In [1]:
```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.model_selection import cross_val_predict, LeaveOneOut
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
from sklearn import metrics

# SVM classifier
from sklearn.svm import SVC
from sklearn import svm

# Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
```

In [2]:
```
data = pd.read_csv(r"C:\Users\prane\OneDrive\Desktop\Jupyter Notebook\Machine
data.drop('id', axis=1, inplace=True)
data
```

Out[2]:

	timestamp	hour	day	month	datetime	timezone	source	destination
664787	1.543753e+09	12	2	12	02-12-2018 12:13	America/New_York	Theatre District	South Station
528020	1.543857e+09	17	3	12	03-12-2018 17:13	America/New_York	Northeastern University	Theatre District
488499	1.543409e+09	12	28	11	28-11-2018 12:44	America/New_York	Beacon Hill	Haymarket Square
186991	1.544677e+09	4	13	12	13-12-2018 04:50	America/New_York	Fenway	North Station
152200	1.545137e+09	12	18	12	18-12-2018 12:50	America/New_York	Boston University	North Station
...	...	...	...	...	...	...	...	...
266230	1.543328e+09	14	27	11	27-11-2018 14:12	America/New_York	Boston University	Theatre District
542782	1.544872e+09	11	15	12	15-12-2018 11:10	America/New_York	Boston University	Theatre District
315432	1.543421e+09	16	28	11	28-11-2018 16:01	America/New_York	Beacon Hill	Fenway
279308	1.543506e+09	15	29	11	29-11-2018 15:48	America/New_York	Haymarket Square	Beacon Hill
434905	1.543480e+09	8	29	11	29-11-2018 08:33	America/New_York	Beacon Hill	Haymarket Square

30000 rows × 56 columns

Data Preprocessing

Data Inspection

In [3]:
```
# To get number of rows and columns in the dataset
num_rows, num_columns = data.shape
num_rows, num_columns
```

Out[3]: (30000, 56)

In [4]:
```
# To find the number of missing values in each column
data.isnull().sum()
```

Out[4]:
```
timestamp                         0
hour                              0
day                               0
month                             0
datetime                          0
timezone                          0
source                            0
destination                       0
cab_type                          0
product_id                        0
name                              0
price                          2370
distance                          0
surge_multiplier                  0
latitude                          0
longitude                         0
temperature                       0
apparentTemperature               0
short_summary                     0
long_summary                      0
precipIntensity                   0
precipProbability                 0
humidity                          0
windSpeed                         0
windGust                          0
windGustTime                      0
visibility                        0
temperatureHigh                   0
temperatureHighTime               0
temperatureLow                    0
temperatureLowTime                0
apparentTemperatureHigh           0
apparentTemperatureHighTime       0
apparentTemperatureLow            0
apparentTemperatureLowTime        0
icon                              0
dewPoint                          0
pressure                          0
windBearing                       0
cloudCover                        0
uvIndex                           0
visibility.1                      0
ozone                             0
sunriseTime                       0
sunsetTime                        0
moonPhase                         0
precipIntensityMax                0
uvIndexTime                       0
temperatureMin                    0
temperatureMinTime                0
temperatureMax                    0
temperatureMaxTime                0
apparentTemperatureMin            0
apparentTemperatureMinTime        0
apparentTemperatureMax            0
```

```
apparentTemperatureMaxTime       0
dtype: int64
```

Handling missing values

In [5]:
```
# To fill the missing values in the "price" attribute with the mean
mean_price = data['price'].mean()
data['price'].fillna(mean_price, inplace = True)
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
timestamp                      0
hour                           0
day                            0
month                          0
datetime                       0
timezone                       0
source                         0
destination                    0
cab_type                       0
product_id                     0
name                           0
price                          0
distance                       0
surge_multiplier               0
latitude                       0
longitude                      0
temperature                    0
apparentTemperature            0
short_summary                  0
long_summary                   0
precipIntensity                0
precipProbability              0
humidity                       0
windSpeed                      0
windGust                       0
windGustTime                   0
visibility                     0
temperatureHigh                0
temperatureHighTime            0
temperatureLow                 0
temperatureLowTime             0
apparentTemperatureHigh        0
apparentTemperatureHighTime    0
apparentTemperatureLow         0
apparentTemperatureLowTime     0
icon                           0
dewPoint                       0
pressure                       0
windBearing                    0
cloudCover                     0
uvIndex                        0
visibility.1                   0
ozone                          0
sunriseTime                    0
sunsetTime                     0
moonPhase                      0
precipIntensityMax             0
uvIndexTime                    0
temperatureMin                 0
temperatureMinTime             0
temperatureMax                 0
temperatureMaxTime             0
apparentTemperatureMin         0
apparentTemperatureMinTime     0
apparentTemperatureMax         0
```

```
apparentTemperatureMaxTime     0
dtype: int64
```

Discretizing the price attribute

In [7]:
```
price_bins = [float('-inf'), 13, 26, float('inf')]
price_labels = ['low', 'medium', 'high']
```

In [8]:
```
data['price_category'] = pd.cut(data['price'], bins=price_bins, labels=price_l
```

In [9]:
```
data.shape
```

Out[9]:
```
(30000, 57)
```

In [10]:
```
data['price_category']
```

Out[10]:
```
664787       low
528020    medium
488499       low
186991    medium
152200       low
           ...
266230      high
542782    medium
315432    medium
279308      high
434905       low
Name: price_category, Length: 30000, dtype: object
```

In [11]:
```
data.shape
```

Out[11]:
```
(30000, 57)
```

Data Visualization

In [12]:
```
categorical_columns = data.select_dtypes(include=['object']).columns.tolist()
categorical_columns
```

