**Experiment No. 6**

**Aim :** Diagnose disease risk from Patient data.

**Theory :**

**Objective:** The aim of this analysis is to diagnose disease risk, specifically diabetes, from patient data. By analyzing various health metrics and demographic information, we can develop a predictive model that identifies patients at risk of diabetes.

**Dataset Description:** The dataset used in this analysis is a diabetes dataset, which contains multiple features related to patients’ health metrics. The columns typically include:

* **Pregnancies:** Number of times pregnant.
* **Glucose:** Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
* **Blood Pressure:** Diastolic blood pressure (mm Hg).
* **Skin Thickness:** Triceps skin fold thickness (mm).
* **Insulin:** 2-Hour serum insulin (mu U/ml).
* **BMI:** Body mass index (weight in kg/(height in m)^2).
* **Diabetes Pedigree Function:** A function that scores the likelihood of diabetes based on family history.
* **Age:** Age of the patient.
* **Outcome:** Indicates whether the patient has diabetes (1 = positive, 0 = negative).

**Methodology:**

1. **Data Exploration:** Understanding the dataset through summary statistics and class distribution helps identify potential issues, such as imbalanced classes or outliers.
2. **Model Selection:** A Random Forest Classifier is chosen for this analysis due to its robustness, ability to handle non-linear relationships, and feature importance evaluation.
3. **Model Training and Evaluation:** The dataset is split into training and testing sets to train the model and evaluate its performance. Key metrics such as accuracy, precision, recall, and specificity are computed to assess model effectiveness.

**Key Steps in the Code**

1. **Import Libraries:** Libraries for data handling (Pandas, NumPy) and machine learning (Scikit-learn) are imported.
2. **Load Data:** The diabetes dataset is loaded using **pd.read\_csv().**
3. **Exploratory Data Analysis (EDA):**
   * Summary statistics are generated using **data.describe()** to understand the range and distribution of each feature.
   * The class distribution of the target variable (Outcome) is checked to identify any potential class imbalance.
4. **Data Preparation:**
   * Features (X) and target variable (y) are defined. The target variable is the 'Outcome', indicating diabetes presence.
5. **Data Splitting:** The data is split into training and testing sets (70% for training, 30% for testing) using **train\_test\_split(),** ensuring reproducibility with a fixed random state.
6. **Model Training:**
   * A Random Forest Classifier is instantiated and trained on the training data with fit().
7. **Predictions:** Predictions are made on the test set using **predict().**
8. **Model Evaluation:**
   * Several metrics are computed:
     + **Accuracy:** Overall correctness of the model.
     + **Confusion Matrix:** A matrix that shows true vs. predicted classifications.
     + **Precision:** The ratio of true positives to the sum of true and false positives.
     + **Recall:** The ratio of true positives to the sum of true positives and false negatives.
     + **Specificity:** The ratio of true negatives to the sum of true negatives and false positives.
9. **Results:** The evaluation metrics are printed to assess the model's performance.

**Code :**

import pandas as pd

import numpy as np

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (accuracy\_score, confusion\_matrix,

precision\_score, recall\_score)

# Load the dataset

data = pd.read\_csv("/kaggle/input/diabetes-dataset/diabetes.csv")

data.head()

# Perform exploratory data analysis

# Summary statistics

summary\_stats = data.describe()

print(summary\_stats)

# Class distribution

class\_distribution = data['Outcome'].value\_counts()

print(class\_distribution)

# Split the data into features and target variable

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

# Split the data into training and testing sets (70% train, 30% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Create a Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the classifier on the training data

rf\_classifier.fit(X\_train, y\_train)

# Make predictions on the testing data

y\_pred = rf\_classifier.predict(X\_test)

# Evaluate the model's performance

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

specificity = confusion[0][0] / (confusion[0][0] + confusion[0][1]) # Calculating specificity

# Print the results

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", confusion)

print("Precision:", precision)

print("Recall:", recall)

print("Specificity:", specificity)

