In [1]:
```
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 scipy import stats
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [2]:
```
data = pd.read_csv(r"rideshare_kaggle.csv").sample(30000)
data.drop('id', axis=1, inplace=True)
data
```

Out[2]:

	timestamp	hour	day	month	datetime	timezone	source	destination
238205	1.543303e+09	7	27	11	2018-11-27 07:24:21	America/New_York	Boston University	North Station
543637	1.544711e+09	14	13	12	2018-12-13 14:25:13	America/New_York	Boston University	Back Bay
589607	1.543333e+09	15	27	11	2018-11-27 15:36:22	America/New_York	South Station	Beacon Hill
183879	1.545106e+09	4	18	12	2018-12-18 04:10:10	America/New_York	Theatre District	Northeastern University
535959	1.543272e+09	22	26	11	2018-11-26 22:32:28	America/New_York	Financial District	North End
...	...	...	...	...	...	...	...	...
170527	1.543283e+09	1	27	11	2018-11-27 01:51:21	America/New_York	Northeastern University	Beacon Hill
640612	1.544904e+09	20	15	12	2018-12-15 20:05:05	America/New_York	Northeastern University	Financial District
430949	1.545087e+09	22	17	12	2018-12-17 22:50:07	America/New_York	North Station	Northeastern University
424968	1.544866e+09	9	15	12	2018-12-15 09:20:07	America/New_York	Fenway	Theatre District
