



Shaheed Zulfikar Ali Bhutto Institute of Science & Technology University

ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT

Data Mining

Report

Submitted To: Dr. Tehreem Qasim

Student Name: Syed Muhammad Mustafa

Reg. Number: 22108130



ROBOTICS & ARTIFICIAL INTELLIGENCE DEPARTMENT

Instructions: *Copied or shown assignments will be marked zero. Late submissions are not entertained in any case.*

Introduction:

The provided code focuses on predicting the survival of passengers aboard the Titanic using machine learning techniques. The dataset used, "train.csv," contains various attributes such as age, sex, ticket class, and embarkation port, which are utilized to train classification models.

Data Loading and Library Imports:

The code begins by importing necessary libraries such as pandas for data manipulation and scikit-learn modules for machine learning tasks. It loads the dataset into a Pandas DataFrame for further processing.

Exploratory Data Analysis:

A brief exploratory data analysis is conducted to understand the dataset's structure and quality. This includes displaying column names and checking for null values. Additionally, concise summary information about the DataFrame, including column names, data types, and memory usage, is provided.

Data Preprocessing:

Preprocessing is crucial for preparing the data for model training. The code handles tasks such as dropping irrelevant columns ('Name', 'Ticket', 'Cabin'), handling missing values, encoding categorical variables ('Sex', 'Embarked') into numerical format, and splitting the data into training and testing sets.

Model Training and Evaluation:

Two classification models, Gaussian Naive Bayes and Decision Tree, are trained on the preprocessed data. Both models are fitted on the training set and evaluated using the testing set. The accuracy of each model is calculated using the `accuracy_score` function from scikit-learn.

Model Comparison and Performance Analysis:

The performance of the Naive Bayes and Decision Tree classifiers is compared based on their accuracy scores. An analysis of their relative performance is provided, indicating which model performs better or if their performance is comparable.

Conclusion:

The code successfully demonstrates a workflow for predicting Titanic survival using machine learning techniques. It covers essential steps such as data loading, preprocessing, model training, evaluation, and performance analysis. Further improvements could involve exploring additional models, hyperparameter tuning, and more in-depth analysis of feature importance.

Code:

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

df = pd.read_csv("train.csv")
```

This block of code imports necessary libraries for data manipulation and machine learning, then loads the dataset "train.csv" into a Pandas DataFrame.

Code:

```
df.columns
```

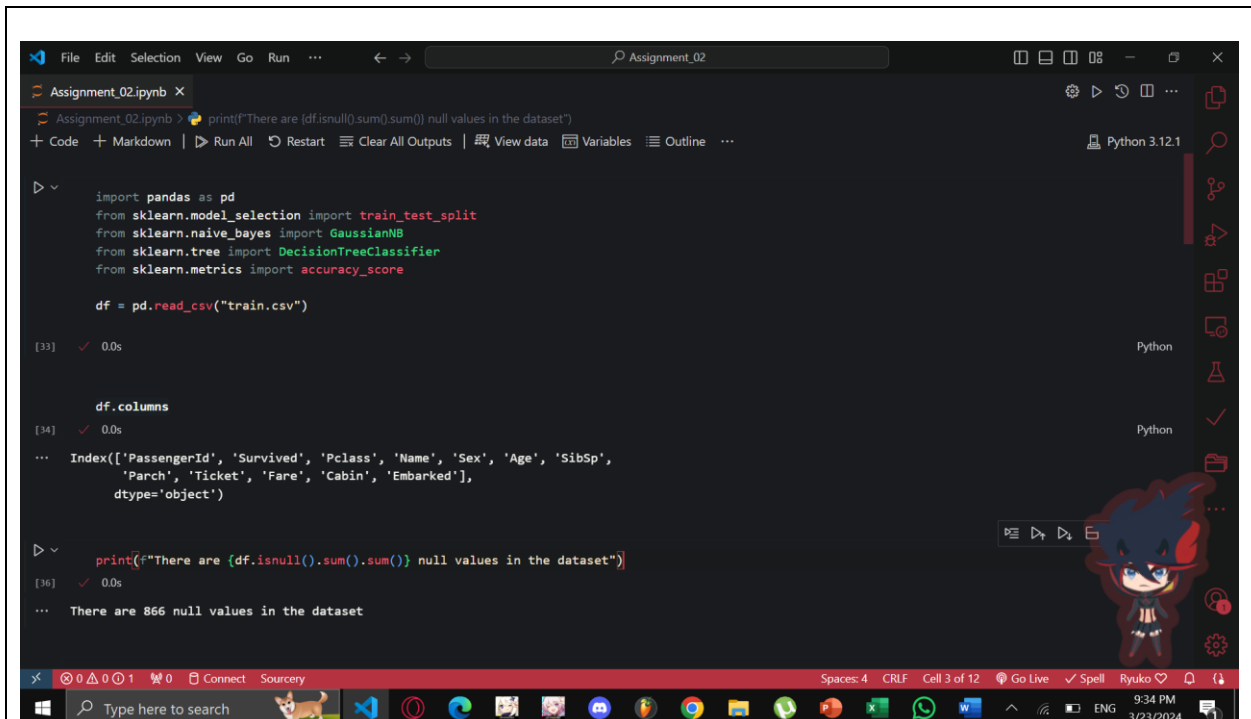
This block of code displays the column names of the DataFrame.

Code:

```
print(f"There are {df.isnull().sum().sum()} null values in the dataset")
```

This block calculates and prints the total number of null values in the dataset.

Output:



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

df = pd.read_csv("train.csv")

[33] ✓ 0.0s

df.columns

[34] ✓ 0.0s
... Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
        'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
        dtype='object')

print(f"There are {df.isnull().sum().sum()} null values in the dataset")

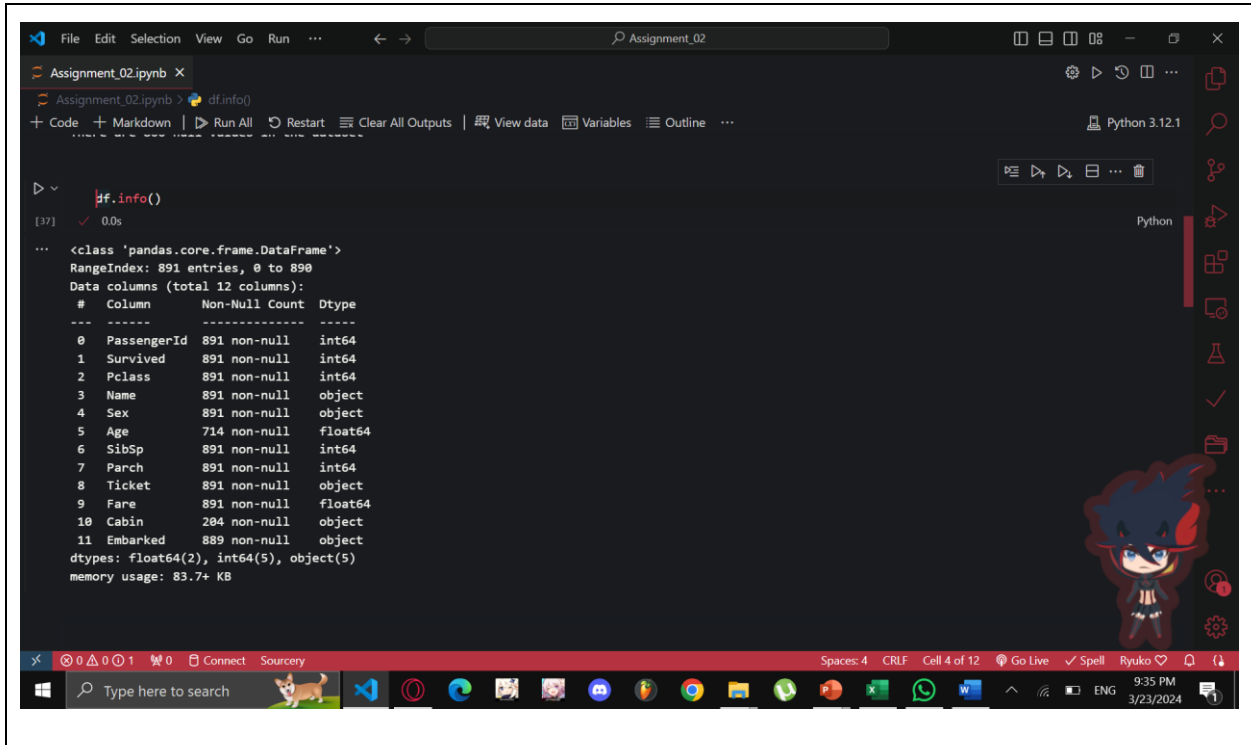
[36] ✓ 0.0s
... There are 866 null values in the dataset
```