**Output:**

Pregnancies Glucose BloodPressure SkinThickness Insulin \

count 768.000000 768.000000 768.000000 768.000000 768.000000

mean 3.845052 120.894531 69.105469 20.536458 79.799479

std 3.369578 31.972618 19.355807 15.952218 115.244002

min 0.000000 0.000000 0.000000 0.000000 0.000000

25% 1.000000 99.000000 62.000000 0.000000 0.000000

50% 3.000000 117.000000 72.000000 23.000000 30.500000

75% 6.000000 140.250000 80.000000 32.000000 127.250000

max 17.000000 199.000000 122.000000 99.000000 846.000000

BMI DiabetesPedigreeFunction Age Outcome

count 768.000000 768.000000 768.000000 768.000000

mean 31.992578 0.471876 33.240885 0.348958

std 7.884160 0.331329 11.760232 0.476951

min 0.000000 0.078000 21.000000 0.000000

25% 27.300000 0.243750 24.000000 0.000000

50% 32.000000 0.372500 29.000000 0.000000

75% 36.600000 0.626250 41.000000 1.000000

max 67.100000 2.420000 81.000000 1.000000

Outcome

0 500

1 268

Name: count, dtype: int64

Accuracy: 0.7532467532467533

Confusion Matrix:

[[121 30]

[ 27 53]]

Precision: 0.6385542168674698

Recall: 0.6625

Specificity: 0.8013245033112583

**Conclusion :**

**Experiment No. 7**

**Aim :** Social Media Analytics for outbreak prediction/ Drug review analytics.

**Theory :**

**Objective:** The objective of this analysis is to perform sentiment analysis on drug reviews to determine patient sentiments towards various medications. By transforming patient feedback into a binary classification problem (positive or negative sentiment), we can gain insights into drug effectiveness and patient satisfaction, ultimately aiding healthcare providers and pharmaceutical companies in improving drug development and patient care strategies.

**Context:**

1. **Importance of Patient Feedback:**
   * Patient reviews provide firsthand accounts of medication experiences, including effectiveness, side effects, and overall satisfaction. Analyzing this feedback is essential for understanding public perception of drugs.
2. **Sentiment Analysis:**
   * Sentiment analysis involves classifying text data to identify the sentiment expressed within. In this context, we classify drug reviews based on patient ratings. Ratings of 3 or higher indicate a positive sentiment, while ratings below 3 indicate a negative sentiment.
3. **Machine Learning Approach:**
   * Using machine learning models, specifically Logistic Regression, we can analyze the patterns in drug reviews and predict the sentiment based on various features. This predictive modeling approach allows us to automate sentiment analysis and derive actionable insights.

**Methodology**

1. **Data Collection:**
   * The dataset used in this analysis contains drug reviews, including attributes such as drug name, condition being treated, review text, and patient ratings.
2. **Data Preprocessing:**
   * The data is cleaned and transformed to prepare it for analysis. This includes creating a binary target variable that reflects patient sentiment based on drug ratings.
   * Categorical features (such as drug name and condition) are encoded into numerical formats using techniques like Label Encoding to make them suitable for machine learning models.
3. **Feature Selection:**
   * Relevant features are selected for the model, excluding non-essential information (like unique identifiers and dates). The features may include encoded categorical variables and any other numerical variables deemed relevant.
4. **Model Development:**
   * The dataset is split into training and testing sets to evaluate model performance. An 80/20 split ensures that the model is trained on a substantial amount of data while still having a robust test set to validate predictions.
   * A Logistic Regression model is selected due to its effectiveness for binary classification tasks. The model learns from the training data to identify patterns associated with positive and negative sentiments.
5. **Model Evaluation:**
   * Predictions are made on the test set, and model performance is evaluated using accuracy as the primary metric. Accuracy indicates the proportion of correctly predicted sentiments in relation to the total predictions made.

**Implications**

1. **Healthcare Insights:**
   * The results of the sentiment analysis can provide healthcare professionals with valuable insights into patient experiences with medications. This can inform decisions regarding drug prescriptions and patient counseling.
2. **Pharmaceutical Development:**
   * Pharmaceutical companies can leverage the findings from sentiment analysis to enhance drug formulations, address adverse effects reported by patients, and improve marketing strategies based on patient feedback.
3. **Future Enhancements:**
   * The analysis can be extended by exploring more advanced models (e.g., support vector machines, random forests, or neural networks) or by incorporating natural language processing techniques for a deeper understanding of sentiment expressed in the review text.
   * Additional features could be engineered from the review text itself (e.g., sentiment scores, keyword extraction) to improve model accuracy and insightfulness.

