Divesh Lulla **D15C** **Roll No:31**

EXPERIMENT-6

Problem Statement: Classification modelling.

Aim:
a. Choose classifier for classification problem.
b. Evaluate the performance of the classifier.

Theory:

Classification Modeling: Theory & Techniques

Classification modeling is a type of supervised learning in machine learning where the goal is to predict the category or class of a given data point based on input features. The model is trained using labeled data (i.e., data where the output class is known).

Classification problems can be:

● Binary Classification: Two classes (e.g., spam vs. not spam).
● Multiclass Classification: More than two classes (e.g., classifying types of flowers).
● Multi-label Classification: Each sample can belong to multiple classes.

1. K-Nearest Neighbors (KNN)
K-Nearest Neighbors (KNN) is a simple, non-parametric classification algorithm based on proximity to labeled examples.
Working Principle:
1. Choose a value for K (number of neighbors).
2. Compute the distance between the new data point and all training samples.
3. Select the K nearest neighbors.
4. Assign the majority class among the K neighbors as the predicted class.

2. Naive Bayes (NB)
Naïve Bayes is a probabilistic classifier based on Bayes' Theorem, assuming independence between predictors.
Bayes' Theorem:

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

Where:
● P(A|B) = Probability of class A given data B
● P(B|A) = Probability of data B given class A
● P(A) = Prior probability of class A
● P(B) = Prior probability of data B

3. Support Vector Machines (SVMs)
SVM is a powerful classification algorithm that finds the optimal hyperplane to separate data points into different classes.

Working Principle:
1. Hyperplane: A decision boundary that maximizes the margin between two classes.
2. Support Vectors: Data points that lie closest to the hyperplane and influence its position.
3. Kernel Trick: SVM can handle non-linearly separable data using kernel functions to transform the input space.

4. Decision Tree
A Decision Tree is a flowchart-like structure where internal nodes represent features, branches represent decisions, and leaves represent class labels.

Working Principle:
1. Splitting Criteria: Choose the best feature to split the data.
Gini Index: Measures impurity (Gini=1−∑pi2).
Entropy (Information Gain): Measures information gained from a split.
2. Recursive Splitting: Continue splitting nodes until a stopping criterion is met.
3. Pruning reduces overfitting by trimming branches.

Steps:

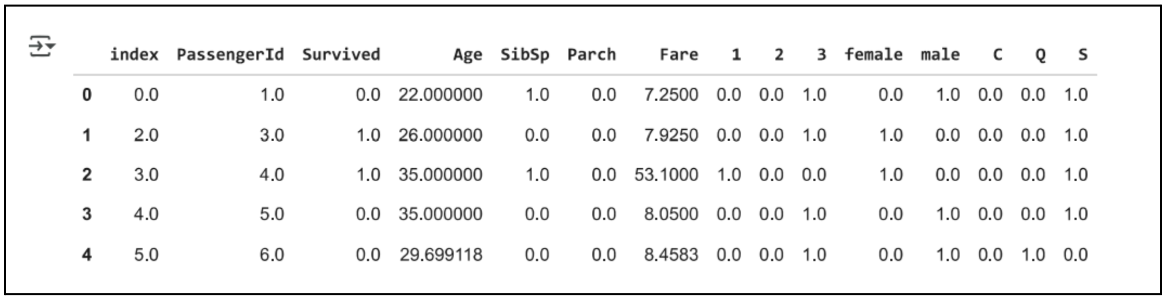
1)Import all the necessary libraries

All necessary Python libraries, like pandas, numpy, sklearn, and plotting libraries, are imported to handle data preprocessing, model building, and visualization.

```
import matplotlib.pyplot as pyplot
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd
import io
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn import preprocessing
plt.rc("font", size = 14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
sns.set(style = 'white')
sns.set(style="whitegrid",color_codes=True)
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

2) Encode Categorical Variables

Categorical features in the dataset are converted into numerical values using techniques like Label Encoding or One-Hot Encoding to make them usable by ML models.

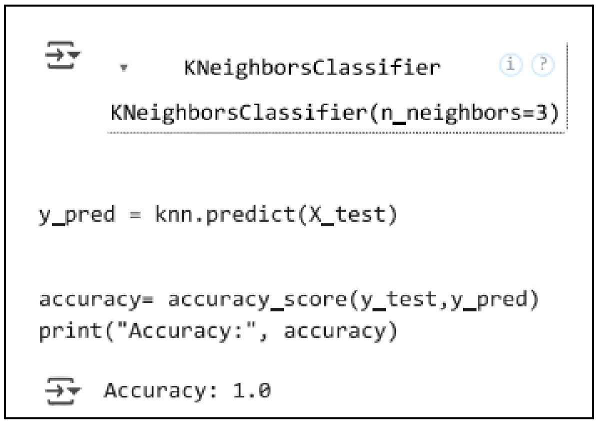


	index	PassengerId	Survived	Age	SibSp	Parch	Fare	1	2	3	female	male	C	Q	S
0	0.0	1.0	0.0	22.000000	1.0	0.0	7.2500	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
1	2.0	3.0	1.0	26.000000	0.0	0.0	7.9250	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0
2	3.0	4.0	1.0	35.000000	1.0	0.0	53.1000	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
3	4.0	5.0	0.0	35.000000	0.0	0.0	8.0500	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
4	5.0	6.0	0.0	29.699118	0.0	0.0	8.4583	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

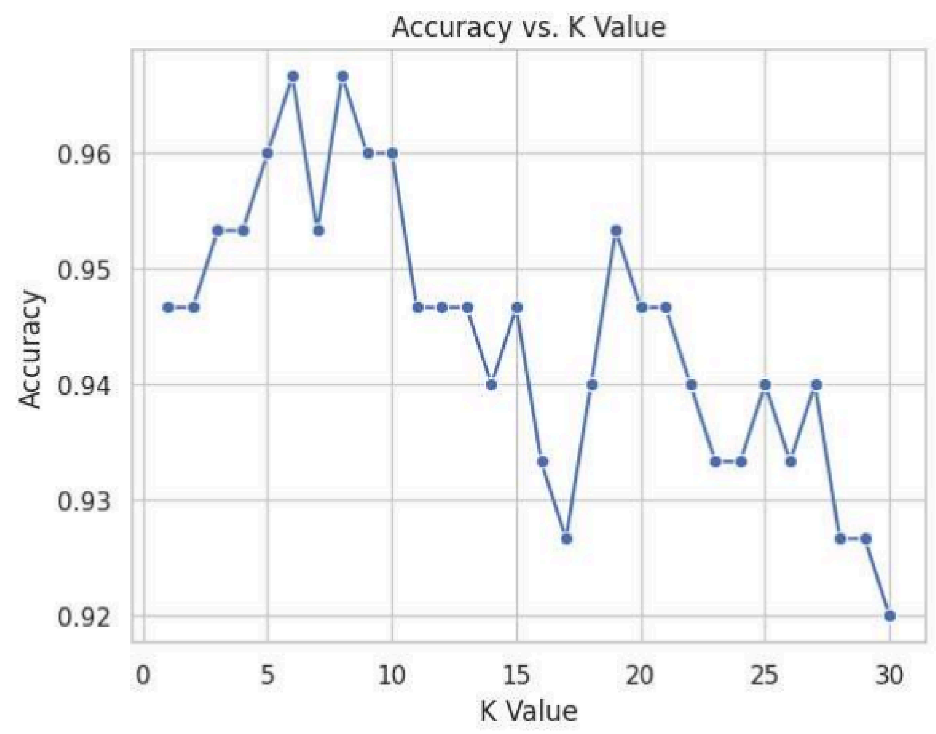
3) Train & Evaluate Classifiers

● K-Nearest Neighbors (KNN)

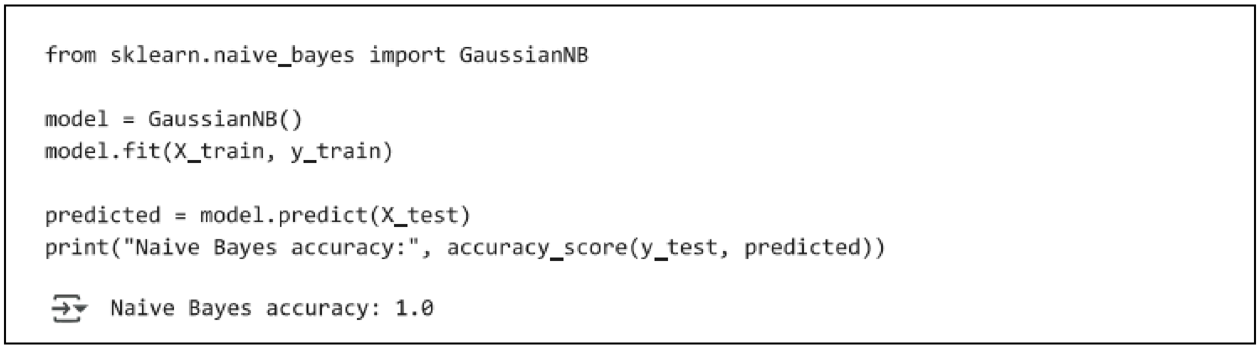
A KNN model is trained on the dataset, predictions are made, and performance is evaluated using metrics like accuracy, confusion matrix, etc.