627605	1.543618e+09	22	30	11	2018-11-30 22:42:58	America/New_York	Northeastern University	Back Bay

30000 rows × 56 columns

Data Preprocessing

Data Inspection

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

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                          2408
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
meanofprice = data['price'].mean()
data['price'].fillna(meanofprice, 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 [10]:
```
pricebins = [float('-inf'), 13, 26, float('inf')]
pricelabels = ['low', 'medium', 'high']
```

In [14]:
```
data['pricecategory'] = pd.cut(data['price'], bins=pricebins, labels=pricelabe
```

In [15]:
```
data.shape
```

Out[15]:
```
(30000, 58)
```

In [16]:
```
data['pricecategory']
```

Out[16]:
```
238205       low
543637       low
589607       low
183879    medium
535959       low
           ...
170527    medium
640612    medium
430949    medium
424968       low
627605       low
Name: pricecategory, Length: 30000, dtype: object
```

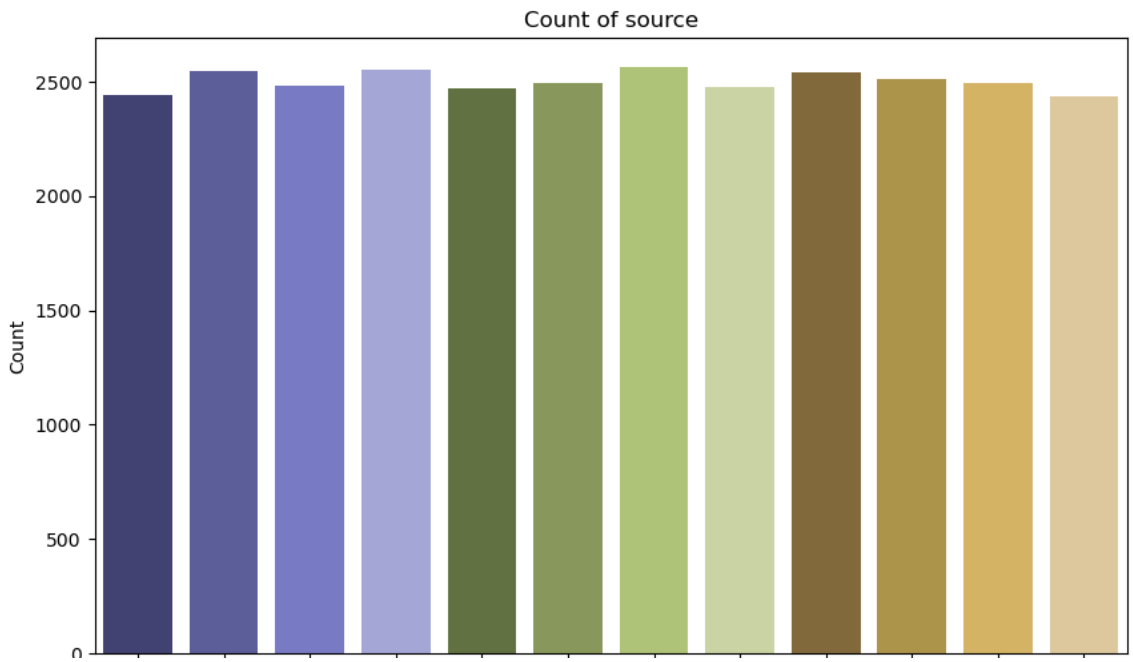
In [17]:
```
data.shape
```

Out[17]:
```
(30000, 58)
```

Data Visualization

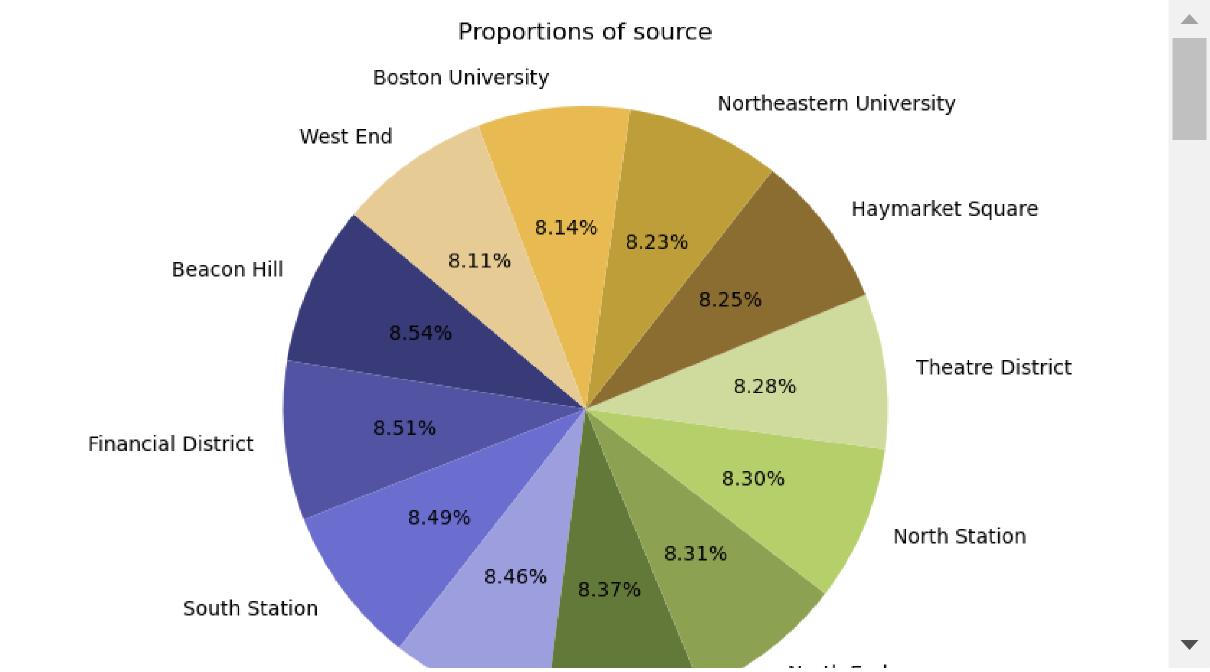
In [21]:
```
categoricalattributes = ['source', 'destination', 'cab_type', 'name', 'priceca