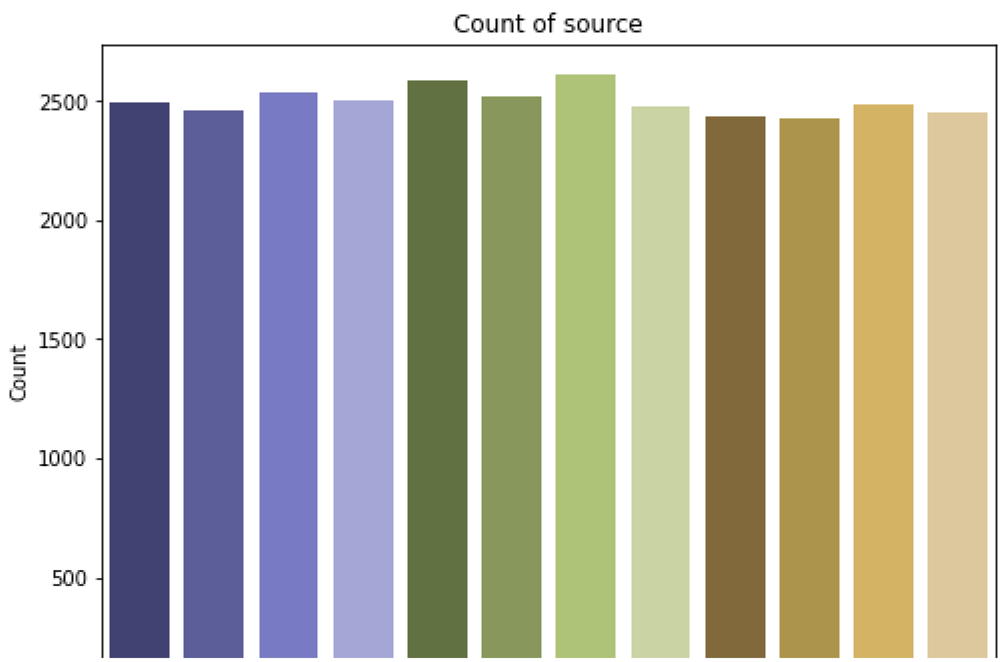
Out[12]:
```
['datetime',
 'timezone',
 'source',
 'destination',
 'cab_type',
 'product_id',
 'name',
 'short_summary',
 'long_summary',
 'icon',
 'price_category']
```

In [13]:
```
numeric_columns = data.select_dtypes(include=['int', 'float']).columns.tolist(
numeric_columns
```

Out[13]:
```
['timestamp',
 'hour',
 'day',
 'month',
 'price',
 'distance',
 'surge_multiplier',
 'latitude',
 'longitude',
 'temperature',
 'apparentTemperature',
 'precipIntensity',
 'precipProbability',
 'humidity',
 'windSpeed',
 'windGust',
 'windGustTime',
 'visibility',
 'temperatureHigh',
 'temperatureHighTime',
 'temperatureLow',
 'temperatureLowTime',
 'apparentTemperatureHigh',
 'apparentTemperatureHighTime',
 'apparentTemperatureLow',
 'apparentTemperatureLowTime',
 'dewPoint',
 'pressure',
 'windBearing',
 'cloudCover',
 'uvIndex',
 'visibility.1',
 'ozone',
 'sunriseTime',
 'sunsetTime',
 'moonPhase',
 'precipIntensityMax',
 'uvIndexTime',
 'temperatureMin',
 'temperatureMinTime',
 'temperatureMax',
 'temperatureMaxTime',
 'apparentTemperatureMin',
 'apparentTemperatureMinTime',
 'apparentTemperatureMax',
 'apparentTemperatureMaxTime']
```

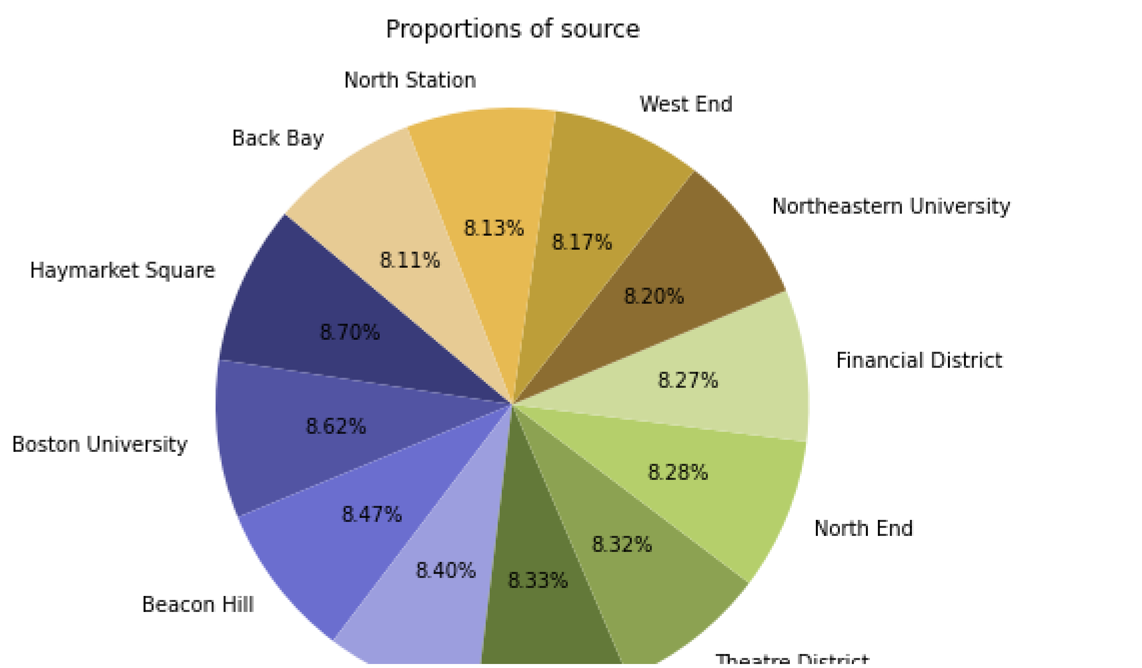
In [15]:
```
categorical_attributes = ['source', 'destination', 'cab_type', 'name', 'price_

for col in categorical_attributes:
    plt.figure(figsize=(8, 6))
    sns.countplot(x=col, data=data)
    sns.set_palette('tab20b')
    plt.title(f'Count of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=70)
    plt.show()
```



In [16]:
```
# Pie charts
for col in categorical_attributes:
    # Calculate value counts for each category
    counts = data[col].value_counts()

    plt.figure(figsize=(8, 6))
    plt.title(f'Proportions of {col}\n')
    sns.set_palette('tab20b')
    plt.pie(counts, labels=counts.index, autopct='%1.2f%%', startangle=140)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a cir
    plt.show()
```



