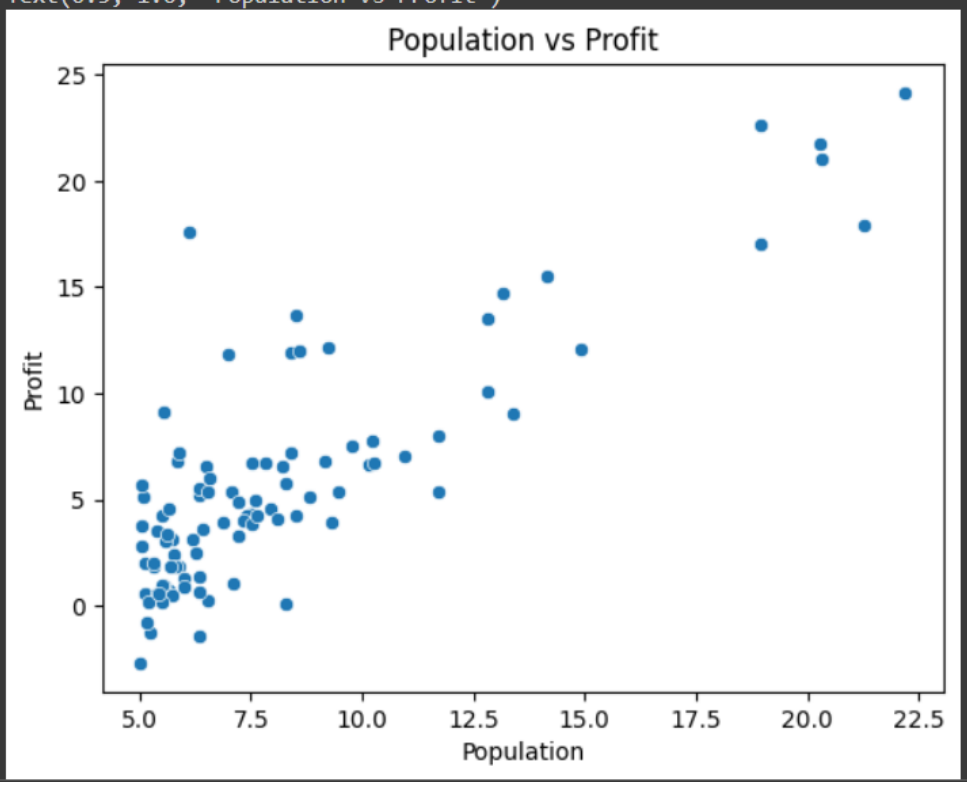



```
df2 = pd.read_csv('/home/food_truck_data.txt')
df2.head()
```

```
import seaborn as sns
ax= sns.scatterplot(x= "Population" , y= "Profit" , data=df2)
ax.set_title("Population vs Profit")
```

Output:

```
Text(0.5, 1.0, 'Population vs Profit')
```




```
def cost_function(X,y,theta):
  m= len(y)
  y_pred= X.dot(theta)
  error= (y_pred - y) **2

  return 1/(2*m) * np.sum(error)
m = df2.Population.size
X= np.append(np.ones((m,1)), df2.Population.values.reshape(m,1),axis=1)
y= df2.Profit.values.reshape(m,1)
theta= np.zeros((2,1))
cost_function(X,y,theta)
```

```
32.072733877455676
```

```
def gradient_descent(X,y,theta,alpha,iterations):
  m= len(y)
  costs=[]

  for i in range(iterations):
    y_pred= X.dot(theta)
    error= np.dot(X.transpose(), (y_pred - y))

    theta -= alpha * 1/m * error
    costs.append(cost_function(X,y,theta))

    return theta,costs
theta,costs = gradient_descent(X,y,theta,alpha= 0.01, iterations=10)
print("h(x)= {} +
{}x1".format(str(round(theta[0,0],2)),str(round(theta[1,0],2))))
```

```
h(x)= 0.06 + 0.65x1
```

2. Use sklearn to perform linear regression on Dataset 2, show the scatter plot for best fit line using matplotlib and show the results using MSE.

Ans.

```
from sklearn.model_selection import train_test_split
X = df2.drop('Profit',axis = 1)
X_train,X_test,y_train,y_test = train_test_split(X,df2['Profit'],test_size
= 0.3,random_state = 42)
```

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X_train,y_train)
reg.score(X_test,y_test)
```

Output:

```
0.5905441260648103
```

```
predictions = reg.predict(X)
```

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y, predictions)
print("Mean Squared Error:", mse)
```

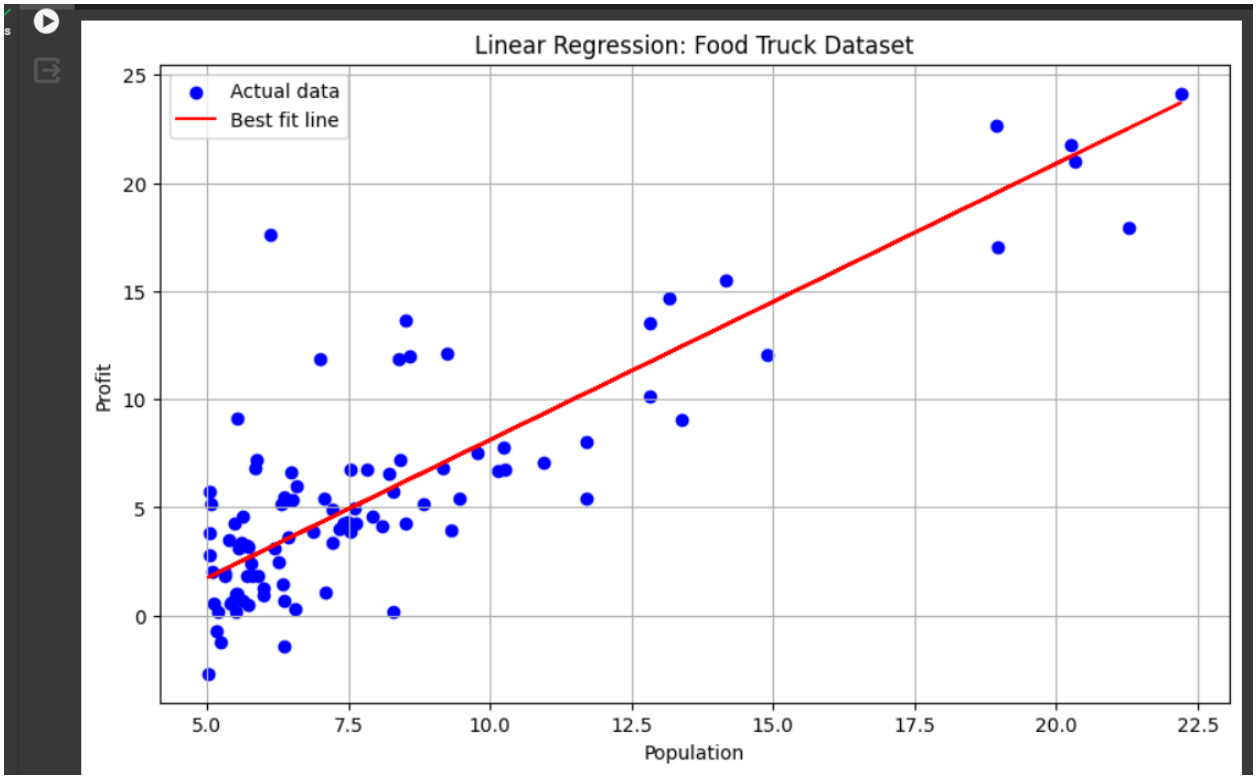
Output:

```
Mean Squared Error: 9.064384699744403
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, predictions, color='red', label='Best fit line')
plt.xlabel('Population')
plt.ylabel('Profit')
plt.title('Linear Regression: Food Truck Dataset')
```

```
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Department of Computer Science and Engineering (Data Science)

3. To perform regularization on linear model build using Linear Regression on Dataset3.

Ans.

```
df2 = pd.read_csv("/home/home_data.csv")
df2.head()
df2.columns
```

Output:

```
[39] df2.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built	yr_renovated	zipcode	
0	7129300520	20141013T000000	221900	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955	0	98178	47.5
1	6414100192	20141209T000000	538000	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951	1991	98125	47.7
2	5631500400	20150225T000000	180000	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933	0	98028	47.7
3	2487200875	20141209T000000	604000	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965	0	98136	47.5
4	1954400510	20150218T000000	510000	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987	0	98074	47.6

5 rows × 21 columns

```
df2.columns

Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
col_to_be_used = ['bedrooms','bathrooms','sqft_living']
df1 = df2[col_to_be_used]
df1.head()
```

Output:

	bedrooms	bathrooms	sqft_living
0	3	1.00	1180
1	3	2.25	2570
2	2	1.00	770
3	4	3.00	1960
4	3	2.00	1680

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(df1,df2['price'],test_size = 0.3,random_state = 42)
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X_train,y_train)
reg.score(X_test,y_test)
```

Output:

```
0.4978201421834113
```

```
reg.score(X_test,y_test)
```

Output:

```
0.4978201421834113
```

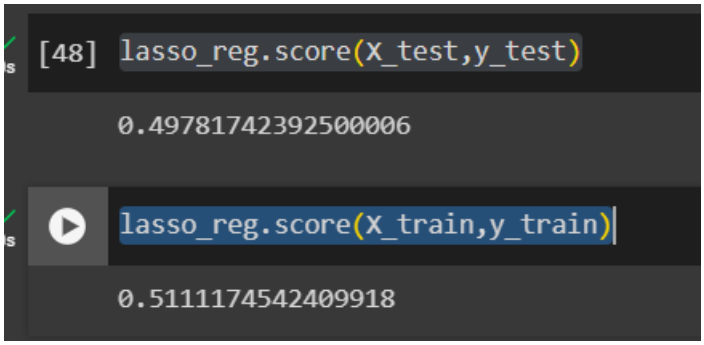
```
from sklearn import linear_model
lasso_reg = linear_model.Lasso(alpha = 50,max_iter = 100,tol = 0.1)
lasso_reg.fit(X_train,y_train)
```

Output:

```
Lasso
Lasso(alpha=50, max_iter=100, tol=0.1)
```

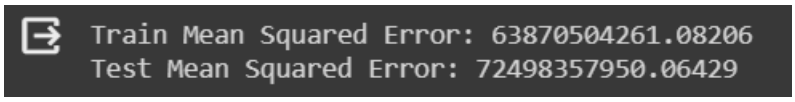
```
lasso_reg.score(X_test,y_test)
lasso_reg.score(X_train,y_train)
```

Output:



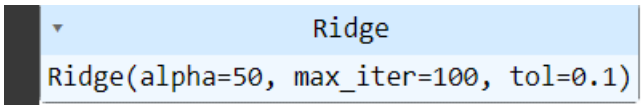

```
train_predictions = lasso_reg.predict(X_train)
test_predictions = lasso_reg.predict(X_test)
train_mse = mean_squared_error(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
print("Train Mean Squared Error:", train_mse)
print("Test Mean Squared Error:", test_mse)
```

Output:




```
ridge_reg = linear_model.Ridge(alpha = 50,max_iter = 100,tol = 0.1)
ridge_reg.fit(X_train,y_train)
```

Output:



```
ridge_reg.score(X_test,y_test)
ridge_reg.score(X_train,y_train)
```

Output:

```
[52] ridge_reg.score(X_test,y_test)

     0.49781207108257

[53] ridge_reg.score(X_train,y_train)

     0.5111169913547382
```

```
train_predictions = ridge_reg.predict(X_train)
test_predictions = ridge_reg.predict(X_test)
train_mse = mean_squared_error(y_train, train_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
print("Train Mean Squared Error:", train_mse)
print("Test Mean Squared Error:", test_mse)
```

Output:

```
Train Mean Squared Error: 63870564735.277115
Test Mean Squared Error: 72499130721.37305
```

Conclusion:

In summary, the experiment demonstrated the application of linear regression and regularization techniques to address overfitting. By implementing gradient descent from scratch and utilizing sklearn for linear regression, the experiment provided hands-on experience in building predictive models. Regularization methods like Lasso and Ridge regression were employed to improve model generalization. Overall, the experiment enhanced understanding of linear regression fundamentals and their practical implications in machine learning tasks.

Google Collab Code Link:

https://colab.research.google.com/drive/1_-VqQQPkUjiEztlJPIwJhGAgbom247WM?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 2 - 3

(Decision Tree)

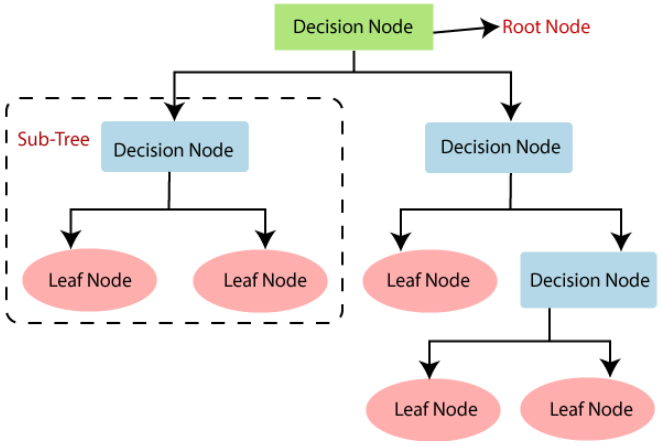
Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: Implement Decision Tree on the given Datasets to build a classifier and Regressor. Apply appropriate pruning method to overcome overfitting.

Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.** In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.**
Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
The decisions or the test are performed on the basis of features of the given dataset.
It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:



Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Steps in building a Tree

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

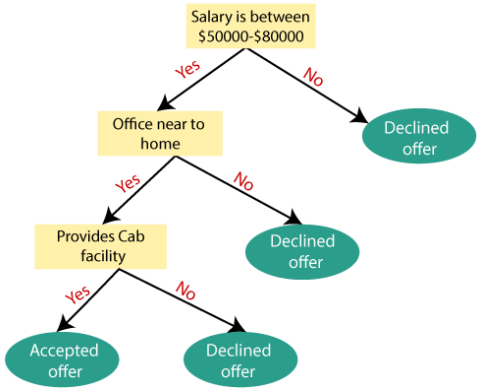
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1. Information Gain:

Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class.

According to the value of information gain, we split the node and build the decision tree.

A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data.

Entropy can be calculated as:

$$Entropy(s)= -P(yes)\log_2 P(yes)- P(no) \log_2 P(no)$$

Where,

S= Total number of samples

P(yes)= probability of yes

P(no)= probability of no

2. Gini Index:

Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.

An attribute with the low Gini index should be preferred as compared to the high Gini index.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree. A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used: • Cost Complexity Pruning

- Reduced Error Pruning.

Department of Computer Science and Engineering (Data Science)

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Iris.csv
Dataset 2: Breastcancer.csv
Dataset 3: car prediction.csv
Dataset 4: car prediction.csv

Q.1 Decision Tree on Playtennis from scratch and sklearn
Ans.

```
import pandas as pd
df=pd.read_csv('/content/PlayTennis.csv')
df.head()
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

```
df['Play Tennis']=df['Play Tennis'].replace({'Yes':1,'No':0})
df
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	0
1	Sunny	Hot	High	Strong	0
2	Overcast	Hot	High	Weak	1
3	Rain	Mild	High	Weak	1
4	Rain	Cool	Normal	Weak	1
5	Rain	Cool	Normal	Strong	0
6	Overcast	Cool	Normal	Strong	1
7	Sunny	Mild	High	Weak	0
8	Sunny	Cool	Normal	Weak	1
9	Rain	Mild	Normal	Weak	1
10	Sunny	Mild	Normal	Strong	1
11	Overcast	Mild	High	Strong	1
12	Overcast	Hot	Normal	Weak	1
13	Rain	Mild	High	Strong	0

```
import pandas as pd
import math

def calculate_entropy_outlook(df, outlook_value):

    filtered_df = df[df['Outlook'] == outlook_value]

    entropy = 0
    total_count = len(filtered_df)
    if total_count == 0:
        return entropy
    value_counts = filtered_df['Play Tennis'].value_counts()
    for count in value_counts:
        probability = count / total_count
        entropy -= probability * math.log2(probability)

    return entropy


outlooks = df['Outlook'].unique()
for outlook in outlooks:
    entropy = calculate_entropy_outlook(df, outlook)
    print(f"Entropy for Outlook '{outlook}': {entropy}")
```

```
Entropy for Outlook 'Sunny': 0.9709505944546686
Entropy for Outlook 'Overcast': 0.0
Entropy for Outlook 'Rain': 0.9709505944546686
```

```
import pandas as pd
import math

def calculate_entropy_temperature(df, temperature_value):

    filtered_df = df[df['Temperature'] == temperature_value]
    entropy = 0
    total_count = len(filtered_df)
    if total_count == 0:
        return entropy
    value_counts = filtered_df['Play Tennis'].value_counts()
    for count in value_counts:
        probability = count / total_count
        entropy -= probability * math.log2(probability)

    return entropy

temperatures = df['Temperature'].unique()
for temp in temperatures:
    entropy = calculate_entropy_temperature(df, temp)
    print(f"Entropy for Temperature '{temp}': {entropy}")
```

```
Entropy for Temperature 'Hot': 1.0
Entropy for Temperature 'Mild': 0.9182958340544896
Entropy for Temperature 'Cool': 0.8112781244591328
```

```
import pandas as pd
import math

def calculate_entropy_humidity(df, humidity_value):

    filtered_df = df[df['Humidity'] == humidity_value]
    entropy = 0
    total_count = len(filtered_df)
    if total_count == 0:
        return entropy
    value_counts = filtered_df['Play Tennis'].value_counts()
    for count in value_counts:
        probability = count / total_count
        entropy -= probability * math.log2(probability)

    return entropy

humidities = df['Humidity'].unique()
for humidity in humidities:
    entropy = calculate_entropy_humidity(df, humidity)
    print(f"Entropy for Humidity '{humidity}': {entropy}")
```

```
Entropy for Humidity 'High': 0.9852281360342515
Entropy for Humidity 'Normal': 0.5916727785823275
```

```
import pandas as pd
import math

def calculate_entropy_wind(df, wind_value):

    filtered_df = df[df['Wind'] == wind_value]


    entropy = 0
    total_count = len(filtered_df)
    if total_count == 0:
        return entropy
    value_counts = filtered_df['Play Tennis'].value_counts()
    for count in value_counts:
        probability = count / total_count
```

```
        entropy -= probability * math.log2(probability)

    return entropy

winds = df['Wind'].unique()
for wind in winds:
    entropy = calculate_entropy_wind(df, wind)
    print(f"Entropy for Wind '{wind}': {entropy}")
```

```
Entropy for Wind 'Weak': 0.8112781244591328
Entropy for Wind 'Strong': 1.0
```

```
import pandas as pd

def calculate_gini_index(df, attribute):
    gini_index = 0
    total_instances = len(df)

    attribute_values = df[attribute].unique()

    for value in attribute_values:
        subset = df[df[attribute] == value]
        proportion = len(subset) / total_instances
        gini_index += proportion * (1 - (subset['Play
Tennis'].value_counts(normalize=True) ** 2).sum())

    return gini_index

attributes = ['Outlook', 'Temperature', 'Humidity', 'Wind']
for attribute in attributes:
    gini_index = calculate_gini_index(df, attribute)
    print(f"Gini Index for {attribute}: {gini_index}")
```

```
Gini Index for Outlook: 0.34285714285714286
Gini Index for Temperature: 0.44047619047619047
Gini Index for Humidity: 0.3673469387755103
Gini Index for Wind: 0.42857142857142855
```