for col in categoricalattributes:
    plt.figure(figsize=(10, 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 [22]:
```
# Pie charts
for col in categoricalattributes:
    # Calculate value counts for each category
    counts = data[col].value_counts()

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



Data Encoding

In [23]:
```
categoricalcolumns = data.select_dtypes(include=['object']).columns.tolist()
categoricalcolumns
```

Out[23]:
```
['datetime',
 'timezone',
 'source',
 'destination',
 'cab_type',
 'product_id',
 'name',
 'short_summary',
 'long_summary',
 'icon',
 'price_category',
 'pricecategory']
```

In [24]:
```
numericcolumns = data.select_dtypes(include=['int', 'float']).columns.tolist()
numericcolumns
```

Out[24]:
```
['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 [26]:
```
# To seperate the ordinal and nominal columns
ordinalcolumns = ['pricecategory']
nominalcolumns = [col for col in categoricalcolumns if col not in ordinalcolum
```

In [27]:
```
# One-hot encoding nominal attributes
dataencoded = pd.get_dummies(data, columns=nominalcolumns, drop_first=True)
```

In [28]:
```
# Label-encoding ordinal attributes
labelencoder = LabelEncoder()
for column in ordinalcolumns:
    dataencoded[column] = labelencoder.fit_transform(data[column])
```

In [29]:
```
dataencoded['pricecategory']
```

Out[29]:
```
238205    1
543637    1
589607    1
183879    2
535959    1
         ..
170527    2
640612    2
430949    2
424968    1
627605    1
Name: pricecategory, Length: 30000, dtype: int32
```

Scaling numeric attributes

In [30]:
```
numericcolumns = dataencoded.select_dtypes(include=['int', 'float']).columns
numericcolumns = numericcolumns.drop('pricecategory', errors='ignore')
```

In [31]:
```
# Creating a StandardScaler instance
scaler = StandardScaler()
```

In [32]:
```
dataencoded[numericcolumns] = scaler.fit_transform(dataencoded[numericcolumns]
```

In [33]:
```
dataencoded
```

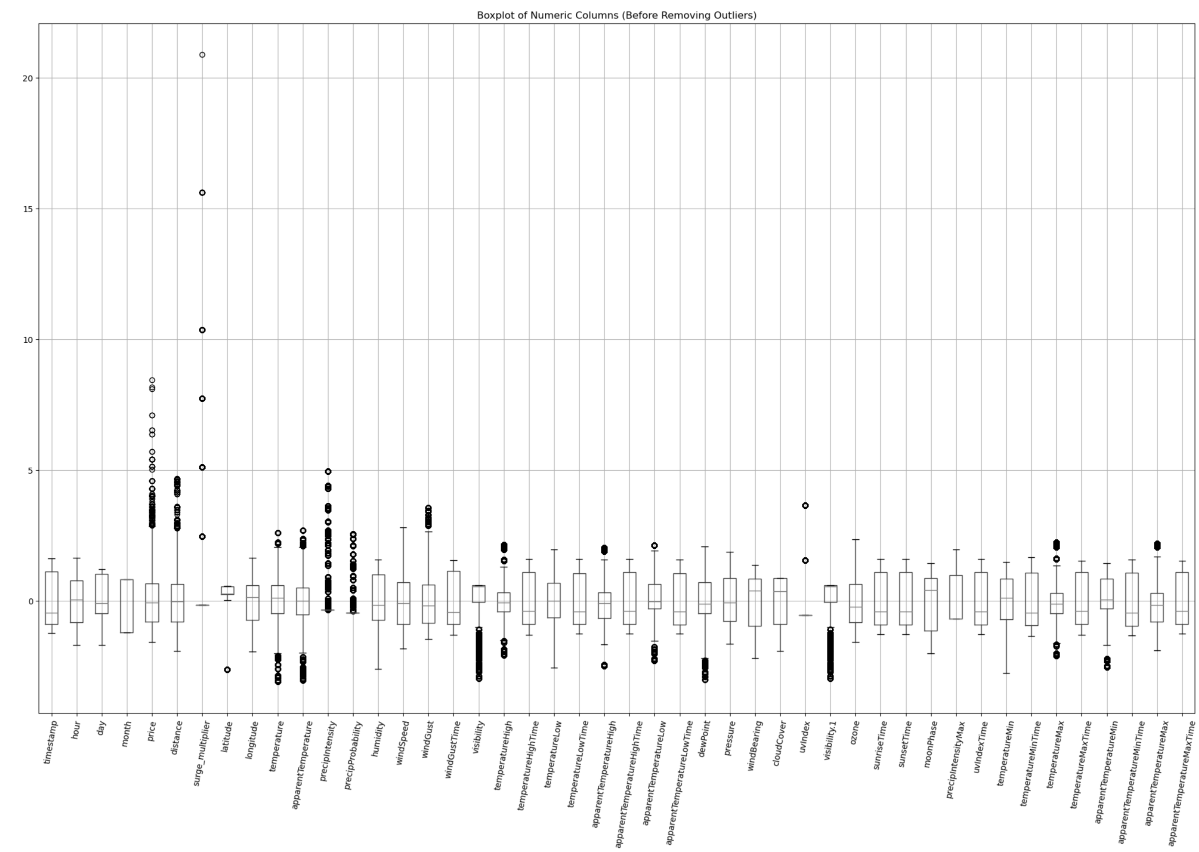
Out[33]:

	timestamp	hour	day	month	price	distance	surge_multiplier	latit
238205	-1.079699	-0.668523	0.923499	-1.194402	-1.063662	1.183507	-0.152039	0.550
543637	0.963381	0.341396	-0.478225	0.837239	-1.063662	0.132024	-0.152039	0.279
589607	-1.036853	0.485670	0.923499	-1.194402	-0.839851	0.317580	-0.152039	0.279
183879	1.536830	-1.101346	0.022391	0.837239	-0.000560	0.202712	-0.152039	0.279
535959	-1.126019	1.495589	0.823376	-1.194402	-1.063662	-0.866444	-0.152039	0.246
...	...	...	...	...	...	...	...	
170527	-1.108698	-1.534168	0.923499	-1.194402	0.670873	0.202712	-0.152039	-2.613
640612	1.243782	1.207041	-0.277979	0.837239	0.447062	2.022927	-0.152039	0.550
430949	1.508957	1.495589	-0.077732	0.837239	0.055393	1.086312	-0.152039	0.425
424968	1.187616	-0.379975	-0.277979	0.837239	-1.511284	0.432448	-0.152039	0.024
627605	-0.623496	1.495589	1.223869	-1.194402	-1.063662	-0.839936	-0.152039	0.550

30000 rows × 16845 columns

Handling the Outliers

In [36]:
```python
# Boxplots before removing outliers
plt.figure(figsize=(25,15))
dataencoded[numericcolumns].boxplot()
plt.title('Boxplot of Numeric Columns (Before Removing Outliers)')
plt.xticks(rotation=80)
plt.show()
```