Scaling numeric attributes

In [17]:
```
numeric_columns = data.select_dtypes(include=['int', 'float']).columns
numeric_columns = numeric_columns.drop('price_category', errors='ignore')
```

In [18]:
```
# Creating a MinMaxScaler instance
scaler = MinMaxScaler()
```

In [19]:
```
data[numeric_columns] = scaler.fit_transform(data[numeric_columns])
```

In [20]:
```
data
```

Out[20]:

	timestamp	hour	day	month	datetime	timezone	source	destina
664787	0.080798	0.521739	0.034483	1.0	02-12-2018 12:13	America/New_York	Theatre District	S Sta
528020	0.096160	0.739130	0.068966	1.0	03-12-2018 17:13	America/New_York	Northeastern University	The Dis
488499	0.030222	0.521739	0.931034	0.0	28-11-2018 12:44	America/New_York	Beacon Hill	Hayma Sq
186991	0.216728	0.173913	0.413793	1.0	13-12-2018 04:50	America/New_York	Fenway	N Sta
152200	0.284530	0.521739	0.586207	1.0	18-12-2018 12:50	America/New_York	Boston University	N Sta
...	...	...	...	...	...	...	...	
266230	0.018288	0.608696	0.896552	0.0	27-11-2018 14:12	America/New_York	Boston University	The Dis
542782	0.245509	0.478261	0.482759	1.0	15-12-2018 11:10	America/New_York	Boston University	The Dis
315432	0.031964	0.695652	0.931034	0.0	28-11-2018 16:01	America/New_York	Beacon Hill	Fer
279308	0.044558	0.652174	0.965517	0.0	29-11-2018 15:48	America/New_York	Haymarket Square	Beacor
434905	0.040718	0.347826	0.965517	0.0	29-11-2018 08:33	America/New_York	Beacon Hill	Hayma Sq

30000 rows × 57 columns

Data Encoding

In [21]:
```
# To seperate the ordinal and nominal columns
ordinal_columns = ['price_category']
nominal_columns = [col for col in categorical_columns if col not in ordinal_co
```

In [22]:
```python
# One-hot encoding nominal attributes
data_encoded = pd.get_dummies(data, columns=nominal_columns, drop_first=True)
```

In [23]:
```python
# Label-encoding ordinal attributes
label_encoder = LabelEncoder()
for column in ordinal_columns:
    data_encoded[column] = label_encoder.fit_transform(data[column])
```

In [24]:
```python
data_encoded
```

Out[24]:

	timestamp	hour	day	month	price	distance	surge_multiplier	latitude
664787	0.080798	0.521739	0.034483	1.0	0.052941	0.071237	0.000000	0.895572
528020	0.096160	0.739130	0.068966	1.0	0.276471	0.258065	0.000000	0.896894
488499	0.030222	0.521739	0.931034	0.0	0.058824	0.192204	0.000000	0.951751
186991	0.216728	0.173913	0.413793	1.0	0.200000	0.438172	0.000000	0.895572
152200	0.284530	0.521739	0.586207	1.0	0.100000	0.452957	0.000000	0.846662
...	...	...	...	...	...	...	...	...
266230	0.018288	0.608696	0.896552	0.0	0.505882	0.392473	0.166667	0.846662
542782	0.245509	0.478261	0.482759	1.0	0.164466	0.326613	0.000000	0.990747
315432	0.031964	0.695652	0.931034	0.0	0.235294	0.309140	0.000000	1.000000
279308	0.044558	0.652174	0.965517	0.0	0.294118	0.178763	0.000000	0.906147
434905	0.040718	0.347826	0.965517	0.0	0.064706	0.178763	0.000000	0.896894

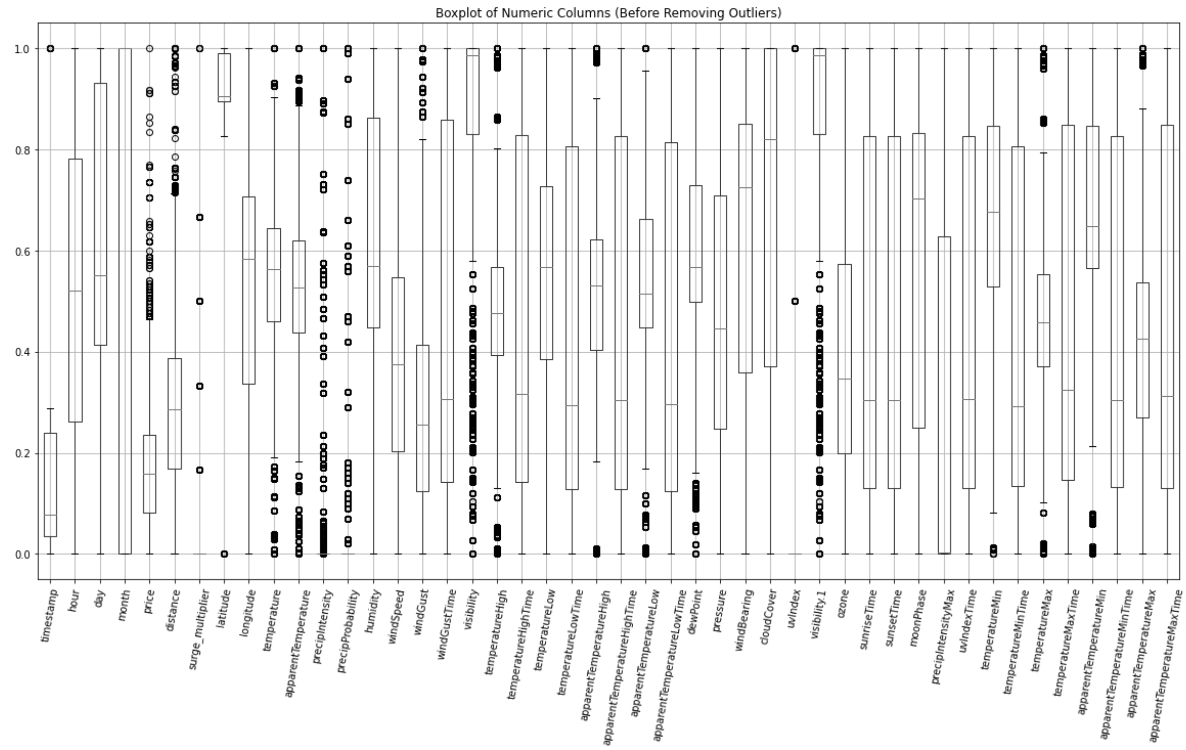
30000 rows × 5860 columns

In [25]:
```python
data_encoded['price_category']
```

Out[25]:
```
664787    1
528020    2
488499    1
186991    2
152200    1
         ..
266230    0
542782    2
315432    2
279308    0
434905    1
Name: price_category, Length: 30000, dtype: int32
```