```
best_attribute = None
best_gini = float('inf')
for attribute in attributes:
    gini = calculate_gini_index(df, attribute)
    if gini < best_gini:
        best_attribute = attribute
        best_gini = gini

print(f"Best attribute to use as root node: {best_attribute}")
```

```
Best attribute to use as root node: Outlook
```

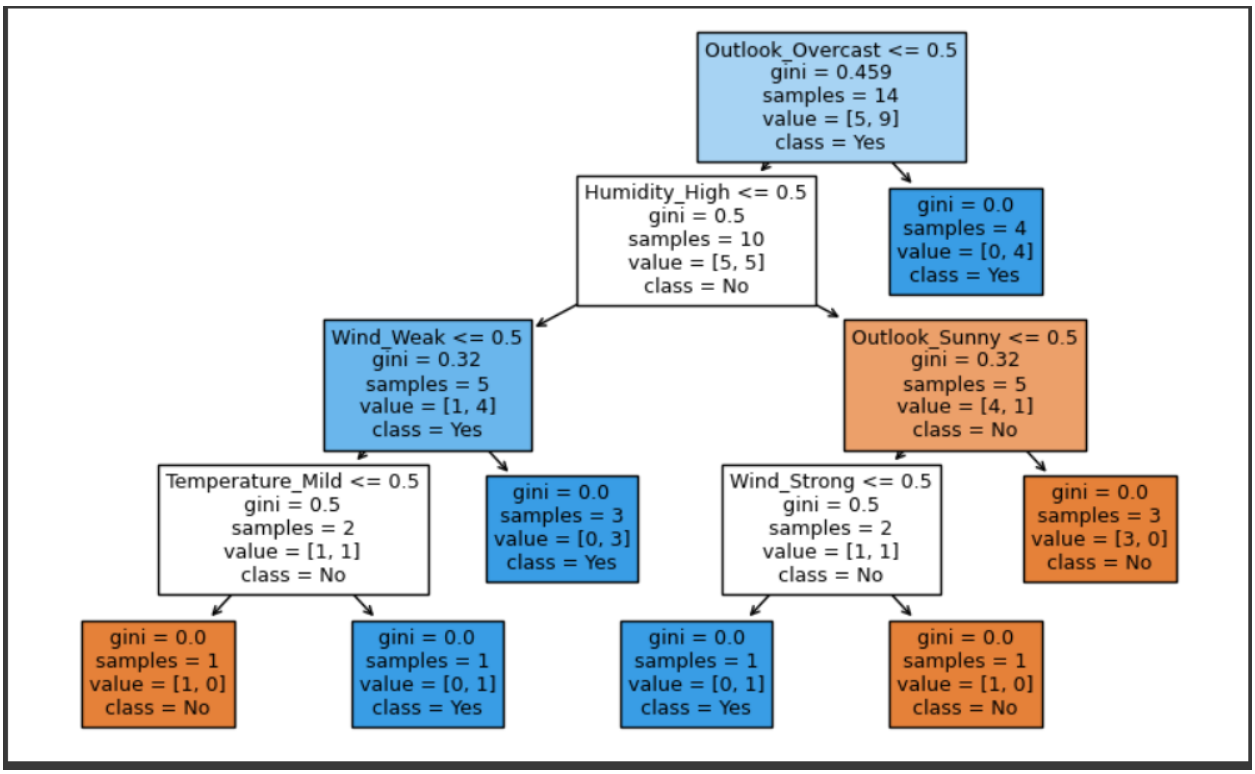
```
class Node:
    def __init__(self, attribute=None, value=None, results=None,
true_branch=None, false_branch=None):
        self.attribute = attribute
        self.value = value
        self.results = results
        self.true_branch = true_branch
        self.false_branch = false_branch

def build_decision_tree(data):
    if len(set(data['Play Tennis'])) == 1:
        return Node(results=data['Play Tennis'].value_counts().to_dict())
    if len(data.columns) == 1:
        return Node(results=data['Play Tennis'].value_counts().to_dict())
    best_attribute = min(data.columns[:-1], key=lambda att:
calculate_gini_index(data, att))
    root = Node(attribute=best_attribute)
    for value in data[best_attribute].unique():
        true_data = data[data[best_attribute] == value]
        false_data = data[data[best_attribute] != value]
        root.true_branch = build_decision_tree(true_data)
        root.false_branch = build_decision_tree(false_data)
    return root

def predict(node, instance):
    if node.results is not None:
        return max(node.results, key=node.results.get)
```

```
    else:
        if instance[node.attribute] == node.value:
            return predict(node.true_branch, instance)
        else:
            return predict(node.false_branch, instance)


def print_tree(node, depth=0):
    if node.results is not None:
        print('  ' * depth + str(node.results))
    else:
        print('  ' * depth + node.attribute + ' = ' + str(node.value) + '
:')

        print_tree(node.true_branch, depth + 1)
        print_tree(node.false_branch, depth + 1)

root_node = build_decision_tree(df)
print_tree(root_node)
```

```
Outlook = None :
  Wind = None :
    {0: 2}
    {1: 3}
  Outlook = None :
    {1: 4}
    Humidity = None :
      {1: 2}
      {0: 3}
```

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

df_encoded = pd.get_dummies(df)

X = df_encoded.drop(columns=['Play Tennis'])
y = df_encoded['Play Tennis']
```

```
clf = DecisionTreeClassifier(criterion='gini', random_state=40)
clf.fit(X, y)

plt.figure(figsize=(10, 6))
plot_tree(clf, feature_names=X.columns, class_names=['No', 'Yes'],
filled=True)
plt.show()
```



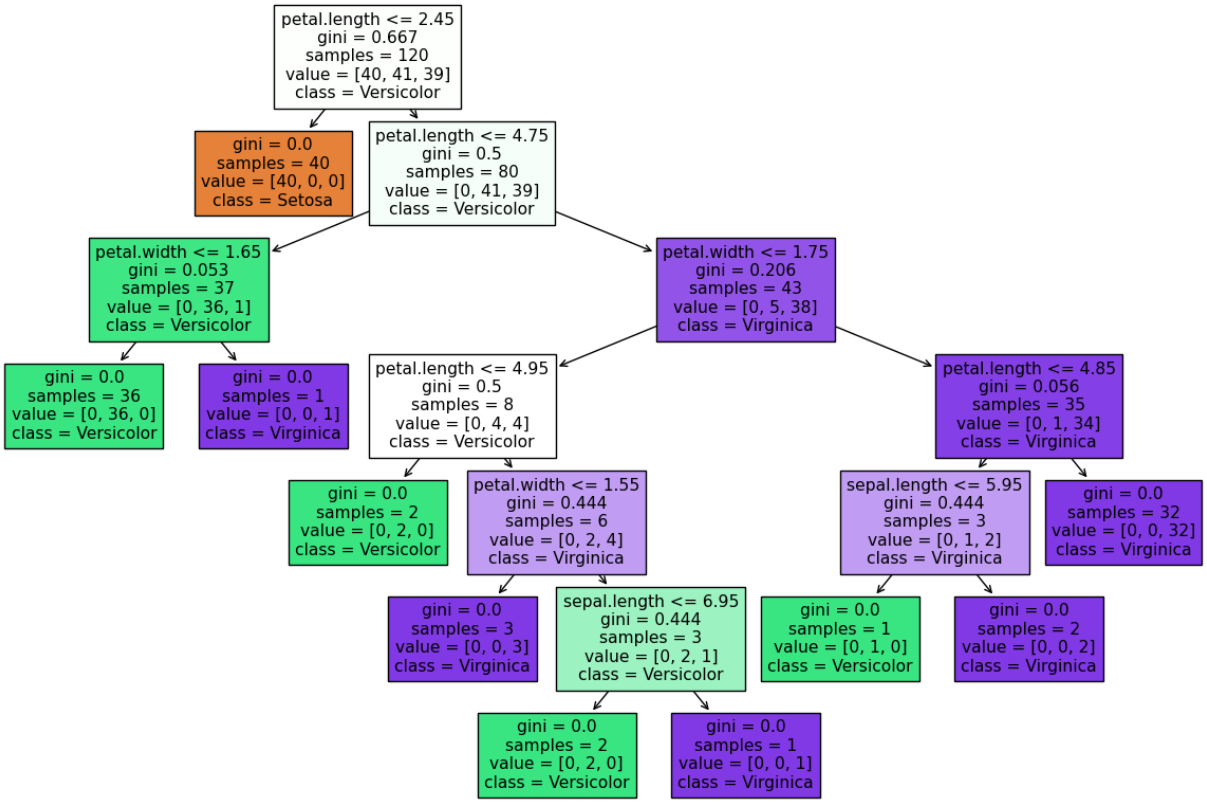
Q.2. Iris.csv using sklearn and add confusion matrix
Ans.

```
import pandas as pd
df2=pd.read_csv('/content/iris.csv')
df2
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica
147	6.5	3.0	5.2	2.0	Virginica
148	6.2	3.4	5.4	2.3	Virginica
149	5.9	3.0	5.1	1.8	Virginica

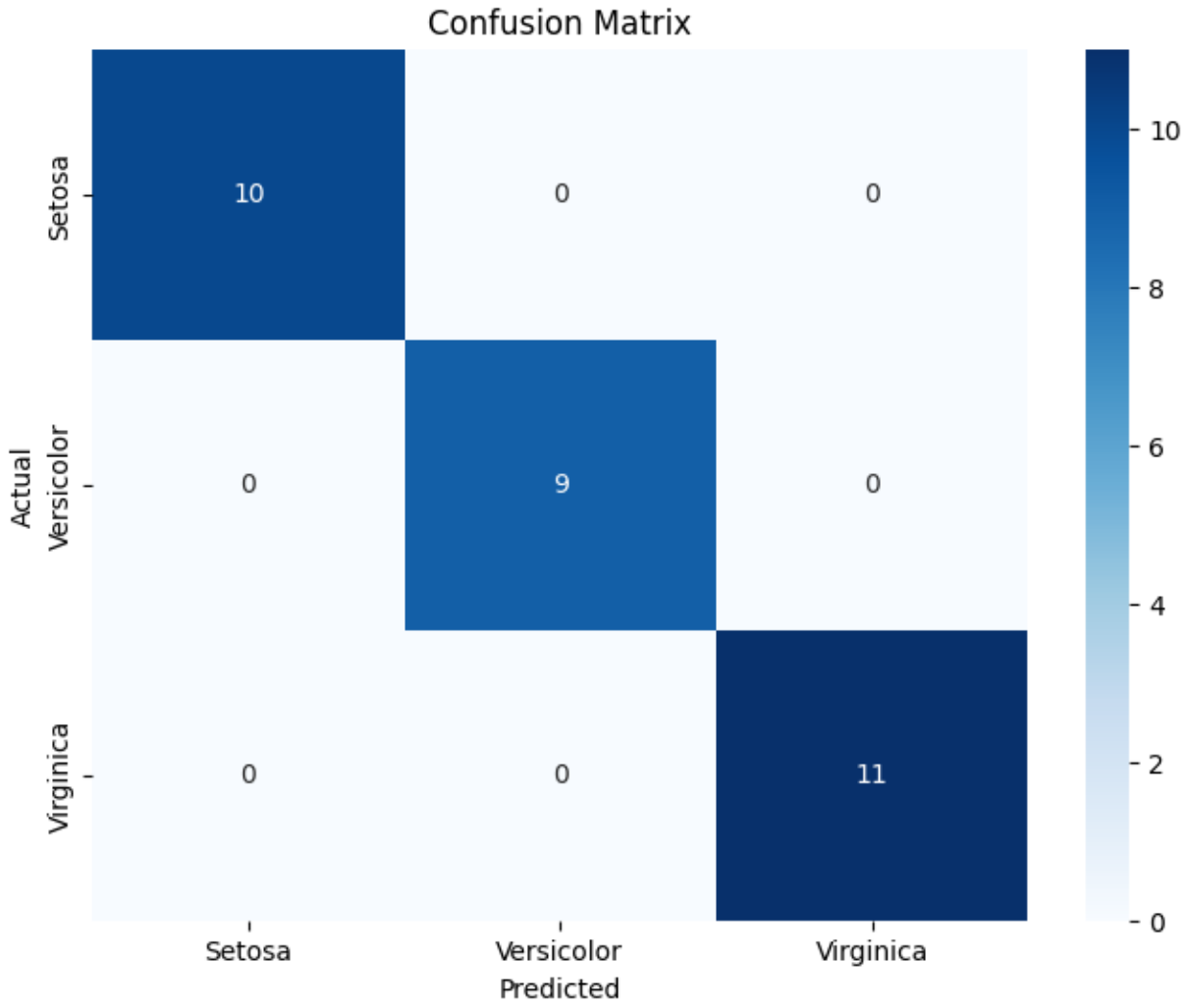
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

X = df2.drop(columns=['variety'])
y = df2['variety']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

plt.figure(figsize=(15, 10))
plot_tree(clf, feature_names=X.columns, class_names=clf.classes_,
filled=True)
plt.show()
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=clf.classes_, yticklabels=clf.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
plt.title('Confusion Matrix')
plt.show()
```



Q.3. Check Overfitting on breastcancer.csv
Ans.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer

#Load Dataset
df_breast_cancer = pd.read_csv('/content/data.csv')
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(df_breast_cancer.drop(columns=['diagnosis']))
y = df_breast_cancer['diagnosis']

#Train,Test and Accuracy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)

# Check for overfitting
if train_accuracy - test_accuracy > 0.2:
    print("Model may be overfitting to the training data.")
else:
    print("Model is not overfitting.")
```

```
Training Accuracy: 1.0
Testing Accuracy: 0.9385964912280702
Model is not overfitting.
```

Q.4 Use regressor on carprediction.csv
Ans.

```
df_car = pd.read_csv('/content/OLX_Car_Data_CSV.csv', encoding='latin1')
df_car.head(5)
```

	Brand	Condition	Fuel	KMs Driven	Model	Price	Registered City	Transaction Type	Year
0	Toyota	Used	Diesel	1.0	Prado	2100000	Karachi	Cash	1997.0
1	Suzuki	Used	Petrol	100000.0	Bolan	380000	Karachi	Cash	2006.0
2	Suzuki	Used	CNG	12345.0	Bolan	340000	Karachi	Cash	1998.0
3	Suzuki	Used	Petrol	94000.0	Alto	535000	Karachi	Cash	2010.0
4	Toyota	Used	Petrol	100000.0	Corolla XLI	1430000	Karachi	Cash	2013.0

```
      import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score


df_car = pd.read_csv('/content/OLX_Car_Data_CSV.csv', encoding='latin1')
df_car_numeric = df_car.select_dtypes(include=['number'])

imputer = SimpleImputer(strategy='mean')
df_car_imputed = pd.DataFrame(imputer.fit_transform(df_car_numeric),
columns=df_car_numeric.columns)
X = df_car_imputed.drop(columns=['Price'])
y = df_car_imputed['Price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
Mean Squared Error: 1953191182282.7637
R-squared: 0.06765135034739511
```

Conclusion:

After implementing Decision Trees for classification and regression tasks on the datasets, pruning was crucial to overcome overfitting, resulting in improved generalization performance. Pruning helped balance model complexity, leading to more robust classifiers and regressors with better accuracy and predictive power.

Collab Link:
https://colab.research.google.com/drive/1MYD0ignTJAcRfXA-SqntY2Rve0G2UKQ4?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 4 (Naïve

Bayes Classifier)

Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: Implement Naïve Bayes Classifier on a given Dataset.

Theory:

Naïve Bayes Classifier Algorithm ○ Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.

- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
P(B) is Marginal Probability: Probability of Evidence.
Types of Naïve Bayes Model:
There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.
  The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Breastcancer.csv
Dataset 2: Social_Network_Ads.csv
Dataset 3: german_credit_data.csv

1. **Perform required preprocessing on Dataset 1 and fit a Naïve Bayes classifier built from scratch. Evaluate the f1 score of classifiers built for categorical and continuous features.**

Ans.

Name: Krish Thakkar
SAP: 60009230213
Batch : D2-2
Lab: ML Lab 4

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ]  !ls
```

```
     'Breast_cancer_data (1).csv'   german_credit_data.csv   sample_data   Social_Network_Ads.csv
```

```
[ ]  df = pd.read_csv('Breast_cancer_data (1).csv')
```

```
[ ]  df.head(5)
```

```
[ ] df = pd.read_csv('Breast_cancer_data (1).csv')
```

```
[ ] df.head(5)
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

```
[ ] df.isna().sum()
```

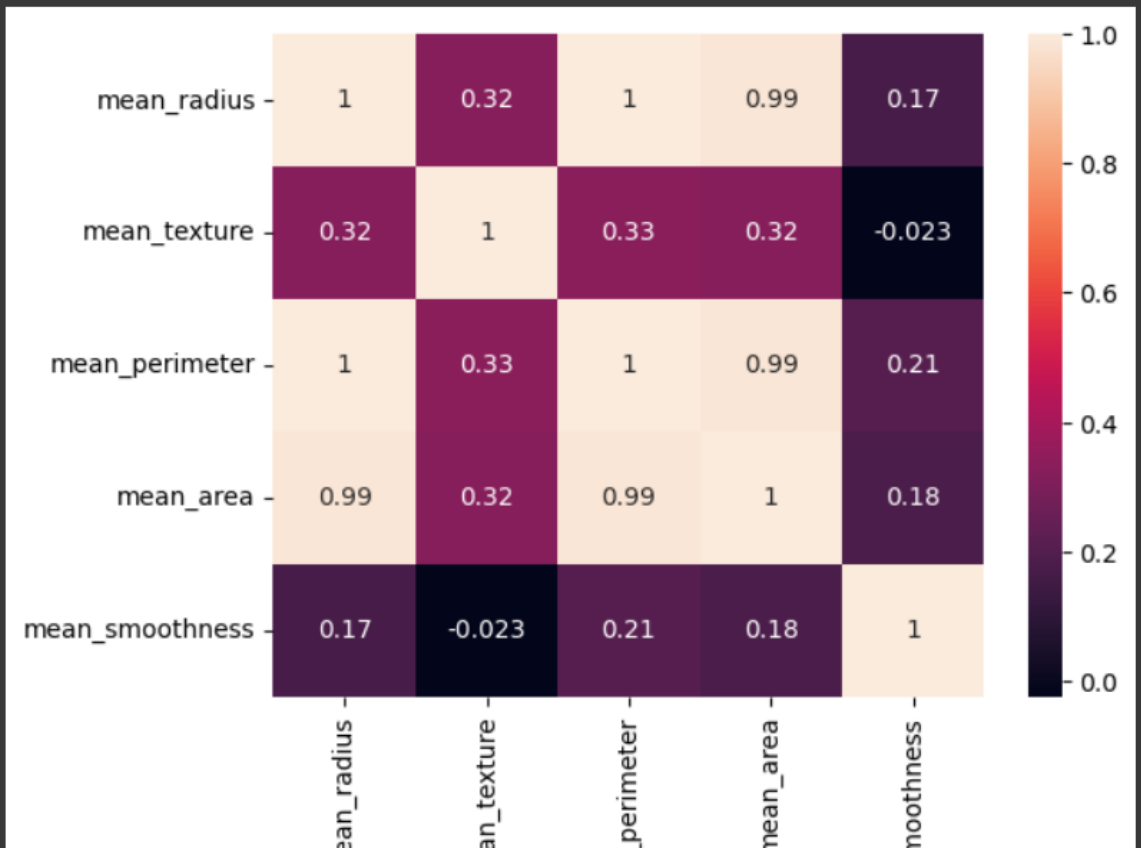
```
mean_radius        0
mean_texture       0
mean_perimeter     0
mean_area          0
mean_smoothness    0
diagnosis          0
dtype: int64
```