**Code :**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the drug review dataset

drug\_reviews = pd.read\_csv('/kaggle/input/kuc-hackathon-winter-2018/drugsComTrain\_raw.csv')

# Display the first few rows and the columns of the dataset

print(drug\_reviews.head())

print(drug\_reviews.columns)

# Create a binary target variable based on rating

# Assuming a rating of 3 or higher is positive

drug\_reviews['Sentiment'] = (drug\_reviews['rating'] >= 3).astype(int)

# Initialize LabelEncoder

label\_encoder = LabelEncoder()

# Encode categorical features

categorical\_features = ['condition', 'drugName', 'review'] # Use actual column names

for feature in categorical\_features:

drug\_reviews[feature] = label\_encoder.fit\_transform(drug\_reviews[feature])

# Split the data into features and target variable

X = drug\_reviews.drop(['Sentiment', 'uniqueID', 'rating', 'date', 'usefulCount'], axis=1)

y = drug\_reviews['Sentiment']

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# Create and fit the Logistic Regression model

model = LogisticRegression(max\_iter=200)

model.fit(X\_train, y\_train)

# Make predictions

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

# Calculate accuracy

acc\_logreg = accuracy\_score(y\_test, y\_pred) \* 100

# Print the accuracy

print("Accuracy of Logistic Regression:", acc\_logreg)

**Output :**

uniqueID drugName condition \

0 206461 Valsartan Left Ventricular Dysfunction

1 95260 Guanfacine ADHD

2 92703 Lybrel Birth Control

3 138000 Ortho Evra Birth Control

4 35696 Buprenorphine / naloxone Opiate Dependence

review rating date \

0 "It has no side effect, I take it in combinati... 9 20-May-12

1 "My son is halfway through his fourth week of ... 8 27-Apr-10

2 "I used to take another oral contraceptive, wh... 5 14-Dec-09

3 "This is my first time using any form of birth... 8 3-Nov-15

4 "Suboxone has completely turned my life around... 9 27-Nov-16

usefulCount

0 27

1 192

2 17

3 10

4 37

Index(['uniqueID', 'drugName', 'condition', 'review', 'rating', 'date',

'usefulCount'],

dtype='object')

Accuracy of Logistic Regression: 82.4922504649721

Conclusion:

**Experiment No. 8**

**Aim :** Visual Analytics for Healthcare Data.

**Theory :**

**Objective:** The objective of this analysis is to diagnose disease risk based on patient data, specifically focusing on survival outcomes in patients who have undergone surgery for breast cancer. The goal is to identify patterns and risk factors that correlate with patient outcomes.

**Dataset Description:** The dataset used is the Haberman dataset, which contains records of patients who underwent surgery for breast cancer. The columns in the dataset are as follows:

* **Age:** Age of the patient at the time of operation.
* **Year:** Year of the operation.
* **Nodes:** Number of positive axillary nodes detected (lymph nodes that contain cancer).
* **Status:** Survival status (1 = patient survived 5 years or longer, 2 = patient died within 5 years).

**Data Exploration and Visualization:**

1. **Data Overview:** Understanding the distribution of features and identifying any patterns or correlations within the data is crucial. The **describe()** function provides summary statistics, and visualizations can help highlight important trends.
2. **Histograms and KDE:** A histogram combined with a Kernel Density Estimate (KDE) allows us to visualize the distribution of patient ages and survival status, helping us understand the relationship between age and survival outcomes.

**Key Steps in the Code**

1. **Import Libraries:** Libraries such as NumPy, Pandas, Matplotlib, and Seaborn are imported for data manipulation and visualization.
2. **Load Data:** The dataset is loaded using **pd.read\_csv(),** and columns are renamed for easier reference.
3. **Data Overview:**
   * **data.head()** shows the first few entries of the dataset to get an idea of the structure and contents.
   * **data.describe()** provides summary statistics (mean, min, max, etc.) for numerical columns.
4. **Visualization:**
   * A histogram of the patient ages is created using Seaborn's **histplot(),** with KDE overlaid to visualize the density of age distribution.