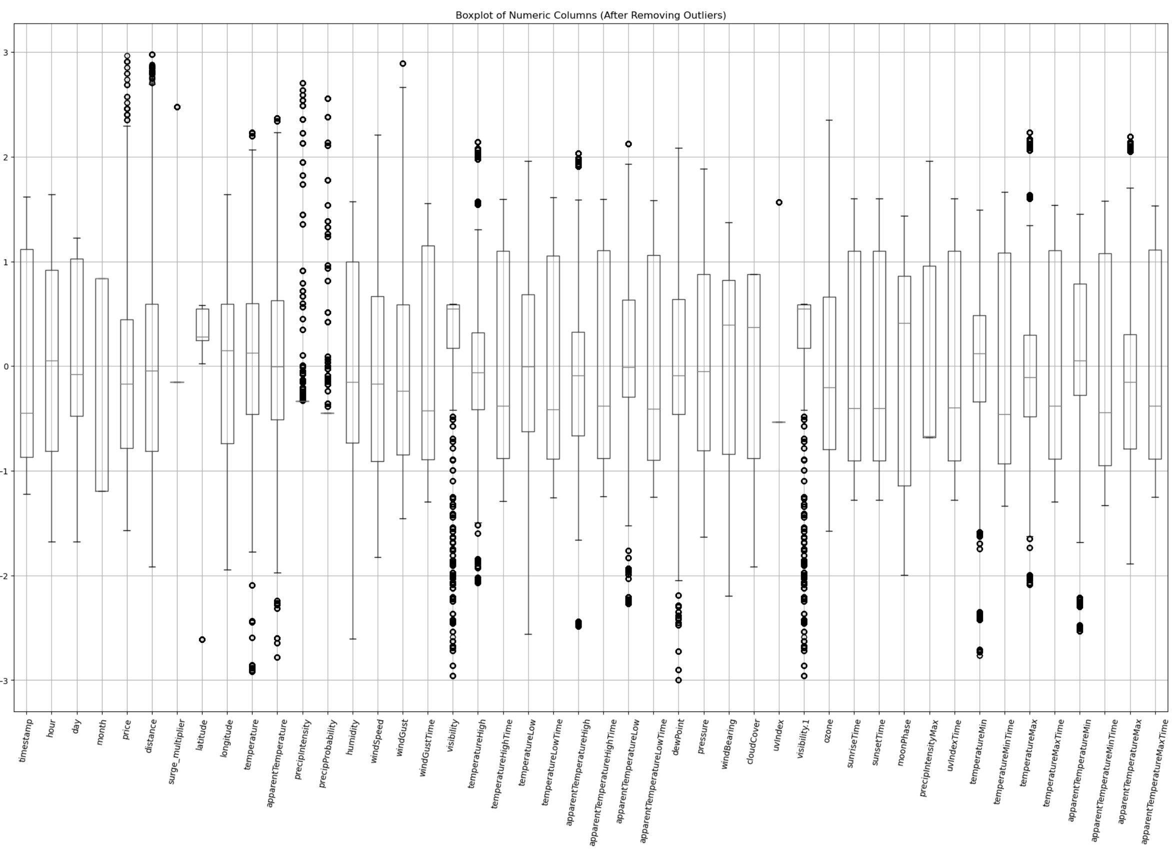


In [38]:
```python
# Removing outliers using the z-score method
dataencoded_no_outliers = dataencoded[(np.abs(stats.zscore(dataencoded[numeric

print("Original Dataset Shape:", dataencoded.shape)
print("Dataset Shape After Removing Outliers:", dataencoded_no_outliers.shape)
```

```
Original Dataset Shape: (30000, 16845)
Dataset Shape After Removing Outliers: (26651, 16845)
```

In [39]:
```
# Boxplots after removing outliers
plt.figure(figsize=(25,15))
dataencoded_no_outliers[numericcolumns].boxplot()
plt.title('Boxplot of Numeric Columns (After Removing Outliers)')
plt.xticks(rotation=80)
plt.show()
```



Data Split

In [40]:
```
X = dataencoded_no_outliers.drop('pricecategory', axis=1)
y = dataencoded_no_outliers['pricecategory']
```

In [41]:
```
# 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 [42]:
```
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (18655, 16844) (18655,)
Testing set shape: (7996, 16844) (7996,)
```

Model Training

Support Vector Machine(SVM)

Linear Kernel

In [43]:
```
# Initializing the SVM classifier
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear
```

Out[43]:
```
SVC
SVC(kernel='linear', random_state=42)
```

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

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

In [*]:
```
#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 [*]:
```
# 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))
```

Polynomial Kernel

In [37]:
```
svm_poly = SVC(kernel='poly', degree=3, random_state=42)
svm_poly
```

