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

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

```
with open("/content/drive/MyDrive/Colab Notebooks/Suicide_Detection.csv", 'rb') as f:
  encoding = chardet.detect(f.read())['encoding']
```

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Suicide_Detection.csv", encoding=encoding)
```

```
# Define a mapping from labels to numbers
label_mapping = {'suicide': 1, 'non-suicide': 0}

# Replace the labels in the 'label' column
data['class'] = data['class'].map(label_mapping)
```

```
data=data.dropna()
data.describe()
# data.head()
```

	Unnamed: 0	class
count	41337.000000	41337.000000
mean	31045.271573	0.497859
std	17901.162606	0.500001
min	2.000000	0.000000
25%	15534.000000	0.000000
50%	31114.000000	0.000000
75%	46572.000000	1.000000
max	61976.000000	1.000000

```
vectorizer=CountVectorizer(analyzer='word')
X=vectorizer.fit_transform(data['text'])
X
```

```
<41337x64859 sparse matrix of type '<class 'numpy.int64'>'
        with 2943354 stored elements in Compressed Sparse Row format>
```

```
X_train, X_test, y_train, y_test = train_test_split(X, data['class'], test_size=0.2, random_state=1)
```

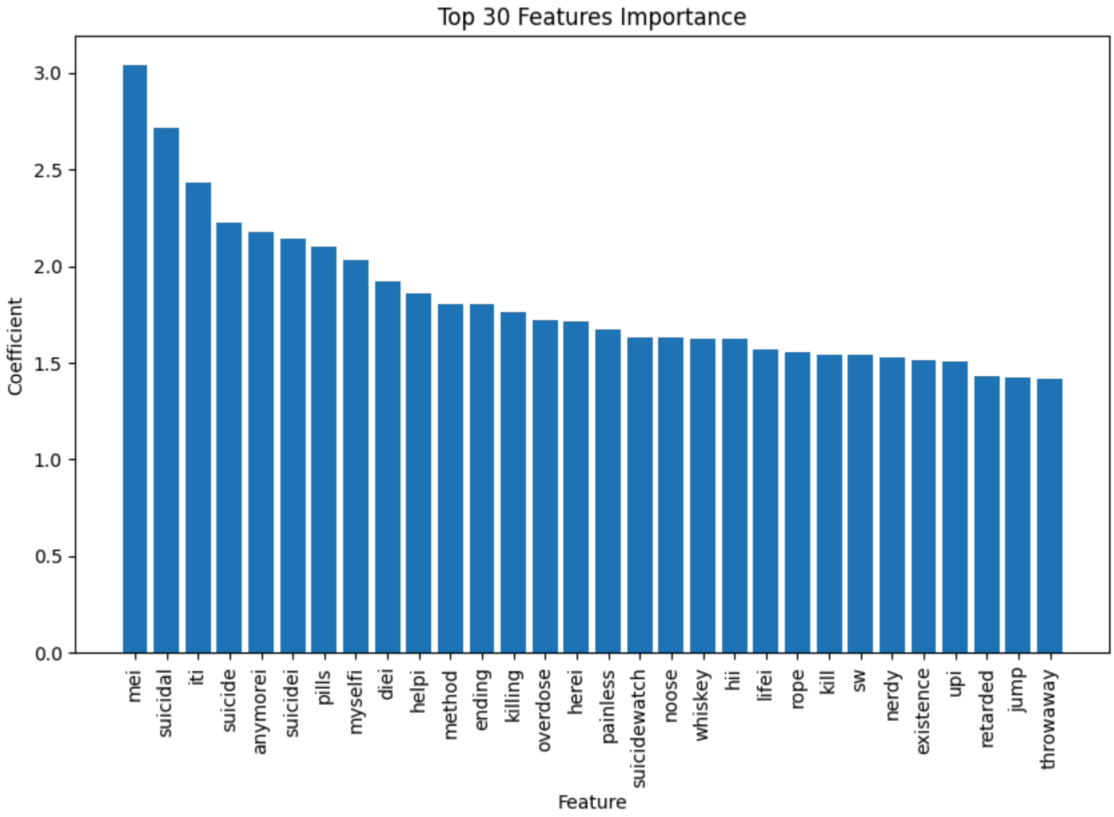
```
model=LogisticRegression(max_iter=10000)
```

```
model.fit(X_train,y_train)
```

```
▾        LogisticRegression
LogisticRegression(max_iter=10000)
```

```
import matplotlib.pyplot as plt

feature_names = vectorizer.get_feature_names_out()

coefficients = model.coef_.tolist()[0]

df_coef = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

df_coef = df_coef.sort_values(by='Coefficient', ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(df_coef['Feature'][:30], df_coef['Coefficient'][:30])
plt.xticks(rotation=90)
plt.xlabel('Feature')
plt.ylabel('Coefficient')
plt.title('Top 30 Features Importance')
plt.show()
```

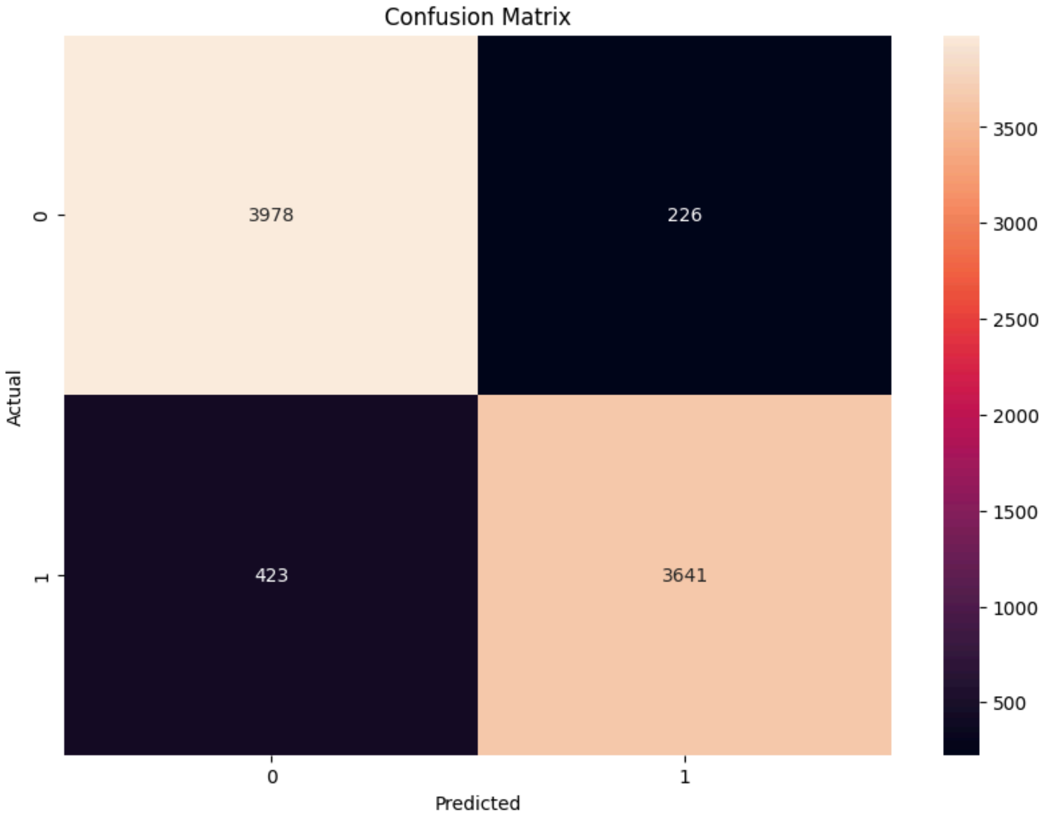
```
--INSERT--
```



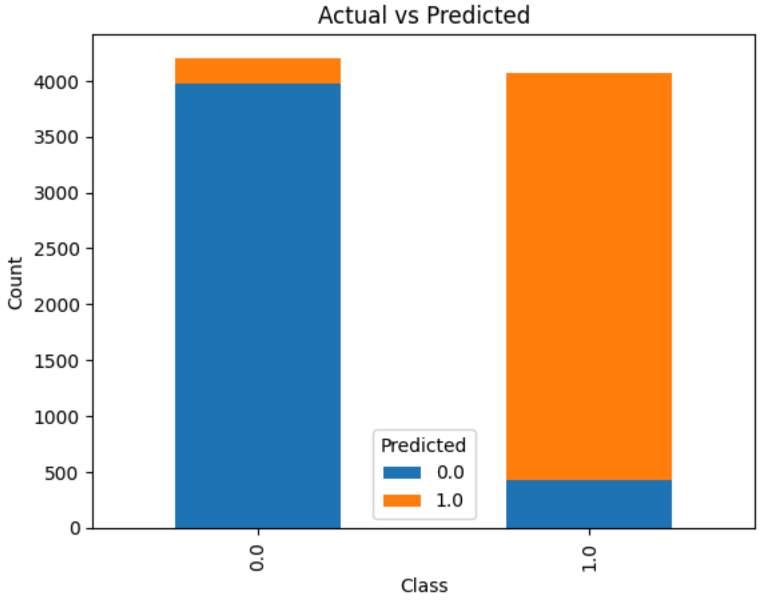
```
# Evaluate the model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.95      0.92      4204
         1.0       0.94      0.90      0.92      4064

    accuracy                           0.92      8268
   macro avg       0.92      0.92      0.92      8268
weighted avg       0.92      0.92      0.92      8268
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming y_test are your true labels and y_pred are the predicted labels
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

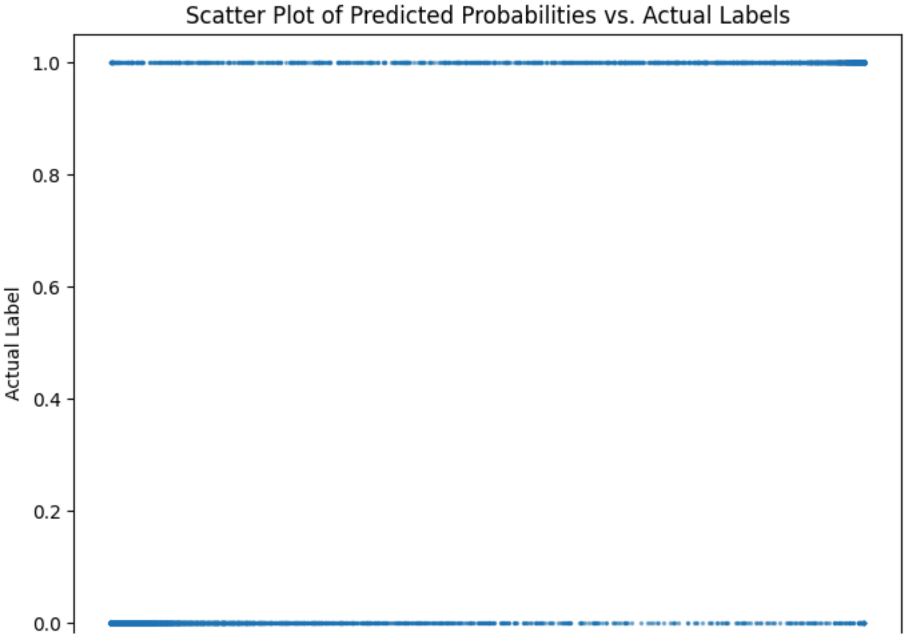


```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Count the number of instances for each class
class_counts = df.groupby(['Actual', 'Predicted']).size().unstack(fill_value=0)

# Plot the counts
class_counts.plot(kind='bar', stacked=True)
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Actual vs Predicted')
plt.show()
```



Generate print hello world using rot13 Close

```
plt.figure(figsize=(8, 6))
plt.scatter(y_pred_prob, y_test,s=2, alpha=0.5)

# Add labels and title
plt.xlabel('Predicted Probability of Suicide')
plt.ylabel('Actual Label')
plt.title('Scatter Plot of Predicted Probabilities vs. Actual Labels')

# Show the plot
plt.show()
```


Scatter Plot of Predicted Probabilities vs. Actual Labels
Actual Label