Code:

```
df.info()
```

This block displays concise summary information about the DataFrame, including the column names, data types, and memory usage.

Output:



```
df.info()
```

```
[37]: ✓ 0.0s
```

```
>>> <class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  -
 0   PassengerId    891 non-null    int64  
 1   Survived       891 non-null    int64  
 2   Pclass        891 non-null    int64  
 3   Name          891 non-null    object  
 4   Sex           891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket        891 non-null    object  
 9   Fare         891 non-null    float64 
10   Cabin        294 non-null    object  
11   Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Code:

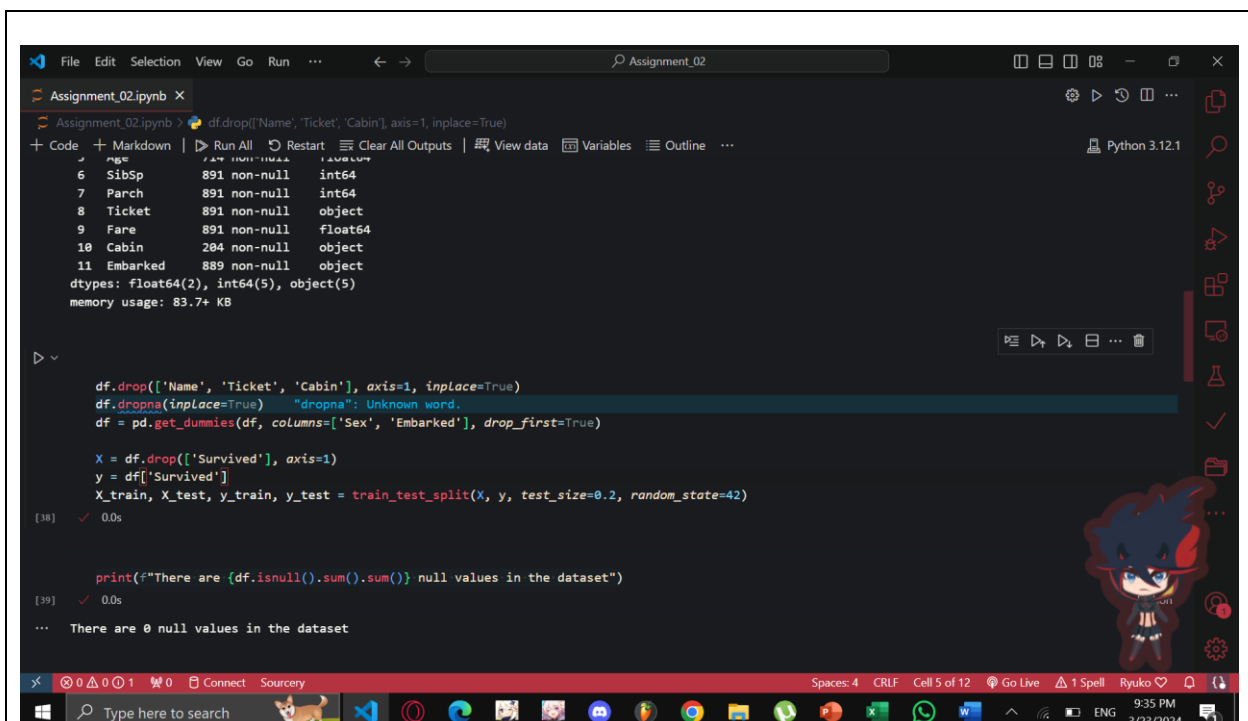
```
df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
df.dropna(inplace=True)
df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)