```
[ ] df.describe()
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.627417
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.483918
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.000000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.000000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	1.000000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	1.000000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	1.000000

```
k=df.iloc[:,:-1]
sns.heatmap(k.corr(), annot=True)
```

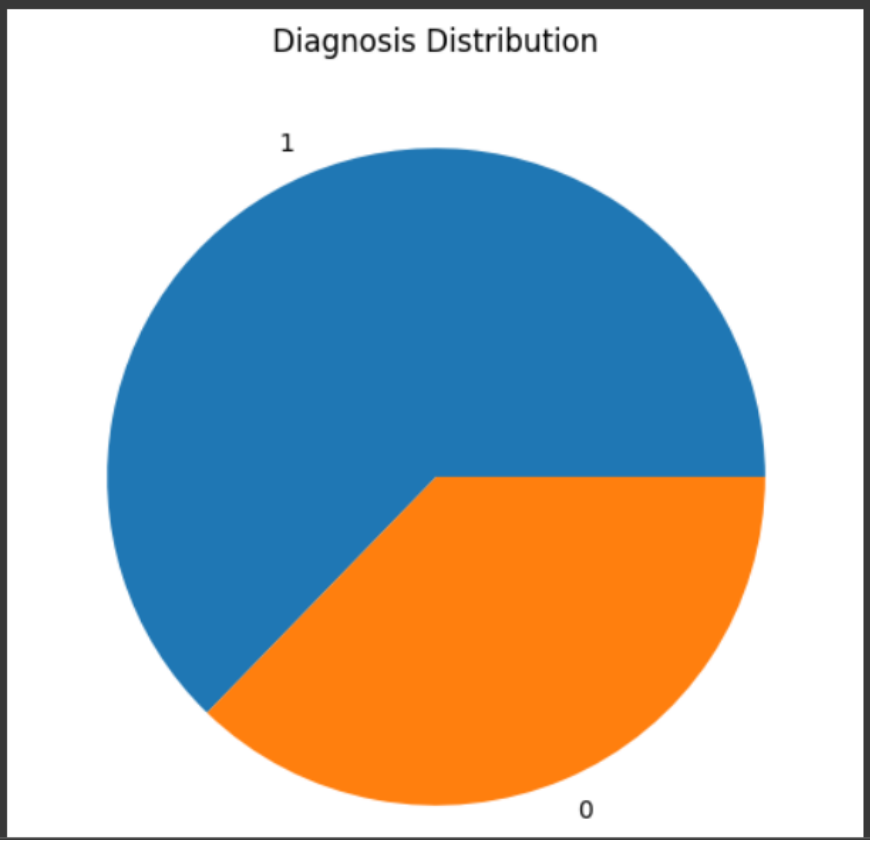
```
<Axes: >
```



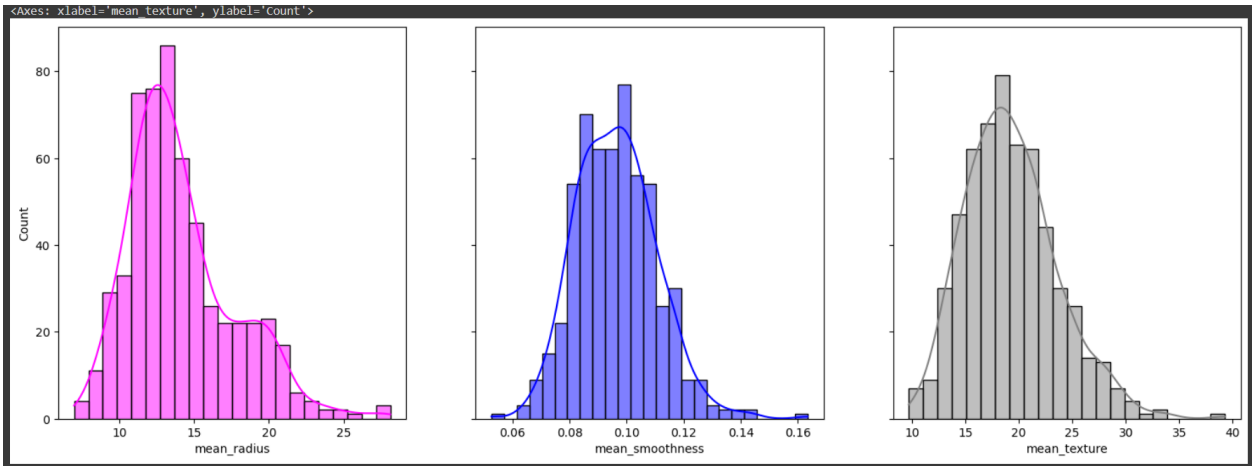

```
[ ]  df['diagnosis'].value_counts()
```

```
1    357
0    212
Name: diagnosis, dtype: int64
```

```
plt.figure(figsize=(6, 6))
plt.pie(df['diagnosis'].value_counts(), labels=df['diagnosis'].value_counts().index)
plt.title('Diagnosis Distribution')
plt.show()
```




```
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(df, ax=axes[0], x="mean_radius", kde=True, color='magenta')
sns.histplot(df, ax=axes[1], x="mean_smoothness", kde=True, color='blue')
sns.histplot(df, ax=axes[2], x="mean_texture", kde=True,color='grey')
```




```
df = df[["mean_radius", "mean_texture", "mean_smoothness", "diagnosis"]]
```

```
[ ] df.head(5)
```

	mean_radius	mean_texture	mean_smoothness	diagnosis
0	17.99	10.38	0.11840	0
1	20.57	17.77	0.08474	0
2	19.69	21.25	0.10960	0
3	11.42	20.38	0.14250	0
4	20.29	14.34	0.10030	0

```
def calc_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

```
def calc_likelihood(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-
mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

```
def naive_bayes_gaussian(df, X, Y):
    features = list(df.columns)[:-1]
    prior = calc_prior(df, Y)
    Y_pred = []
    for x in X:
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calc_likelihood(df, features[i], x[i], Y,
labels[j])
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]
        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

```
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.25, random_state=33)
X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="diagnosis")
print(confusion_matrix(Y_test, Y_pred))
print(f1_score(Y_test, Y_pred))
```

Output:

```
[[44 10]
 [ 1 88]]
0.9411764705882354
```

2. **Using sklearn library fit a Naïve Bayes classifier on Dataset 2 and Dataset 3.**
Ans.

```
[ ] df2 = pd.read_csv('Social_Network_Ads.csv')
```

```
[ ] df2.head(5)
```

	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

```
[ ] df2.isna().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

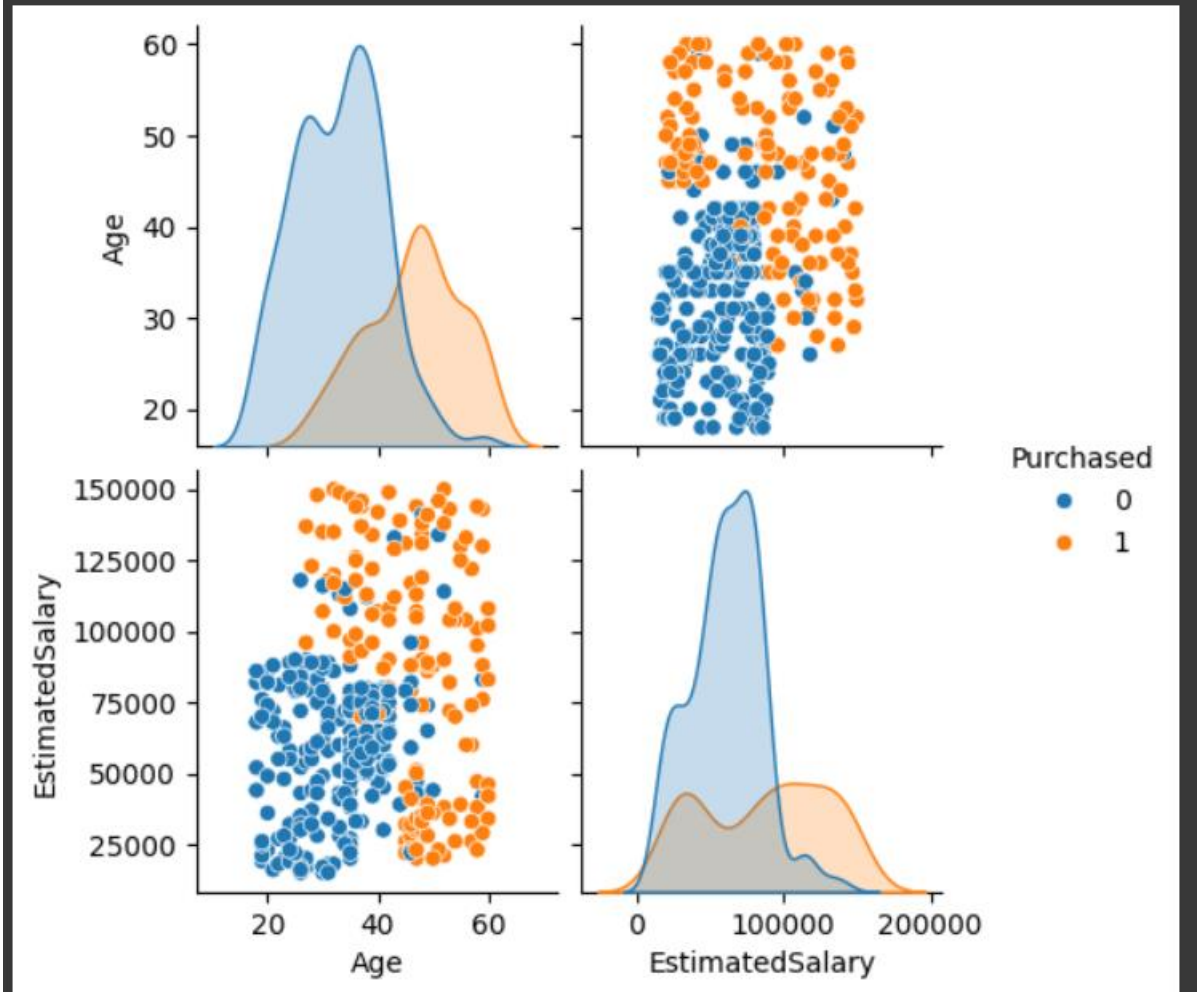
```
df2.describe()
```

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

```
df2.columns
```

```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

```
sns.pairplot(df2[['Age', 'EstimatedSalary', 'Purchased']], hue='Purchased')
plt.show()
```



```
x = df2.iloc[:,[1,2,3]].values
y = df2.iloc[:,-1].values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x[:, 0] = le.fit_transform(x[:, 0])
```

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=61)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
gnb = GaussianNB()
```

```
gnb.fit(x_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
[ ] y_pred = gnb.predict(x_test)

    print(accuracy_score(y_test, y_pred))

    0.92

[ ] print(f1_score(y_test, y_pred))

    0.8857142857142857
```

On Dataset 3:

```
[ ] # dataset 3

df3 = pd.read_csv('german_credit_data.csv')

[ ] df3.head(5)
```

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	2	49	male	1	own	little	NaN	2096	12	education	good
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	4	53	male	2	free	little	little	4870	24	car	bad

```
df3.isna().sum()
```

```
Unnamed: 0            0
Age                   0
Sex                   0
Job                   0
Housing               0
Saving accounts     183
Checking account    394
Credit amount         0
Duration              0
Purpose               0
Risk                  0
dtype: int64
```

```
[ ] df3.dropna(inplace=True)

[ ] df3 = df3.drop(['Unnamed: 0'], axis=1)
```

```
df3.describe()
```

	Age	Job	Credit amount	Duration
count	522.000000	522.000000	522.000000	522.000000
mean	34.888889	1.875479	3278.745211	21.339080
std	11.787918	0.682811	2929.155177	12.474079
min	19.000000	0.000000	276.000000	6.000000
25%	26.000000	2.000000	1297.500000	12.000000
50%	31.500000	2.000000	2326.500000	18.000000
75%	41.000000	2.000000	3971.250000	26.750000
max	75.000000	3.000000	18424.000000	72.000000

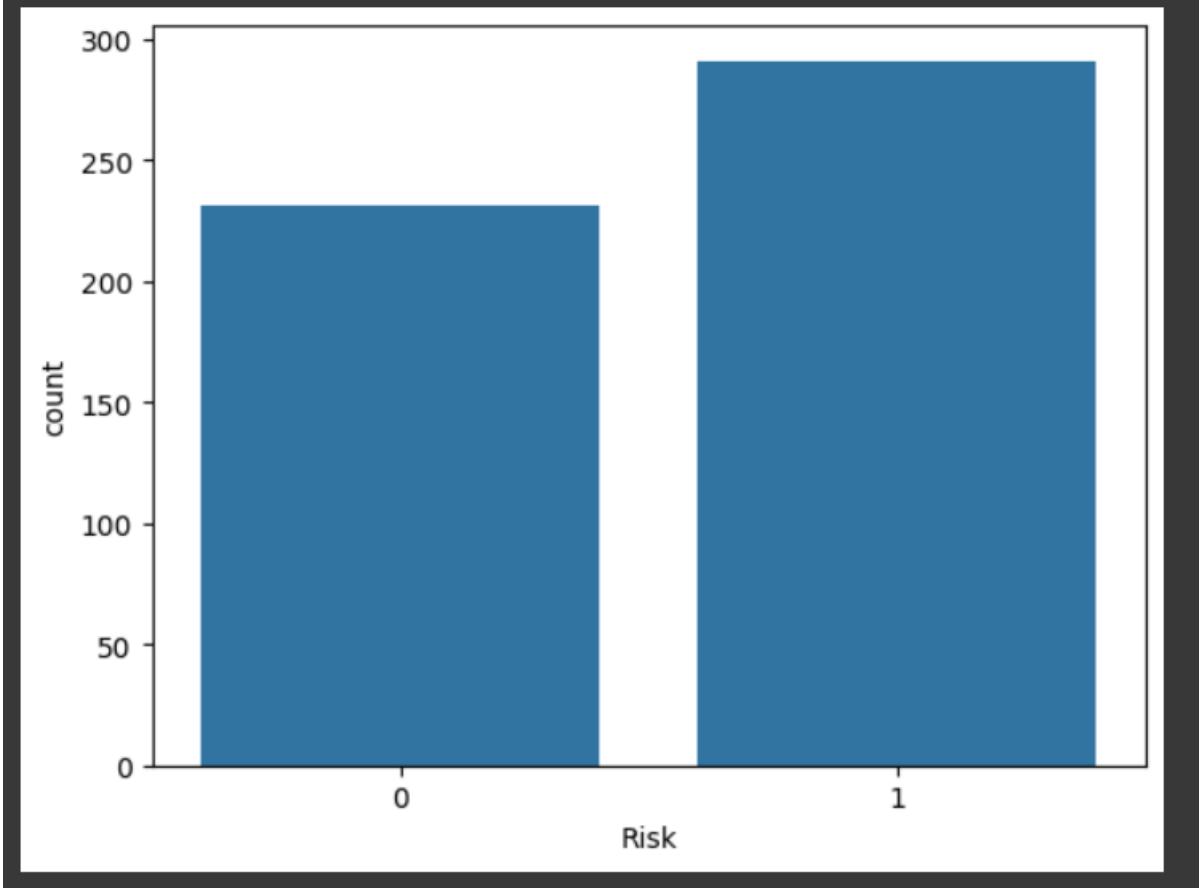
```
df3.columns
```

```
Index(['Age', 'Sex', 'Job', 'Housing', 'Saving accounts', 'Checking account',
       'Credit amount', 'Duration', 'Purpose', 'Risk'],
      dtype='object')
```

```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 522 entries, 1 to 999
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Age               522 non-null    int64
 1   Sex               522 non-null    object
 2   Job               522 non-null    int64
 3   Housing           522 non-null    object
 4   Saving accounts   522 non-null    object
 5   Checking account  522 non-null    object
 6   Credit amount     522 non-null    int64
 7   Duration          522 non-null    int64
 8   Purpose           522 non-null    object
 9   Risk              522 non-null    int64
dtypes: int64(5), object(5)
memory usage: 61.0+ KB
```

```
sns.countplot(x='Risk', data=df3)
plt.show()
```




```
X = df3.drop('Risk', axis=1)
Y = df3['Risk']
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=99)
```

```
cols = ['Sex', 'Housing', 'Saving accounts', 'Checking account', 'Purpose']
```

```
df3.columns
```

```
Index(['Age', 'Sex', 'Job', 'Housing', 'Saving accounts', 'Checking account',
       'Credit amount', 'Duration', 'Purpose', 'Risk'],
      dtype='object')
```

```
label_encoder = LabelEncoder()

for col in cols:
    df3[col] = label_encoder.fit_transform(df3[col])

df3.head()
```

	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
1	22	0	2	1	0	1	5951	48	5	0
3	45	1	2	0	0	0	7882	42	4	1
4	53	1	2	0	0	0	4870	24	1	0
7	35	1	3	2	0	1	6948	36	1	1
9	28	1	3	1	0	1	5234	30	1	0

```
g_bayes = GaussianNB()

g_bayes.fit(X_train, Y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
Y_pred = g_bayes.predict(X_test)

print(f"Accuracy: {accuracy_score(Y_test, Y_pred)}")
```

```
Accuracy: 0.6641221374045801
```

```
print(f"F1-Score: {f1_score(Y_test, Y_pred)}")
```

```
F1-Score: 0.738095238095238
```

Conclusion:

Hence, we have Implemented Naïve Bayes from Scratch Classifier on a given Dataset.

Google Collab Link:
https://colab.research.google.com/drive/1rq7VUMq7qWFoBr_9A8I-WKH4inzF_vT0?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 5

(Random Forest)

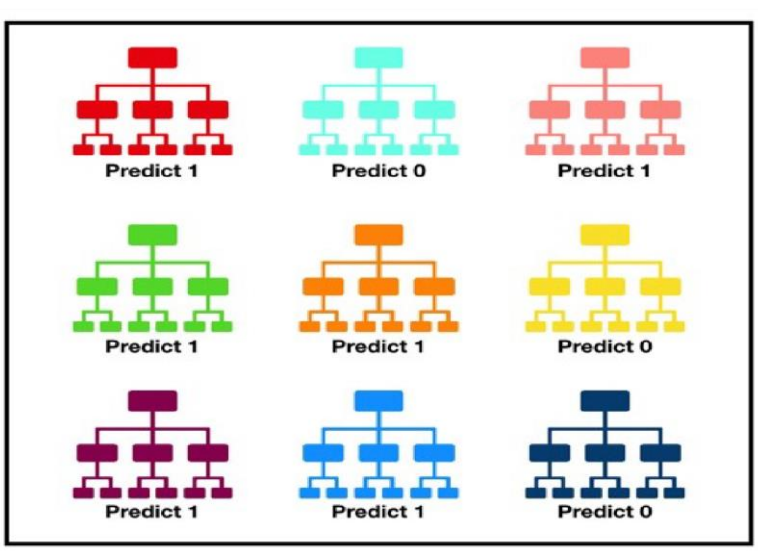
Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: Implement Random Forest algorithm on given datasets and compare the results with Decision Tree classifiers for the same datasets.

Theory:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).



A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all err in the same direction). While

some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.