```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)


y_pred = knn.predict(X_test)


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

Accuracy: 1.0
```

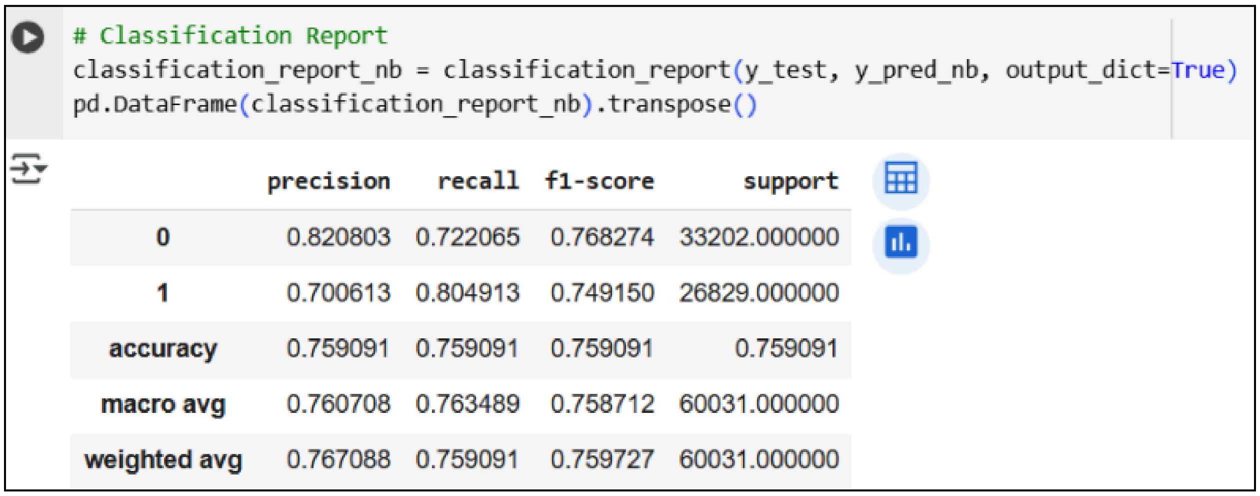


● **Naive Bayes**
The Naive Bayes algorithm is applied, where the model is trained and evaluated based on its probabilistic predictions.



