```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
import numpy as np
import pandas as pd
import matplotlib as mplot
import matplotlib.cm as cmplot
import matplotlib.pyplot as plt
import plotly.graph_objects as gp
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
import re
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import metrics
from time import time
from nltk.tokenize import word_tokenize
```

```
data = pd.read_csv("/content/drive/MyDrive/spam.csv")
```

```
print(data.head())
```

```
  Category                                            Message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                      Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
```

```
print(data.shape)
```

```
(5572, 2)
```

```
data.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Next steps: Generate code with data View recommended plots

```
print(data.columns)
```

```
Index(['Category', 'Message'], dtype='object')
```

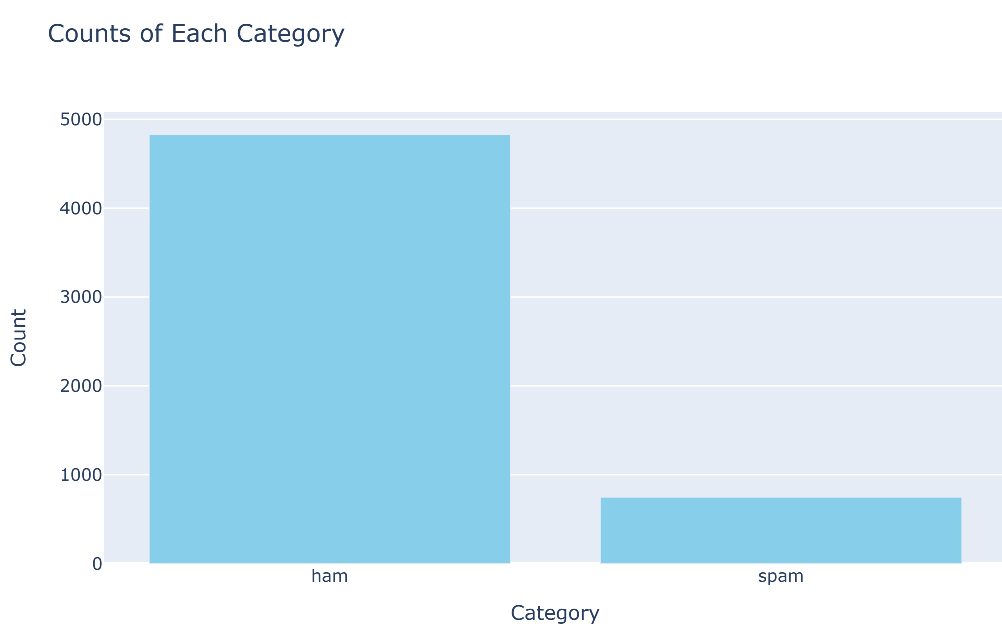
```
label = data['Category']
tdata = data['Message']
data = pd.DataFrame({'text': tdata, 'label': label})
```

```
data.head()
```

	text	label
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham

Next steps: Generate code with data View recommended plots

```
# Create a Figure object with a bar plot
bar_fig = gp.Figure([gp.Bar(x=data['label'].value_counts().index,
                            y=data['label'].value_counts().tolist(),
                            marker_color='skyblue')])

# Update layout of the figure
bar_fig.update_layout(
    title="Counts of Each Category",
    xaxis_title="Category",
    yaxis_title="Count",
    width=800,
    height=500
)

# Show the figure
bar_fig.show()
```



```
def clean_text(text_data):
    # Convert text to lowercase
    text_data = text_data.lower()

    # Remove non-alphanumeric characters and spaces
    text_data = re.sub(r'[^a-zA-Z0-9\s]', '', text_data)

    # Tokenize the text
    tokens = word_tokenize(text_data)

    # Remove stopwords
    stop_words_set = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words_set]

    # Lemmatize the tokens
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Join tokens into a single string
    cleaned_text = ' '.join(tokens)

    return cleaned_text

# Clean the dataset: tokenization, stop words removal, and lemmatization
data['cleaned_text'] = data['text'].apply(clean_text)



# Extracting features (X) and labels (y)
X_data = data['cleaned_text']
y_data = data['label']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=42)



# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the training data to TF-IDF features
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Transform the testing data to TF-IDF features
X_test_tfidf = tfidf_vectorizer.transform(X_test)



# Initialize Multinomial Naive Bayes classifier
classifier = MultinomialNB()

# Train the classifier and measure time
start_time = time()
classifier.fit(X_train_tfidf, y_train)
training_time = time() - start_time

# Predict labels for training and testing data
y_pred_train = classifier.predict(X_train_tfidf)
y_pred_test = classifier.predict(X_test_tfidf)

# Calculate accuracy scores
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print accuracy scores
print("\nTraining Accuracy score:", train_accuracy)
print("Testing Accuracy score:", test_accuracy)
print("Training Time:", training_time)
```

```
Training Accuracy score: 0.9755440879515369
Testing Accuracy score: 0.968609865470852
Training Time: 0.016905546188354492
```

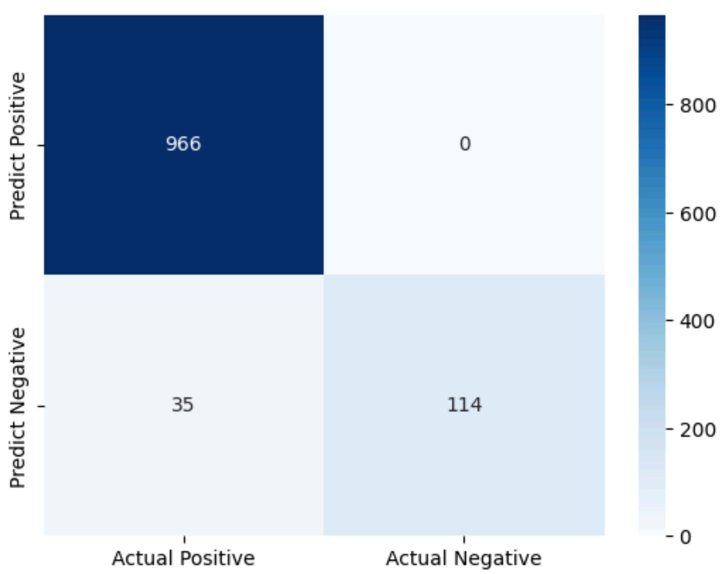
```
report = classification_report(y_test, y_pred_test, target_names=['no', 'yes'])
print(report)
```

```
              precision    recall  f1-score   support

          no       0.97      1.00      0.98       966
         yes       1.00      0.77      0.87       149

    accuracy                           0.97      1115
   macro avg       0.98      0.88      0.92      1115
weighted avg       0.97      0.97      0.97      1115
```

```
cm_custom = confusion_matrix(y_test, y_pred_test)
print(cm_custom)
```

```
[[966   0]
 [ 35 114]]
```

```
cm_custom = pd.DataFrame(data=cm_custom, columns=['Actual Positive', 'Actual Negative'],
                         index=['Predict Positive', 'Predict Negative'])

sns.heatmap(cm_custom, annot=True, fmt='d', cmap='Blues')
plt.show()
```




```
from sklearn.naive_bayes import BernoulliNB
import time

# Create an instance of BernoulliNB
nb = BernoulliNB()

# Measure the time taken to fit the model
start_time = time.time()
nb.fit(X_train_tfidf, y_train)
end_time = time.time()

# Calculate the elapsed time
elapsed_time = end_time - start_time
print("Time taken to fit the model:", elapsed_time, "seconds")
```

```
Time taken to fit the model: 0.018539905548095703 seconds
```

```
from sklearn.metrics import accuracy_score

# Predictions on the training set
y_pred_train = nb.predict(X_train_tfidf)

# Predictions on the test set
y_pred_test = nb.predict(X_test_tfidf)

# Calculate and print the training accuracy score
train_accuracy = accuracy_score(y_train, y_pred_train)
print("\nTraining Accuracy score:", train_accuracy)

# Calculate and print the testing accuracy score
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Testing Accuracy score:", test_accuracy)
```