**Code :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv("/kaggle/input/haberman.csv/haberman.csv")

data.columns = ["age", "year", "nodes", "status"]

print(data.head())

print(data.describe())

sns.histplot(data["age"], bins = 30, kde = True)

plt.title("Operation Year")

plt.xlabel("Patient Age")

plt.ylabel("Survival Status")

plt.show()

**Output :**

age year nodes status

0 30 64 1 1

1 30 62 3 1

2 30 65 0 1

3 31 59 2 1

4 31 65 4 1

age year nodes status

count 305.000000 305.000000 305.000000 305.000000

mean 52.357377 62.868852 4.032787 1.262295

std 10.678010 3.242783 7.200528 0.440605

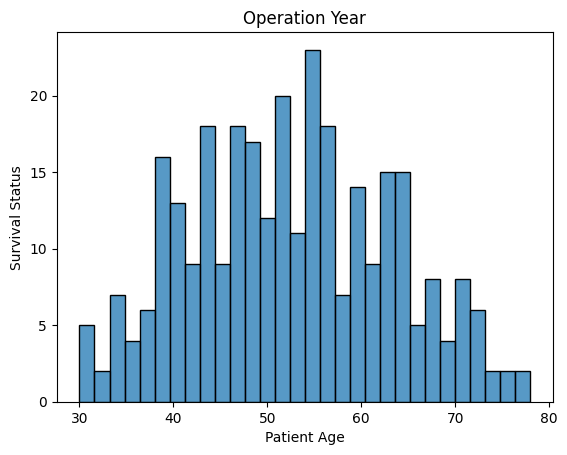
min 30.000000 58.000000 0.000000 1.000000

25% 44.000000 60.000000 0.000000 1.000000

50% 52.000000 63.000000 1.000000 1.000000

75% 60.000000 66.000000 4.000000 2.000000

max 78.000000 69.000000 52.000000 2.000000



**Conclusion:**

**Experiment No 9**

**Aim :** Implement an innovative Data Science application based on Healthcare Data.

**Theory :**

**Code :**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image\_path = "/kaggle/input/brain-mri-images-for-brain-tumor-detection/yes/Y109.JPG"

mri\_image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

plt.imshow(mri\_image, cmap='gray')

plt.title('Original MRI Image')

plt.axis('off')

plt.show()

\_, binary\_image = cv2.threshold(mri\_image, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

plt.imshow(binary\_image, cmap='gray')

plt.title('Binary Image')

plt.axis('off')

plt.show()

kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (15, 15))

opened\_image = cv2.morphologyEx(binary\_image, cv2.MORPH\_OPEN, kernel)

plt.imshow(opened\_image, cmap='gray')

plt.title('Opened Image')

plt.axis('off')

plt.show()

num\_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(opened\_image, connectivity=8)

brain\_label = 9

opened\_image[labels != brain\_label] = 0

plt.imshow(opened\_image, cmap='gray')

plt.title('Brain Part')

plt.axis('off')

plt.show()

closed\_image = cv2.morphologyEx(opened\_image, cv2.MORPH\_CLOSE, kernel)

plt.imshow(closed\_image, cmap='gray')

plt.title('Closed Brain Part')

plt.axis('off')

plt.show()

extracted\_brain = cv2.multiply(closed\_image, mri\_image)