X = df.drop(['Survived'], axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print(f"There are {df.isnull().sum().sum()} null values in the dataset")
```

This block preprocesses the data by dropping irrelevant columns, handling missing values, converting categorical variables into dummy/indicator variables, and splitting the data into training and testing sets.

Output:



```
File Edit Selection View Go Run ... Assignment_02
Assignment_02.ipynb X
Assignment_02.ipynb > df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
+ Code + Markdown | ▶ Run All ⏮ Restart ⏭ Clear All Outputs | 🔍 View data 📄 Variables 📖 Outline ... Python 3.12.1
6 SibSp 891 non-null int64
7 Parch 891 non-null int64
8 Ticket 891 non-null object
9 Fare 891 non-null float64
10 Cabin 284 non-null object
11 Embarked 889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

df.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
df.dropna(inplace=True) "dropna": Unknown word.
df = pd.get_dummies(df, columns=['Sex', 'Embarked'], drop_first=True)

X = df.drop(['Survived'], axis=1)
y = df['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[38] ✓ 0.0s

print(f"There are {df.isnull().sum().sum()} null values in the dataset")

[39] ✓ 0.0s

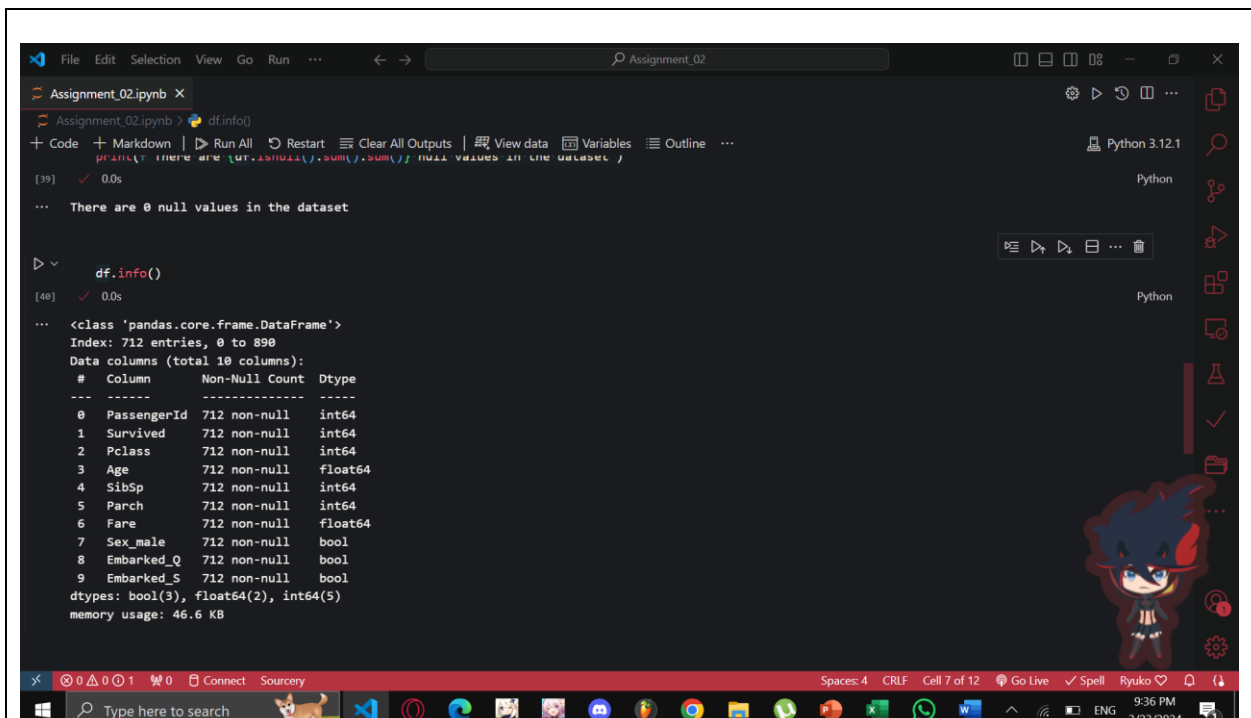
... There are 0 null values in the dataset
```

Code:

```
df.info()
```

This block displays updated information about the DataFrame after preprocessing.

Output:



```
Assignment_02.ipynb X
Assignment_02.ipynb > df.info()
+ Code + Markdown | Run All | Restart | Clear All Outputs | View data | Variables | Outline ...
Python 3.12.1

[39] ✓ 0.0s
... There are 0 null values in the dataset

df.info()
[40] ✓ 0.0s
... <class 'pandas.core.frame.DataFrame'>
Index: 712 entries, 0 to 890
Data columns (total 10 columns):
# Column Non-Null Count Dtype
---
0 PassengerId 712 non-null int64
1 Survived 712 non-null int64
2 Pclass 712 non-null int64
3 Age 712 non-null float64
4 SibSp 712 non-null int64
5 Parch 712 non-null int64
6 Fare 712 non-null float64
7 Sex_male 712 non-null bool
8 Embarked_Q 712 non-null bool
9 Embarked_S 712 non-null bool
dtypes: bool(3), float64(2), int64(5)
memory usage: 46.6 KB
```


Code:

```
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
y_pred_nb = naive_bayes.predict(X_test)
accuracy_naive_bayes = accuracy_score(y_test, y_pred_nb)
accuracy_naive_bayes_percentage = accuracy_naive_bayes * 100
accuracy_naive_bayes_percentage = round(accuracy_naive_bayes_percentage, 2)
```

This block trains a Gaussian Naive Bayes classifier, predicts the target variable for the test set, calculates the accuracy, and converts it to a percentage.

Code:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
y_pred_dt = decision_tree.predict(X_test)
accuracy_decision_tree = accuracy_score(y_test, y_pred_dt)
accuracy_decision_tree_percentage = accuracy_decision_tree * 100
accuracy_decision_tree_percentage = round(accuracy_decision_tree_percentage, 2)
```

This block trains a Decision Tree classifier, predicts the target variable for the test set, calculates the accuracy, and converts it to a percentage.