```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train, y_train)

predicted = model.predict(X_test)
print("Naive Bayes accuracy:", accuracy_score(y_test, predicted))
```

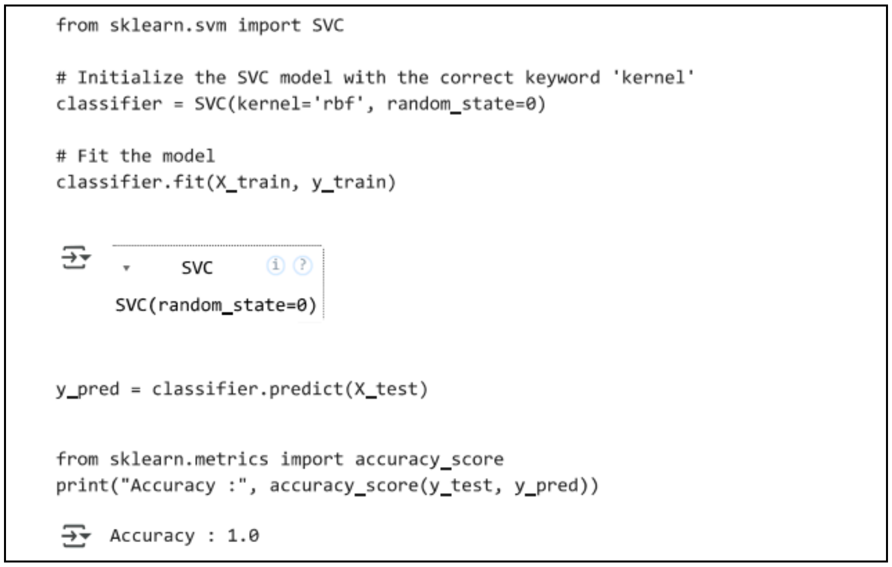
```
Naive Bayes accuracy: 1.0
```



```
# Classification Report
classification_report_nb = classification_report(y_test, y_pred_nb, output_dict=True)
pd.DataFrame(classification_report_nb).transpose()
```

	precision	recall	f1-score	support
0	0.820803	0.722065	0.768274	33202.000000
1	0.700613	0.804913	0.749150	26829.000000
accuracy	0.759091	0.759091	0.759091	0.759091
macro avg	0.760708	0.763489	0.758712	60031.000000
weighted avg	0.767088	0.759091	0.759727	60031.000000

● **Support Vector Machines (SVM)**
An SVM model is trained with the dataset, and its decision boundary is used to classify and evaluate the accuracy of predictions.



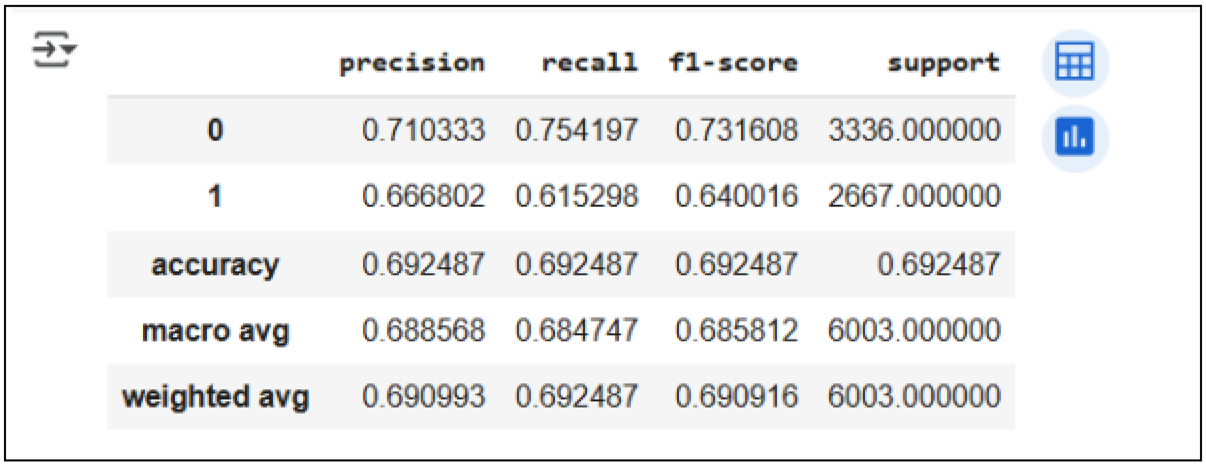
```
from sklearn.svm import SVC