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

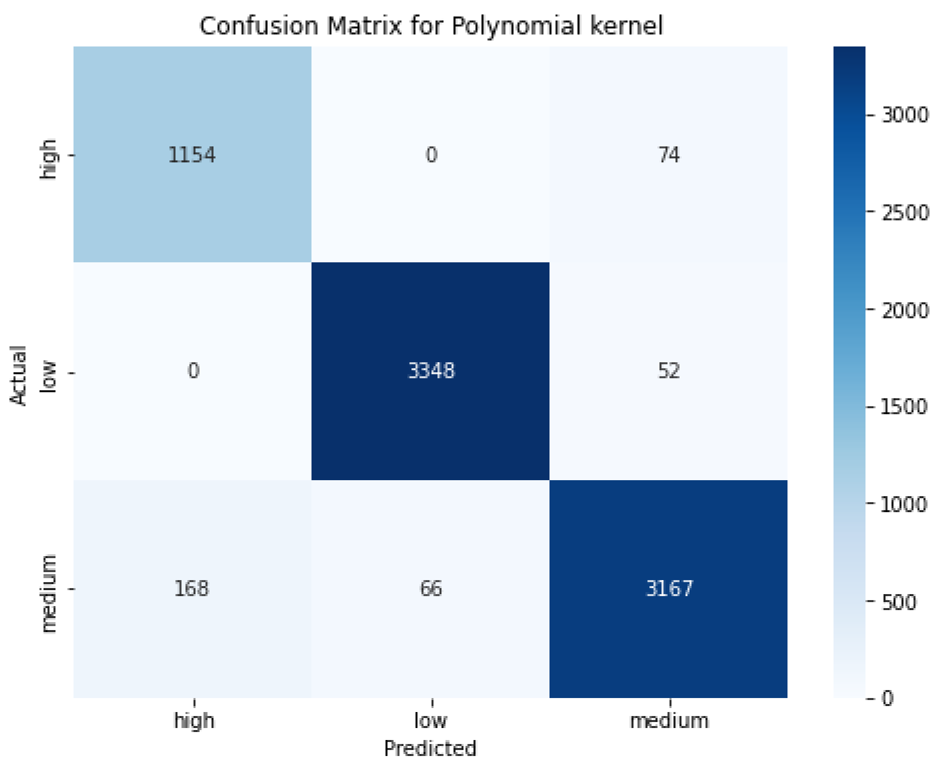
Out[38]:
```
SVC(kernel='poly', random_state=42)
```

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 [39]:
```
# Predict on the test set
y_pred_poly = svm_poly.predict(X_test)
```

In [40]:
```
#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 [41]:
```
# 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: 0.955162535807697
Precision: 0.9561693592307494
Recall: 0.955162535807697
F1 Score: 0.9553714684212993

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.91      1228
           1       0.98      0.98      0.98      3400
           2       0.96      0.93      0.95      3401

    accuracy                           0.96      8029
   macro avg       0.94      0.95      0.94      8029
weighted avg       0.96      0.96      0.96      8029
```

RBF Kernel

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

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

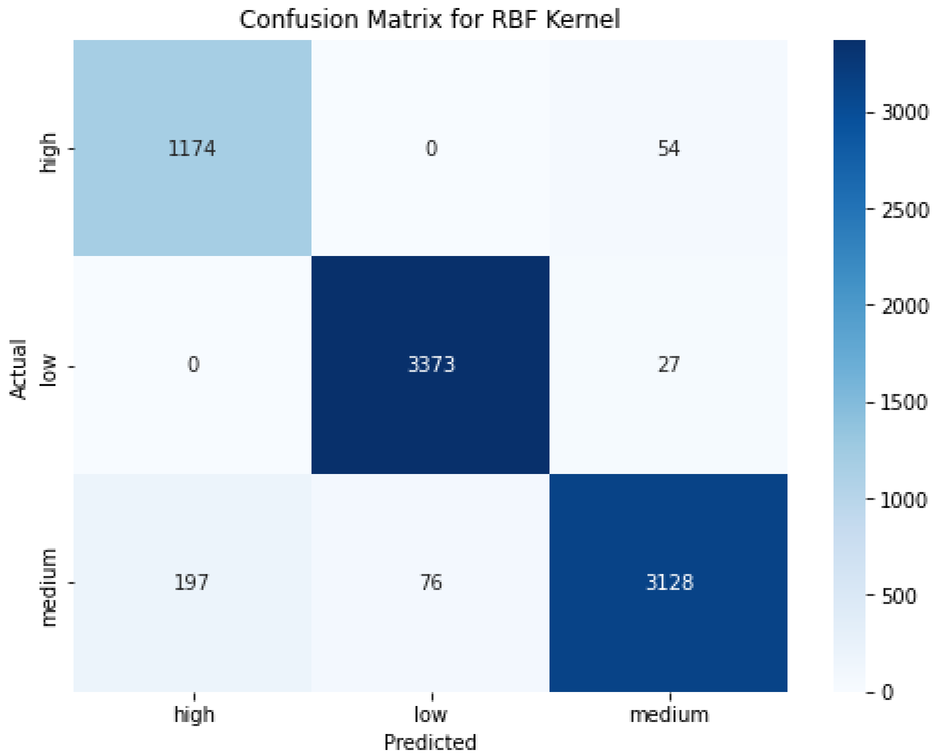
Out[43]:
```
SVC(random_state=42)
```

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 [44]:
```
# Predict on the test set
y_pred_rbf = svm_rbf.predict(X_test)
```

In [45]:
```
#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 [46]:
```
# 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: 0.9559098268775688
Precision: 0.9579998927658411
Recall: 0.9559098268775688
F1 Score: 0.9561754020075687

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.96      0.90      1228
           1       0.98      0.99      0.98      3400
           2       0.97      0.92      0.95      3401

    accuracy                           0.96      8029
   macro avg       0.94      0.96      0.94      8029
weighted avg       0.96      0.96      0.96      8029
```