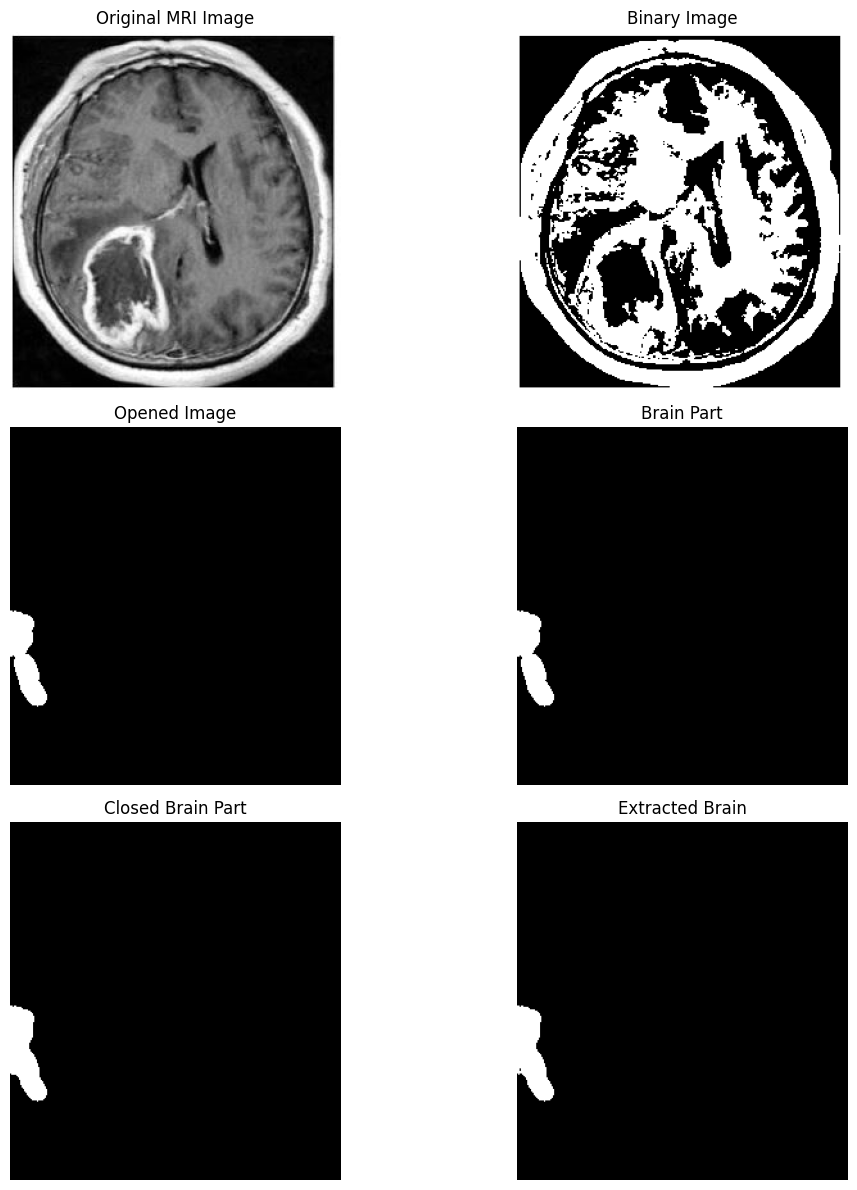
plt.imshow(extracted\_brain, cmap='gray')

plt.title('Extracted Brain')

plt.axis('off')

plt.show()

**Output :**

****

**Conclusion :**

**Experiment No 10**

**Aim :** Documentation and Presentation of Mini Project.

**Theory :**

**Objective: The objective of this mini project is to analyze comments from the trailer of a film (in this case, "Black Adam") to determine public sentiment using natural language processing (NLP) techniques and machine learning algorithms. The project aims to document the entire process, including data preprocessing, sentiment analysis, model training, and evaluation, and to present the findings effectively.**

**Context and Importance:**

1. **Sentiment Analysis:**
   * **Sentiment analysis involves using NLP techniques to determine the emotional tone behind a body of text. This is particularly useful in understanding public opinion about media content, products, or services.**
   * **In the context of film trailers, analyzing comments can provide insights into audience reception and expectations.**
2. **Natural Language Processing (NLP):**
   * **NLP is a field at the intersection of computer science, artificial intelligence, and linguistics. It enables computers to understand and process human languages.**
   * **Techniques such as tokenization, stemming, and stop-word removal are employed to clean and prepare textual data for analysis.**
3. **Machine Learning Models:**
   * **Machine learning algorithms, including Random Forest and XGBoost classifiers, are utilized to classify the sentiment of comments. These models are trained on features derived from the processed text data.**
4. **Ensemble Learning:**
   * **The project incorporates ensemble methods, specifically a Voting Classifier, which combines the predictions of multiple models to improve overall accuracy.**

**Methodology:**

1. **Data Collection:**
   * **The dataset consists of comments on the film trailer, which is read from a CSV file.**
2. **Data Preprocessing:**
   * **Unnecessary columns are dropped, and sentiment scores are computed using the VADER sentiment analysis tool.**
   * **Comments are processed to remove URLs, punctuation, and stop words, and words are stemmed to their root forms.**
3. **Feature Extraction:**
   * **The TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique is applied to convert the cleaned comments into numerical feature vectors suitable for machine learning.**
4. **Model Training:**
   * **The dataset is split into training and testing sets.**
   * **Two classifiers, Random Forest and XGBoost, are trained using Grid Search for hyperparameter tuning.**
5. **Model Evaluation:**
   * **A Voting Classifier is used to combine the best-performing models.**
   * **The confusion matrix and accuracy score are computed to evaluate model performance.**

**Code :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

from nltk.corpus import stopwords

import re

import nltk

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

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

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

from xgboost import XGBClassifier

nltk.download('vader\_lexicon', quiet=True)

nltk.download('punkt', quiet=True)

nltk.download('stopwords', quiet=True)

data = pd.read\_csv('/kaggle/input/d/pratyushranjan01/blackadam-trailer-comments/comments.csv')

data1 = data.drop(columns=['Unnamed: 0', 'Likes', 'Time', 'user', 'UserLink'])

sentiments = SentimentIntensityAnalyzer()

stemmer = PorterStemmer()

data1['sentiment\_scores'] = data1['Comment'].apply(sentiments.polarity\_scores)

data1['Sentiment'] = data1['sentiment\_scores'].apply(

lambda x: 'Positive' if x['compound'] >= 0.05

else 'Negative' if x['compound'] <= -0.05

else 'Neutral'

)

data2 = data1.drop(columns=['sentiment\_scores'])

def text\_processing(text):

text = re.sub(r'\n', ' ', text.lower())

text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

text = re.sub(r'[^\w\s]', '', text)

stop\_words = set(stopwords.words('english'))

text = ' '.join([stemmer.stem(word) for word in word\_tokenize(text) if word not in stop\_words])

return text

data\_copy = data2.copy()

data\_copy['Comment'] = data\_copy['Comment'].apply(text\_processing)

le = LabelEncoder()

data\_copy['Sentiment'] = le.fit\_transform(data\_copy['Sentiment'])

corpus = data\_copy['Comment'].values

tfidf = TfidfVectorizer(max\_features=3000, ngram\_range=(1, 2))

X = tfidf.fit\_transform(corpus).toarray()

y = data\_copy['Sentiment'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0, stratify=y)

rf\_classifier = RandomForestClassifier(random\_state=0)

rf\_param\_grid = {

'n\_estimators': [100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5, 10]

}

rf\_grid\_search = GridSearchCV(rf\_classifier, rf\_param\_grid, cv=3, scoring='accuracy', n\_jobs=-1)

rf\_grid\_search.fit(X\_train, y\_train)

xgb\_classifier = XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss')

xgb\_param\_grid = {

'n\_estimators': [100, 200],

'max\_depth': [3, 5, 7],

'learning\_rate': [0.01, 0.1, 0.2]

}

xgb\_grid\_search = GridSearchCV(xgb\_classifier, xgb\_param\_grid, cv=3, scoring='accuracy', n\_jobs=-1)

xgb\_grid\_search.fit(X\_train, y\_train)

voting\_classifier = VotingClassifier(

estimators=[

('rf', rf\_grid\_search.best\_estimator\_),

('xgb', xgb\_grid\_search.best\_estimator\_)

],

voting='hard'

)

voting\_classifier.fit(X\_train, y\_train)

y\_pred = voting\_classifier.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

voting\_score = accuracy\_score(y\_test, y\_pred)

print('Confusion Matrix:\n', cm)

print('Accuracy:', voting\_score)

**Output :**

Confusion Matrix:

[[ 1 1 9]

[ 0 6 6]

[ 1 4 56]]

Accuracy: 0.75

**Conclusion :**