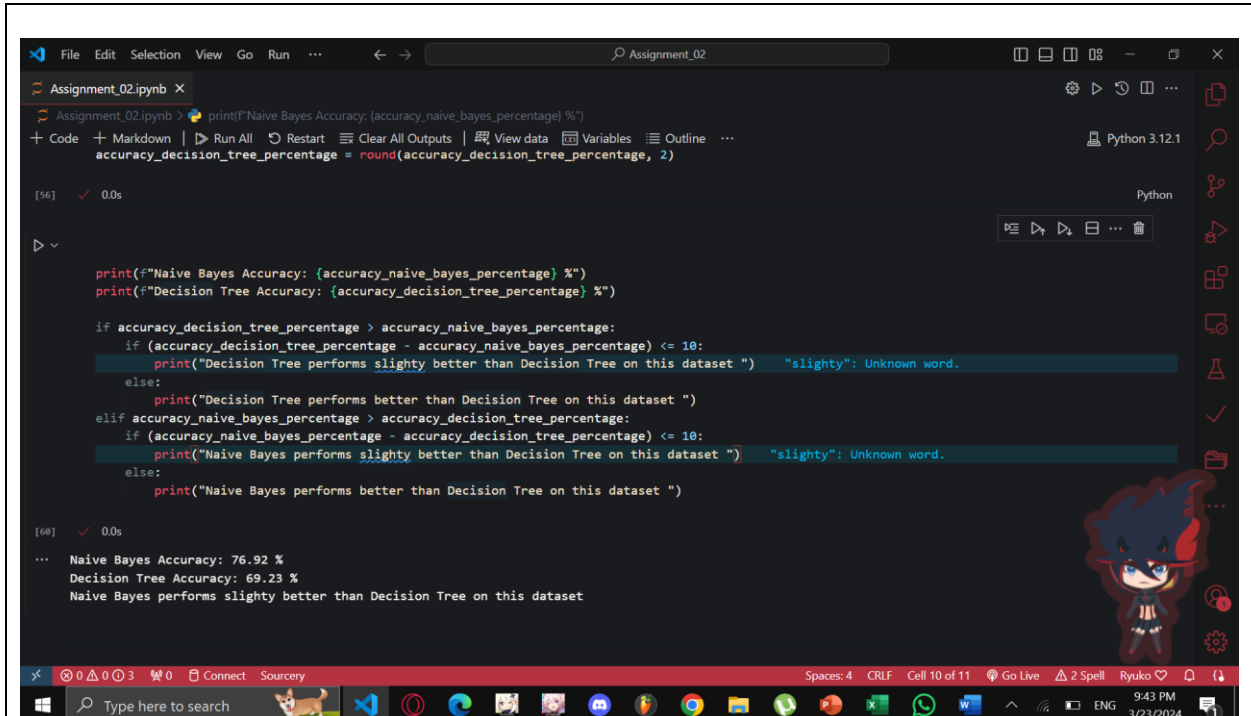
Code:

```
print(f"Naive Bayes Accuracy: {accuracy_naive_bayes_percentage} %")
print(f"Decision Tree Accuracy: {accuracy_decision_tree_percentage} %")

if accuracy_decision_tree_percentage > accuracy_naive_bayes_percentage:
    if (accuracy_decision_tree_percentage - accuracy_naive_bayes_percentage) <=
10:
        print("Decision Tree performs slightly better than Decision Tree on this
dataset ")
    else:
        print("Decision Tree performs better than Decision Tree on this dataset
")
elif accuracy_naive_bayes_percentage > accuracy_decision_tree_percentage:
    if (accuracy_naive_bayes_percentage - accuracy_decision_tree_percentage) <=
10:
        print("Naive Bayes performs slightly better than Decision Tree on this
dataset ")
    else:
        print("Naive Bayes performs better than Decision Tree on this dataset
")
```

This block compares the accuracies of the Naive Bayes and Decision Tree classifiers and provides an analysis based on their performance. If one model performs significantly better than the other (with a difference of more than 10%), it indicates which model performs better. Otherwise, it states that they perform similarly.

Output:



```
Assignment_02.ipynb X
Assignment_02.ipynb > print(f"Naive Bayes Accuracy: {accuracy_naive_bayes_percentage} %")
+ Code + Markdown | ▶ Run All ↺ Restart ≡ Clear All Outputs | 🔍 View data 📄 Variables 📖 Outline ... Python 3.12.1
[56] ✓ 0.0s

print(f"Naive Bayes Accuracy: {accuracy_naive_bayes_percentage} %")
print(f"Decision Tree Accuracy: {accuracy_decision_tree_percentage} %")

if accuracy_decision_tree_percentage > accuracy_naive_bayes_percentage:
    if (accuracy_decision_tree_percentage - accuracy_naive_bayes_percentage) <= 10:
        print("Decision Tree performs slightly better than Decision Tree on this dataset ") "slightly": Unknown word.
    else:
        print("Decision Tree performs better than Decision Tree on this dataset ")
elif accuracy_naive_bayes_percentage > accuracy_decision_tree_percentage:
    if (accuracy_naive_bayes_percentage - accuracy_decision_tree_percentage) <= 10:
        print("Naive Bayes performs slightly better than Decision Tree on this dataset ") "slightly": Unknown word.
    else:
        print("Naive Bayes performs better than Decision Tree on this dataset ")

[58] ✓ 0.0s

... Naive Bayes Accuracy: 76.92 %
    Decision Tree Accuracy: 69.23 %
    Naive Bayes performs slightly better than Decision Tree on this dataset
```