Handling the Outliers

In [26]:
```
# Boxplots before removing outliers
plt.figure(figsize=(20,10))
data_encoded[numeric_columns].boxplot()
plt.title('Boxplot of Numeric Columns (Before Removing Outliers)')
plt.xticks(rotation=80)
plt.show()
```

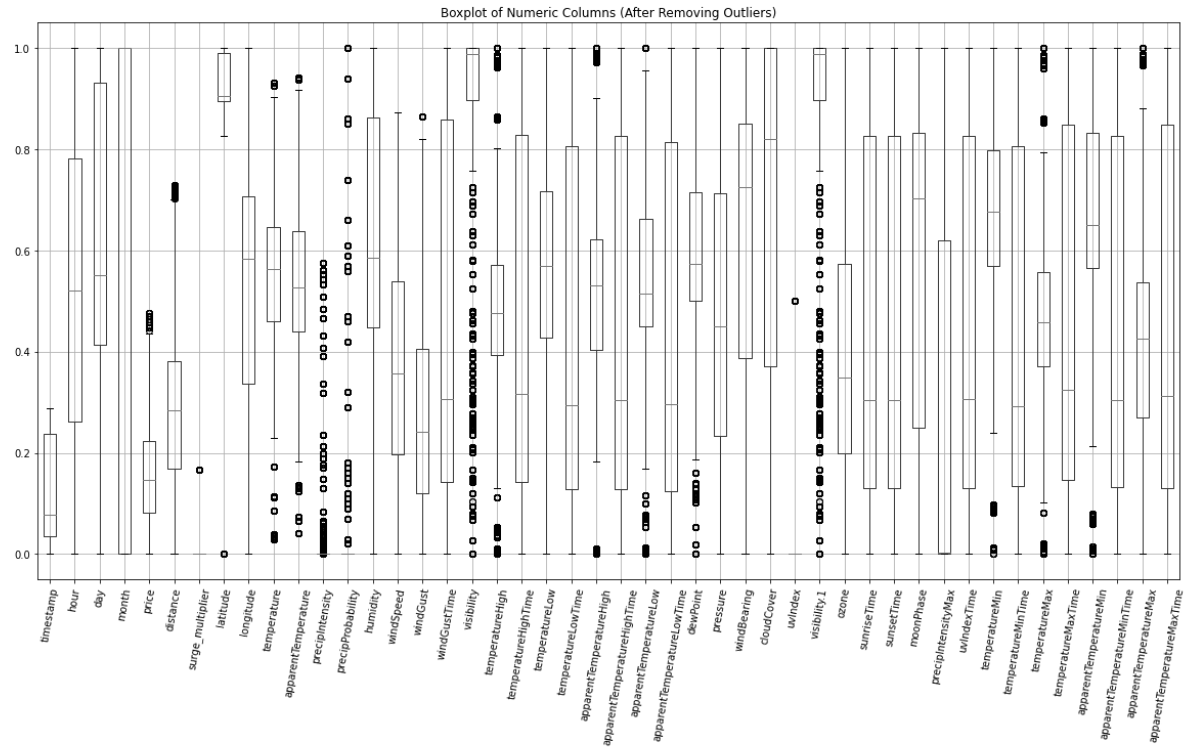


In [27]:
```
# Removing outliers using the z-score method
data_encoded_no_outliers = data_encoded[(np.abs(stats.zscore(data_encoded[nume

print("Original Dataset Shape:", data_encoded.shape)
print("Dataset Shape After Removing Outliers:", data_encoded_no_outliers.shape
```

```
Original Dataset Shape: (30000, 5860)
Dataset Shape After Removing Outliers: (26663, 5860)
```

In [28]:
```
# Boxplots after removing outliers
plt.figure(figsize=(20,10))
data_encoded_no_outliers[numeric_columns].boxplot()
plt.title('Boxplot of Numeric Columns (After Removing Outliers)')
plt.xticks(rotation=80)
plt.show()
```



Data Split(for classifiers)

In [29]:
```
X = data_encoded_no_outliers.drop('price_category', axis=1)
y = data_encoded_no_outliers['price_category']
```

In [30]:
```
# Splitting the data into training and testing (7:3 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

In [31]:
```
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (18664, 5859) (18664,)
Testing set shape: (7999, 5859) (7999,)
```

Data Split(for regressors)

In [48]:
```
X_reg = data_encoded_no_outliers.drop('price', axis=1)
y_reg = data_encoded_no_outliers['price']
```

In [49]:
```
# Splitting the data into training and testing (7:3 ratio)
X_reg_train, X_reg_test, y_reg_train, y_reg_test = train_test_split(X_reg, y_r
```

In [50]:
```
print("Training set shape:", X_reg_train.shape, y_reg_train.shape)
print("Testing set shape:", X_reg_test.shape, y_reg_test.shape)
```