```
Training Accuracy score: 0.9836212699124972
Testing Accuracy score: 0.9739910313901345
```

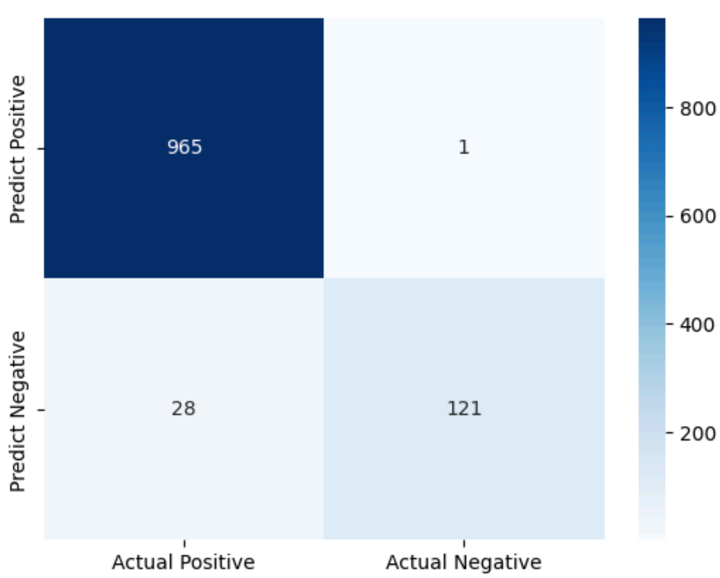
```
# Print the classification report
report = classification_report(y_test, y_pred_test, target_names=['no', 'yes'])
print(report)
```

```
              precision    recall  f1-score   support

          no       0.97      1.00      0.99       966
         yes       0.99      0.81      0.89       149

    accuracy                           0.97      1115
   macro avg       0.98      0.91      0.94      1115
weighted avg       0.97      0.97      0.97      1115
```

```
cm_custom = confusion_matrix(y_test, y_pred_test)
print(cm_custom)
```

```
[[965   1]
 [ 28 121]]
```

```
cm_custom = pd.DataFrame(data=cm_custom, columns=['Actual Positive', 'Actual Negative'],
                         index=['Predict Positive', 'Predict Negative'])

sns.heatmap(cm_custom, annot=True, fmt='d', cmap='Blues')
plt.show()
```



```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score as acc_score,
    precision_score as prec_score,
    recall_score as rec_score,
    f1_score as f1,
    log_loss as ll
)
```

```
from sklearn.linear_model import LogisticRegression

# Creating a logistic regression classifier with specified parameters
custom_classifier = LogisticRegression(solver='lbfgs', class_weight='balanced')

# Fitting the classifier to the training data
custom_classifier.fit(X_train_tfidf, y_train)
```

```
▾            LogisticRegression
LogisticRegression(class_weight='balanced')
```

```
# Making predictions using the classifier
custom_pred = custom_classifier.predict(X_test_tfidf)

# Printing accuracy score
print('Accuracy score:', acc_score(y_test, custom_pred))

# Defining the positive label
custom_positive_label = 'yes'

# Printing precision score
print('Precision score:', prec_score(y_test, custom_pred, pos_label='spam'))

# Printing recall score
print('Recall score:', rec_score(y_test, custom_pred, pos_label='spam'))

# Printing F1 score
print('F1 score:', f1(y_test, custom_pred, pos_label='spam'))

# Calculating predicted probabilities
custom_probs = custom_classifier.predict_proba(X_test_tfidf)

# Printing log loss
print('Log loss:', ll(y_test, custom_probs))
```

```
Accuracy score: 0.9766816143497757
Precision score: 0.9019607843137255
Recall score: 0.9261744966442953
F1 score: 0.9139072847682119
Log loss: 0.16960314412077115
```

```
# Predictions on the training set
custom_y_pred_train = custom_classifier.predict(X_train_tfidf)

# Predictions on the test set
custom_y_pred_test = custom_classifier.predict(X_test_tfidf)

# Calculate and print the training accuracy score
train_accuracy = acc_score(y_train, custom_y_pred_train)
print("\nTraining Accuracy score:", train_accuracy)

