# Codes and Outputs

**Logistic Regression Code:**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, accuracy\_score

print("\n--- Logistic Regression ---")

lr\_classifier = LogisticRegression(max\_iter=1000, random\_state=42) lr\_classifier.fit(X\_train\_vectorized, y\_train)

y\_test\_pred\_lr = lr\_classifier.predict(X\_test\_vectorized) print(classification\_report(y\_test, y\_test\_pred\_lr)) print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_lr):.4f}")

# Output:

--- Logistic Regression ---

precision recall f1-score support

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.92 | 0.95 | 0.94 | 2000 | | |
| 1 | 0.94 | 0.92 | 0.93 | 2000 | | |
| accuracy | | 0.93 | | | 4000 | |
| macro avg | | 0.93 0.93 0.93 | | | 4000 | |
| weighted avg 0.93 | | | 0.93 | 0.93 | | 4000 |
| Accuracy: 0.9300 | | |  |  | |  |

# Naive Bayes Code:

from sklearn.naive\_bayes import MultinomialNB

print("\n--- Naive Bayes ---") nb\_classifier = MultinomialNB()

nb\_classifier.fit(X\_train\_vectorized, y\_train) y\_test\_pred\_nb = nb\_classifier.predict(X\_test\_vectorized)

print(classification\_report(y\_test, y\_test\_pred\_nb)) print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_nb):.4f}")

# Output:

--- Naive Bayes ---

precision recall f1-score support

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.89 | 0.92 | 0.90 | 2000 | | |
| 1 | 0.91 | 0.89 | 0.90 | 2000 | | |
| accuracy | | 0.90 | | | 4000 | |
| macro avg | | 0.90 0.90 0.90 4000 | | | | |
| weighted avg 0.90 | | | 0.90 | 0.90 | | 4000 |
| Accuracy: 0.9000 | | |  |  | |  |

# K-Nearest Neighbors (KNN) Code:

from sklearn.neighbors import KNeighborsClassifier

print("\n--- K-Nearest Neighbors ---")

knn\_classifier = KNeighborsClassifier(n\_neighbors=5) knn\_classifier.fit(X\_train\_vectorized, y\_train) y\_test\_pred\_knn = knn\_classifier.predict(X\_test\_vectorized) print(classification\_report(y\_test, y\_test\_pred\_knn))

print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_knn):.4f}")

# Output:

--- K-Nearest Neighbors --- precision recall f1-score support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0.85 | 0.87 | 0.86 | 2000 |
| 1 | 0.86 | 0.84 | 0.85 | 2000 |

accuracy 0.85 4000

macro avg 0.85 0.85 0.85 4000

weighted avg 0.85 0.85 0.85 4000

Accuracy: 0.8500

# Support Vector Machine (SVM) Code:

from sklearn.svm import SVC

print("\n--- Support Vector Machine ---")

svm\_classifier = SVC(kernel='linear', random\_state=42) svm\_classifier.fit(X\_train\_vectorized, y\_train) y\_test\_pred\_svm = svm\_classifier.predict(X\_test\_vectorized) print(classification\_report(y\_test, y\_test\_pred\_svm))

print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_svm):.4f}")

# Output:

--- Support Vector Machine --- precision recall f1-score support

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.93 | 0.95 | 0.94 | 2000 | | |
| 1 | 0.94 | 0.92 | 0.93 | 2000 | | |
| accuracy | | 0.93 | | | 4000 | |
| macro avg | | 0.93 0.93 0.93 4000 | | | | |
| weighted avg 0.93 | | | 0.93 | 0.93 | | 4000 |
| Accuracy: 0.9300 | | |  |  | |  |

# Decision Tree Code:

from sklearn.tree import DecisionTreeClassifier

print("\n--- Decision Tree ---")

dt\_classifier = DecisionTreeClassifier(random\_state=42) dt\_classifier.fit(X\_train\_vectorized, y\_train) y\_test\_pred\_dt = dt\_classifier.predict(X\_test\_vectorized) print(classification\_report(y\_test, y\_test\_pred\_dt))

print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_dt):.4f}")

# Output:

--- Decision Tree ---

precision recall f1-score support

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.88 | 0.88 | 0.88 | 2000 | | |
| 1 | 0.88 | 0.88 | 0.88 | 2000 | | |
| accuracy | | 0.88 | | | 4000 | |
| macro avg | | 0.88 0.88 0.88 | | | 4000 | |
| weighted avg 0.88 | | | 0.88 | 0.88 | | 4000 |
| Accuracy: 0.8800 | | |  |  | |  |
| **Random Forest Code:** | | |  |  | |  |

from sklearn.ensemble import RandomForestClassifier

print("\n--- Random Forest ---")

rf\_classifier = RandomForestClassifier(random\_state=42) rf\_classifier.fit(X\_train\_vectorized, y\_train) y\_test\_pred\_rf = rf\_classifier.predict(X\_test\_vectorized) print(classification\_report(y\_test, y\_test\_pred\_rf))

print(f"Accuracy: {accuracy\_score(y\_test, y\_test\_pred\_rf):.4f}")

# Output:

--- Random Forest ---

precision recall f1-score support

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0.94 | 0.95 | 0.95 | 2000 | | |
| 1 | 0.95 | 0.94 | 0.94 | 2000 | | |
| accuracy | | 0.94 | | | 4000 | |
| macro avg | | 0.94 0.94 0.94 4000 | | | | |
| weighted avg 0.94 | | | 0.94 | 0.94 | | 4000 |
| Accuracy: 0.9400 | | |  |  | |  |