# Initialize the SVC model with the correct keyword 'kernel'
classifier = SVC(kernel='rbf', random_state=0)

# Fit the model
classifier.fit(X_train, y_train)
```

```
SVC
SVC(random_state=0)
```

```
y_pred = classifier.predict(X_test)
```

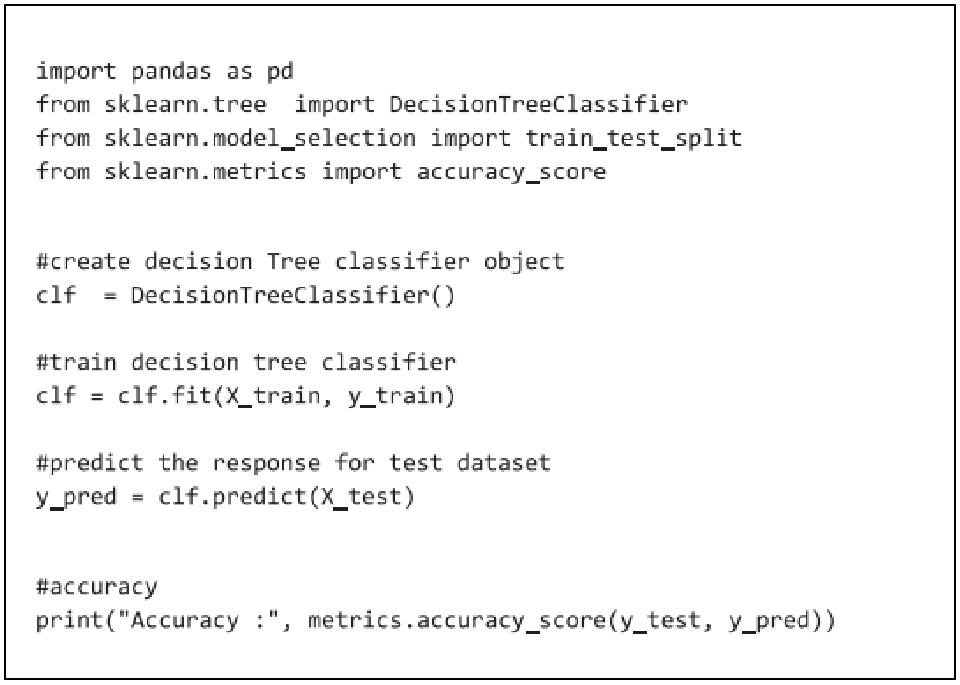
```
from sklearn.metrics import accuracy_score
print("Accuracy :", accuracy_score(y_test, y_pred))
```

```
Accuracy : 1.0
```

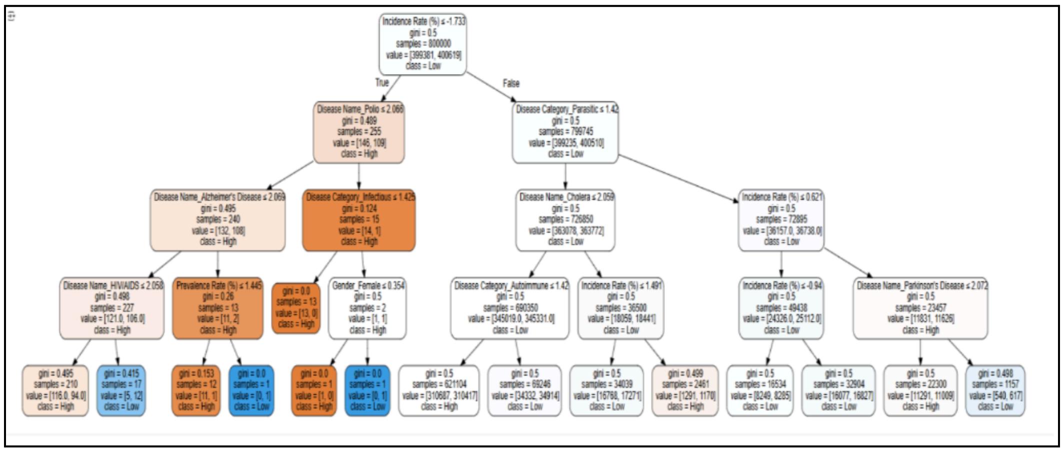


	precision	recall	f1-score	support
0	0.710333	0.754197	0.731608	3336.000000
1	0.666802	0.615298	0.640016	2667.000000
accuracy	0.692487	0.692487	0.692487	0.692487
macro avg	0.688568	0.684747	0.685812	6003.000000
weighted avg	0.690993	0.692487	0.690916	6003.000000

● **Decision Tree (With Visualization)**
A decision tree is trained, and the tree structure is visualized using tools like plot_tree() or graphviz to understand the splits and decision rules.



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


#create decision Tree classifier object
clf  = DecisionTreeClassifier()

#train decision tree classifier
clf = clf.fit(X_train, y_train)

#predict the response for test dataset
y_pred = clf.predict(X_test)


#accuracy
print("Accuracy :", metrics.accuracy_score(y_test, y_pred))
```