```
Training set shape: (18664, 5859) (18664,)
Testing set shape: (7999, 5859) (7999,)
```

Model Training

Suppoer Vector Machine

Linear Kernel

In [31]:
```
# Initializing the SVM classifier
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear
```

Out[31]:
```
SVC(kernel='linear', random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```
# Fit the classifier on the training data
svm_linear.fit(X_train, y_train)
```

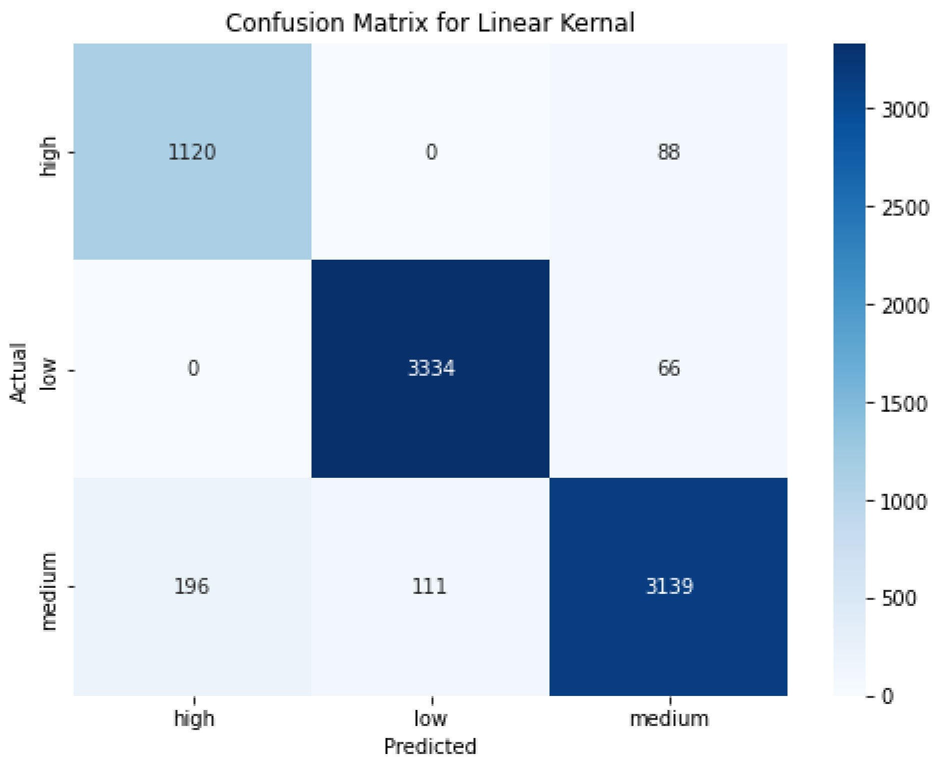
Out[32]:
```
SVC(kernel='linear', random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [33]:
```
# Predict on the test set
y_pred_linear = svm_linear.predict(X_test)
```

In [34]:
```
#To print the confusion matrix
cm_linear = confusion_matrix(y_test, y_pred_linear)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_linear, annot=True, cmap='Blues', fmt='d', xticklabels=label_en
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Linear Kernal')
plt.show()
```



In [35]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_linear))
precision_linear, recall_linear, fscore_linear, _ = precision_recall_fscore_su
print("Precision:", precision_linear)
print("Recall:", recall_linear)
print("F1 Score:", fscore_linear)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_linear))
```

```
Accuracy Score: 0.9427613608145021
Precision: 0.9440501435515866
Recall: 0.9427613608145021
F1 Score: 0.9429382525994325

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      1208
           1       0.97      0.98      0.97      3400
           2       0.95      0.91      0.93      3446

    accuracy                           0.94      8054
   macro avg       0.92      0.94      0.93      8054
weighted avg       0.94      0.94      0.94      8054
```

Polynomial Kernel

In [36]:
```
svm_poly = SVC(kernel='poly', degree=3, random_state=42)
svm_poly
```

Out[36]: SVC(kernel='poly', random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [37]:
```
# Fit the classifier on the training data
svm_poly.fit(X_train, y_train)
```

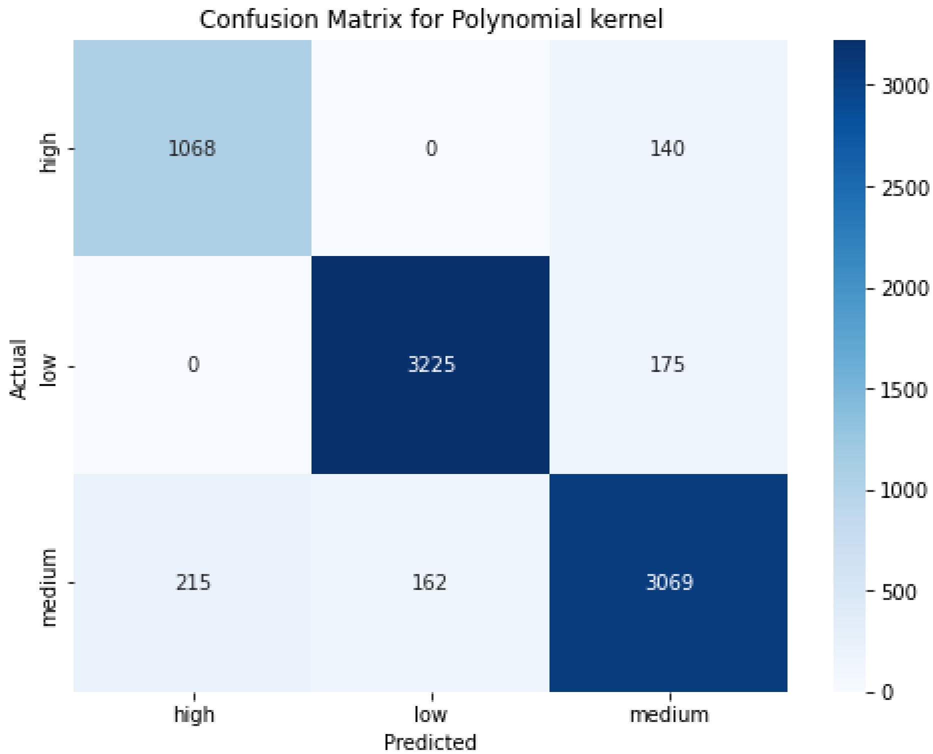
Out[37]: SVC(kernel='poly', random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [38]:
```
# Predict on the test set
y_pred_poly = svm_poly.predict(X_test)
```

In [39]:
```
#To print the confusion matrix
cm_poly = confusion_matrix(y_test, y_pred_poly)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_poly, annot=True, cmap='Blues', fmt='d', xticklabels=label_enco
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Polynomial kernel')
plt.show()
```



In [40]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_poly))
precision_poly, recall_poly, fscore_poly, _ = precision_recall_fscore_support(
print("Precision:", precision_poly)
print("Recall:", recall_poly)
print("F1 Score:", fscore_poly)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_poly))
```

```
Accuracy Score: 0.9140799602681897
Precision: 0.9148466766504375
Recall: 0.9140799602681897
F1 Score: 0.9143134642032293

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      1208
           1       0.95      0.95      0.95      3400
           2       0.91      0.89      0.90      3446

    accuracy                           0.91      8054
   macro avg       0.90      0.91      0.90      8054
weighted avg       0.91      0.91      0.91      8054
```