# Calculate and print the testing accuracy score
test_accuracy = acc_score(y_test, custom_y_pred_test)
print("Testing Accuracy score:", test_accuracy)
```

```
Training Accuracy score: 0.9908009872111285
Testing Accuracy score: 0.9766816143497757
```

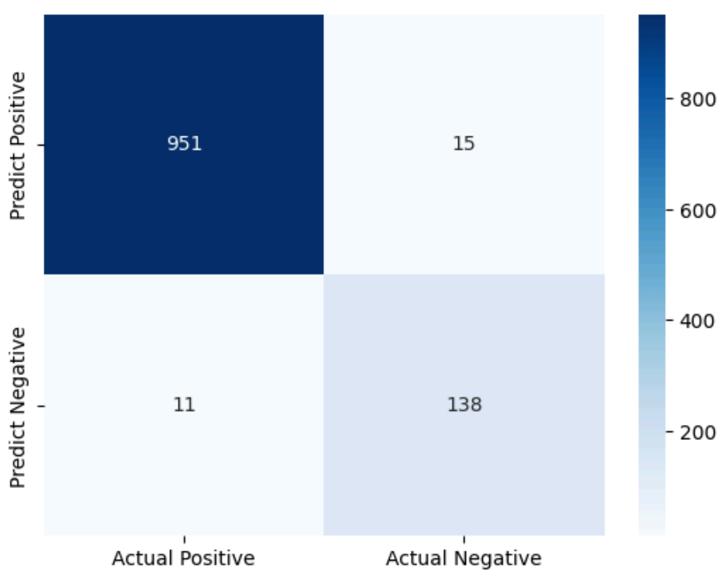
```
# Print the classification report
report = classification_report(y_test, custom_y_pred_test, target_names=['no', 'yes'])
print(report)
```

```
              precision    recall  f1-score   support

          no       0.99      0.98      0.99       966
         yes       0.90      0.93      0.91       149

    accuracy                           0.98      1115
   macro avg       0.95      0.96      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

```
cm_custom = confusion_matrix(y_test, custom_y_pred_test)
print(cm_custom)
```

```
[[951  15]
 [ 11 138]]
```

```
cm_custom = pd.DataFrame(data=cm_custom, columns=['Actual Positive', 'Actual Negative'],
                         index=['Predict Positive', 'Predict Negative'])

sns.heatmap(cm_custom, annot=True, fmt='d', cmap='Blues')
plt.show()
```




```
from sklearn.neural_network import MLPClassifier

# Define the MLPClassifier with specified parameters
classifier = MLPClassifier(solver='lbfgs', alpha=1e-5,
                           hidden_layer_sizes=(8, 2), random_state=1)

# Fit the classifier to the training data
classifier.fit(X_train_tfidf, y_train)
```

```
▾                        MLPClassifier
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(8, 2), random_state=1,
              solver='lbfgs')
```

```
# Making predictions using the classifier
custom_pred = classifier.predict(X_test_tfidf)

# Printing accuracy score
print('Accuracy score:', acc_score(y_test, custom_pred))

# Defining the positive label
custom_positive_label = 'yes'

# Printing precision score
print('Precision score:', prec_score(y_test, custom_pred, pos_label='spam'))

# Printing recall score
print('Recall score:', rec_score(y_test, custom_pred, pos_label='spam'))

# Printing F1 score
print('F1 score:', f1(y_test, custom_pred, pos_label='spam'))

# Calculating predicted probabilities
custom_probs = custom_classifier.predict_proba(X_test_tfidf)

# Printing log loss
print('Log loss:', ll(y_test, custom_probs))
```

```
Accuracy score: 0.9847533632286996
Precision score: 0.9647887323943662
Recall score: 0.9194630872483222
F1 score: 0.9415807560137458
Log loss: 0.16960314412077115
```

```
# Predictions on the training set
custom_y_pred_train = classifier.predict(X_train_tfidf)

# Predictions on the test set
custom_y_pred_test = classifier.predict(X_test_tfidf)

# Calculate and print the training accuracy score
train_accuracy = acc_score(y_train, custom_y_pred_train)
print("\nTraining Accuracy score:", train_accuracy)

# Calculate and print the testing accuracy score
test_accuracy = acc_score(y_test, custom_y_pred_test)
print("Testing Accuracy score:", test_accuracy)
```

```
Training Accuracy score: 1.0
Testing Accuracy score: 0.9847533632286996
```