Incidence Rate (%) ≤ -1.733 gini = 0.5 samples = 800000 value = [399381, 400619] class = Low
True
False
Disease Name_Polio ≤ 2.066 gini = 0.489 samples = 255 value = [146, 109] class = High
Disease Category_Parasitic ≤ 1.42 gini = 0.5 samples = 799745 value = [399235, 400510] class = Low
Disease Name_Alzheimer's Disease ≤ 2.069 gini = 0.495 samples = 240 value = [132, 108] class = High
Disease Category_Infectious ≤ 1.425 gini = 0.124 samples = 15 value = [14, 1] class = High
Disease Name_Cholera ≤ 2.059 gini = 0.5 samples = 726850 value = [363078, 363772] class = Low
Incidence Rate (%) ≤ 0.621 gini = 0.5 samples = 72895 value = [36157.0, 36738.0] class = Low
Disease Name_HIV/AIDS ≤ 2.058 gini = 0.498 samples = 227 value = [121.0, 106.0] class = High
Prevalence Rate (%) ≤ 1.445 gini = 0.26 samples = 13 value = [11, 2] class = High
gini = 0.0 samples = 13 value = [13, 0] class = High
Gender_Female ≤ 0.354 gini = 0.5 samples = 2 value = [1, 1] class = High
Disease Category_Autoimmune ≤ 1.42 gini = 0.5 samples = 690350 value = [345019.0, 345331.0] class = Low
Incidence Rate (%) ≤ 1.491 gini = 0.5 samples = 36500 value = [18059, 18441] class = Low
Incidence Rate (%) ≤ -0.94 gini = 0.5 samples = 49438 value = [24326.0, 25112.0] class = Low
Disease Name_Parkinson's Disease ≤ 2.072 gini = 0.5 samples = 23457 value = [11831, 11626] class = High
gini = 0.495 samples = 210 value = [116.0, 94.0] class = High
gini = 0.415 samples = 17 value = [5, 12] class = Low
gini = 0.153 samples = 12 value = [11, 1] class = High
gini = 0.0 samples = 1 value = [0, 1] class = Low
gini = 0.0 samples = 1 value = [1, 0] class = High
gini = 0.0 samples = 1 value = [0, 1] class = Low
gini = 0.5 samples = 621104 value = [310687, 310417] class = High
gini = 0.5 samples = 69246 value = [34332, 34914] class = Low
gini = 0.5 samples = 34039 value = [16768, 17271] class = Low
gini = 0.499 samples = 2461 value = [1291, 1170] class = High
gini = 0.5 samples = 16534 value = [8249, 8285] class = Low
gini = 0.5 samples = 32904 value = [16077, 16827] class = Low
gini = 0.5 samples = 22300 value = [11291, 11009] class = High
gini = 0.498 samples = 1157 value = [540, 617] class = Low

Conclusion:

In this experiment, we worked with four classification algorithms—KNN, Naive Bayes, SVM, and Decision Trees applying them to a dataset after essential preprocessing. KNN is classified based on neighbor proximity, while Naive Bayes offered quick, reliable predictions using probability. SVM stood out with its ability to handle complex data using kernel tricks, and Decision Trees made results easy to interpret through visual flowcharts. Overall, this gave us a practical understanding of how different models perform and when to use each one effectively in real-world situations.