RBF Kernel

In [41]:
```
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf
```

Out[41]: SVC(random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [42]:
```
# Fit the classifier on the training data
svm_rbf.fit(X_train, y_train)
```

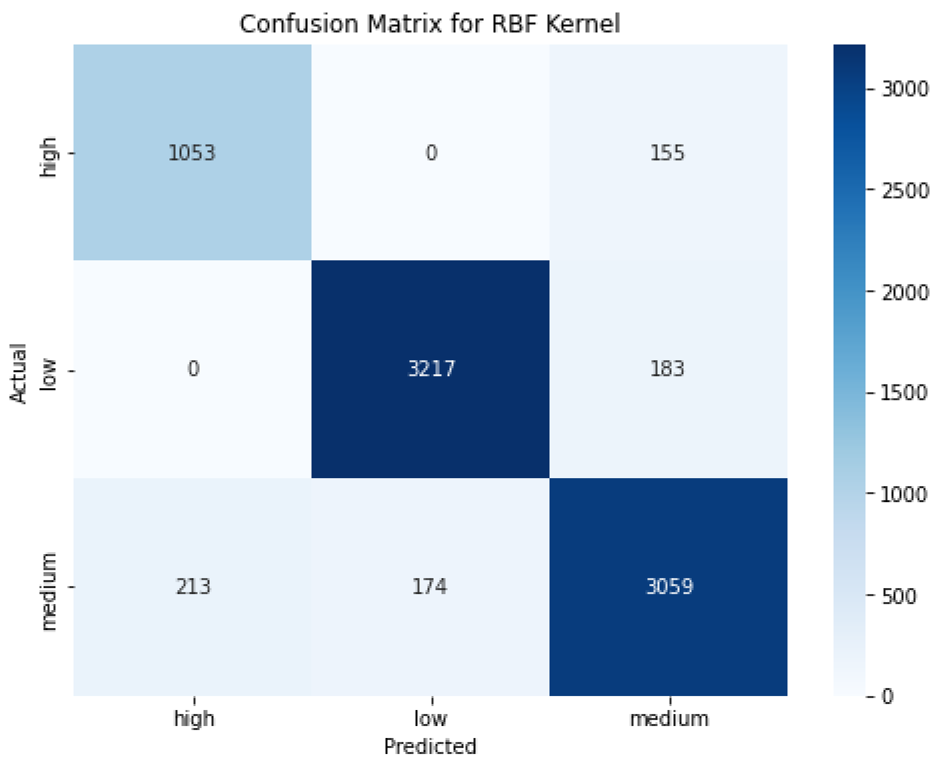
Out[42]: SVC(random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [43]:
```
# Predict on the test set
y_pred_rbf = svm_rbf.predict(X_test)
```

In [44]:
```
#To print the confusion matrix
cm_rbf = confusion_matrix(y_test, y_pred_rbf)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_rbf, annot=True, cmap='Blues', fmt='d', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for RBF Kernel')
plt.show()
```



In [45]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_rbf))
precision_rbf, recall_rbf, fscore_rbf, _ = precision_recall_fscore_support(y_t
print("Precision:", precision_rbf)
print("Recall:", recall_rbf)
print("F1 Score:", fscore_rbf)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_rbf))
```

```
Accuracy Score: 0.9099826173330022
Precision: 0.9105315377204227
Recall: 0.9099826173330022
F1 Score: 0.9101665483902712

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.87      0.85      1208
           1       0.95      0.95      0.95      3400
           2       0.90      0.89      0.89      3446

    accuracy                           0.91      8054
   macro avg       0.89      0.90      0.90      8054
weighted avg       0.91      0.91      0.91      8054
```

Sigmoid Kernel

In [46]:
```
svm_sigmoid = SVC(kernel='sigmoid', random_state=42)
svm_sigmoid
```

Out[46]:
```
SVC(kernel='sigmoid', random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [47]:
```
# Fit the classifier on the training data
svm_sigmoid.fit(X_train, y_train)
```

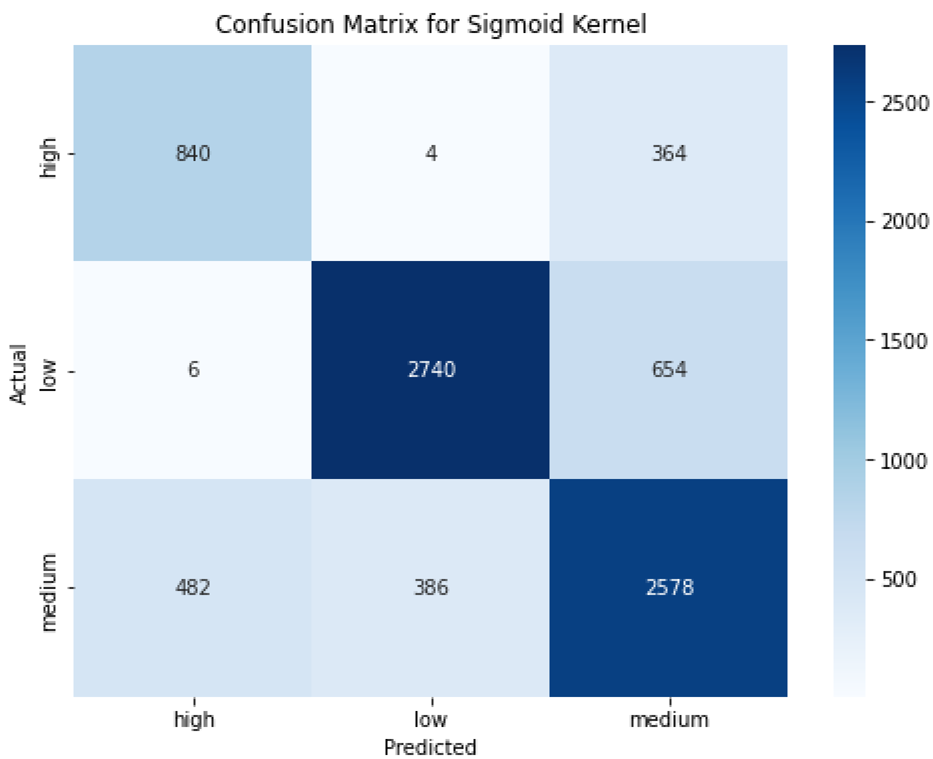
Out[47]:
```
SVC(kernel='sigmoid', random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [48]:
```
# Predict on the test set
y_pred_sigmoid = svm_sigmoid.predict(X_test)
```

In [49]:
```
#To print the confusion matrix
cm_sigmoid = confusion_matrix(y_test, y_pred_sigmoid)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_sigmoid, annot=True, cmap='Blues', fmt='d', xticklabels=label_e
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Sigmoid Kernel')
plt.show()
```



In [50]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_sigmoid))
precision_sigmoid, recall_sigmoid, fscore_sigmoid, _ = precision_recall_fscore
print("Precision:", precision_sigmoid)
print("Recall:", recall_sigmoid)
print("F1 Score:", fscore_sigmoid)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_sigmoid))
```

```
Accuracy Score: 0.76458902408741
Precision: 0.7711594315543839
Recall: 0.76458902408741
F1 Score: 0.7669023448249996

Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.70      0.66      1208
           1       0.88      0.81      0.84      3400
           2       0.72      0.75      0.73      3446

    accuracy                           0.76      8054
   macro avg       0.74      0.75      0.74      8054
weighted avg       0.77      0.76      0.77      8054
```