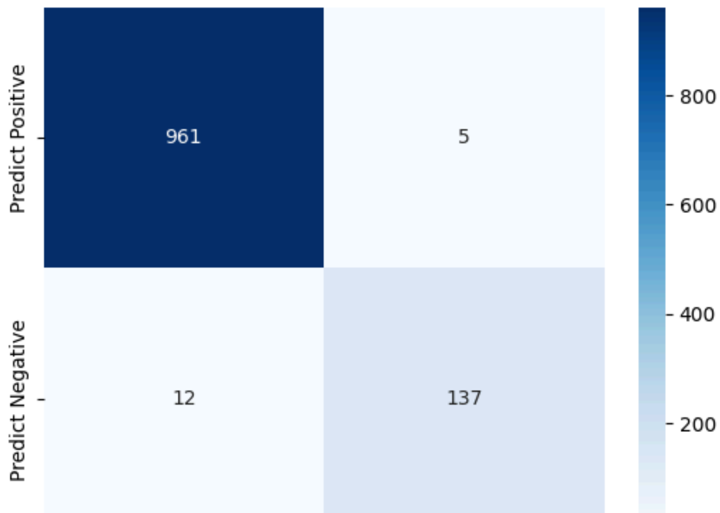
```
# Print the classification report
report = classification_report(y_test, custom_y_pred_test, target_names=['no', 'yes'])
print(report)
```

```
              precision    recall  f1-score   support

          no       0.99      0.99      0.99       966
         yes       0.96      0.92      0.94       149

    accuracy                           0.98      1115
   macro avg       0.98      0.96      0.97      1115
weighted avg       0.98      0.98      0.98      1115
```

```
cm_custom = confusion_matrix(y_test, custom_y_pred_test)
print(cm_custom)
```

```
[[961   5]
 [ 12 137]]
```

```
cm_custom = pd.DataFrame(data=cm_custom, columns=['Actual Positive', 'Actual Negative'],
                          index=['Predict Positive', 'Predict Negative'])

sns.heatmap(cm_custom, annot=True, fmt='d', cmap='Blues')
plt.show()
```



The Multinomial Naive Bayes classifier achieved a training accuracy score of approximately 97.55% and a testing accuracy score of approximately 96.86%. It exhibited high precision (97%) for the 'no' class and slightly lower precision (100%) for the 'yes' class. However, it showed lower recall (77%) for the 'yes' class, resulting in a slightly lower F1-score (87%) for that class compared to the 'no' class. Overall, the classifier demonstrated robust performance with a weighted average F1-score of 97%.

The Bernoulli Naive Bayes classifier achieved a training accuracy score of approximately 98.36% and a testing accuracy score of approximately 97.40%. It exhibited high precision (97%) for the 'no' class and slightly lower precision (99%) for the 'yes' class. However, it showed lower recall (81%) for the 'yes' class, resulting in a slightly lower F1-score (89%) for that class compared to the 'no' class. Overall, the classifier demonstrated robust performance with a weighted average F1-score of 97%.

The Logistic Regression model achieved an accuracy score of approximately 97.67%. It demonstrated a precision of approximately 90.20%, indicating the proportion of correctly identified positive cases out of all positive predictions. The model also exhibited a recall of approximately 92.62%, indicating the proportion of correctly identified positive cases out of all actual positive cases. The F1-score, a harmonic mean of precision and recall, was approximately 91.39%.

The neural network model achieved perfect training accuracy (100%) and a high testing accuracy of approximately 98.48%. It demonstrated excellent precision of 99% for the 'no' class and strong precision of 96% for the 'yes' class. Additionally, the model exhibited high recall scores, with 99% for the 'no' class and 92% for the 'yes' class, resulting in an overall balanced F1-score of 94%. Overall, the neural network model showed robust performance on both training and testing data.

Start coding or generate with AI.