Sigmoid Kernel

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

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

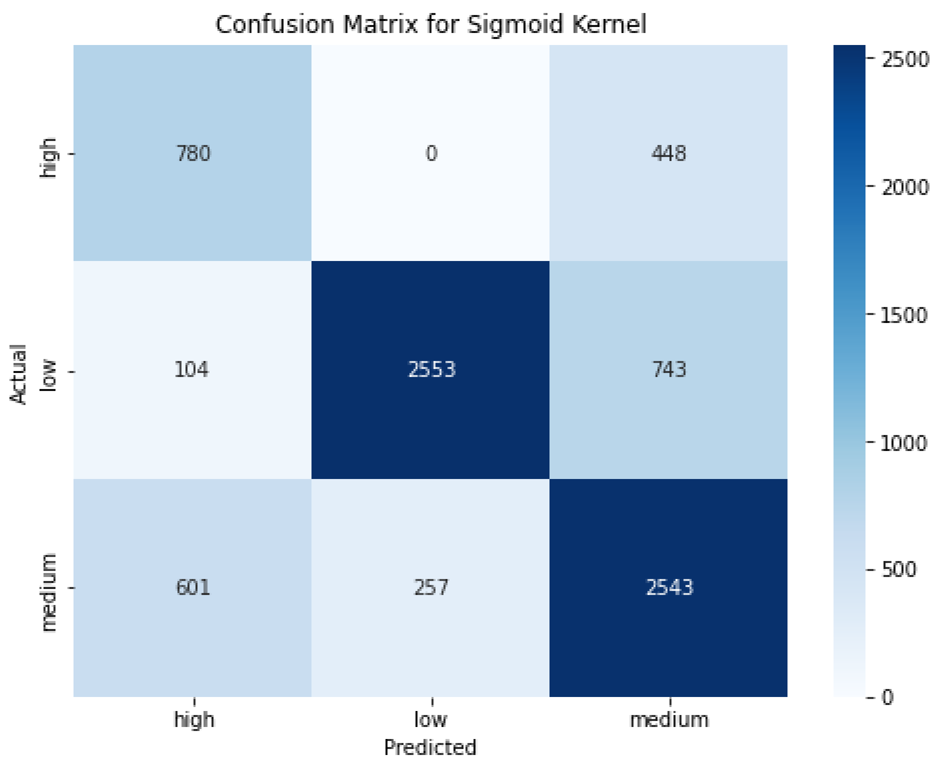
Out[48]: SVC(kernel='sigmoid', random_state=42)

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 [49]:
```
# Predict on the test set
y_pred_sigmoid = svm_sigmoid.predict(X_test)
```

In [50]:
```
#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 [51]:
```
# 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: 0.7318470544276996
Precision: 0.7535512799307592
Recall: 0.7318470544276996
F1 Score: 0.7380721955046605

Classification Report:
              precision    recall  f1-score   support

           0       0.53      0.64      0.58      1228
           1       0.91      0.75      0.82      3400
           2       0.68      0.75      0.71      3401

    accuracy                           0.73      8029
   macro avg       0.70      0.71      0.70      8029
weighted avg       0.75      0.73      0.74      8029
```