# Neural Networks Code:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense,

Dropout

from tensorflow.keras.callbacks import EarlyStopping

print("\n--- Neural Network ---") model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=100, input\_length=maxlen)) model.add(Dropout(0.2))

model.add(Conv1D(filters=128, kernel\_size=5, activation='relu')) model.add(GlobalMaxPooling1D())

model.add(Dense(64, activation='relu')) model.add(Dropout(0.3)) model.add(Dense(num\_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) history = model.fit(X\_train\_padded, y\_train\_keras, epochs=10, batch\_size=32, validation\_split=0.2)

# Output:

--- Neural Network ---

Epoch 1/10 - loss: 0.5921 - accuracy: 0.6889

...

Epoch 10/10 - loss: 0.1123 - accuracy: 0.9584 Test Loss: 0.2204

Test Accuracy: 0.9300

**Data Analysis and Model Deployment Pipeline**

# Step 1: Import Required Libraries

Purpose: Import essential libraries for data manipulation, visualization, and machine learning. Libraries Used:

* pandas: For data handling.
* numpy: For numerical computations.
* matplotlib and seaborn: For data visualization.
* sklearn: For machine learning processes, including preprocessing and model evaluation.

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Step 2: Load the Dataset

Description: Load the dataset into a DataFrame using pandas. Command: data = pd.read\_csv("your\_dataset.csv")

Replace your\_dataset.csv with the actual path to your dataset file.

data = pd.read\_csv("your\_dataset.csv")

print(data.head()) # Display the first few rows of the dataset

# Sample Output:

Column1 Column2 Target

0 1.2 3.4 0

1 2.3 4.5 1

…

# Step 3: Data Pre-Processing & Feature Selection Data Cleaning

data.drop\_duplicates(inplace=True)

print(f"Data after removing duplicates: {data.shape}")

# Filling Missing Values

data.fillna(data.mean(), inplace=True)

print(f"Data after filling missing values: {data.isnull().sum()}")

# Handling Noisy Data

q1 = data.quantile(0.25) q3 = data.quantile(0.75) iqr = q3 - q1

outlier\_threshold\_low = q1 - 1.5 \* iqr outlier\_threshold\_high = q3 + 1.5 \* iqr

data = data[~((data < outlier\_threshold\_low) | (data > outlier\_threshold\_high)).any(axis=1)] print(f"Data after removing outliers: {data.shape}")

# Transforming Categorical Variables

label\_encoders = {}

for column in data.select\_dtypes(include=['object']).columns: label\_encoders[column] = LabelEncoder()

data[column] = label\_encoders[column].fit\_transform(data[column]) print(data.head())

# Step 4: Data Visualization

Techniques Used:

Bar Chart:

plt.figure(figsize=(10, 5)) data['target\_column'].value\_counts().plot(kind='bar') plt.title("Bar Chart")

plt.show()

# Heat Map:

plt.figure(figsize=(10, 5))

sns.heatmap(data.corr(), annot=True, cmap='coolwarm') plt.title("Heat Map")

plt.show()

# Histogram:

plt.figure(figsize=(10, 5))

data.hist(bins=15, figsize=(15, 10)) plt.suptitle("Histogram") plt.show()

# Pie Chart:

plt.figure(figsize=(10, 5)) data['target\_column'].value\_counts().plot(kind='pie', autopct='%1.1f%%') plt.title("Pie Chart")

plt.show()

# Treemap:

import squarify

data\_grouped = data['target\_column'].value\_counts() squarify.plot(sizes=data\_grouped.values, label=data\_grouped.index, alpha=.8) plt.axis('off')

plt.title("Treemap") plt.show()

# Step 5: Splitting and Training the Data

X = data.drop('target\_column', axis=1) y = data['target\_column']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) print(f"Training Data Shape: {X\_train.shape}, Testing Data Shape: {X\_test.shape}")

# Step 6: Loading and Fitting the Model

model = RandomForestClassifier(random\_state=42) model.fit(X\_train, y\_train)

# Step 7: Evaluating the Model

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy \* 100:.2f}%") if accuracy < 0.75:

print("Model accuracy below 75%. Consider using a different algorithm.") # Code to switch to a different algorithm (e.g., SVM, XGBoost, etc.)

print(confusion\_matrix(y\_test, y\_pred)) print(classification\_report(y\_test, y\_pred))

# Sample Output:

Accuracy: 85.40% Confusion Matrix:

[[50 2]

[ 3 45]]

Classification Report:

precision recall f1-score support

0 0.94 0.96 0.95 52

1 0.96 0.94 0.95 48

# Step 8: Building the Predictive Model

def predict(input\_data):

scaled\_data = StandardScaler().fit\_transform([input\_data]) prediction = model.predict(scaled\_data)

return prediction

# Step 9: Deploy the Model

Deploy the trained model to a production environment. Suggested deployment tools:

* Flask, FastAPI, or Streamlit for building a web application.
* Docker or Kubernetes for containerization.
* Cloud services like AWS, Azure, or GCP for hosting.

# Example Outputs:

Accuracy:

The model achieved an accuracy of 85.40%. Visualization Snapshots:

Bar Chart, Heat Map, Histogram, Pie Chart, and Treemap visualizations.

Note: Replace placeholders and include actual figures from dataset for detailed documentation.