2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: IRIS.csv
Dataset 2: Breast_Cancer_Data.csv
Dataset 3: BehaviouralRskFactorSurvillanceSystem.csv (The objective of the BRFSS is to collect uniform, state-specific data on preventive health practices and risk behaviors that are linked to chronic diseases, injuries, and preventable infectious diseases in the adult population. Factors assessed by the BRFSS include tobacco use, health care coverage, HIV/AIDS knowledge or prevention, physical activity, and fruit and vegetable consumption. Data are collected from a random sample of adults (one per household) through a telephone survey. The Behavioral Risk Factor Surveillance System (BRFSS) is the nation's premier system of health-related telephone surveys that collect state data about U.S. residents regarding their health-related risk behaviors, chronic health conditions, and use of preventive services. Established in 1984 with 15 states, BRFSS now collects data in all 50 states as well as the District of Columbia and three U.S. territories. BRFSS completes more than 400,000 adult interviews each year, making it the largest continuously conducted health survey system in the world.)

1. Build a Random Forest classifier from scratch using Dataset 2.
Ans.

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import datasets
from collections import Counter
```

```
class Node:
    def __init__(self, feature=None, threshold=None, left=None,
right=None,*,value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

```
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100,
n_features=None):
        self.min_samples_split=min_samples_split
        self.max_depth=max_depth
        self.n_features=n_features
        self.root=None

    def fit(self, X, y):
        self.n_features = X.shape[1] if not self.n_features else
min(X.shape[1],self.n_features)
        self.root = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_feats = X.shape
        n_labels = len(np.unique(y))

        # check the stopping criteria
```

```
        if (depth>=self.max_depth or n_labels==1 or
n_samples<self.min_samples_split):
            leaf_value = self._most_common_label(y)
            return Node(value=leaf_value)

        feat_idxs = np.random.choice(n_feats, self.n_features,
replace=False)

        # find the best split
        best_feature, best_thresh = self._best_split(X, y, feat_idxs)

        # create child nodes
        left_idxs, right_idxs = self._split(X[:, best_feature],
best_thresh)
        left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth+1)
        right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth+1)
        return Node(best_feature, best_thresh, left, right)


    def _best_split(self, X, y, feat_idxs):
        best_gain = -1
        split_idx, split_threshold = None, None

        for feat_idx in feat_idxs:
            X_column = X[:, feat_idx]
            thresholds = np.unique(X_column)

            for thr in thresholds:
                # calculate the information gain
                gain = self._information_gain(y, X_column, thr)

                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_threshold = thr

        return split_idx, split_threshold


    def _information_gain(self, y, X_column, threshold):
        # parent entropy
        parent_entropy = self._entropy(y)

        # create children
```

```
        left_idxs, right_idxs = self._split(X_column, threshold)

        if len(left_idxs) == 0 or len(right_idxs) == 0:
            return 0

        # calculate the weighted avg. entropy of children
        n = len(y)
        n_l, n_r = len(left_idxs), len(right_idxs)
        e_l, e_r = self._entropy(y[left_idxs]),
self._entropy(y[right_idxs])
        child_entropy = (n_l/n) * e_l + (n_r/n) * e_r

        # calculate the IG
        information_gain = parent_entropy - child_entropy
        return information_gain

    def _split(self, X_column, split_thresh):
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        return left_idxs, right_idxs

    def _entropy(self, y):
        hist = np.bincount(y)
        ps = hist / len(y)
        return -np.sum([p * np.log(p) for p in ps if p>0])


    def _most_common_label(self, y):
        counter = Counter(y)
        value = counter.most_common(1)[0][0]
        return value

    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x in X])

    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value

        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)
```

```
class RandomForest:
    def __init__(self, n_trees=10, max_depth=10, min_samples_split=2,
n_feature=None):
        self.n_trees = n_trees
        self.max_depth=max_depth
        self.min_samples_split=min_samples_split
        self.n_features=n_feature
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(max_depth=self.max_depth,
                                min_samples_split=self.min_samples_split,
                                n_features=self.n_features)
            X_sample, y_sample = self._bootstrap_samples(X, y)
            tree.fit(X_sample, y_sample)
            self.trees.append(tree)

    def _bootstrap_samples(self, X, y):
        n_samples = X.shape[0]
        idxs = np.random.choice(n_samples, n_samples, replace=True)
        return X[idxs], y[idxs]

    def _most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common

    def predict(self, X):
        predictions = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(predictions, 0, 1)
        predictions = np.array([self._most_common_label(pred) for pred in
tree_preds])
        return predictions
```

```
data = pd.read_csv("/content/Breast_cancer_data.csv")
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
clf = RandomForest(n_trees=20)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
```

```
acc = accuracy_score(y_test, predictions)
print("Accuracy:", acc)
```

```
Accuracy: 0.9473684210526315
```

2. Build a Random Forest classifier using libraries on Dataset 1.

Ans.

```
df = pd.read_csv('Iris.csv')
df.isna().sum()
df.head(5)
```

	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

```
X = df.iloc[:,1:-1]
y=df['Species']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=61)
```

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, class_weight='balanced')
clf.fit(X_train, y_train)
y_pred=clf.predict(X_test)
```

```
y_train_pred = clf.predict(X_train)
training_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", training_accuracy)
```

```
Training Accuracy: 1.0
```

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=45)
clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)
training_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", training_accuracy)
y_pred = clf.predict(X_test)
testing_accuracy = accuracy_score(y_test, y_pred)
print("Testing Accuracy:", testing_accuracy)
```

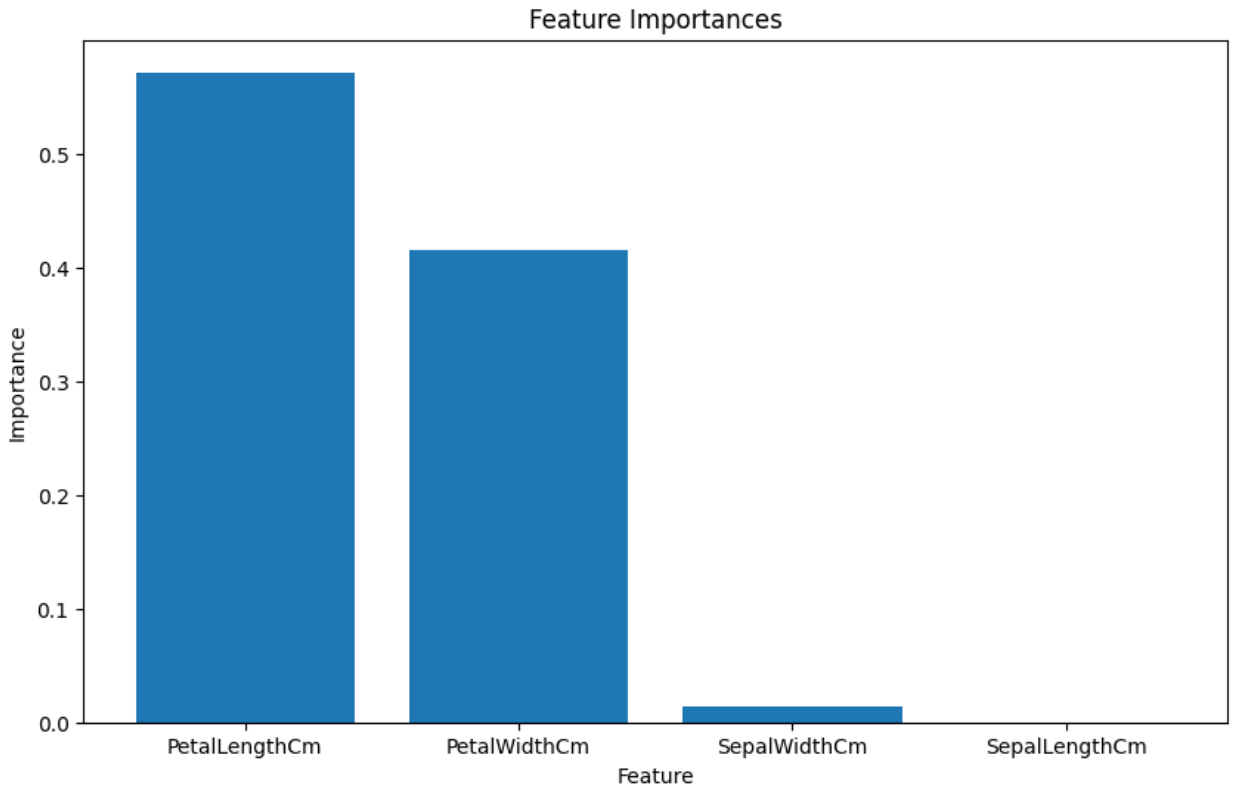
```
Training Accuracy: 1.0
Testing Accuracy: 0.9473684210526315
```

3. Compare the results of random forest with and without selecting important features only for building the classifier on dataset 1, 2 and 3.

```
feature_imp = pd.Series(clf.feature_importances_, index=df.columns[1:-1]).sort_values(ascending=False)
print(feature_imp)
```

```
PetalLengthCm    0.571144
PetalWidthCm     0.415462
SepalWidthCm     0.013394
SepalLengthCm    0.000000
dtype: float64
```

```
plt.figure(figsize=(10, 6))
plt.bar(feature_imp.index, feature_imp.values)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importances')
```




```
X = df[['PetalLengthCm', 'PetalWidthCm','SepalWidthCm']]
y = df['Species']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
```

```
clf = RandomForestClassifier(n_estimators=100)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
y_train_pred = clf.predict(X_train)
train_acc = accuracy_score(y_train, y_train_pred)
y_test_pred = clf.predict(X_test)
test_acc = accuracy_score(y_test, y_test_pred)

print("Training Accuracy:", train_acc)
print("Testing Accuracy:", test_acc)
```

```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

Conclusion:

Random Forests consistently delivered higher accuracy compared to standalone Decision Trees across all datasets. Feature selection further improved model efficiency by focusing on key predictors. The ensemble nature of Random Forests makes them robust against overfitting and suitable for a wide range of classification tasks.

Collab File:

https://colab.research.google.com/drive/1hDrTpFPsstFCy0CylMxVrh9Xj_QV2eQe?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 6

(Logistic Regression)

Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: To Implement Logistic Regression algorithm on the given dataset

```
import numpy as np
from numpy import log,dot,exp,shape
import matplotlib.pyplot as plt
import numpy as np
from numpy import log,dot,exp,shape
```

```
from sklearn.datasets import make_classification
X,y = make_classification(n_features = 4,n_classes=2)
from sklearn.model_selection import train_test_split
X_tr,X_te,y_tr,y_te = train_test_split(X,y,test_size=0.1)
```

```
def standardize(X_tr):
  for i in range(shape(X_tr)[1]):
    X_tr[:,1] = (X_tr[:,1] - np.mean(X_tr[:,i]))/np.std(X_tr[:,i])
```

```
class LogisticRegression:
    def sigmoid(self,z):
        sig = 1/(1+exp(-z))
        return sig
    def initialize(self,X):
        weights = np.zeros((shape(X)[1]+1,1))
        X = np.c_[np.ones((shape(X)[0],1)),X]
        return weights,X
    def fit(self,X,y,alpha=0.001,iter=400):
```

```
        weights,X = self.initialize(X)
        def cost(theta):
            z = dot(X,theta)
            cost0 = y.T.dot(log(self.sigmoid(z)))
            cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))
            cost = -((cost1 + cost0))/len(y)
            return cost
        cost_list = np.zeros(iter,)
        for i in range(iter):
            weights = weights - alpha*dot(X.T,self.sigmoid(dot(X,weights))-
np.reshape(y,(len(y),1)))
            cost_list[i] = cost(weights)
        self.weights = weights
        return cost_list
    def predict(self,X):
        z = dot(self.initialize(X)[1],self.weights)
        lis = []
        for i in self.sigmoid(z):
            if i>0.5:
                lis.append(1)
            else:
                lis.append(0)
        return lis
standardize(X_tr)
standardize(X_te)
obj1 = LogisticRegression()
model= obj1.fit(X_tr,y_tr)
y_pred = obj1.predict(X_te)
y_train = obj1.predict(X_tr)
```

```
def f1_score(y,y_hat):
    tp,tn,fp,fn = 0,0,0,0
    for i in range(len(y)):
        if y[i] == 1 and y_hat[i] == 1:
            tp += 1
        elif y[i] == 1 and y_hat[i] == 0:
            fn += 1
        elif y[i] == 0 and y_hat[i] == 1:
            fp += 1
        elif y[i] == 0 and y_hat[i] == 0:
            tn += 1
```

```
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    return f1_score
```

```
f1_score_tr = f1_score(y_tr,y_train)
f1_score_te = f1_score(y_te,y_pred)
print(f1_score_tr)
print(f1_score_te)
```

```
0.8666666666666667
0.8000000000000002
```

Using Library:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
model1=LogisticRegression().fit(X_tr,y_tr)
y_pred=model1.predict(X_te)
y_pred_train=model1.predict(X_tr)
print("Testing F1 Score: ",f1_score(y_te,y_pred))
print("Training F1 Score: ",f1_score(y_tr,y_pred_train))
```

```
Testing F1 Score:  0.8000000000000002
Training F1 Score:  0.8666666666666667
```

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
```

```
X_test_std = sc.transform(X_test)
```

```
logreg = LogisticRegression()
logreg.fit(X_train_std, y_train)
```

```
from sklearn.metrics import f1_score

y_train_pred = logreg.predict(X_train_std)
y_test_pred = logreg.predict(X_test_std)
f1_train = f1_score(y_train, y_train_pred, average='macro')
f1_test = f1_score(y_test, y_test_pred, average='macro')
print("F1 Score")
print("Training Set:", f1_train)
print("Testing Set:", f1_test)
```

```
F1 Score
Training Set: 0.9714687562788829
Testing Set: 0.975983436853002
```

```
from sklearn.metrics import accuracy_score

y_train_pred = logreg.predict(X_train_std)
y_test_pred = logreg.predict(X_test_std)

accuracy_train = accuracy_score(y_train, y_train_pred)
accuracy_test = accuracy_score(y_test, y_test_pred)
print("Accuracy : ")
print("Training Set:", accuracy_train)
print("Testing Set:", accuracy_test)
```

```
Accuracy :
Training Set: 0.9714285714285714
Testing Set: 0.9777777777777777
```

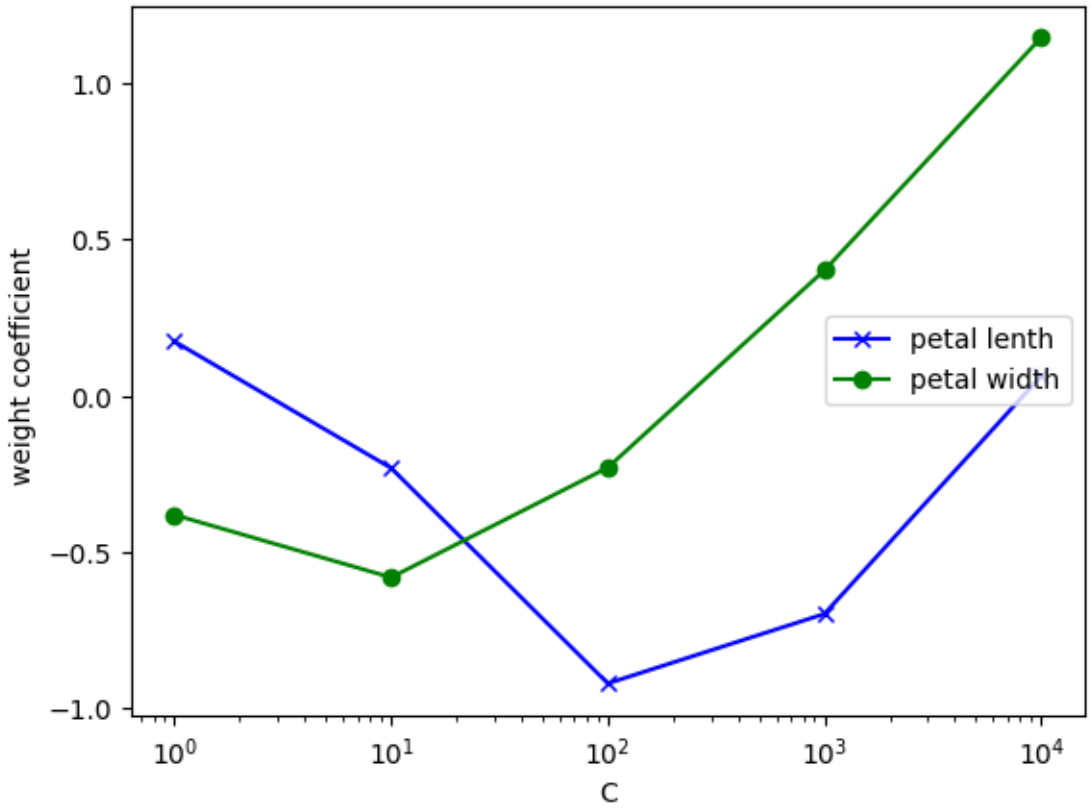
```
X = iris.data[:,[2,3]]
y = iris.target
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
from sklearn.preprocessing import StandardScaler
```

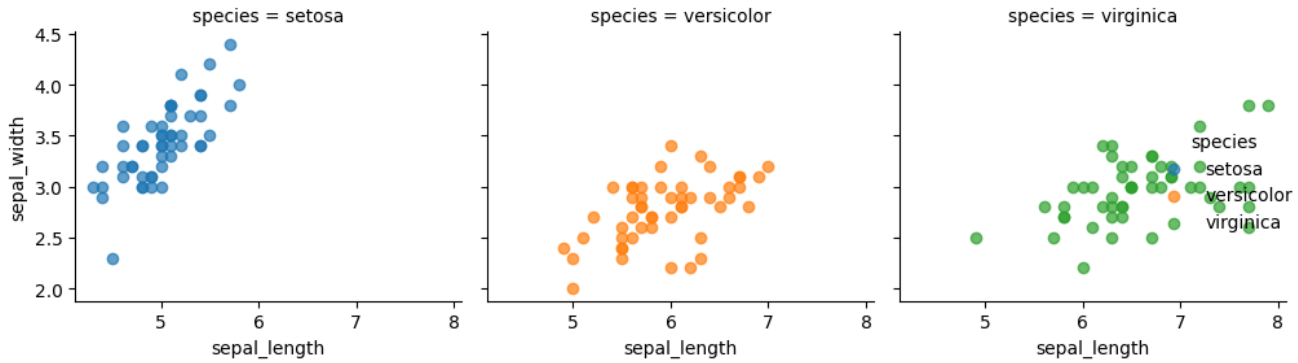
```
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression

weights, params = [], []
for c in np.arange(0, 5):
    lr = LogisticRegression(C=10**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)
```

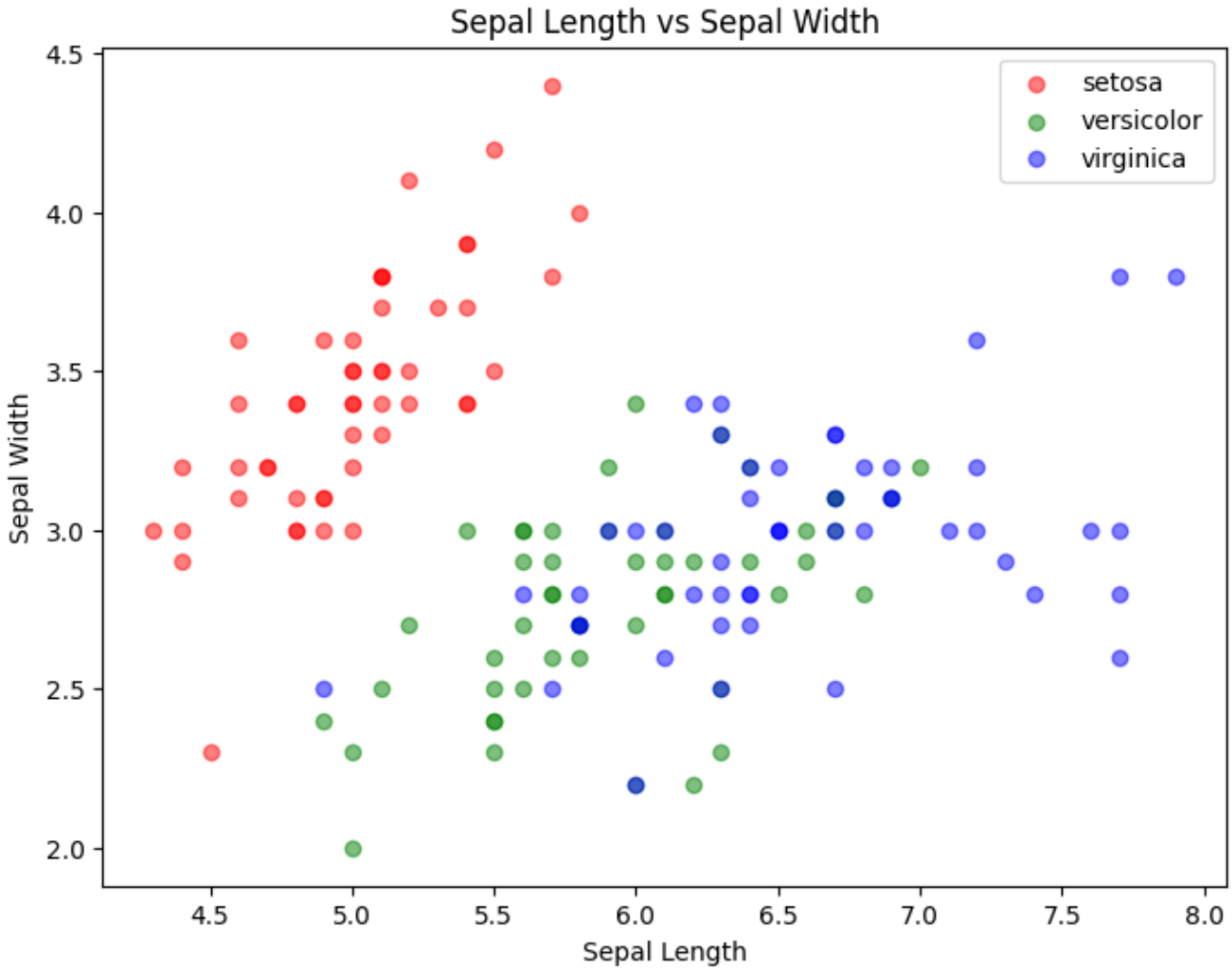
```
import matplotlib.pyplot as plt

plt.plot(params,weights[:,0],color="blue",marker="x",label="petal lenth")
plt.plot(params,weights[:,1],color="green",marker="o",label="petal width")
plt.ylabel('weight coefficient ')
plt.xlabel('C')
plt.legend(loc="right")
plt.xscale("log")
plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset('iris')

g = sns.FacetGrid(iris, col="species", hue="species", col_wrap=3)
g.map(plt.scatter, "sepal_length", "sepal_width", alpha=0.7)

g.add_legend()
plt.tight_layout()
plt.show()
```



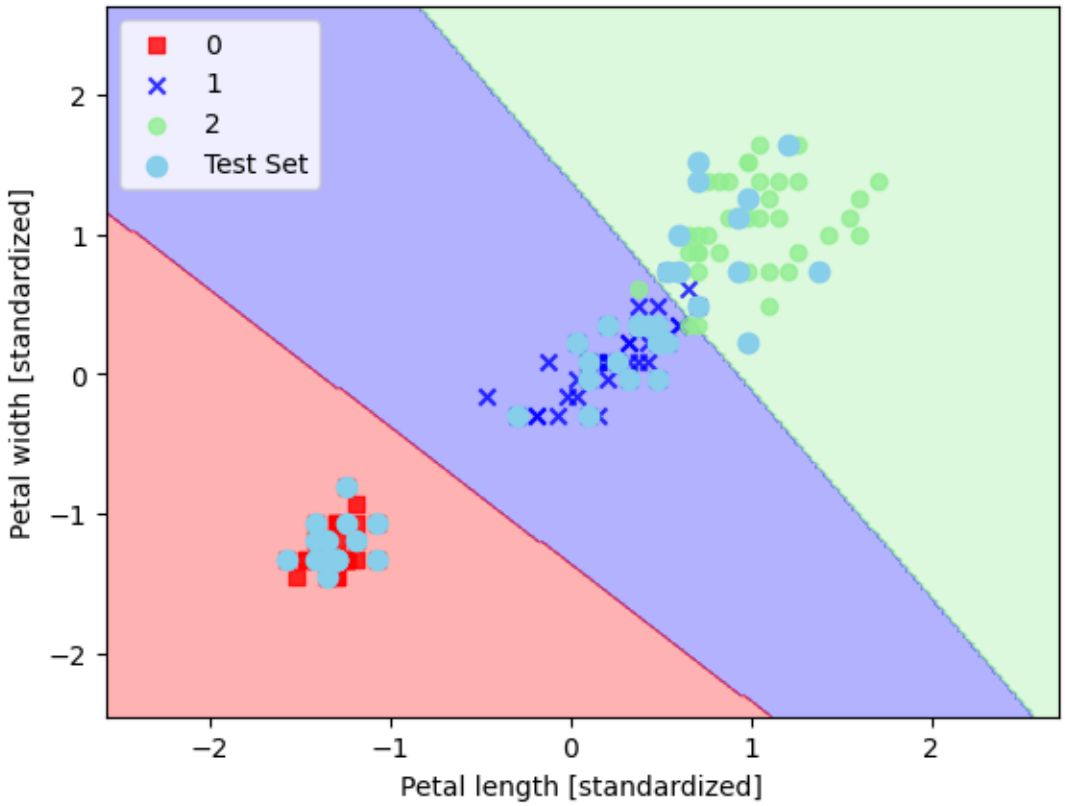

```
import seaborn as sns

plt.figure(figsize=(8, 6))

for species, color in zip(iris['species'].unique(), ['r', 'g', 'b']):
    species_data = iris[iris['species'] == species]
    plt.scatter(species_data['sepal_length'], species_data['sepal_width'],
label=species, color=color, alpha=0.5)

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Sepal Length vs Sepal Width')
plt.legend()
plt.show()
```




```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from matplotlib.colors import ListedColormap
import numpy as np
import matplotlib.pyplot as plt

def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    markers = ('s', 'x', 'o', '^')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
```

```
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=[cmap(idx)],
                    marker=markers[idx], label=cl)

    if test_idx:
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1], c='skyblue',
                    alpha=1.0, linewidths=1, marker='o', s=55, label='Test
Set')

iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)

lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)

plot_decision_regions(X=np.vstack((X_train_std, X_test_std)),
                      y=np.hstack((y_train, y_test)),
                      classifier=lr,
                      test_idx=range(105, 150))

plt.xlabel('Petal length [standardized]')
plt.ylabel('Petal width [standardized]')
```

```
plt.legend(loc='upper left')
plt.show()
```



Conclusion:

Hence, we have implemented Logistic Regression algorithm on iris dataset, Logistic regression proves effective for binary and multiclass classification, offering a straightforward yet powerful method for predictive modelling with interpretable results.

Collab File:

https://colab.research.google.com/drive/1WivrBVnxheCYoNQ7SBlKydtHWBRY4iLw?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 7

(AdaBoost)

Name: Krish Thakkar **BATCH: D2-2**

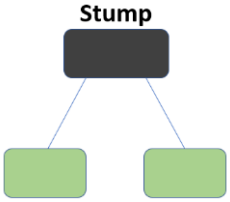
SAP ID: 60009230213 **ROLL NO: D138**

Aim: Evaluate the performance of boosting algorithm (AdaBoost) with different base learners and hyperparameter tuning.

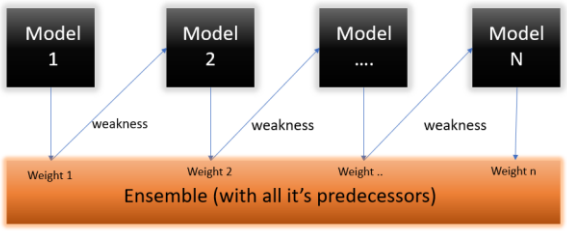
Theory:

Boosting algorithms improve the prediction power by **converting a number of weak learners to strong learners.** The principle behind boosting algorithms is first built a model on the training dataset, then a second model is built to rectify the errors present in the first model. This procedure is continued until and unless the errors are minimized, and the dataset is predicted correctly. Let's take an example to understand this, suppose you built a decision tree algorithm on the Titanic dataset and from there you get an accuracy of 80%. After this, you apply a different algorithm and check the accuracy and it comes out to be 75% for KNN and 70% for Linear Regression. The accuracy differs when we built a different model on the same dataset. But what if we use combinations of all these algorithms for making the final prediction? We'll get more accurate results by taking the average of results from these models. We can increase the prediction power in this way.

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called **Decision Stumps.**



Algorithm builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.



Step 1 – The Image is shown below is the actual representation of our dataset. Since the target column is binary it is a classification problem. First of all these data points will be assigned some weights. Initially, all the weights will be equal.

Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \; i = 1, 2, \ldots . n$$

Where N is the total number of datapoints. since we have 5 data points so the sample weights assigned will be 1/5.

Step 2 – We start by seeing how well "*Gender*" classifies the samples and will see how the variables (Age, Income) classifies the samples.

We'll create a decision stump for each of the features and then calculate the ***Gini Index*** of each tree. The tree with the lowest Gini Index will be our first stump.

Here in our dataset let's say ***Gender*** has the lowest gini index so it will be our first stump.

Step 3 – Calculate the **"Amount of Say"** or **"Importance"** or **"Influence"** for this classifier in classifying the datapoints using this formula:

$$\frac{1}{2}\log\frac{1 - Total\ Error}{Total\ Error}$$

The total error is nothing, but the summation of all the sample weights of misclassified data points. Here in our dataset let's assume there is 1 wrong output, so our total error will be 1/5, and alpha(performance of the stump) will be:

$$Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e\left(\frac{1-Total\ Error}{Total\ Error}\right)$$

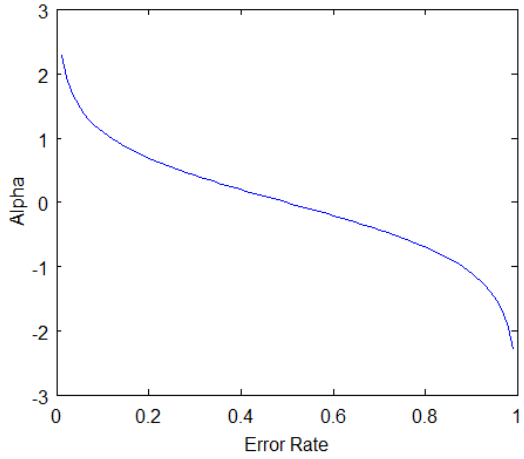
$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{1-\frac{1}{5}}{\frac{1}{5}}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e(4)\ =\ \frac{1}{2}*(1.38)$$

$$\alpha\ =\ 0.69$$

Note: Total error will always be between 0 and 1.
0 Indicates perfect stump and 1 indicates horrible stump.



From the graph above we can see that when there is no misclassification then we have no error (Total Error = 0), so the "amount of say (alpha)" will be a large number.
When the classifier predicts half right and half wrong then the Total Error = 0.5 and the importance (amount of say) of the classifier will be 0. If all the samples have been incorrectly classified then the error will be very high (approx. to 1) and hence our alpha value will be a negative integer.
Step 4 –We need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model. The wrong predictions will be given more weight whereas the correct predictions weights will be decreased. Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.
After finding the importance of the classifier and total error we need to finally update the weights and for this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

The amount of say (alpha) will be ***negative*** when the sample is **correctly classified**.
The amount of say (alpha) will be ***positive*** when the sample is **miss-classified.**
There are four correctly classified samples and 1 wrong, here the ***sample weight*** of that datapoint is *1/5* and the ***amount of say/performance of the stump*** of ***Gender*** is *0.69*.

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(-0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 0.502\ =\ 0.1004$$

For *wrongly classified* samples the updated weights will be:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 1.994\ =\ 0.3988$$

Note: See the sign of alpha when I am putting the values, the **alpha is negative** when the data point is correctly classified, and this *decreases the sample weight* from 0.2 to 0.1004. It is **positive** when there is **misclassification**, and this will *increase the sample weight* from 0.2 to 0.3988

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

We know that the total sum of the sample weights must be equal to 1 but here if we sum up all the new sample weights, we will get 0.8004. To bring this sum equal to 1 we will normalize these weights by dividing all the weights by the total sum of updated weights that is 0.8004. So, after normalizing the sample weights we get this dataset and now the sum is equal to 1.

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

Step 5 – Now we need to make a new dataset to see if the errors decreased or not. For this we will remove the "sample weights" and "new sample weights" column and then based on the "new sample weights" we will divide our data points into buckets.

Row No.	Gender	Age	Income	Illness	New Sample Weights	Buckets
1	Male	41	40000	Yes	0.1004/0.8004= 0.1254	0 to 0.1254
2	Male	54	30000	No	0.1004/0.8004= 0.1254	0.1254 to 0.2508
3	Female	42	25000	No	0.1004/0.8004= 0.1254	0.2508 to 0.3762
4	Female	40	60000	Yes	0.3988/0.8004= 0.4982	0.3762 to 0.8744
5	Male	46	50000	Yes	0.1004/0.8004= 0.1254	0.8744 to 0.9998

Step 6 – We are almost done, now what the algorithm does is selects random numbers from 0-1. Since incorrectly classified records have higher sample weights, the probability to select those records is very high. Suppose the 5 random numbers our algorithm take is 0.38,0.26,0.98,0.40,0.55.

Now we will see where these random numbers fall in the bucket and according to it, we'll make our new dataset shown below.

Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

This comes out to be our new dataset and we see the datapoint which was wrongly classified has been selected 3 times because it has a higher weight.

Step 9 – Now this act as our new dataset and we need to repeat all the above steps i.e.

1. Assign ***equal weights*** to all the datapoints
2. Find the stump that does the ***best job classifying*** the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index
3. Calculate the ***"Amount of Say"*** and ***"Total error"*** to update the previous sample weights.
4. Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.

Suppose with respect to our dataset we have constructed 3 decision trees (DT1, DT2, DT3) in a ***sequential manner.*** If we send our **test data** now it will pass through all the decision trees and finally, we will see which class has the majority, and based on that we will do predictions for our test dataset.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Synthetic dataset
Dataset 2: CreditcardFraud.csv: The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data are not provided. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

1. Implement Decision Tree classifier and Logistic Regression on Dataset 1 using K fold cross validation and compare the results with AdaBoost classifier with base learner as Decision tree and Logistic Regression.

2. Check if there is class imbalance problem in Dataset 2. Compare the results of decision tree classifier and AdaBoost classifier on Dataset 2 and write your analysis.
3. Implement AdaBoost with base learner as decision tree on dataset 2 using K fold cross validation. Perform Hyperparameter tuning using (a) different depth, (b) different learning rate and (c) grid search CV. Show your results using Boxplot.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.datasets import load_iris
```

```
iris = load_iris()

X = iris.data
y = iris.target
```

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

```
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1)

model = abc.fit(X_train,y_train)
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9111111111111111
```

```
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)
y_pred_dt = dt_classifier.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", accuracy_dt)
```

```
Decision Tree Accuracy: 0.9777777777777777
```

```
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
```

```
abc =
AdaBoostClassifier(n_estimators=50,base_estimator=NB,learning_rate=1)

model = abc.fit(X_train,y_train)
y_pred = model.predict(X_test)

accuracy_nb = accuracy_score(y_test, y_pred)
print("Naive Bayes Accuracy:", accuracy_nb)
```

```
Naive Bayes Accuracy: 0.9777777777777777
```

```
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

X, y = make_classification(n_samples=1000, n_features=20, n_informative=5,
n_redundant=5, random_state=0)
model = AdaBoostClassifier()

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,
n_jobs=-1, error_score='raise')
print('Accuracy : %.3f (%.3f)' % (np.mean(n_scores), np.std(n_scores)))
```

```
Accuracy : 0.918 (0.029)
```

```
from sklearn.datasets import make_regression
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15,
noise=0.1,random_state=6)

print(X.shape,y.shape)
```

```
(1000, 20) (1000,)
```

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import cross_val_score, RepeatedKFold

X, y = make_regression(n_samples=1000, n_features=20, n_informative=15,
noise=0.1,random_state=6)
```

```
model = AdaBoostRegressor()
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error',
cv=cv, n_jobs=-1, error_score='raise')
mse_scores = -n_scores
print('Mean Squared Error: %.3f (%.3f)' % (np.mean(mse_scores),
np.std(mse_scores)))
```

```
Mean Squared Error: 8708.836 (999.167)
```

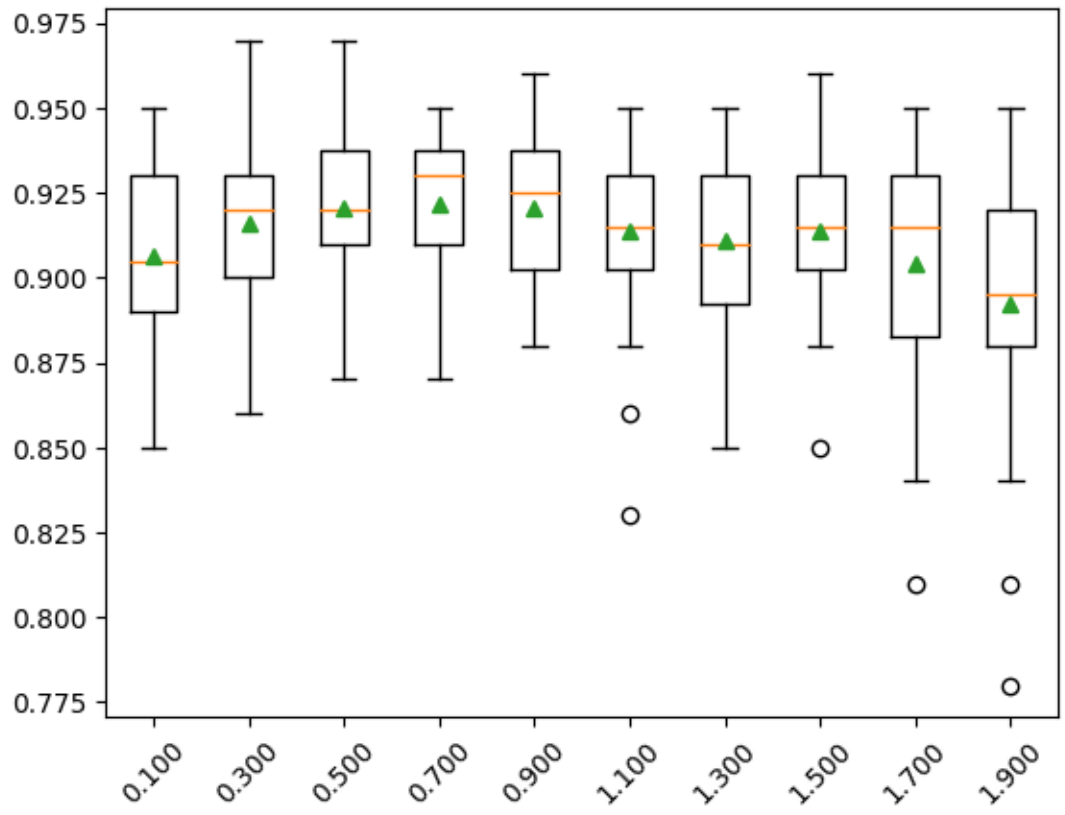
```
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15,
noise=0.1,random_state=6)
model = AdaBoostRegressor()
model.fit(X,y)
row = [[-1.345, 0.2445, 5.4656, 3.678, -2.345, 1.678] +
list(np.random.rand(14))]
yhat = model.predict(row)
print("prediction : %d" % yhat[0])
```

```
prediction : 161
```

```
import numpy as np
from numpy import arange,mean,std
```

```
def get_dataset():
  X,y = make_classification(n_samples=1000, n_features=20,
n_informative=5, n_redundant=5, random_state=0)
  return X,y
def get_models():
  models = dict()
  for i in arange(0.1,2.1,0.2):
    key = "%.3f" % i
    models[key] = AdaBoostClassifier(learning_rate=i)
  return models

def evaluate_model(model,X,y):
  cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
  scores = cross_val_score(model,X,y,scoring='accuracy',cv=cv,n_jobs=-1)
  return scores
```

```
X,y = get_dataset()
models = get_models()
results,names = list() , list()
for name,model in models.items():
  scores = evaluate_model(model,X,y)
  results.append(scores)
  names.append(name)

  print(">%s %.3f (%.3f)" % (name,mean(scores),std(scores)))
3
plt.boxplot(results,labels=names,showmeans=True)
plt.xticks(rotation=45)
plt.show()
```



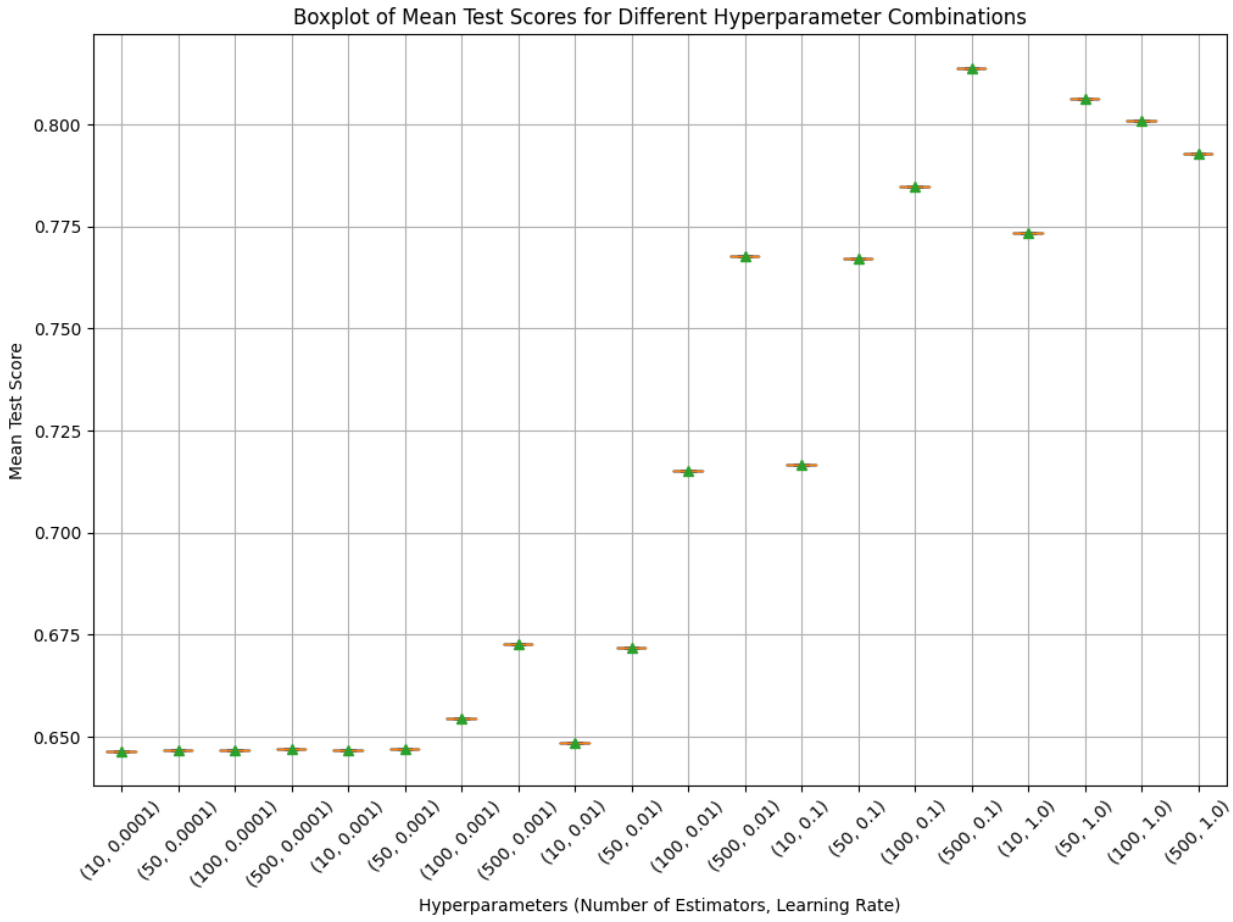

```
from sklearn.datasets import make_classification
from sklearn.model_selection import RepeatedStratifiedKFold
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier

X, y = make_classification(n_samples=1000, n_features=20,
n_informative=15, n_redundant=5, random_state=6)
model = AdaBoostClassifier()

grid = dict()
grid['n_estimators'] = [10, 50, 100, 500]
grid['learning_rate'] = [0.0001, 0.001, 0.01, 0.1, 1.0]

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
cv=cv, scoring='accuracy')

grid_result = grid_search.fit(X, y)

print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))

means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.813667 using {'learning_rate': 0.1, 'n_estimators': 500}
0.646333 (0.036376) with: {'learning_rate': 0.0001, 'n_estimators': 10}
0.646667 (0.036545) with: {'learning_rate': 0.0001, 'n_estimators': 50}
0.646667 (0.036545) with: {'learning_rate': 0.0001, 'n_estimators': 100}
0.647000 (0.038136) with: {'learning_rate': 0.0001, 'n_estimators': 500}
0.646667 (0.036545) with: {'learning_rate': 0.001, 'n_estimators': 10}
0.647000 (0.038136) with: {'learning_rate': 0.001, 'n_estimators': 50}
0.654333 (0.045511) with: {'learning_rate': 0.001, 'n_estimators': 100}
0.672667 (0.046543) with: {'learning_rate': 0.001, 'n_estimators': 500}
0.648333 (0.042197) with: {'learning_rate': 0.01, 'n_estimators': 10}
0.671667 (0.045613) with: {'learning_rate': 0.01, 'n_estimators': 50}
0.715000 (0.053213) with: {'learning_rate': 0.01, 'n_estimators': 100}
0.767667 (0.045948) with: {'learning_rate': 0.01, 'n_estimators': 500}
0.716667 (0.048876) with: {'learning_rate': 0.1, 'n_estimators': 10}
0.767000 (0.049271) with: {'learning_rate': 0.1, 'n_estimators': 50}
0.784667 (0.042874) with: {'learning_rate': 0.1, 'n_estimators': 100}
0.813667 (0.032092) with: {'learning_rate': 0.1, 'n_estimators': 500}
0.773333 (0.038759) with: {'learning_rate': 1.0, 'n_estimators': 10}
0.806333 (0.040701) with: {'learning_rate': 1.0, 'n_estimators': 50}
0.801000 (0.032491) with: {'learning_rate': 1.0, 'n_estimators': 100}
0.792667 (0.027801) with: {'learning_rate': 1.0, 'n_estimators': 500}
```

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

mean_test_scores = grid_result.cv_results_['mean_test_score']
n_estimators = [params['n_estimators'] for params in
grid_result.cv_results_['params']]
learning_rates = [params['learning_rate'] for params in
grid_result.cv_results_['params']]

scores_dict = {}
for i in range(len(mean_test_scores)):
    key = (n_estimators[i], learning_rates[i])
    if key not in scores_dict:
        scores_dict[key] = []
    scores_dict[key].append(mean_test_scores[i])

plt.figure(figsize=(12, 8))
```

```
plt.boxplot(scores_dict.values(), labels=scores_dict.keys(),
showmeans=True)
plt.xlabel('Hyperparameters (Number of Estimators, Learning Rate)')
plt.ylabel('Mean Test Score')
plt.title('Boxplot of Mean Test Scores for Different Hyperparameter
Combinations')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



Conclusion:

This experiment showcases AdaBoost's efficacy in improving model performance through ensemble learning with different base learners like decision trees and logistic regression. It also demonstrates the importance of hyperparameter tuning, particularly for boosting algorithms, to optimize performance, especially in scenarios with imbalanced datasets like credit card fraud detection.

Collab file:

https://colab.research.google.com/drive/1ljcy2mCxaY6Fv6RXNFG0X9tleruoiQkJ?usp=sharing

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I

(DJ19DSC402) AY: 2023-24

Experiment 8

(SVM)

Name: Krish Thakkar **BATCH: D2-2**

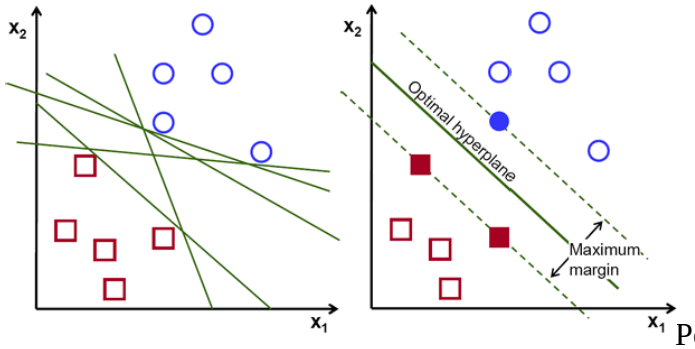
SAP ID: 60009230213 **ROLL NO: D138**

Aim: Perform SVM using soft margin SVC, Kernels and improve the accuracies using hyperparameter tuning.

Theory:

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N

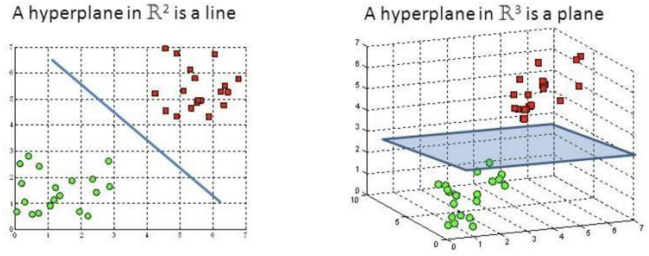
— the number of features) that distinctly classifies the data points.



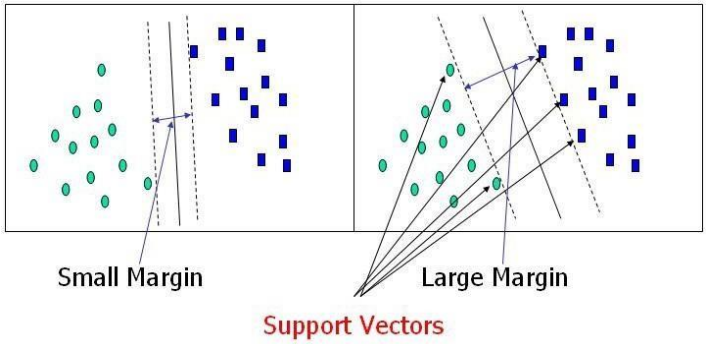
Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors



Hyperplanes in 2D and 3D feature space: Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also,the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class.

Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

Cost Function and Gradient Update: In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

$$c(x, y, f(x)) = (1 - y * f(x))_+$$

margin is hinge loss.

Hinge loss function (function on left can be represented as a function on the right)
The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the

regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM
Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Gradients
When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification
When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient

update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update — Misclassification

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: IRIS.cvs
Dataset 2: mnist_784 : The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the first 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size- normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

Task 1: Build a linear classifier on Dataset 1 using SVC.
Task 2: Build a classifier on Dataset 1 using Linear, Polynomial and RBF kernel and show the decision boundary using matplotlib.
Task 3. Find the accuracy of svc classifier (M1) built on Dataset 3 using linear csv and RBF kernel.
Task 4: Improve the accuracy of M1 by varying C and gamma values and using RandomizedSearchCV.
Task 5: Calculate the computational time of Task 3 and 4.

```
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.datasets import load_iris, make_classification
```

```
x,y = make_classification(n_samples=5000,n_features=10,
                          n_classes=3,n_clusters_per_class=1)
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.15)
```

```
lsvc = LinearSVC(verbose=0)
print(lsvc)

LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

```
LinearSVC()
▾ LinearSVC
LinearSVC()
```

```
lsvc.fit(xtrain, ytrain)
score = lsvc.score(xtrain, ytrain)
print("Score: ", score)
```

```
Score:  0.8934117647058823
```

```
cv_scores = cross_val_score(lsvc, xtrain, ytrain, cv=10)
print("CV average score: %.2f" % cv_scores.mean())
```

```
CV average score: 0.89
```

```
ypred = lsvc.predict(xtest)

cm = confusion_matrix(ytest, ypred)
print(cm)
```

```
[[235  11   4]
 [ 51 188   9]
 [  4   0 248]]
```

```
cr = classification_report(ytest, ypred)
print(cr)
```

	precision	recall	f1-score	support
0	0.81	0.94	0.87	250
1	0.94	0.76	0.84	248
2	0.95	0.98	0.97	252
accuracy			0.89	750
macro avg	0.90	0.89	0.89	750
weighted avg	0.90	0.89	0.89	750

```
print("Iris dataset classification with SVC")

iris = load_iris()
x, y = iris.data, iris.target
xtrain, xtest, ytrain, ytest=train_test_split(x, y, test_size=0.15)
```

```
Iris dataset classification with SVC
```

```
lsvc = LinearSVC(verbose=0)
print(lsvc)
```

```
lsvc.fit(xtrain, ytrain)
score = lsvc.score(xtrain, ytrain)
print("Score: ", score)
```

```
Score:  0.968503937007874
```

```
cv_scores = cross_val_score(lsvc, xtrain, ytrain, cv=10)
print("CV average score: %.2f" % cv_scores.mean())
```

```
CV average score: 0.95
```

```
ypred = lsvc.predict(xtest)
```

```
cm = confusion_matrix(ytest, ypred)
print(cm)
```

```
[[ 6  0  0]
 [ 0  7  0]
 [ 0  0 10]]
```

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

```
▾ LinearSVC
LinearSVC()
```

```
cr = classification_report(ytest, ypred)
print(cr)
```

```
                precision    recall  f1-score   support

             0       1.00      1.00      1.00         6
             1       1.00      1.00      1.00         7
             2       1.00      1.00      1.00        10

      accuracy                           1.00        23
     macro avg       1.00      1.00      1.00        23
  weighted avg       1.00      1.00      1.00        23
```

```
from __future__ import division,print_function
import matplotlib.pyplot as plt
from sklearn import datasets,svm
```

```
iris = datasets.load_iris()
X = iris.data[:,:2]
y = iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
```

```
def evaluate_on_test_data(model=None):
    predictions = model.predict(X_test)
    correct_classifications = 0
    for i in range(len(y_test)):
        if predictions[i] == y_test[i]:
```

```
            correct_classifications += 1
    accuracy = 100*correct_classifications/len(y_test) #Accuracy as a
percentage
    return accuracy
```

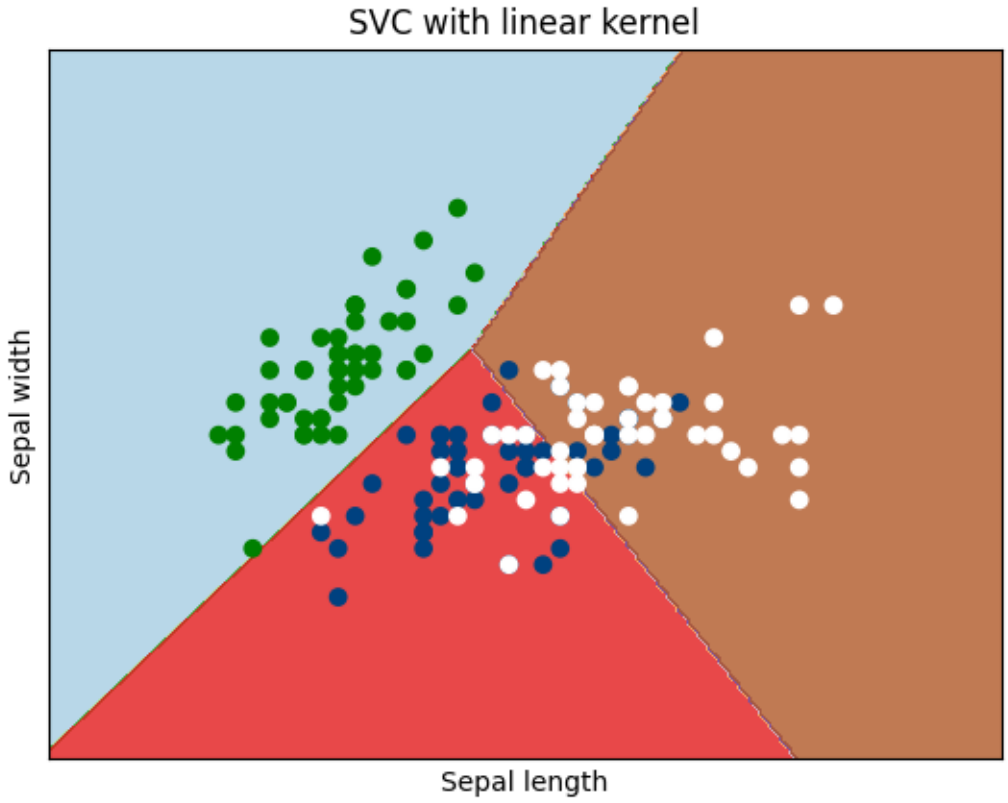
```
kernels = ('linear','poly','rbf')
accuracies = []
for index, kernel in enumerate(kernels):
    model = svm.SVC(kernel=kernel)
    model.fit(X_train, y_train)
    acc = evaluate_on_test_data(model)
    accuracies.append(acc)
    print("{} % accuracy obtained with kernel = {}".format(acc, kernel))
```

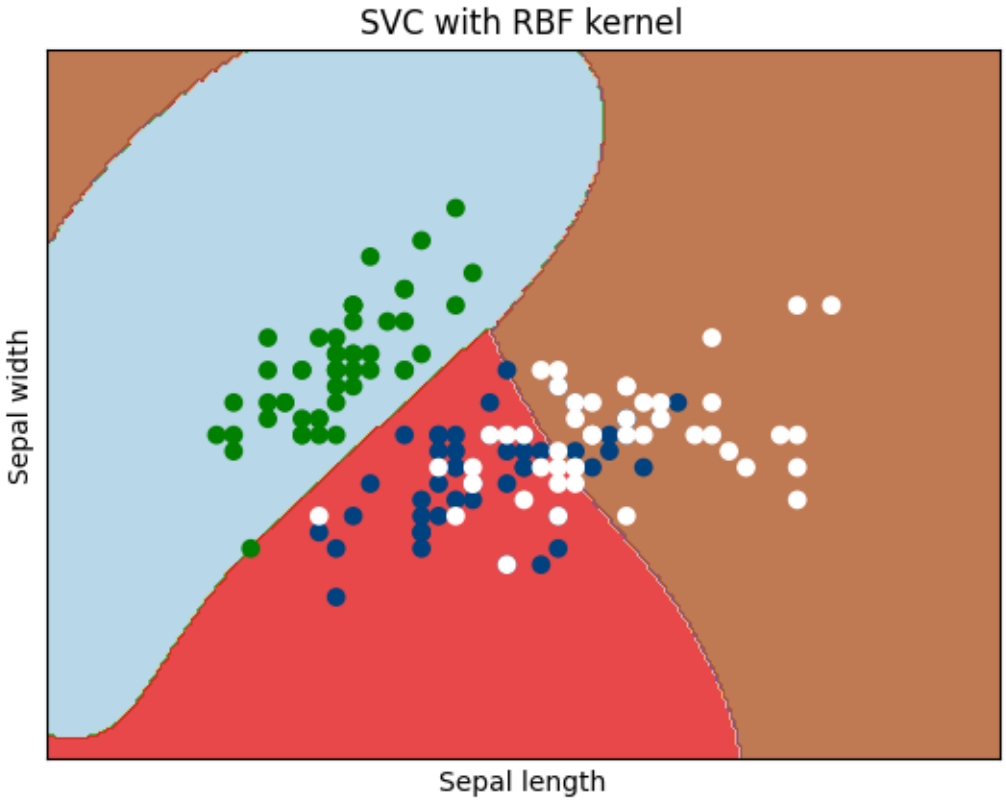
```
84.21052631578948 % accuracy obtained with kernel = linear
76.3157894736842 % accuracy obtained with kernel = poly
84.21052631578948 % accuracy obtained with kernel = rbf
```

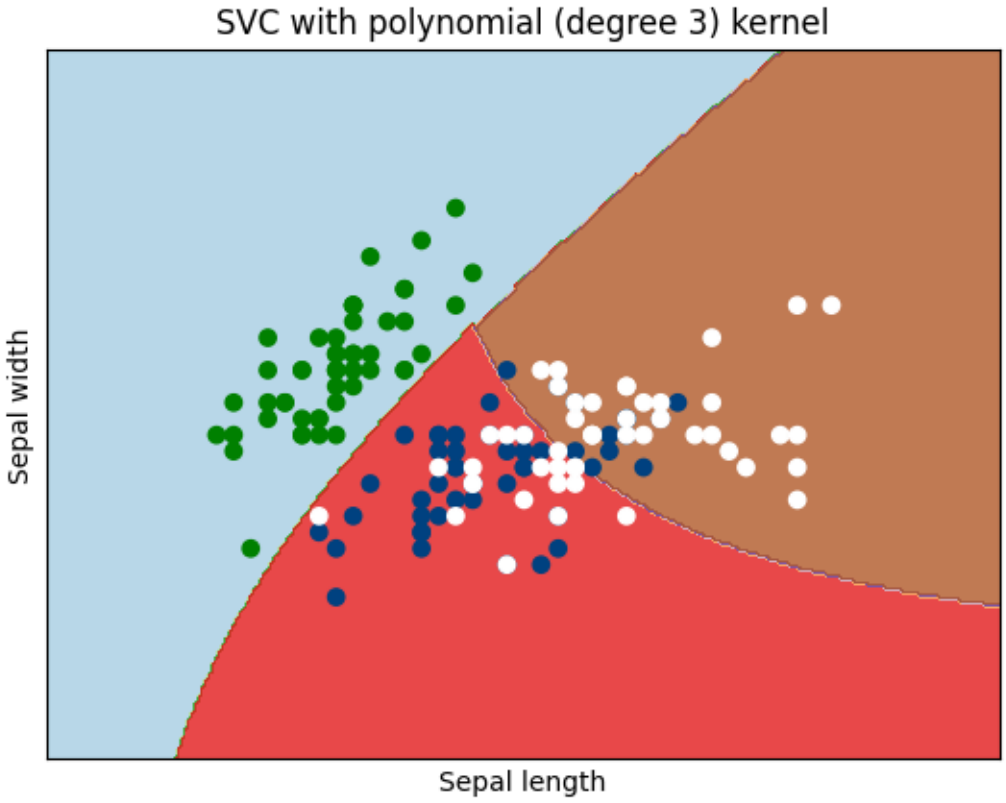
```
svc = svm.SVC(kernel='linear').fit(X_train, y_train)
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7).fit(X_train, y_train)
poly_svc = svm.SVC(kernel='poly', degree=3).fit(X_train, y_train)
```

```
h = .02
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 = ['SVC with linear kernel',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel']


for i, clf in enumerate((svc, rbf_svc, poly_svc)):
    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('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
```

```
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```




SVC with RBF kernel
Sepal width
Sepal length


SVC with polynomial (degree 3) kernel
Sepal width
Sepal length

```
import warnings
from sklearn.datasets import fetch_openml
from sklearn.exceptions import ConvergenceWarning
```

```
X,y = fetch_openml("mnist_784",version=1,return_X_y=True,as_frame=False)
X_train = X[:60000]
y_train = y[:60000]
X_test = X[60000:]
y_test = y[60000:]
```

```
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train, y_train)
y_pred = lin_clf.predict(X_train)
accuracy = accuracy_score(y_train, y_pred)
print("Training Accuracy:", accuracy)
```

```
Training Accuracy: 0.8348666666666666
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float32))
X_test_scaled = scaler.transform(X_test.astype(np.float32))

lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train_scaled, y_train)

y_pred_train = lin_clf.predict(X_train_scaled)
train_accuracy = accuracy_score(y_train, y_pred_train)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9214
```

```
from sklearn.svm import SVC
svm_clf = SVC(kernel='rbf', gamma='scale')
svm_clf.fit(X_train_scaled[:10000], y_train[:10000])
```

```
y_pred_train = svm_clf.predict(X_train_scaled)
train_accuracy = accuracy_score(y_train, y_pred_train)
print("Training Accuracy:", train_accuracy)

y_pred_test = svm_clf.predict(X_test_scaled)
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Test Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.9455333333333333
Test Accuracy: 0.9389
```

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import reciprocal, uniform

param_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1,
10)}

print(X_train_scaled.shape)
print(y_train.shape)

X_train_scaled_subset = X_train_scaled[:112]
y_train_subset = y_train[:112]

rnd_search_cv = RandomizedSearchCV(svm_clf, param_distributions, n_iter=10,
verbose=2, cv=3)
rnd_search_cv.fit(X_train_scaled_subset, y_train_subset)
```

```
(60000, 784)
(60000,)
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[CV] END ....C=6.963187740539605, gamma=0.002349047830068742; total time=   0.0s
[CV] END ....C=6.963187740539605, gamma=0.002349047830068742; total time=   0.0s
[CV] END ....C=6.963187740539605, gamma=0.002349047830068742; total time=   0.0s
[CV] END ....C=2.663080448416122, gamma=0.001553396371540171; total time=   0.0s
[CV] END ....C=2.663080448416122, gamma=0.001553396371540171; total time=   0.0s
[CV] END ....C=2.663080448416122, gamma=0.001553396371540171; total time=   0.0s
[CV] END ...C=8.060257203198415, gamma=0.0010778726666161812; total time=   0.0s
[CV] END ...C=8.060257203198415, gamma=0.0010778726666161812; total time=   0.0s
[CV] END ...C=8.060257203198415, gamma=0.0010778726666161812; total time=   0.0s
[CV] END ...C=2.978416419527705, gamma=0.0011769207911320394; total time=   0.0s
[CV] END ...C=2.978416419527705, gamma=0.0011769207911320394; total time=   0.0s
[CV] END ...C=2.978416419527705, gamma=0.0011769207911320394; total time=   0.0s
[CV] END ....C=7.134659723450092, gamma=0.010515948661761853; total time=   0.0s
[CV] END ....C=7.134659723450092, gamma=0.010515948661761853; total time=   0.0s
[CV] END ....C=7.134659723450092, gamma=0.010515948661761853; total time=   0.0s
[CV] END ...C=3.6515021131652396, gamma=0.008224796952115818; total time=   0.0s
[CV] END ...C=3.6515021131652396, gamma=0.008224796952115818; total time=   0.0s
```

```
rnd_search_cv.best_estimator_
```

```
SVC
SVC(C=8.060257203198415, gamma=0.0010778726666161812)
```

Conclusion:

In this experiment on Support Vector Machines (SVM), we explored various aspects including building SVM classifiers using different kernels (linear, polynomial, RBF), improving accuracy through hyperparameter tuning (varying C and gamma), and evaluating computational time using RandomizedSearchCV

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2023-24

Experiment 9

(K-Means Clustering)

Name: Krish Thakkar **BATCH: D2-2**

SAP ID: 60009230213 **ROLL NO: D138**

Aim: Explore K means clustering with variations on different datasets.

Theory: The K-means clustering algorithm computes centroids and repeats until the optimal centroid is found. It is presumptively known how many clusters there are. It is also known as the flat clustering algorithm. The number of clusters found from data by the method is denoted by the letter 'K' in Kmeans. In this method, data points are assigned to clusters in such a way that the sum of the squared distances between the data points and the centroid is as small as possible. It is essential to note that reduced diversity within clusters leads to more identical data points within the same cluster.

The following stages will help us understand how the K-Means clustering technique works

Step 1: First, we need to provide the number of clusters, K, that need to be generated by this algorithm.

Step 2: Next, choose K data points at random and assign each to a cluster. Briefly, categorize the data based on the number of data points.

Step 3: The cluster centroids will now be computed.

Step 4: Iterate the steps below until we find the ideal centroid, which is the assigning of data points to clusters that do not vary.

4.1 The sum of squared distances between data points and centroids would be calculated first.

4.2 At this point, we need to allocate each data point to the cluster that is closest to the others (centroid).

4.3 Finally, compute the centroids for the clusters by averaging all of the cluster's data points.

When using the K-means algorithm, we must keep the following points in mind:

It is suggested to normalize the data while dealing with clustering algorithms such as K-Means since such algorithms employ distance-based measurement to identify the similarity between data points. Because of the iterative nature of K-Means and the random initialization of centroids, K-Means may become stuck in a local optimum and fail to converge to the global optimum. As a result, it is advised to employ distinct centroids' initializations.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:

Dataset 1: Synthetic Data (200 samples, 3 clusters and cluster_std = 2.7)

Dataset 2: TCGA-PANCAN-HiSeq-801x20531.tar.gz gene expression cancer RNA-Seq Data Set. This collection of data is part of the RNA-Seq (HiSeq) PANCAN data set, it is a random extraction of gene expressions of patients having different types of tumor: BRCA, KIRC, COAD, LUAD and PRAD.

Dataset 3:Titanic dataset
(http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv And
http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv)

Task 1: Perform Kmeans clustering on Dataset 1 with random initialization, 10 variations of initial means, 300 iterations. Find Lowest SSE value, final location of centroids and number of iterations to converge. Show the predicted labels for the first 10 points.

Task 2: Perform elbow method and silhouette method to find appropriate clustering value on Dataset 1.

Task 3. Use dataset 2 and create a clustering pipeline with pre-processing using PCA (2 components) and clustering using Kmeans on Dataset 2. Predict the label, calculate the silhouette score and plot a scatterplot for 2 PCA components.

Task 4: Perform data cleaning and pre-processing on dataset 3. Form three clustering using Kmeans++ initialization.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import seaborn as sns
```

```
features, true_labels =
make_blobs(n_samples=200,centers=3,cluster_std=2.75,random_state=42)
```

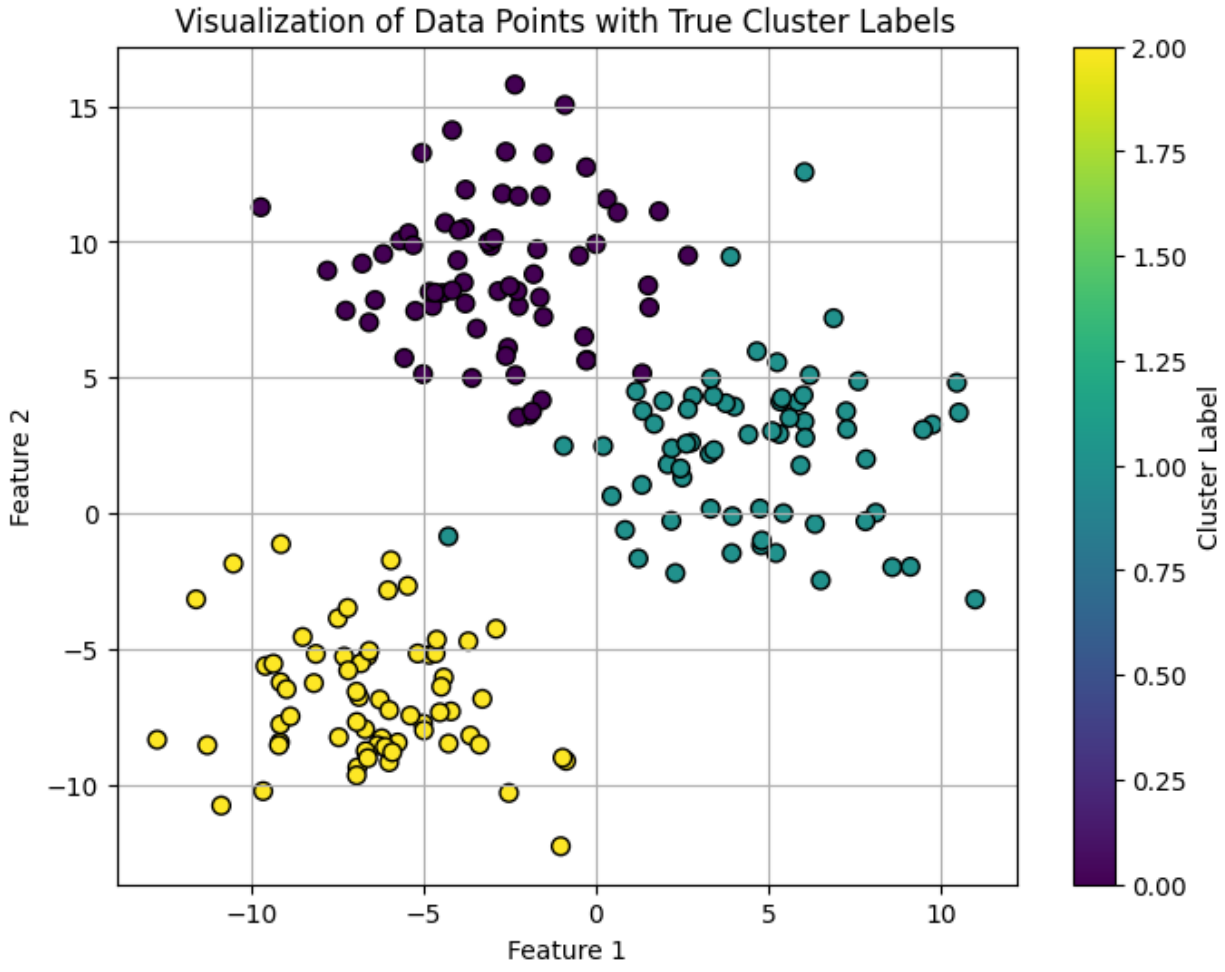
```
features[:5]
```

```
array([[  9.77075874,   3.27621022],
       [ -9.71349666,  11.27451802],
       [ -6.91330582,  -9.34755911],
       [-10.86185913, -10.75063497],
       [ -8.50038027,  -4.54370383]])
```

```
true_labels[:5]
```

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

```
plt.figure(figsize=(8, 6))
plt.scatter(features[:, 0], features[:, 1], c=true_labels, cmap='viridis',
s=50, edgecolor='k')
plt.title('Visualization of Data Points with True Cluster Labels')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Cluster Label')
plt.grid(True)
plt.show()
```



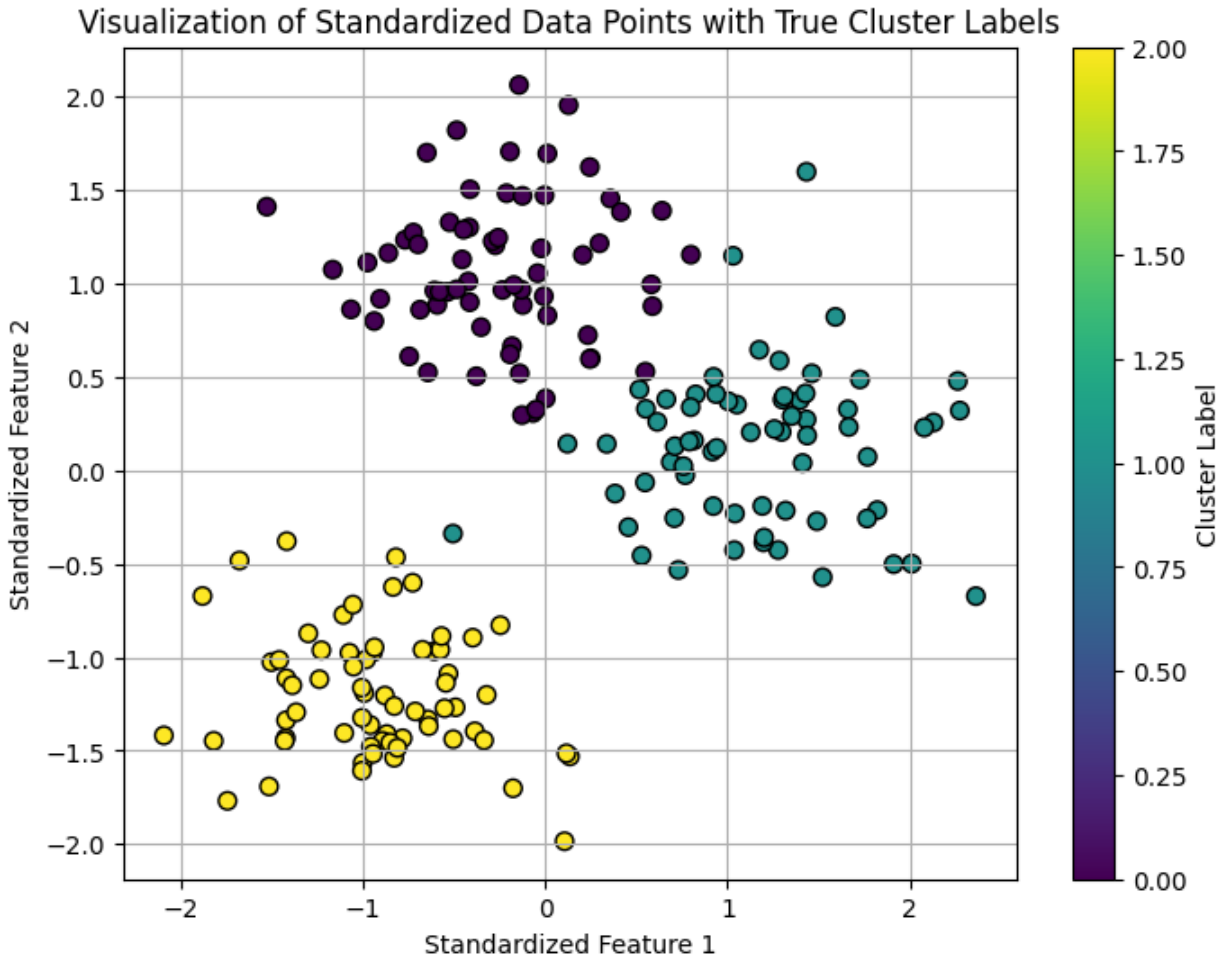