Naive Bayes

Gaussian Naive Bayes (GNB)

In [32]:
```
# Initializing Gaussian Naive Bayes classifier
gnb = GaussianNB()
```

In [33]:
```
# Fit the classifier on the training data
gnb.fit(X_train, y_train)
```

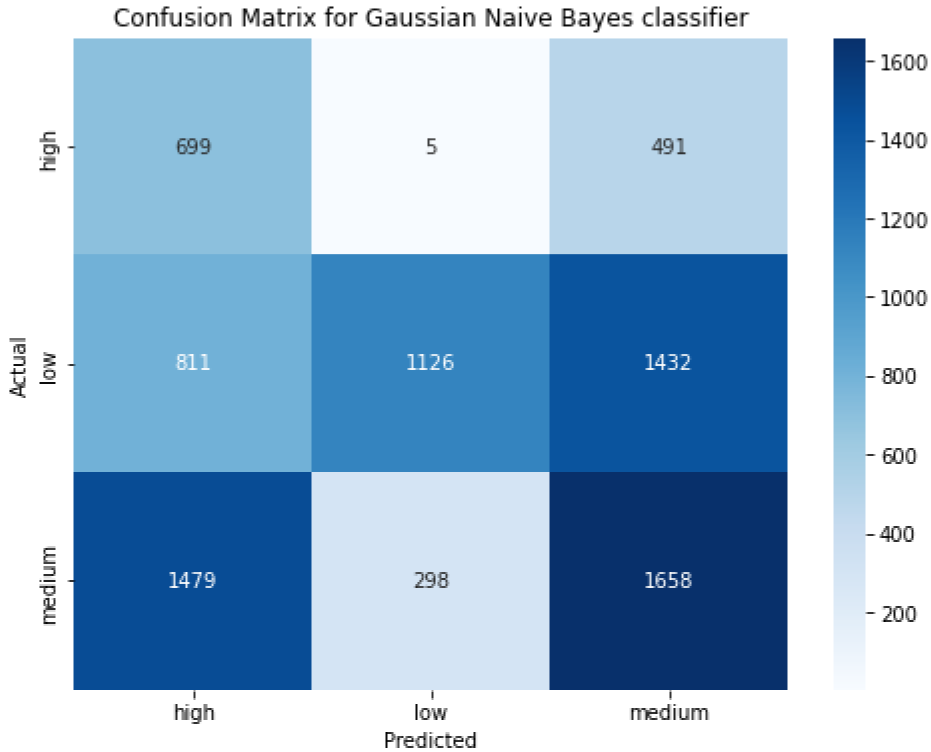
Out[33]: GaussianNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```
# Predict on the test data
y_pred_gnb = gnb.predict(X_test)
```

In [35]:
```
# To print the confusion matrix
cm_gnb = confusion_matrix(y_test, y_pred_gnb)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_gnb, annot=True, cmap='Blues', fmt='d', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Gaussian Naive Bayes classifier')
plt.show()
```



In [36]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_gnb))
precision_gnb, recall_gnb, fscore_gnb, _ = precision_recall_fscore_support(y_t
print("Precision:", precision_gnb)
print("Recall:", recall_gnb)
print("F1 Score:", fscore_gnb)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_gnb))
```

```
Accuracy Score: 0.4354294286785848
Precision: 0.5656346059512688
Recall: 0.4354294286785848
F1 Score: 0.4505643779951977

Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.58      0.33      1195
           1       0.79      0.33      0.47      3369
           2       0.46      0.48      0.47      3435

    accuracy                           0.44      7999
   macro avg       0.49      0.47      0.43      7999
weighted avg       0.57      0.44      0.45      7999
```

Multinomial Naive Bayes (MNB)

In [37]:
```
# Initializing Multinomial Naive Bayes classifier
mnb = MultinomialNB()
```

In [38]:
```
# Fit the classifier on the training data
mnb.fit(X_train, y_train)
```