```
scaler = StandardScaler()
```

```
scaled_features = scaler.fit_transform(features)
```

```
scaled_features[:5]
```

```
array([[ 2.13082109,  0.25604351],
       [-1.52698523,  1.41036744],
       [-1.00130152, -1.56583175],
       [-1.74256891, -1.76832509],
       [-1.29924521, -0.87253446]])
```

```
plt.figure(figsize=(8, 6))
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=true_labels,
cmap='viridis', s=50, edgecolor='k')
plt.title('Visualization of Standardized Data Points with True Cluster
Labels')
plt.xlabel('Standardized Feature 1')
plt.ylabel('Standardized Feature 2')
plt.colorbar(label='Cluster Label')
plt.grid(True)
plt.show()
```




```
kmeans =
KMeans(init="random",n_clusters=3,n_init=10,max_iter=300,random_state=42)
```

```
kmeans.fit(scaled_features)
```

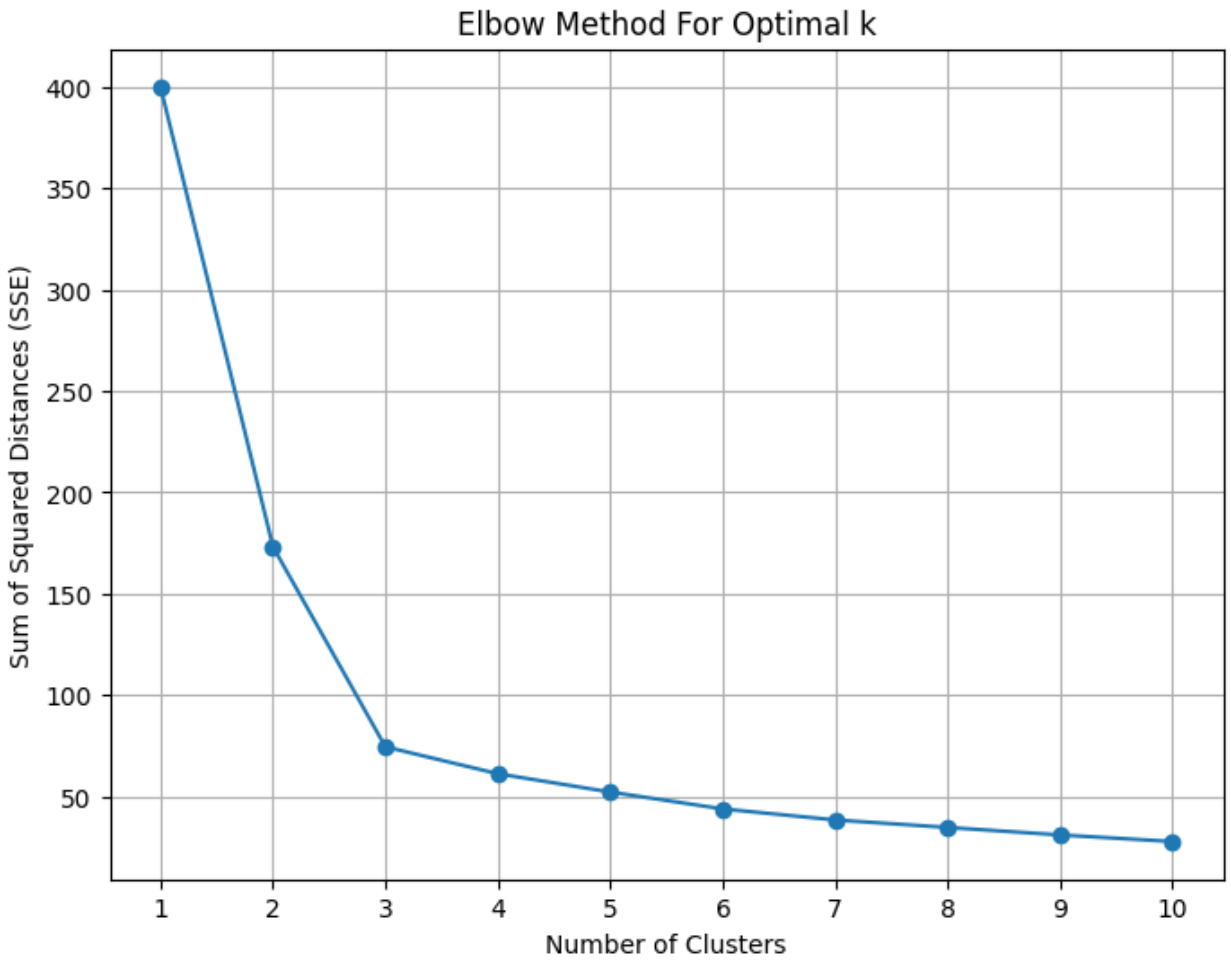
```
KMeans
KMeans(init='random', n_clusters=3, n_init=10, random_state=42)
```

```
kmeans.inertia_
```

```
kmeans.cluster_centers_
```

```
array([[-0.25813925,  1.05589975],
       [-0.91941183, -1.18551732],
       [ 1.19539276,  0.13158148]])
```

```
kmeans_kwargs = {"init": "random","n_init": 10,"max_iter":
300,"random_state": 42}

sse_list = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    sse_list.append(kmeans.inertia_)
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), sse_list, marker='o', linestyle='-')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances (SSE)')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()
```




```
sse_list
```

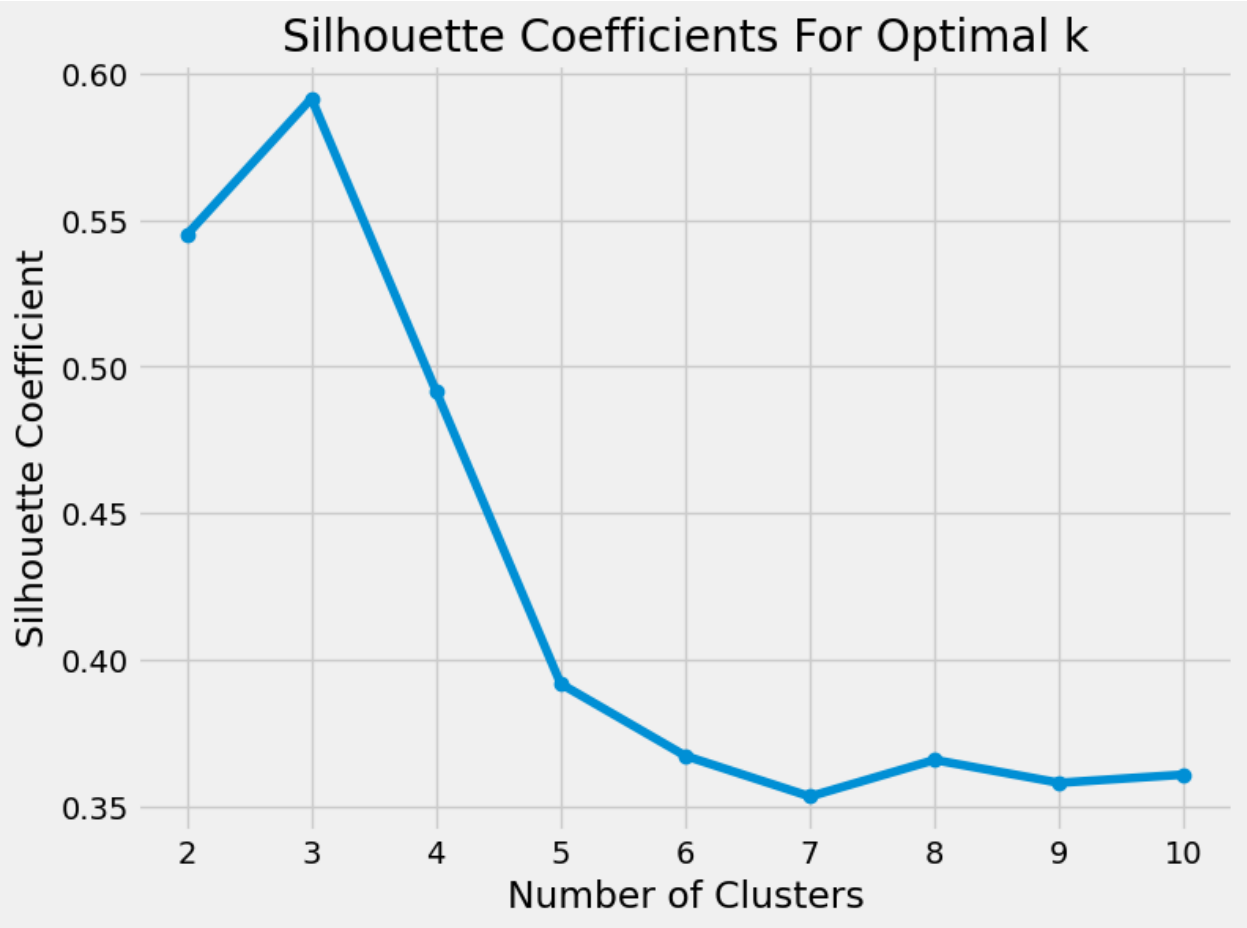
```
[400.0,
 173.2307489387777,
 74.57960106819854,
 61.30474344497751,
 52.2753872590211,
 43.963117678328196,
 38.52881875287544,
 34.826374070261245,
 31.138657464397568,
 27.896572369402648]
```

```
silhouette_coefficients = []
for k in range(2, 11):
  kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
  kmeans.fit(scaled_features)
  score = silhouette_score(scaled_features, kmeans.labels_)
  silhouette_coefficients.append(score)
```

```
silhouette_coefficients
```

```
[0.5449728784485278,
 0.5915043942870359,
 0.4916111520533458,
 0.39172760330275125,
 0.36713672897176236,
 0.35326132806361693,
 0.36574407217979027,
 0.3579272766766677,
 0.3607265826184605]
```

```
plt.figure(figsize=(8, 6))
plt.plot(range(2, 11), silhouette_coefficients, marker='o', linestyle='-')
plt.title('Silhouette Coefficients For Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Coefficient')
plt.xticks(range(2, 11))
plt.grid(True)
plt.show()
```




```
import tarfile
import urllib
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
```

```
uci_tcga_url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/00401/"
archive_name = "TCGA-PANCAN-HiSeq-801x20531.tar.gz"

full_download_url = urllib.parse.urljoin(uci_tcga_url, archive_name)
r = urllib.request.urlretrieve (full_download_url, archive_name)
tar = tarfile.open(archive_name, "r:gz")
tar.extractall()
```

```
tar.close()
```

```
datafile = "TCGA-PANCAN-HiSeq-801x20531/data.csv"
labels_file = "TCGA-PANCAN-HiSeq-801x20531/labels.csv"
data = np.genfromtxt(datafile,delimiter=",",usecols=range(1,
20532),skip_header=1)
true_label_names =
np.genfromtxt(labels_file,delimiter=",",usecols=(1,),skip_header=1,dtype="
str")
```

```
data[:5, :3]
```

```
array([[0.        , 2.01720929, 3.26552691],
       [0.        , 0.59273209, 1.58842082],
       [0.        , 3.51175898, 4.32719872],
       [0.        , 3.66361787, 4.50764878],
       [0.        , 2.65574107, 2.82154696]])
```

```
true_label_names[:5]
```

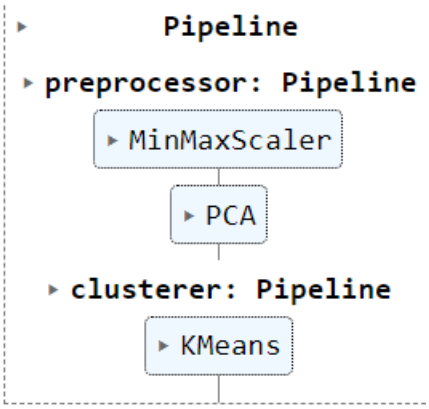
```
array(['PRAD', 'LUAD', 'PRAD', 'PRAD', 'BRCA'], dtype='<U4')
```

```
label_encoder = LabelEncoder()
true_labels = label_encoder.fit_transform(true_label_names)
true_labels[:10]
```

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

```
n_clusters = len(label_encoder.classes_)
preprocessor = Pipeline([("scaler", MinMaxScaler()),("pca",
PCA(n_components=2, random_state=42)),])
clusterer = Pipeline([("kmeans", KMeans(n_clusters=n_clusters, init="k-
means++", n_init=50, max_iter=500, random_state=42,),),])
pipe = Pipeline([("preprocessor", preprocessor), ("clusterer",
clusterer)])
```

```
pipe.fit(data)
```




```
preprocessed_data = pipe["preprocessor"].transform(data)
predicted_labels = pipe["clusterer"]["kmeans"].labels_
silhouette_score(preprocessed_data, predicted_labels)
```

```
0.5118775528450281
```

```
adjusted_rand_score(true_labels, predicted_labels)
```

```
0.722276752060253
```

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
train_url =
"http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv"
```

```
train = pd.read_csv(train_url)
test_url =
"http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test = pd.read_csv(test_url)
```

```
print("***** Train_Set *****")
print(train.head() )
print ("\n")
print("***** Test_Set *****")
print((test.head))
```

```
***** Train_Set *****
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
***** Test_Set *****
<bound method NDFrame.head of      PassengerId  Pclass
0            892       3                                Kelly, Mr. James
1            893       3                Wilkes, Mrs. James (Ellen Needs)
2            894       2                       Myles, Mr. Thomas Francis
3            895       3                               Wirz, Mr. Albert
4            896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
..           ...     ...                                           ...
413         1305       3                            Spector, Mr. Woolf
414         1306       1                  Oliva y Ocana, Dona. Fermina
415         1307       3                  Saether, Mr. Simon Sivertsen
416         1308       3                           Ware, Mr. Frederick
417         1309       3                      Peter, Master. Michael J

        Sex   Age  SibSp  Parch              Ticket      Fare Cabin Embarked
0      male  34.5      0      0              330911    7.8292   NaN        Q
1    female  47.0      1      0              363272    7.0000   NaN        S
2      male  62.0      0      0              240276    9.6875   NaN        Q
3      male  27.0      0      0              315154    8.6625   NaN        S
4    female  22.0      1      1             3101298   12.2875   NaN        S
..      ...   ...    ...    ...                 ...       ...   ...      ...
413    male   NaN      0      0           A.5. 3236    8.0500   NaN        S
414  female  39.0      0      0            PC 17758  108.9000  C105        C
415    male  38.5      0      0  SOTON/O.Q. 3101262    7.2500   NaN        S
416    male   NaN      0      0              359309    8.0500   NaN        S
417    male   NaN      1      1                2668   22.3583   NaN        C

[418 rows x 11 columns]>
```

```
train.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
test.isna().sum()
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
test = test.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
```

```
train.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	1	0	3	male	22.0	1	0	7.2500
1	2	1	1	female	38.0	1	0	71.2833
2	3	1	3	female	26.0	0	0	7.9250
3	4	1	1	female	35.0	1	0	53.1000
4	5	0	3	male	35.0	0	0	8.0500

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
train['Sex'] = label_encoder.fit_transform(train['Sex'])
```

```
test['Sex'] = label_encoder.fit_transform(test['Sex'])
test.head()
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare
0	892	3	1	34.5	0	0	7.8292
1	893	3	0	47.0	1	0	7.0000
2	894	2	1	62.0	0	0	9.6875
3	895	3	1	27.0	0	0	8.6625
4	896	3	0	22.0	1	1	12.2875

```
train.fillna(train.mean(), inplace=True)
test.fillna(test.mean(), inplace=True)
train.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Fare'],
      dtype='object')
```

```
X = np.array(train.drop(['Survived'], axis=1).astype(float))
y = np.array(train['Survived'])
```

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
```

```
KMeans
KMeans(n_clusters=2)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, random_state=None, tol=0.0001, verbose=0)
```

```
KMeans
KMeans(algorithm='auto', n_clusters=2, n_init=10)
```

```
correct = 0.
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.49270482603815935
```

```
kmeans = kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
kmeans.fit(X)
```

```
KMeans
KMeans(algorithm='auto', max_iter=600, n_clusters=2)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=600,
    n_clusters=2, n_init=10, random_state=None, tol=0.0001, verbose=0)
```

```
KMeans
KMeans(algorithm='auto', max_iter=600, n_clusters=2, n_init=10)
```

```
correct = 0.
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.49270482603815935
```

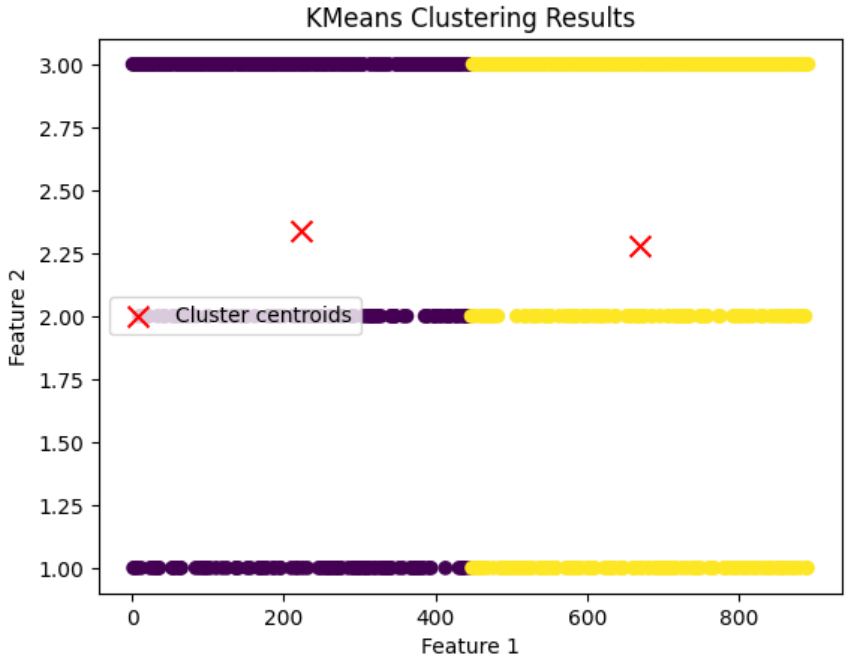
```
steps = [ ('kmeans', KMeans(n_clusters=2, random_state=None))  ]

pipeline = Pipeline(steps)

pipeline.fit(X)
```

```
Pipeline
  KMeans
```

```
cluster_labels = pipeline.predict(X)
```

```
centroids = pipeline.named_steps['kmeans'].cluster_centers_
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
```

```
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100, c='red',
label='Cluster centroids')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('KMeans Clustering Results')
plt.legend()
plt.show()
```




```
silhouette_avg = silhouette_score(X, cluster_labels)
print("Silhouette Score:", silhouette_avg)
```

```
Silhouette Score: 0.5951870787803006
```

Conclusion

The experiment involved applying K-means clustering on different datasets, exploring initialization methods, determining optimal cluster numbers using the elbow method and silhouette scores, and integrating preprocessing techniques like PCA. The outcomes provided insights into cluster formation and model performance, emphasizing the importance of preprocessing and careful initialization in achieving effective clustering results across diverse datasets.