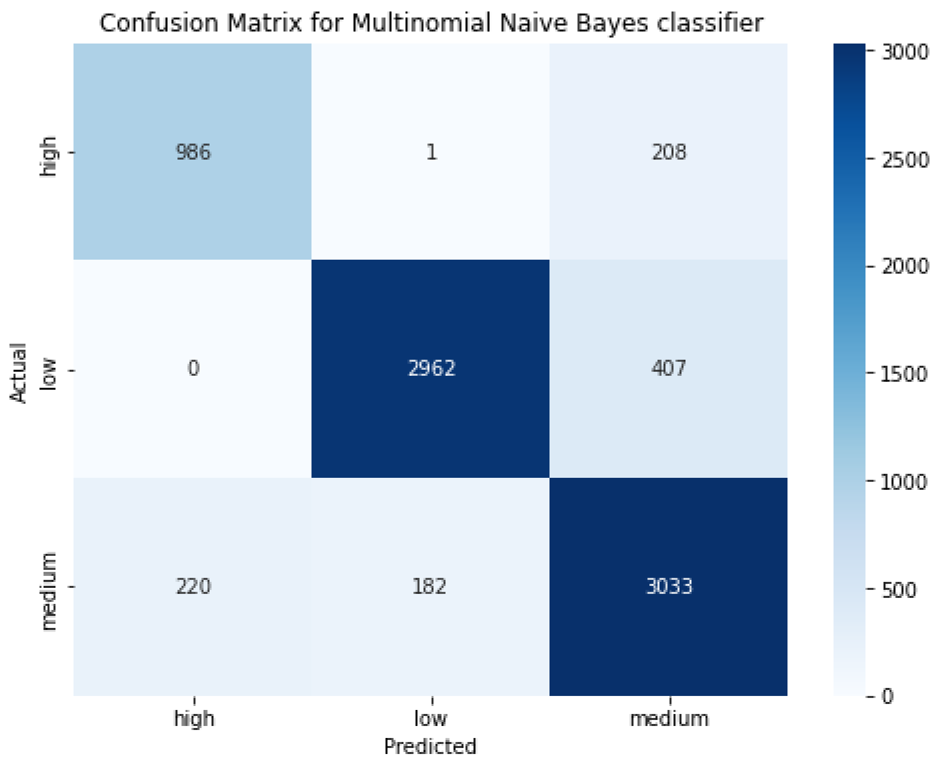
Out[38]:
```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [39]:
```
# Predict on the test data
y_pred_mnb = mnb.predict(X_test)
```

In [40]:
```
# To print the confusion matrix
cm_mnb = confusion_matrix(y_test, y_pred_mnb)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_mnb, annot=True, cmap='Blues', fmt='d', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Multinomial Naive Bayes classifier')
plt.show()
```



In [41]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_mnb))
precision_mnb, recall_mnb, fscore_mnb, _ = precision_recall_fscore_support(y_t
print("Precision:", precision_mnb)
print("Recall:", recall_mnb)
print("F1 Score:", fscore_mnb)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_mnb))
```

```
Accuracy Score: 0.8727340917614702
Precision: 0.8758446466241884
Recall: 0.8727340917614702
F1 Score: 0.8735004406405089

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.82      1195
           1       0.94      0.88      0.91      3369
           2       0.83      0.88      0.86      3435

    accuracy                           0.87      7999
   macro avg       0.86      0.86      0.86      7999
weighted avg       0.88      0.87      0.87      7999
```

Bernoulli Naive Bayes (BNB)

In [42]:
```
# Initializing Bernoulli Naive Bayes classifier
bnb = BernoulliNB()
```

In [43]:
```
# Fit the classifier on the training data
bnb.fit(X_train, y_train)
```

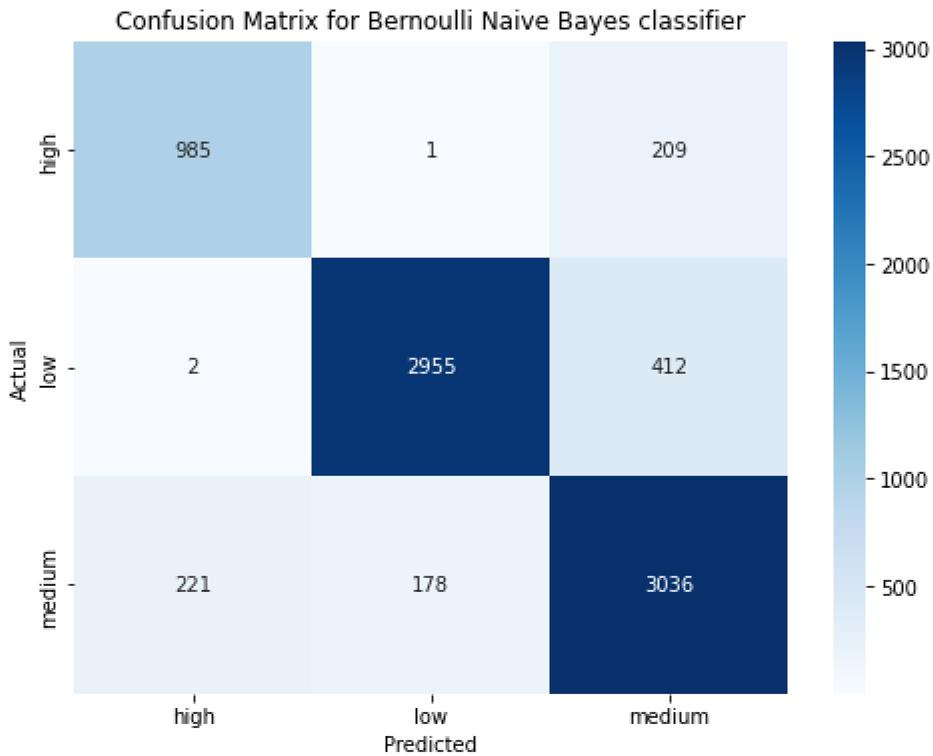
Out[43]:
```
BernoulliNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
# Predict on the test data
y_pred_bnb = bnb.predict(X_test)
```

In [45]:
```
# To print the confusion matrix
cm_bnb = confusion_matrix(y_test, y_pred_bnb)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_bnb, annot=True, cmap='Blues', fmt='d', xticklabels=label_encod
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Bernoulli Naive Bayes classifier')
plt.show()
```



In [46]:
```
# Evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred_bnb))
precision_bnb, recall_bnb, fscore_bnb, _ = precision_recall_fscore_support(y_t
print("Precision:", precision_bnb)
print("Recall:", recall_bnb)
print("F1 Score:", fscore_bnb)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_bnb))
```

```
Accuracy Score: 0.8721090136267033
Precision: 0.8754439059157396
Recall: 0.8721090136267033
F1 Score: 0.8729117441093668

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1195
           1       0.94      0.88      0.91      3369
           2       0.83      0.88      0.86      3435

    accuracy                           0.87      7999
   macro avg       0.86      0.86      0.86      7999
weighted avg       0.88      0.87      0.87      7999
```

In [ ]: