Prediction of the Canadian Inflation through Machine Learning and Sentiment Analysis

Sergio Torres ID: 300366657 Github User

Yeison Montoya ID: 300375916 Github User

```
- Special Topics in Data Analytics CSIS 4260-002
- Douglas College - Winter 2024
```

Developed in Python 3.12.0 and Jupyter Notebook

Project Proposal

In this project, we aim to predict trends in the Canadian market basket using sentiment analysis techniques. Sentiment analysis involves analyzing text data to determine the sentiment expressed, whether positive, negative, or neutral. By applying sentiment analysis to market-related text data, such as customer reviews, social media comments, or news articles, we can gain insights into consumer sentiment towards various products or brands.

Objectives

- **Data Collection** a diverse range of textual data from multiple sources, including e-commerce websites, census and statistics, and news articles, pertaining to products or brands within the Canadian market basket.
- **Data Preprocessing** the collected textual data using Python to remove noise, and apply techniques like stopwords removal, punctuation removal, and stemming to prepare the data for sentiment analysis.
- **Sentiment Analysis Model Training** using Python and machine learning techniques, including supervised learning algorithms and deep learning architectures, to assign sentiment scores to the textual data.
- **Prediction and Trend Analysis** assigned to the textual data using Python to identify trends within the Canadian market basket.

- **Insights Generation** from the sentiment analysis results using Python to inform business decisions and identify opportunities for improvement and areas for innovation in the sentiment trends.

- **Evaluation and Validation** Evaluate the performance of the sentiment analysis models using Python and validate the accuracy of the predictions through cross-validation and comparison with ground truth data.

Data Dictionary

- `Time` : Time period for data recording or analysis.
- `Basket Weights` : Weights assigned to items in the Consumer Price Index (CPI) basket.
- `CPI` : Consumer Price Index representing changes in the price level of goods and services.
- `Interest Rate` : Interest rate offered by Canada's chartered banks.
- `Keywords` : Keywords associated with news articles.
- `Title` : Title of news articles.
- `Average Sentiment` : Average sentiment score of news articles related to a keyword.
- `Average Subjectivity` : Average subjectivity score of news articles related to a keyword.
- `Inflation Rate` : Rate of change in the general price level of goods and services.
- `Selected Features` : Features selected for modeling and analysis.
- `Accuracy` : Proportion of correctly classified instances in model evaluation.
- `Recall` : Proportion of actual positives correctly identified in model evaluation.
- `Precision` : Proportion of true positives among all positive predictions in model evaluation.
- `F1 Score` : Weighted average of precision and recall in model evaluation.
- `Word Cloud Data` : Words included in the word cloud and their frequencies.

Exepcted Outcomes

- A comprehensive report detailing sentiment trends within the Canadian market basket.
- A predictive model capable of accurately determining sentiment for gathered news data.
- A validation section that confirms the reliability of your model through statistical metrics.

Table of Contents

- Chapter 1 - Data Collection, Preprocessing and Cleaning

- 1.1. Basket Weights in Relation to the CPI
- 1.2. Understanding Canada's Chartered Banks' Interest Rates
- 1.3. Understanding the Consumer Price Index - CPI
- 1.4. Data Preprocessing and Cleaning

- Chapter 2 - Exploratory Data Analysis - EDA
  - 2.1. Correlation Matrix
  - 2.2. Removing Columns and Separating Features and Response
  - 2.3. Feature Selection
  - 2.4. Summary Statistics
  - 2.5. Feature Selection Pipeline with Logistic Regression

- Chapter 3 - Initial Model Training
  - 3.1. Splitting the Data into Training and Testing Sets
  - 3.2. Hyperparameter Tuning for Support Vector Classifier (SVC)
  - 3.3. Choosing the best classifiers
  - 3.4. Evaluation Metrics for Model Performance
  - 3.5. Confusion Matrix
  - 3.6. The ROC curve

- Chapter 4 - Evaluation of our Model and Validation
  - 4.1. Loading and Preprocessing Google News Data
  - 4.2. Preprocessing News Dates
  - 4.3. Preprocessing and Filtering News Data
  - 4.4. Sentiment Analysis of News Data
  - 4.5. Sentiment Analysis of Selected Keywords Over Time and Subjectivity
    - 4.5.1. Sentiment Analysis of Selected Keywords Over Time
    - 4.5.2. Sentiment Analysis of Subjectivity
  - 4.6. Importing and Preprocessing Inflation Data
  - 4.7. Preprocessing Sentiment Results
  - 4.8. Transforming Sentiment Data
  - 4.9. Sentiment Data with Pivot
  - 4.10. Merging DataFrames and Handling Missing Values
  - 4.11. Removing Unnecessary Columns and Updating Feature and Response Data

    - 4.12. Creating Pipeline for Feature Scaling, Feature Selection, and Logistic Regression
    - 4.13. Splitting the dataset into training and testing sets
    - 4.14. Model Selection and Evaluation
    - 4.15. Best fitted Model Selection
    - 4.16. Model Evaluation
    - 4.17. Visualizing Model Performance with Confusion Matrix
    - 4.18. The ROC curve indicator
    - 4.19. The 'Top 10' Bigrams in the "news2" file:
    - 4.20. The 'Top 10' Trygrams in the "news2" file:
    - 4.21. The 'Top 10' most common words in the "news2" file:
    - 4.22. The word cloud in the "news2" file:
    - 4.23. Named Entity Recognition (NER) Analysis for the "news2" file:

- Chapter 5 - Conclusions

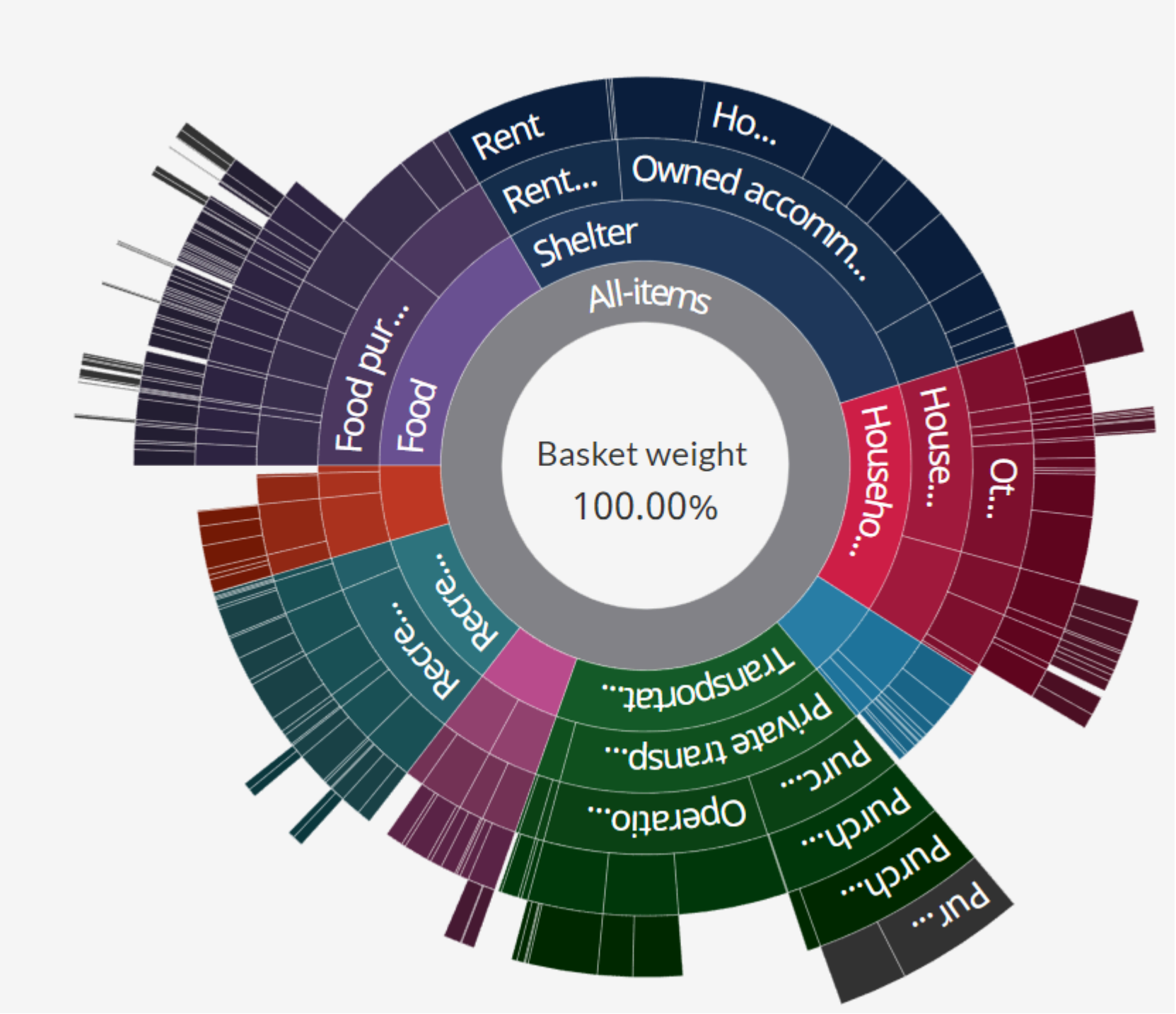
- References

Chapter 1 - Data Collection, Preprocessing and Cleaning

1.1. Basket Weights in Relation to the CPI

The basket weights utilized in this tool correspond to the latest update of the CPI basket, and they are expressed based on prices at the month of basket linkage. Generally, these weights are revised annually to accommodate changes in consumer spending tendencies.

Basket Weights in Canada


All-items
Basket weight
100.00%
Food
Food pur...
Shelter
Rent
Rent...
Owned accomm...
Ho...
Household...
House...
Ot...
Transportat...
Private transp...
Operatio...
Purc...
Purch...
Purch...
Pur...
Recre...
Recre...

The inflation index can be calculated using the following formula:

$$\begin{aligned}\text{Inflation Index} = {} & 28.34\% \times \text{Shelter} + 16.65\% \times \text{Food} + 16.44\% \times \text{Transportation} \\ & + 14.36\% \times \text{Household operations, furnishings and equipment} \\ & + 9.92\% \times \text{Recreation, education and reading} \\ & + 5.03\% \times \text{Health and personal care} \\ & + 4.73\% \times \text{Clothing and footwear} \\ & + 4.53\% \times \text{Alcoholic beverages, tobacco products and recreational cannabis}\end{aligned}$$

1.2. Understanding Canada's Chartered Banks' Interest Rates

Canada's chartered banks

The Bank of Canada's interest rates serve as a crucial benchmark in our model validation process. This platform offers comprehensive insights into the nation's interest rate policies, which are instrumental for tracking economic trends, making strategic financial decisions, or conducting academic research. It stands as a pivotal resource, reflecting the influence of interest rate fluctuations across different economic sectors.

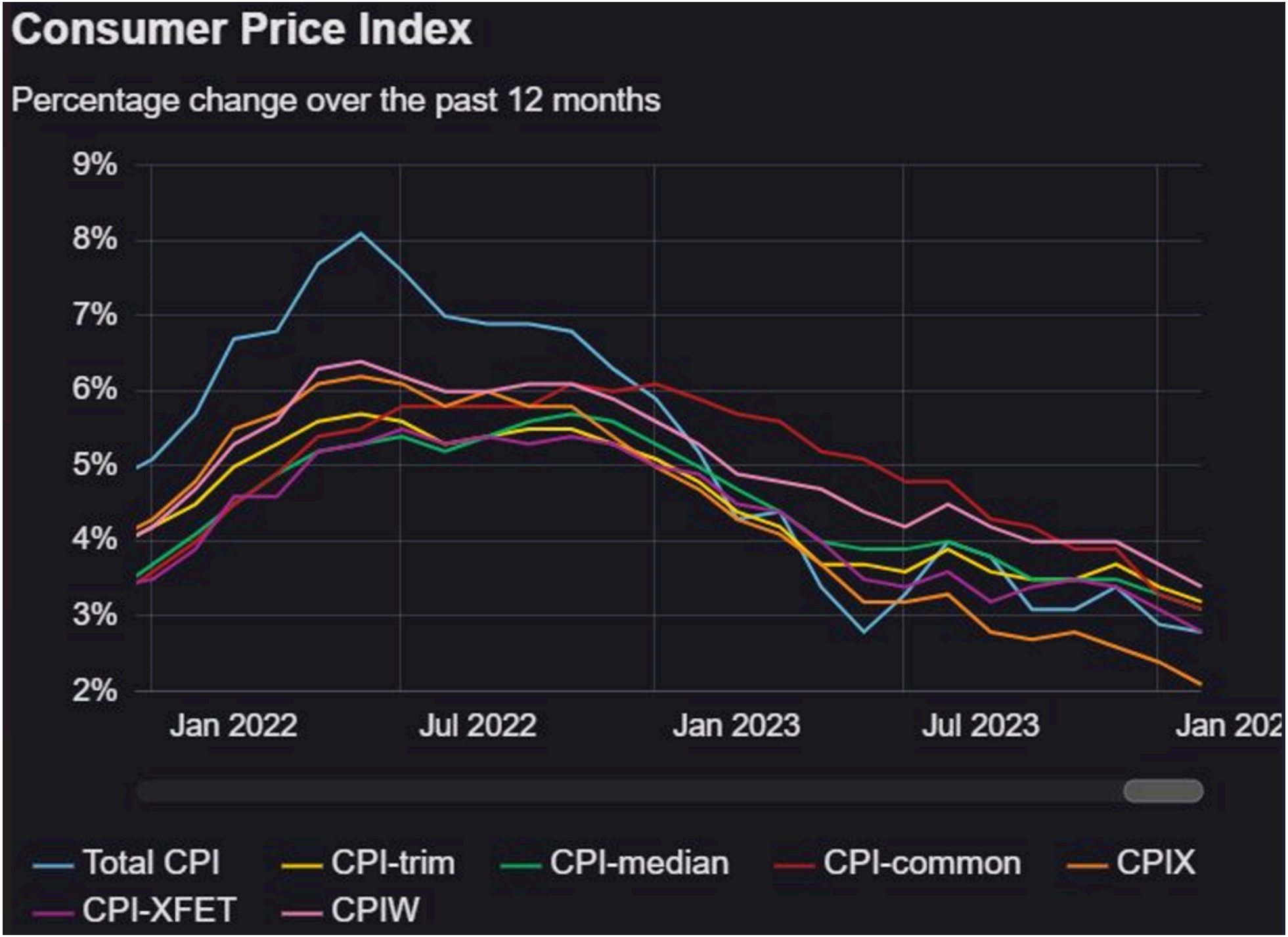
V122530: Bank rate

V122530: Bank rate			
Low		2022-02-01	0.50
Average	2022-01-01 –	2024-03-01	3.81
High		2024-03-01	5.25

1.3. Understanding the Consumer Price Index - CPI

CPI data

The Consumer Price Index (CPI) measures the variation in prices paid by Canadians for a standard set of goods and services over time. It tracks the cost fluctuations of this consistent basket to gauge consumer price changes.


Consumer Price Index
Percentage change over the past 12 months
9%
8%
7%
6%
5%
4%
3%
2%
Jan 2022
Jul 2022
Jan 2023
Jul 2023
Jan 202
Total CPI
CPI-trim
CPI-median
CPI-common
CPIX
CPI-XFET
CPIW

1.4. Data Preprocessing and Cleaning

The code segment demonstrates crucial data preprocessing steps, encompassing loading, cleaning, and transformation, pivotal for subsequent inflation trend analysis.

- Initially, essential libraries, **Pandas** and **NumPy**, are imported, facilitating data manipulation and analysis. Pandas excels in data manipulation, while NumPy supports mathematical operations on arrays and matrices.
- Subsequently, a CSV file named `'1810000401_Inflation_Basket2_new.csv'` is read into a Pandas DataFrame named `'data'`, likely containing historical inflation data.
- The `'Date'` column of the DataFrame is then converted to datetime format using `pd.to_datetime()`, crucial for time-based analysis.
- Following this, the DataFrame is sorted chronologically based on the `'Date'` column using `sort_values()`.
- The DataFrame is filtered to retain rows with `'Date'` on or after September 1, 2022, focusing the analysis on recent trends.
- Rows with NaN values are removed using `dropna()` with `how='all'`, ensuring data integrity.
- Finally, the DataFrame index is reset using `reset_index()` with `drop=True`, maintaining data consistency for subsequent analysis.

In [ ]:
```
# Import necessary libraries
import pandas as pd
import numpy as np

# Load the dataset into a pandas DataFrame
data = pd.read_csv('1810000401_Inflation_Basket2_new.csv')
df = pd.DataFrame(data)

# Convert the 'Date' column to datetime format for proper sorting and manipulation
df['Date'] = pd.to_datetime(df['Date'])

# Sort the DataFrame based on the 'Date' column in ascending order
df.sort_values(by='Date', inplace=True)

# Filter the DataFrame to include only rows from January 1, 2022, onwards
df = df[df['Date'] >= '2022-01-01']

# Remove all rows that contain only NaN values to clean the dataset
df.dropna(how='all', inplace=True)
```

```
# Reset the index of the DataFrame and drop the old index to maintain data integrity
df.reset_index(drop=True, inplace=True)

# Display the first few rows of the cleaned DataFrame
df.head(2)
```

Out[ ]:

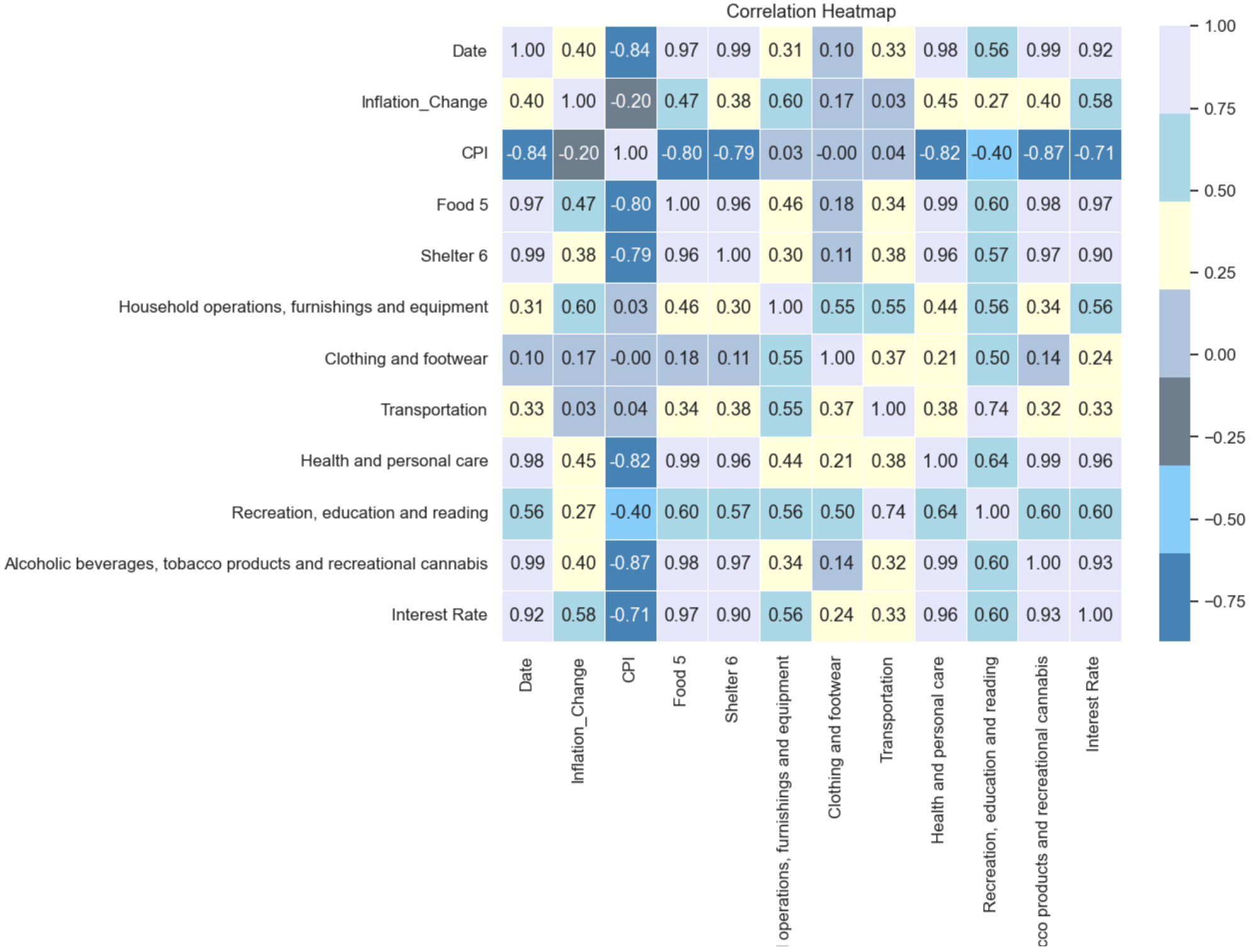
	Date	Inflation_Change	CPI	Food 5	Shelter 6	Household operations, furnishings and equipment	Clothing and footwear	Transportation	Health and personal care	Recreation, education and reading	Alcoholic beverages, tobacco products and recreational cannabis	Interest Rate
0	2022-01-01	0	0.051	163.9	157.6	126.7	93.4	156.9	134.1	119.2	178.2	0.5
1	2022-02-01	0	0.057	166.0	158.6	127.6	93.8	159.9	134.6	120.9	178.5	0.5

Chapter 2 - Exploratory Data Analysis - EDA

2.1. Correlation Matrix

In [ ]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Define the color palette
colors = ['steelblue', 'lightskyblue', 'slategray', 'lightsteelblue', 'lightyellow', 'lightblue', 'lavender']

# Create the correlation heatmap with the defined palette
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap=colors, fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
Date 1.00 0.40 -0.84 0.97 0.99 0.31 0.10 0.33 0.98 0.56 0.99 0.92
Inflation_Change 0.40 1.00 -0.20 0.47 0.38 0.60 0.17 0.03 0.45 0.27 0.40 0.58
CPI -0.84 -0.20 1.00 -0.80 -0.79 0.03 -0.00 0.04 -0.82 -0.40 -0.87 -0.71
Food 5 0.97 0.47 -0.80 1.00 0.96 0.46 0.18 0.34 0.99 0.60 0.98 0.97
Shelter 6 0.99 0.38 -0.79 0.96 1.00 0.30 0.11 0.38 0.96 0.57 0.97 0.90
Household operations, furnishings and equipment 0.31 0.60 0.03 0.46 0.30 1.00 0.55 0.55 0.44 0.56 0.34 0.56
Clothing and footwear 0.10 0.17 -0.00 0.18 0.11 0.55 1.00 0.37 0.21 0.50 0.14 0.24
Transportation 0.33 0.03 0.04 0.34 0.38 0.55 0.37 1.00 0.38 0.74 0.32 0.33
Health and personal care 0.98 0.45 -0.82 0.99 0.96 0.44 0.21 0.38 1.00 0.64 0.99 0.96
Recreation, education and reading 0.56 0.27 -0.40 0.60 0.57 0.56 0.50 0.74 0.64 1.00 0.60 0.60
Alcoholic beverages, tobacco products and recreational cannabis 0.99 0.40 -0.87 0.98 0.97 0.34 0.14 0.32 0.99 0.60 1.00 0.93
Interest Rate 0.92 0.58 -0.71 0.97 0.90 0.56 0.24 0.33 0.96 0.60 0.93 1.00
Date
Inflation_Change
CPI
Food 5
Shelter 6
operations, furnishings and equipment
Clothing and footwear
Transportation
Health and personal care
Recreation, education and reading
cco products and recreational cannabis
Interest Rate
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

Household

Alcoholic beverages, toba

2.2. Removing Columns and Separating Features and Response

In [ ]:
```
# Dropping unnecessary columns from the DataFrame
df = df.drop(['Date','CPI'], axis=1)

# Separating features and response variable
features = df.drop(['Inflation_Change'], axis=1)
response = df['Inflation_Change']

# Displaying the shape of the features DataFrame
features_shape = features.shape
features_shape
```

Out[ ]:
```
(26, 9)
```

2.3. Feature Sellection

In [ ]:
```
features.columns
```

Out[ ]:
```
Index(['Food 5', 'Shelter 6',
       'Household operations, furnishings and equipment',
       'Clothing and footwear', 'Transportation', 'Health and personal care',
       'Recreation, education and reading',
       'Alcoholic beverages, tobacco products and recreational cannabis',
       'Interest Rate'],
      dtype='object')
```

Each of these features represents a different aspect of consumer spending and economic indicators. This information is essential for further analysis and modeling tasks, providing insight into the factors that may influence changes in inflation rates.

In [ ]:

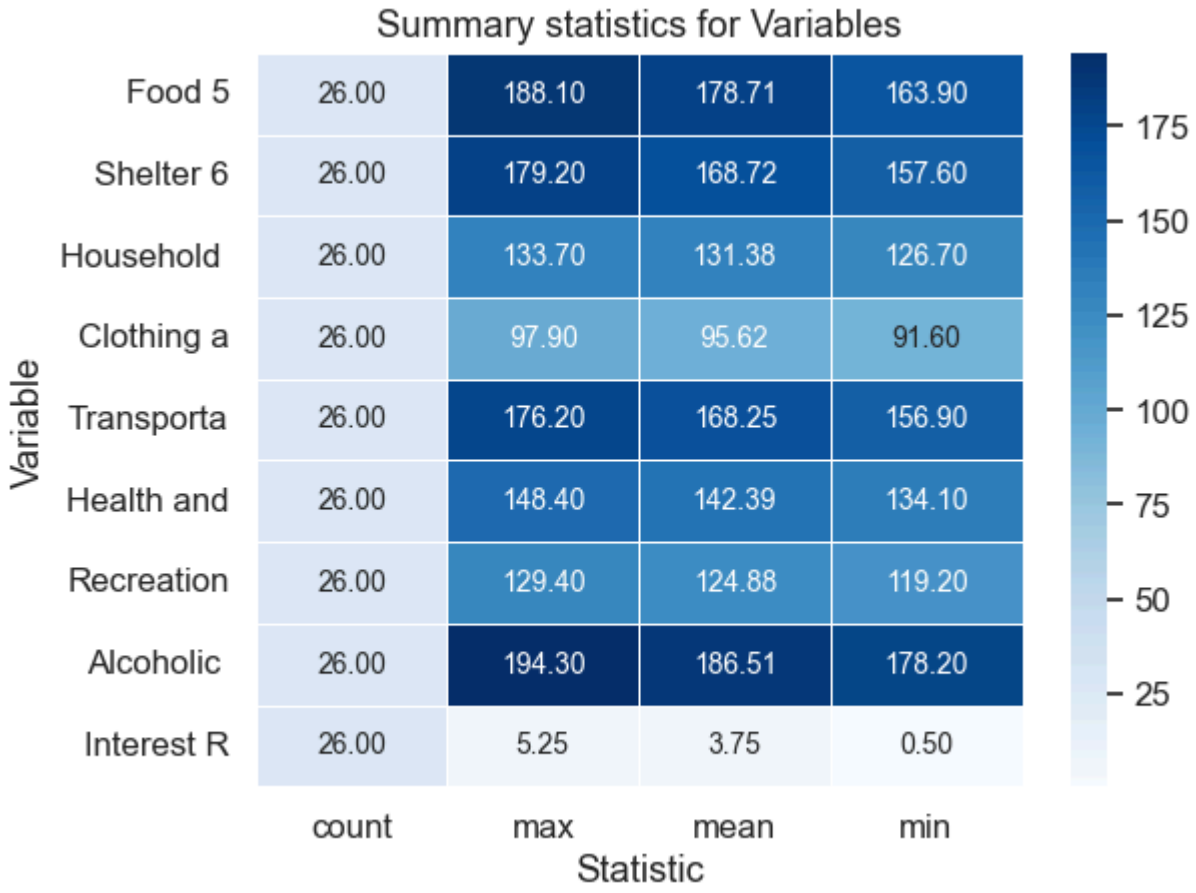
```
# %pip install wordcloud

from wordcloud import WordCloud
import matplotlib.pyplot as plt

# List of words
words = ['Food', 'Shelter', 'Transportation', 'Clothing', 'Healthcare',
         'Recreation', 'Alcoholic beverages', 'Tobacco products',
         'Household operations', 'Interest Rate']

# Convert the list of words into a string
text = ' '.join(words)

# Generate a word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud using matplotlib
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Initial Features', fontsize=14)
plt.show()
```

2.4. Summary Statistics

Now, let's explore some basic summary statistics on the numerical variables based on the variables.

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# List of words
words = ['Food 5', 'Shelter 6',
         'Household operations, furnishings and equipment',
         'Clothing and footwear', 'Transportation', 'Health and personal care',
         'Recreation, education and reading',
         'Alcoholic beverages, tobacco products and recreational cannabis',
         'Interest Rate']
```

```
# Define a function to calculate summary statistics for each variable
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}

# Create a dictionary to store statistics for each variable
stats_dict = {}

# Calculate statistics for each variable in the list
for word in words:
    stats_dict[word] = get_stats(df[word])

# Convert the dictionary to a DataFrame
variable_stats = pd.DataFrame(stats_dict).T

# Sort the statistics DataFrame by 'count'
variable_stats = variable_stats.sort_values(by='count')

# Modify the Y-axis labels to show only the first 10 letters
variable_stats.index = [label[:10] for label in variable_stats.index]

# Create a heatmap using Seaborn
sns.set(font_scale=1.1)

heatmap = sns.heatmap(data=variable_stats[['count', 'max', 'mean', 'min']], annot=True, cmap='Blues', linewidths=0.5, fmt=".2f",
                      annot_kws={'size': 10})

plt.title('Summary statistics for Variables')
plt.xlabel('Statistic')
plt.ylabel('Variable')

plt.tight_layout()
plt.show()
```



In [ ]:
```
df.describe()
```

Out[ ]:

	Inflation_Change	Food 5	Shelter 6	Household operations, furnishings and equipment	Clothing and footwear	Transportation	Health and personal care	Recreation, education and reading	Alcoholic beverages, tobacco products and recreational cannabis	Interest Rate
count	26.000000	26.000000	26.000000	26.000000	26.000000	26.000000	26.000000	26.000000	26.000000	26.000000
mean	0.653846	178.707692	168.715385	131.384615	95.615385	168.250000	142.392308	124.876923	186.511538	3.750000
std	0.485165	7.555815	6.249396	1.683970	1.704510	4.196022	4.653164	2.585700	5.397505	1.714643
min	0.000000	163.900000	157.600000	126.700000	91.600000	156.900000	134.100000	119.200000	178.200000	0.500000
25%	0.000000	172.450000	164.125000	130.825000	94.525000	166.075000	138.725000	123.175000	181.800000	2.750000
50%	1.000000	181.550000	168.150000	131.800000	95.800000	168.500000	142.700000	125.050000	186.900000	4.625000
75%	1.000000	185.150000	173.650000	132.575000	97.175000	170.175000	146.675000	127.025000	191.725000	5.250000
max	1.000000	188.100000	179.200000	133.700000	97.900000	176.200000	148.400000	129.400000	194.300000	5.250000

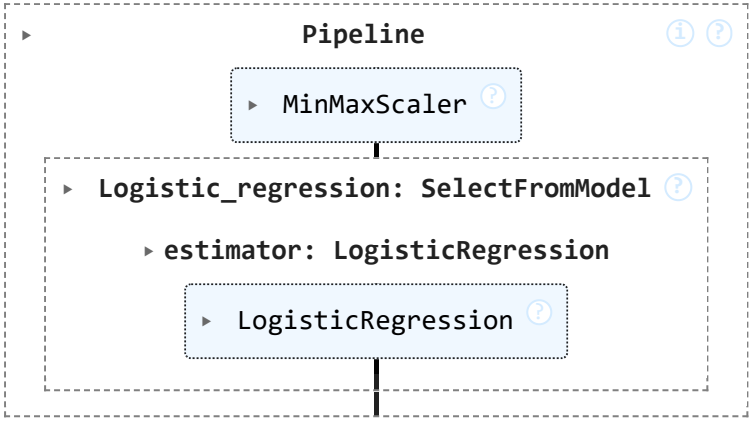
In [ ]:
```
# Showing general info of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 10 columns):
 #   Column                                                  Non-Null Count  Dtype  
---  ------                                                  --------------  -----  
 0   Inflation_Change                                        26 non-null     int64  
 1   Food 5                                                  26 non-null     float64
 2   Shelter 6                                               26 non-null     float64
 3   Household operations, furnishings and equipment         26 non-null     float64
 4   Clothing and footwear                                   26 non-null     float64
 5   Transportation                                          26 non-null     float64
 6   Health and personal care                                26 non-null     float64
 7   Recreation, education and reading                       26 non-null     float64
 8   Alcoholic beverages, tobacco products and recreational cannabis  26 non-null  float64
 9   Interest Rate                                           26 non-null     float64
dtypes: float64(9), int64(1)
memory usage: 2.2 KB
```

2.5. Feature Selection Pipeline with Logistic Regression

In [ ]:
```
# Importing necessary libraries
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

# Defining the machine learning pipeline
pipe_lr = Pipeline([
    ('MinMax_Scaler', MinMaxScaler()),  # Scaling features to a specified range using MinMaxScaler
    ('Logistic_regression', SelectFromModel(estimator=LogisticRegression(max_iter=5000)))  # Selecting features using logistic regression
])

# Fitting the pipeline to the data
pipe_lr.fit(features, response)
```

Out[ ]:



In [ ]:
```
# saving the selected feautures with transformations
X_1= pipe_lr.transform(features)
X_1.shape
```

Out[ ]:
```
(26, 3)
```

In [ ]:
```
#Get the features selected
features_selected_lr = pipe_lr['Logistic_regression'].get_support(indices=True)

selected_feature_names_lr = features.columns[features_selected_lr]
#Create a new dataframe with features selected
df_features_selected_lr = pd.DataFrame(X_1, columns=selected_feature_names_lr)
#df_features_selected_lr.head()
df_features_selected_lr.columns
```

Out[ ]:
```
Index(['Food 5', 'Household operations, furnishings and equipment',
       'Interest Rate'],
      dtype='object')
```

The **target variable** is `selected_feature_names_lr` which holds the names of the features selected by the logistic regression model (`Logistic_regression`) within the **pipeline (`pipe_lr`)**.

In [ ]:
```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Define the list of features
features = df_features_selected_lr.columns

# Combine all features into a single string
text = ' '.join(features)

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Selected Features', fontsize=14)
plt.show()
```

Chapter 3 - Initial Model Training

3.1. Splitting the Data into Training and Testing Sets

In [ ]:
```
# Importing the necessary function from scikit-learn to split the data
from sklearn.model_selection import train_test_split

# Splitting the transformed feature set (X_1) and the response variable (response) into training and testing sets
# The test set size is set to 30% of the total data, and a random seed (random_state) of 42 is used for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X_1, response, test_size=0.30, random_state=42)
```

3.2. Hyperparameter Tuning for Support Vector Classifier (SVC)

In [ ]:
```python
# Importing the Support Vector Classifier (SVC) from scikit-learn
from sklearn.svm import SVC

# Specifying the values to search for the best hyperparameters
gammas = [0.001, 0.01, 0.1, 1, 10]
Cs = [0.01, 0.1, 1, 10, 100]

# Initializing variables to store the best score and corresponding hyperparameters
best_score = 0
best_C = 0
best_gamma = 0

# Iterating over each combination of gamma and C values
for gamma in gammas:
    for C in Cs:
        # Creating an SVC instance with the current gamma and C values
        svm = SVC(gamma=gamma, C=C, random_state=42)

        # Fitting the SVC model to the training data
        svm.fit(X_train, y_train)

        # Calculating the accuracy score on the testing data
        score = svm.score(X_test, y_test)

        # Updating the best score and corresponding hyperparameters if the current score is higher
        if score > best_score:
            best_score = score
            best_C = C
            best_gamma = gamma

# Printing the best score and corresponding hyperparameters
print('Best Score:', best_score)
print('Best C:', best_C)
print('Best gamma:', best_gamma)
```

```
Best Score: 1.0
Best C: 100
Best gamma: 0.01
```

3.3. Choosing the best classifiers

In [ ]:
```python
# Model Selection and Evaluation

# Importing necessary libraries
```

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Defining a list of classifiers with their corresponding names
classifiers = [('Naive_Bayes', GaussianNB()),
               ('Logistic_Regression', LogisticRegression()),
               ('Random_Forest', RandomForestClassifier(max_depth=3, n_estimators=1000)),
               ('Decision_Tree', DecisionTreeClassifier(max_depth=3, random_state=0)),
               ('svc_linear', SVC(kernel='linear', C=0.025, probability=True)),
               ('svm', SVC(gamma=best_gamma, C=best_C, probability=True, random_state=0))
              ]

# Initializing variables to store results
list_accuracy = []
best_classifier = 0
chosen_classifier = 0

# Initializing KFold for cross-validation
kf = KFold(n_splits=3, shuffle=True, random_state=42)

# Looping through each classifier and evaluating its performance using cross-validation
for n, clf in classifiers:
    # Performing cross-validation
    score = cross_val_score(clf, X_train, y_train, cv=kf, scoring='accuracy')

    # Calculating the mean score
    mean_score = np.mean(score)

    # Appending the classifier name and mean score to the list
    list_accuracy.append((n, mean_score))

    # Updating the best classifier if the current mean score is higher
    if best_classifier < mean_score:
        best_classifier = mean_score
        chosen_classifier = clf

# Converting the list of results into a DataFrame
results_df = pd.DataFrame(list_accuracy, columns=['Name', 'MeanScore'])
results_df
```

Out[ ]:

	Name	MeanScore
0	Naive_Bayes	0.722222
1	Logistic_Regression	0.722222
2	Random_Forest	0.722222
3	Decision_Tree	0.722222
4	svc_linear	0.611111
5	svm	0.777778

In [ ]:
```
# Fitting the chosen classifier on the training data
chosen_classifier.fit(X_train, y_train)
```

Out[ ]:
```
SVC
SVC(C=100, gamma=0.01, probability=True, random_state=0)
```

In [ ]:
```
# Function to calculate recall score
def get_recall_score(model):
    '''
    Calculate recall scores for the model on both training and test sets.

    Parameters:
    model (classifier): The classifier to predict values of X.

    Returns:
    None
    '''
    # Predicting labels for training and test sets
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    # Computing recall scores
    recall_train = metrics.recall_score(y_train, pred_train)
    recall_test = metrics.recall_score(y_test, pred_test)

    # Printing recall scores
    print("Recall on training set: ", recall_train)
    print("Recall on test set: ", recall_test)
```

3.4. Evaluation Metrics for Model Performance

Description:

- **Accuracy:** Accuracy measures the proportion of correctly classified instances out of all instances in the dataset. An accuracy of 1.0 on the test set indicates that all instances in the test set were classified correctly by the model, while an accuracy of 0.722 on the training set suggests that around 72.22% of instances in the training set were classified correctly.

- **Recall:** Recall, also known as sensitivity, measures the proportion of actual positive instances that were correctly identified by the model. A recall of 1.0 on the test set indicates that all positive instances in the test set were correctly identified by the model, while a recall of 0.818 on the training set suggests that around 81.82% of positive instances in the training set were correctly identified.

In [ ]:
```
from sklearn import metrics

# Calculate and print accuracy on training and test sets
print("Accuracy on training set: ", chosen_classifier.score(X_train, y_train))
print("Accuracy on test set: ", chosen_classifier.score(X_test, y_test))

# Calculate and print recall scores on training and test sets
get_recall_score(chosen_classifier)
```

```
Accuracy on training set:  0.7222222222222222
Accuracy on test set:  1.0
Recall on training set:  0.8181818181818182
Recall on test set:  1.0
```

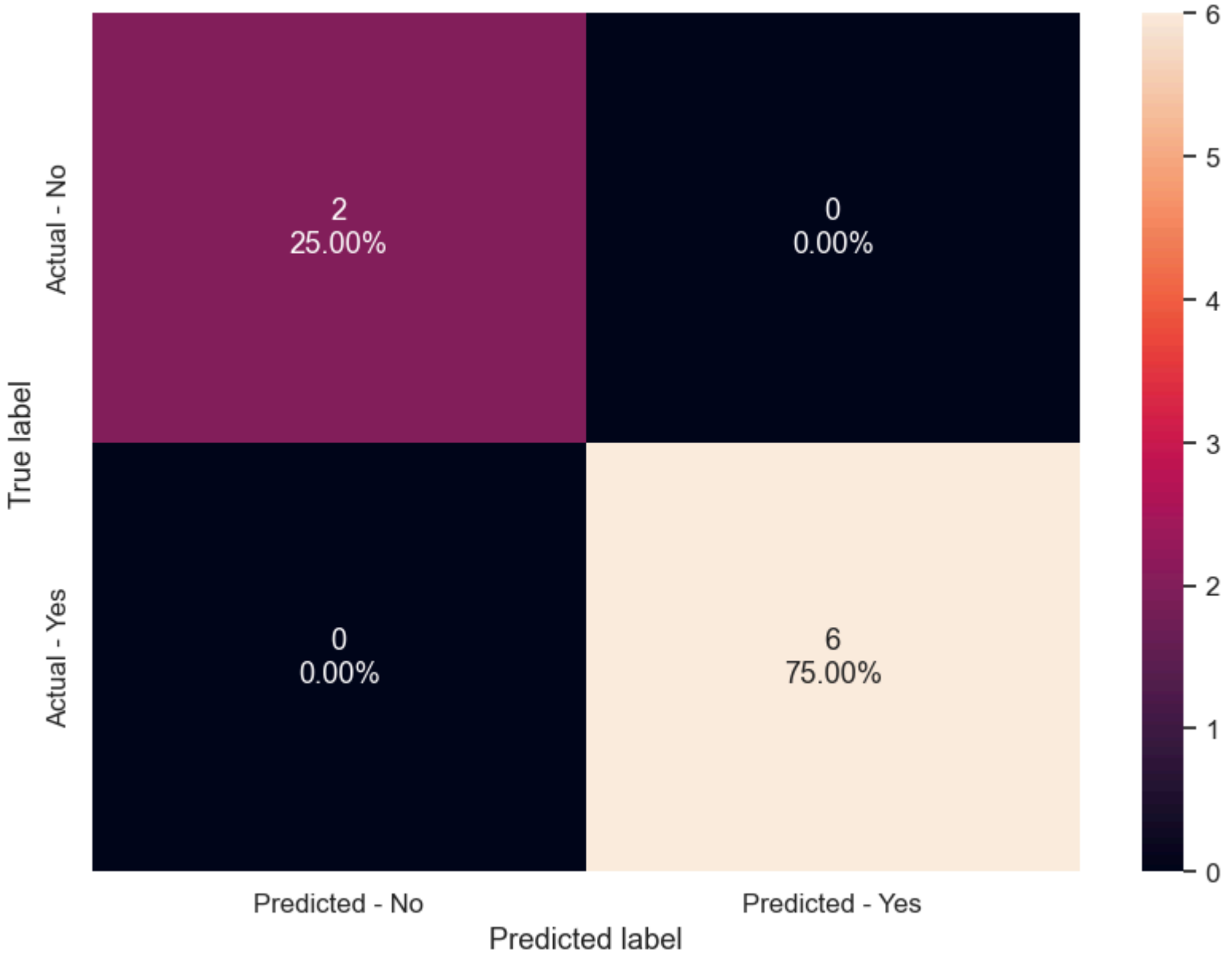
3.5. Confusion Matrix

In [ ]:
```
## Function to create confusion matrix
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
```

```
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [ ]:

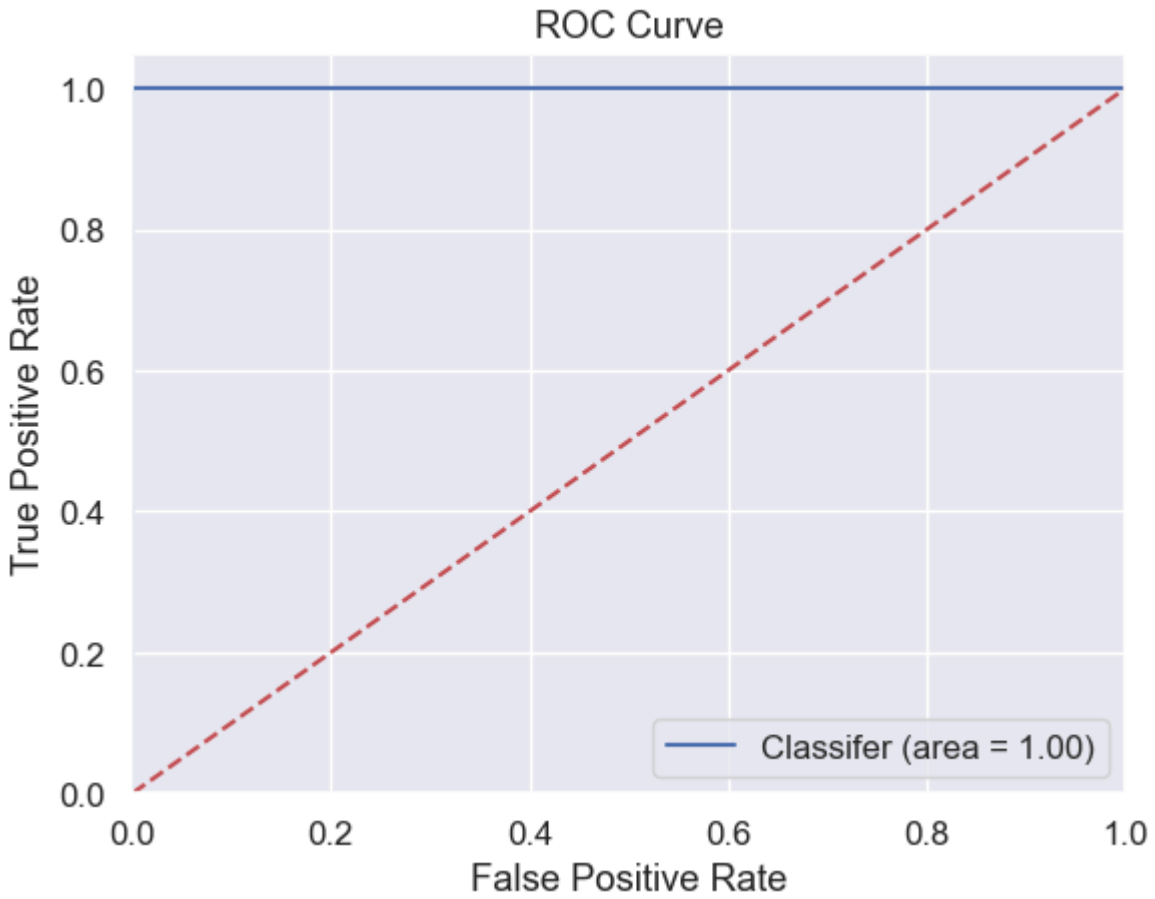
```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt

make_confusion_matrix(chosen_classifier,y_test)
```



3.6. The ROC curve

In [ ]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve #plot the curve
```

```
rf_roc_auc = roc_auc_score(y_test, chosen_classifier.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, chosen_classifier.predict_proba(X_test)[:,1])

plt.figure()
plt.plot(fpr, tpr, label='Classifer (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
# the Receiver Operating Characteristic value obtained from the validation process
print("For this validation the ROC = ", rf_roc_auc)
```

```
For this validation the ROC =  1.0
```

In [ ]:
```python
# Check the unique classes in y_true
unique_classes = np.unique(y_test)
if len(unique_classes) < 2:
    print("Error: Only one class present in y_true. Cannot calculate ROC AUC.")
else:
    # Proceed with ROC AUC calculation
    from sklearn.metrics import roc_auc_score
    from sklearn.metrics import roc_curve

    # Your code for ROC AUC calculation here
    rf_roc_auc = roc_auc_score(y_test, chosen_classifier.predict(X_test))
    fpr, tpr, thresholds = roc_curve(y_test, chosen_classifier.predict_proba(X_test)[:,1])
    plt.figure()
    # Plot ROC curve and other relevant visualizations
```

```
<Figure size 640x480 with 0 Axes>
```

Chapter 4 - Evaluation of our Model and Validation

4.1. Loading and Preprocessing Google News Data

In [ ]:
```python
import pandas as pd
import numpy as np

# Load the news data from a CSV file
data = pd.read_csv('news.csv')
final_news_df = pd.DataFrame(data)

# Remove all rows that contain NaN values in the original DataFrame
final_news_df.dropna(how='all', inplace=True)

# Reset the index for the original DataFrame
final_news_df.reset_index(drop=True, inplace=True)

# Display the first few rows of the DataFrame
final_news_df.head()
```

Out[ ]:

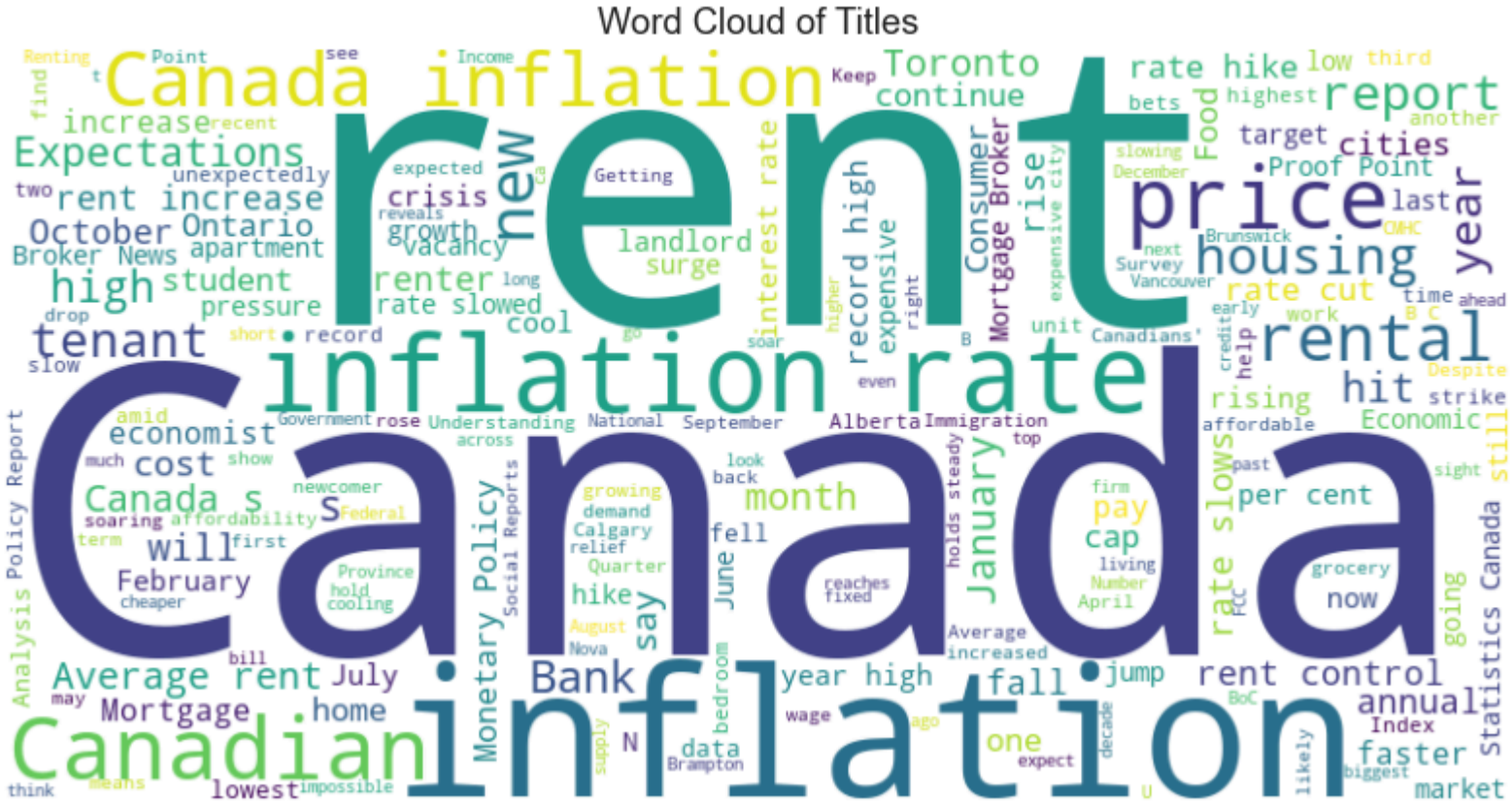
	Title	Source	Time	Link	Keywords
0	A snapshot of how inflation is affecting Canad...	Statistique Canada	Feb 27, 2023	https://news.google.com/articles/CBMiVGh0dHBzO...	Canada inflation
1	What's happening to inflation and why it matters	Bank of Canada	Oct 6, 2022	https://news.google.com/articles/CBMiVGh0dHBzO...	Canada inflation
2	Canadian Inflation: A New Vintage	TD Economics	Apr 27, 2022	https://news.google.com/articles/CBMiMWh0dHBzO...	Canada inflation
3	Canada's inflation rate slowed to 2.9% in Janu...	CBC News	Feb 20	https://news.google.com/articles/CBMiQWh0dHBzO...	Canada inflation
4	Canada inflation rate ticks down to 2.8 per cent	CP24	Mar 19	https://news.google.com/articles/CBMidWh0dHBzO...	Canada inflation

word cloud of news titles

A word cloud of news titles related to Canada's inflation provides a visual representation of the most frequently occurring words in the titles of news articles. The size of each word in the cloud corresponds to its frequency in the dataset, with larger words indicating more common occurrences.

In [ ]:

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all titles into a single string
all_titles = ' '.join(final_news_df['Title'])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_titles)

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Titles')
plt.axis('off')
plt.show()
```



word cloud of news sources

The word cloud of news sources provides a visual representation of the various sources from which data on Canada's inflation has been gathered.

In [ ]:
```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all sources into a single string
all_sources = ' '.join(final_news_df['Source'])

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all_sources)

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud of Sources')
```

```
plt.axis('off')
plt.show()
```

4.2. Preprocessing News Dates

In [ ]:

```
import pandas as pd
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta

# Import re module for regular expressions
import re

# Get the current date
current_date = pd.to_datetime('today')
current_month = current_date.month
current_year = current_date.year
```

```python
# Iterate through each row in the DataFrame
for index, row in final_news_df.iterrows():
    date_str = row['Time']

    # Check if the date string is already in the format 'YYYY-MM-DD'
    if re.match(r'^\d{4}-\d{2}-\d{2}$', date_str):
        # If so, keep the value unchanged'
        continue

    # Check if the date string represents 'X days ago'
    elif re.match(r'^\d+ days ago$', date_str):
        days_ago = int(date_str.split()[0])
        date = current_date - pd.Timedelta(days=days_ago)

    # Check if the date string represents 'X hours ago'
    elif re.match(r'^\d+ hours ago$', date_str):
        hours_ago = int(date_str.split()[0])
        date = current_date - pd.Timedelta(hours=hours_ago)

    # Check if the date string represents 'X hour ago'
    elif re.match(r'^\d+ hour ago$', date_str):
        date = current_date

    # Check if the date string represents 'X minutes ago'
    elif re.match(r'^\d+ minutes ago$', date_str):
        minutes_ago = int(date_str.split()[0])
        date = current_date - pd.Timedelta(minutes=minutes_ago)

    elif re.match(r'^[A-Z][a-z]{2} \d{1,2}, \d{4}$', date_str):
        month_str, day_str, year_str = date_str.split()

        # Convert month abbreviation to month number
        month_dict = {
            'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
            'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12
        }
        month = month_dict[month_str]

        # Convert day to integer
        day = int(day_str.strip(','))

        # Convert year to integer
        year = int(year_str)

        # Construct the date string
        date_str = f'{year}-{month:02d}-{day:02d}'
```

```
        # Convert the date string to a Timestamp object
        date = pd.to_datetime(date_str)

    # Check if the date string is "Yesterday"
    elif date_str == 'Yesterday':
        date = current_date - pd.Timedelta(days=1)

    # If the date string represents 'Month-Day'
    else:
        month_str, day_str = date_str.split(' ')

        # Convert month abbreviation to month number
        month_dict = {
            'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
            'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12
        }
        month = month_dict[month_str]

        # Convert day to integer
        day = int(day_str)

        # Determine the year based on the current month
        year = current_year - 1 if month > current_month else current_year

        # Construct the date string
        date_str = f'{year}-{month:02d}-{day:02d}'

        # Convert the date string to a Timestamp object
        date = pd.to_datetime(date_str)

    # Update the "Time" column with the formatted date
    final_news_df.at[index, 'Time'] = date.strftime('%Y-%m-%d')
```

4.3. Preprocessing and Filtering News Data

In [ ]:

```
# Convert 'Time' column to datetime format
final_news_df['Time'] = pd.to_datetime(final_news_df['Time'], format='%Y-%m-%d')

# Remove duplicate rows based on 'Title' and 'Time'
final_news_df = final_news_df.drop_duplicates(subset=['Title', 'Time'])

# Filter data to include dates starting from January 1, 2022
final_news_df = final_news_df[final_news_df['Time'] >= '2022-01-01']
```

```
# Display information about the DataFrame
print(final_news_df.info())

# Save the preprocessed DataFrame to a CSV file
final_news_df.to_csv('news2.csv', index=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 344 entries, 0 to 359
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Title     344 non-null    object
 1   Source    344 non-null    object
 2   Time      344 non-null    datetime64[ns]
 3   Link      344 non-null    object
 4   Keywords  344 non-null    object
dtypes: datetime64[ns](1), object(4)
memory usage: 16.1+ KB
None
```

4.4. Sentiment Analysis of News Data

In [ ]:
```
# install just once, after that comment the code for better performance
# %pip install textblob
```

In [ ]:
```
# Import necessary libraries
from textblob import TextBlob
import pandas as pd

# Initialize an empty list to store sentiment results
sentiment_results = []

# Group news data by month and keywords, then analyze sentiment
for (month, keyword), group in final_news_df.groupby([pd.Grouper(freq='ME', key='Time'), 'Keywords']):
    # Concatenate titles within each group
    month_keyword_text = ' '.join(group['Title'])

    # Perform sentiment analysis using TextBlob
    blob = TextBlob(month_keyword_text)
    average_sentiment = blob.sentiment.polarity
    average_subjectivity = blob.sentiment.subjectivity

    # Append sentiment results to the list
```

```
    sentiment_results.append({'Month': pd.to_datetime(month).strftime('%Y-%m'), 'Keyword': keyword, 'Average Sentiment': average_sentiment,

# Create a DataFrame from the sentiment results
sentiment_df = pd.DataFrame(sentiment_results)

# Save the sentiment DataFrame to a CSV file
sentiment_df.to_csv('sentiment_results.csv', index=False)

# Display the sentiment DataFrame
print(sentiment_df)
```

	Month	Keyword	Average Sentiment	Average Subjectivity
0	2022-01	Canada inflation	0.098788	0.464848
1	2022-02	Canada inflation	0.000000	0.000000
2	2022-02	Canada rent	0.000000	0.000000
3	2022-03	Canada inflation	0.160000	0.540000
4	2022-03	Canada rent	0.000000	0.000000
5	2022-04	Canada inflation	0.144242	0.483030
6	2022-05	Canada inflation	0.148182	0.497273
7	2022-05	Canada rent	0.166667	0.333333
8	2022-06	Canada inflation	0.173333	0.426667
9	2022-06	Canada rent	0.000000	0.000000
10	2022-07	Canada inflation	0.098788	0.531515
11	2022-08	Canada inflation	0.250000	0.333333
12	2022-08	Canada rent	0.142222	0.401111
13	2022-09	Canada inflation	0.020000	0.276667
14	2022-09	Canada rent	0.200000	0.600000
15	2022-10	Canada inflation	0.000000	0.000000
16	2022-10	Canada rent	-0.666667	1.000000
17	2022-11	Canada inflation	-0.400000	0.700000
18	2022-11	Canada rent	0.100000	0.200000
19	2022-12	Canada rent	0.316667	0.483333
20	2023-01	Canada inflation	0.016667	0.500000
21	2023-01	Canada rent	0.061364	0.397403
22	2023-02	Canada inflation	0.133333	0.383333
23	2023-02	Canada rent	-0.500000	0.700000
24	2023-03	Canada inflation	0.160000	0.540000
25	2023-03	Canada rent	0.007273	0.364242
26	2023-04	Canada inflation	-0.107500	0.582500
27	2023-04	Canada rent	0.300000	0.750000
28	2023-05	Canada inflation	0.066667	0.666667
29	2023-05	Canada rent	0.370000	0.353333
30	2023-06	Canada inflation	-0.030000	0.272500
31	2023-06	Canada rent	0.090341	0.338636
32	2023-07	Canada inflation	0.165556	0.334444
33	2023-07	Canada rent	0.003333	0.496667
34	2023-08	Canada inflation	-0.100000	0.554167
35	2023-08	Canada rent	0.185594	0.469417
36	2023-09	Canada inflation	0.287500	0.537500
37	2023-09	Canada rent	0.132843	0.460772
38	2023-10	Canada inflation	0.009778	0.623556
39	2023-10	Canada rent	-0.006145	0.508838
40	2023-11	Canada inflation	0.092778	0.421111
41	2023-11	Canada rent	0.105359	0.485231
42	2023-12	Canada inflation	0.131111	0.548889
43	2023-12	Canada rent	0.200000	0.437500
44	2024-01	Canada inflation	0.075000	0.562500

```
45  2024-01       Canada rent           0.032068             0.373311
46  2024-02  Canada inflation           0.220000             0.376667
47  2024-02       Canada rent           0.118747             0.414483
48  2024-03  Canada inflation           0.095741             0.509815
49  2024-03       Canada rent          -0.173333             0.540000
50  2024-04       Canada rent           0.000000             0.000000
```

In [ ]:
```python
sentiment_df.info()
```

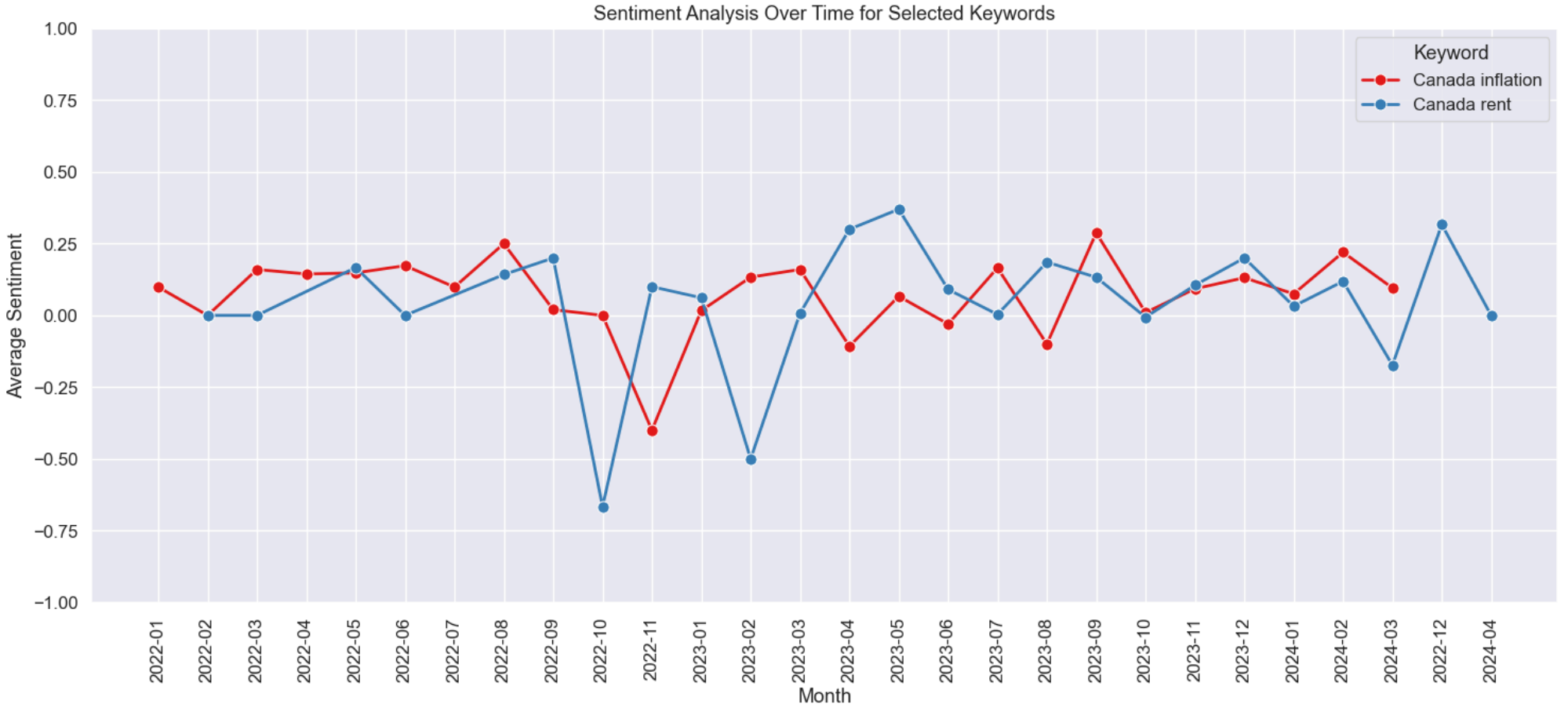
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Month                 51 non-null     object 
 1   Keyword               51 non-null     object 
 2   Average Sentiment     51 non-null     float64
 3   Average Subjectivity  51 non-null     float64
dtypes: float64(2), object(2)
memory usage: 1.7+ KB
```

4.5. Sentiment Analysis of Selected Keywords Over Time and Subjectivity

4.5.1. Sentiment Analysis of Selected Keywords Over Time

In [ ]:
```python
# Importing necessary libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Define the keywords to consider
keywords_to_consider = ['Canada inflation', 'Canada rent']

# Filter sentiment_df to include only specified keywords
filtered_sentiment_df = sentiment_df[sentiment_df['Keyword'].isin(keywords_to_consider)]

# Plot sentiment analysis over time for selected keywords
plt.figure(figsize=(15,7))
plt.title('Sentiment Analysis Over Time for Selected Keywords')

# Use a distinct color palette
colors = sns.color_palette('Set1', n_colors=len(keywords_to_consider))

# Plot each keyword separately with different line styles and markers
for i, keyword in enumerate(keywords_to_consider):
```

```
    sns.lineplot(data=filtered_sentiment_df[filtered_sentiment_df['Keyword'] == keyword],
                 x='Month', y='Average Sentiment',
                 linewidth=2, linestyle='-', marker='o',
                 markersize=8, color=colors[i], label=keyword)

plt.xticks(rotation=90)
plt.ylim(-1, 1)
plt.xlabel('Month')
plt.ylabel('Average Sentiment')
plt.grid(True)
plt.legend(title='Keyword', loc='upper right')
plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```

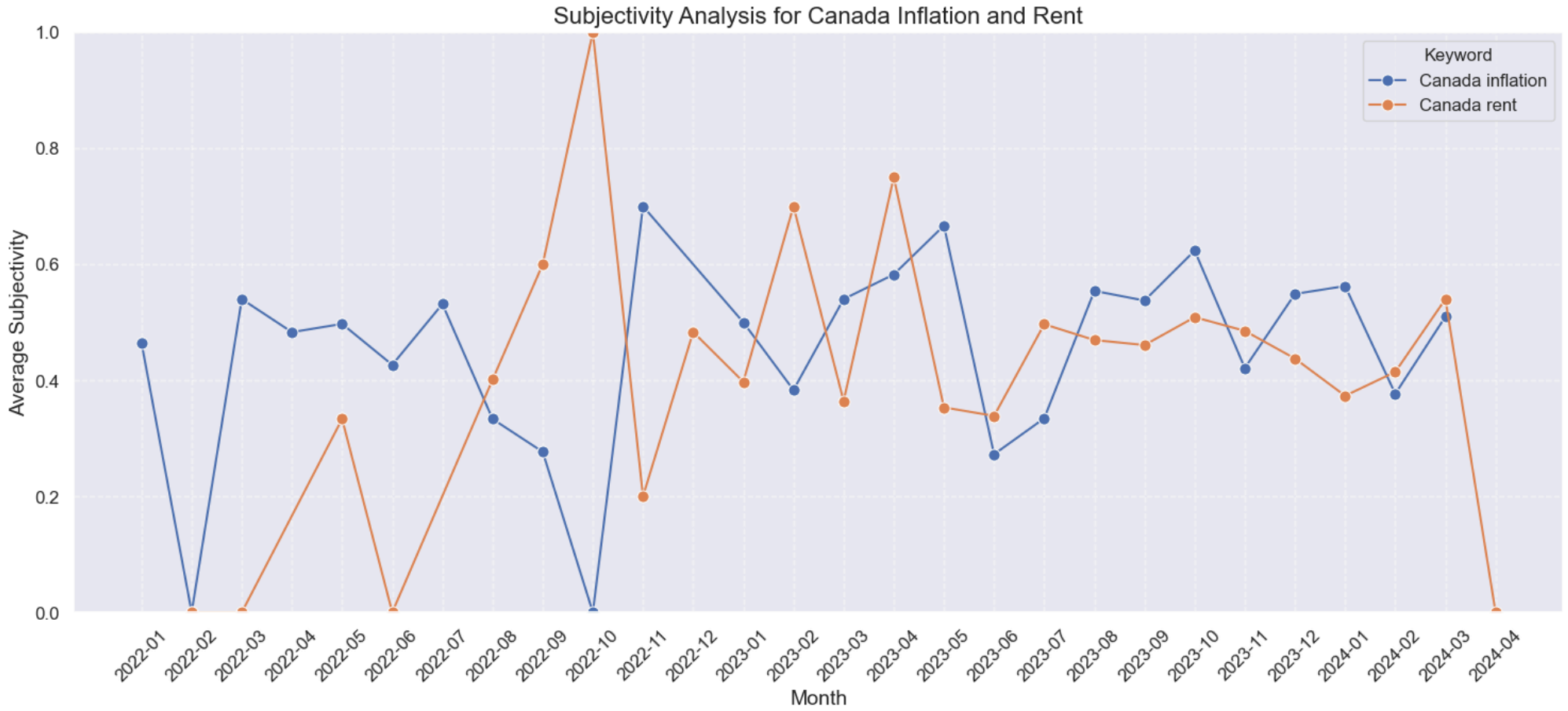


4.5.2. Sentiment Analysis of Subjectivity

In [ ]:

```python
# Libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Define the keywords of interest
keywords_to_consider = ['Canada inflation', 'Canada rent']

# Filter the sentiment dataframe for specified keywords
filtered_sentiment_df = sentiment_df[sentiment_df['Keyword'].isin(keywords_to_consider)]

# Set the figure size and title
plt.figure(figsize=(15, 7))
plt.title('Subjectivity Analysis for Canada Inflation and Rent', fontsize=16)

# Plot the subjectivity analysis using seaborn lineplot
sns.lineplot(data=filtered_sentiment_df, x='Month', y='Average Subjectivity', hue='Keyword', marker='o', markersize=8)

# Rotate x-axis tick labels for better readability
plt.xticks(rotation=45)

# Set y-axis limit
plt.ylim(0, 1)

# Set axis labels
plt.xlabel('Month', fontsize=14)
plt.ylabel('Average Subjectivity', fontsize=14)

# Add grid lines for better readability
plt.grid(True, linestyle='--', alpha=0.5)

# Adjust legend position
plt.legend(title='Keyword', fontsize=12, title_fontsize=12, loc='upper right')

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

```
c:\Users\yeiso\AppData\Local\Programs\Python\Python312\Lib\site-packages\seaborn\_base.py:948: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
c:\Users\yeiso\AppData\Local\Programs\Python\Python312\Lib\site-packages\seaborn\_base.py:948: FutureWarning: When grouping with a length-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



4.6. Importing and Preprocessing Inflation Data

In [ ]:

```
import pandas as pd

# Read the inflation data from CSV file
data = pd.read_csv('1810000401_Inflation_Basket2_new.csv')
df = pd.DataFrame(data)

# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Filter data for dates starting from January 2022
df = df[df['Date'] >= '2022-01-01']

# Sort the DataFrame by date
```

```
df.sort_values(by='Date', inplace=True)

# Remove rows with NaN values
df.dropna(how='all', inplace=True)

# Reset the index of the DataFrame
df.reset_index(drop=True, inplace=True)

# Display the first few rows of the DataFrame
df.head(3)
```

Out[ ]:

	Date	Inflation_Change	CPI	Food 5	Shelter 6	Household operations, furnishings and equipment	Clothing and footwear	Transportation	Health and personal care	Recreation, education and reading	Alcoholic beverages, tobacco products and recreational cannabis	Interest Rate
0	2022-01-01	0	0.051	163.9	157.6	126.7	93.4	156.9	134.1	119.2	178.2	0.50
1	2022-02-01	0	0.057	166.0	158.6	127.6	93.8	159.9	134.6	120.9	178.5	0.50
2	2022-03-01	0	0.067	167.5	160.2	129.1	94.5	165.5	134.9	123.1	179.3	0.75

4.7. Preprocessing Sentiment Results

In [ ]:

```
import pandas as pd

# Read sentiment results from CSV file
data = pd.read_csv('sentiment_results.csv')
sentiment = pd.DataFrame(data)

# Remove rows with NaN values
sentiment.dropna(how='all', inplace=True)

# Reset the index of the DataFrame
sentiment.reset_index(drop=True, inplace=True)

# Display the first few rows of the DataFrame
sentiment.head()
```

Out[ ]:

	Month	Keyword	Average Sentiment	Average Subjectivity
0	2022-01	Canada inflation	0.098788	0.464848
1	2022-02	Canada inflation	0.000000	0.000000
2	2022-02	Canada rent	0.000000	0.000000
3	2022-03	Canada inflation	0.160000	0.540000
4	2022-03	Canada rent	0.000000	0.000000

4.8. Transforming Sentiment Data

In [ ]:

```
import pandas as pd

# Convert 'Month' column to datetime format
sentiment['Month'] = pd.to_datetime(sentiment['Month'], format='%Y-%m', errors='coerce')

# Display information about the transformed DataFrame
print(sentiment.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 4 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   Month                 51 non-null     datetime64[ns]
 1   Keyword               51 non-null     object        
 2   Average Sentiment     51 non-null     float64       
 3   Average Subjectivity  51 non-null     float64       
dtypes: datetime64[ns](1), float64(2), object(1)
memory usage: 1.7+ KB
None
```

4.9. Sentiment Data with Pivot

In [ ]:

```
import pandas as pd

# Assuming you already have the 'sentiment' DataFrame with the required columns

# Pivot the DataFrame to create separate columns for each type of record in "Keyword"
pivot_df = sentiment.pivot(index='Month', columns='Keyword', values=['Average Sentiment', 'Average Subjectivity'])
```

```
# Flatten the multi-level column index
pivot_df.columns = [' '.join(col).strip() for col in pivot_df.columns.values]

# Reset the index to have a sequential index and add a "Month" column
pivot_df.reset_index(inplace=True)
pivot_df['Month'] = pd.to_datetime(pivot_df['Month'])  # Convert 'Month' back to datetime if needed

# Display the updated DataFrame information
print(pivot_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 5 columns):
 #   Column                                Non-Null Count  Dtype         
---  ------                                --------------  -----         
 0   Month                                 28 non-null     datetime64[ns]
 1   Average Sentiment Canada inflation    26 non-null     float64       
 2   Average Sentiment Canada rent         25 non-null     float64       
 3   Average Subjectivity Canada inflation  26 non-null     float64       
 4   Average Subjectivity Canada rent      25 non-null     float64       
dtypes: datetime64[ns](1), float64(4)
memory usage: 1.2 KB
None
```

4.10. Merging DataFrames and Handling Missing Values

In [ ]:

```
import pandas as pd

# Merging the 'df' DataFrame with the 'pivot_df' DataFrame based on common dates
merged_df = pd.merge(df, pivot_df, left_on='Date', right_on='Month', how='inner')

# Dropping rows with missing values
merged_df.dropna(inplace=True)

# Resetting the index of the merged DataFrame
merged_df.reset_index(drop=True, inplace=True)

# Displaying the head of the merged DataFrame
merged_df.head(3)
```

Out[ ]:

	Date	Inflation_Change	CPI	Food 5	Shelter 6	Household operations, furnishings and equipment	Clothing and footwear	Transportation	Health and personal care	Recreation, education and reading	Alcoholic beverages, tobacco products and recreational cannabis	Interest Rate	Month	Average Sentiment Canada inflation	Average Sentiment Canada rent
0	2022-02-01	0	0.057	166.0	158.6	127.6	93.8	159.9	134.6	120.9	178.5	0.50	2022-02-01	0.000000	0.000000
1	2022-03-01	0	0.067	167.5	160.2	129.1	94.5	165.5	134.9	123.1	179.3	0.75	2022-03-01	0.160000	0.000000
2	2022-05-01	0	0.077	170.4	163.0	131.5	97.2	172.2	137.2	123.9	180.4	1.25	2022-05-01	0.148182	0.166667

In [ ]:
```
merged_df.columns
```

Out[ ]:
```
Index(['Date', 'Inflation_Change', 'CPI', 'Food 5', 'Shelter 6',
       'Household operations, furnishings and equipment',
       'Clothing and footwear', 'Transportation', 'Health and personal care',
       'Recreation, education and reading',
       'Alcoholic beverages, tobacco products and recreational cannabis',
       'Interest Rate', 'Month', 'Average Sentiment Canada inflation',
       'Average Sentiment Canada rent',
       'Average Subjectivity Canada inflation',
       'Average Subjectivity Canada rent'],
      dtype='object')
```

4.11. Removing Unnecessary Columns and Updating Feature and Response Data

In [ ]:
```
# Dropping unnecessary columns from the merged DataFrame
merged_df = merged_df.drop(['Date', 'CPI', 'Month'], axis=1)

# Extracting features and response variables from the updated DataFrame
features_new = merged_df.drop(['Inflation_Change'], axis=1)
response_new = merged_df['Inflation_Change']

# Displaying the shape of the feature data
features_new.shape
```

Out[ ]:
```
(22, 13)
```

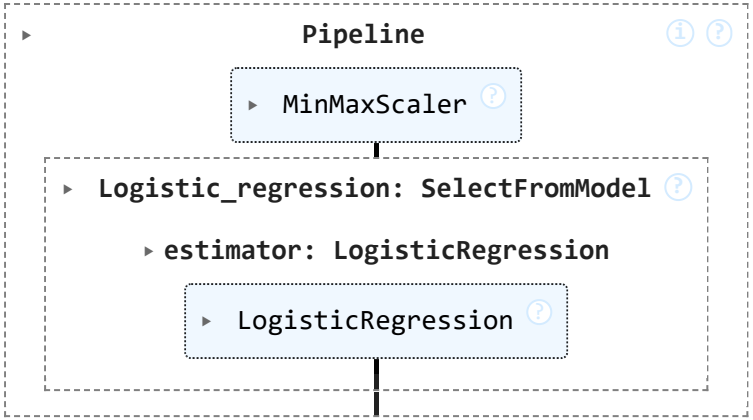
In [ ]:
```
features_new.columns
```

Out[ ]:
```
Index(['Food 5', 'Shelter 6',
       'Household operations, furnishings and equipment',
       'Clothing and footwear', 'Transportation', 'Health and personal care',
       'Recreation, education and reading',
       'Alcoholic beverages, tobacco products and recreational cannabis',
       'Interest Rate', 'Average Sentiment Canada inflation',
       'Average Sentiment Canada rent',
       'Average Subjectivity Canada inflation',
       'Average Subjectivity Canada rent'],
      dtype='object')
```

4.12. Creating Pipeline for Feature Scaling, Feature Selection, and Logistic Regression

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

# Define the pipeline with MinMaxScaler, SelectFromModel, and LogisticRegression
pipe_lr_new = Pipeline([
    ('MinMax_scaler', MinMaxScaler()),  # Scaling features to a specified range using MinMaxScaler
    ('Logistic_regression', SelectFromModel(estimator=LogisticRegression(max_iter=5000)))  # Selecting features using logistic regression
])

# Fit the pipeline to the data
pipe_lr_new.fit(features_new, response_new)
```

Out[ ]:



In [ ]:
```
# saving the selected feautures with transformations
X_1_new= pipe_lr_new.transform(features_new)
X_1_new.shape
```

Out[ ]:
```
(22, 6)
```

In [ ]:
```
#Get the features selected
features_selected_lr_new = pipe_lr_new['Logistic_regression'].get_support(indices=True)

selected_feature_names_lr_new = features_new.columns[features_selected_lr_new]
#Create a new dataframe with features selected
df_features_selected_lr_new = pd.DataFrame(X_1_new, columns=selected_feature_names_lr_new)
#df_features_selected_lr.head()
df_features_selected_lr_new.columns
```

Out[ ]:
```
Index(['Food 5', 'Household operations, furnishings and equipment',
       'Transportation', 'Health and personal care', 'Interest Rate',
       'Average Subjectivity Canada rent'],
      dtype='object')
```

4.13. Splitting the dataset into training and testing sets

In [ ]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets with 70% for training and 30% for testing
X_train, X_test, y_train, y_test = train_test_split(X_1_new, response_new, test_size=0.30, random_state=42)
```

4.14. Model Selection

In [ ]:
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Define a list of classifiers to evaluate
classifiers = [('Naive Bayes', GaussianNB()),
               ('Logistic Regression', LogisticRegression()),
               ('Random Forest', RandomForestClassifier(max_depth=3, n_estimators=1000)),
               ('Decision Tree', DecisionTreeClassifier(max_depth=3, random_state=0)),
               ('SVM Linear', SVC(kernel='linear', C=0.025, probability=True)),
               ('SVM', SVC(gamma=best_gamma, C=best_C, probability=True, random_state=0))]

# Initialize lists to store classifier performance metrics
list_accuracy = []
best_classifier = 0
chosen_classifier = 0

# Define cross-validation parameters
kf = KFold(n_splits=3, shuffle=True, random_state=42)

# Loop through each classifier, perform cross-validation, and record mean accuracy
for name, clf in classifiers:
    score = cross_val_score(clf, X_train, y_train, cv=kf, scoring='accuracy')
    mean_score = np.mean(score)
    list_accuracy.append((name, mean_score))

    # Update the best classifier if a higher mean accuracy is found
    if best_classifier < mean_score:
        best_classifier = mean_score
        chosen_classifier = clf

# Convert accuracy results to a DataFrame for easy visualization
results_df = pd.DataFrame(list_accuracy, columns=['Name', 'Mean Accuracy'])
results_df
```

Out[ ]:

	Name	Mean Accuracy
0	Naive Bayes	0.800000
1	Logistic Regression	0.733333
2	Random Forest	0.800000
3	Decision Tree	0.800000
4	SVM Linear	0.733333
5	SVM	0.800000

4.15. Best fitted Model Selection

In [ ]:

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

#List of classifier
classifiers= [('Naive_Bayes', GaussianNB()),
              ('Logistic_Regression',LogisticRegression()),
              ('Random_Forest', RandomForestClassifier(max_depth=3, n_estimators=1000)),
              ('Decision_Tree', DecisionTreeClassifier(max_depth=3, random_state=0)),
              ('svc_linear', SVC(kernel='linear', C=0.025, probability=True)),
              ('svm',SVC(gamma=best_gamma, C=best_C, probability=True, random_state=0))
             ]

clf_list=[]
list_accuracy=[]
kf = KFold(n_splits=3, shuffle=True, random_state=42)
best_classifier=0

clf_list=[]
list_accuracy=[]
kf = KFold(n_splits=3, shuffle=True, random_state=42)
best_classifier=0
chosen_classifier=0

for n, clf in  classifiers:
    score = cross_val_score(clf, X_train, y_train, cv=kf, scoring='accuracy')
    mean_score = np.mean(score)
```

```
    list_accuracy.append((n, mean_score))

    if best_classifier < mean_score:
        best_classifier = mean_score
        chosen_classifier_new = clf

results_new = np.array(list_accuracy)
results_df_new = pd.DataFrame(results_new, columns=['Name', 'MeanScore'])
results_df_new
```

Out[ ]:

	Name	MeanScore
0	Naive_Bayes	0.8000000000000002
1	Logistic_Regression	0.7333333333333334
2	Random_Forest	0.7999999999999999
3	Decision_Tree	0.7999999999999999
4	svc_linear	0.7333333333333334
5	svm	0.8000000000000002

4.16. Model Evaluation

In [ ]:
```
chosen_classifier_new.fit(X_train, y_train)
```

Out[ ]:

GaussianNB

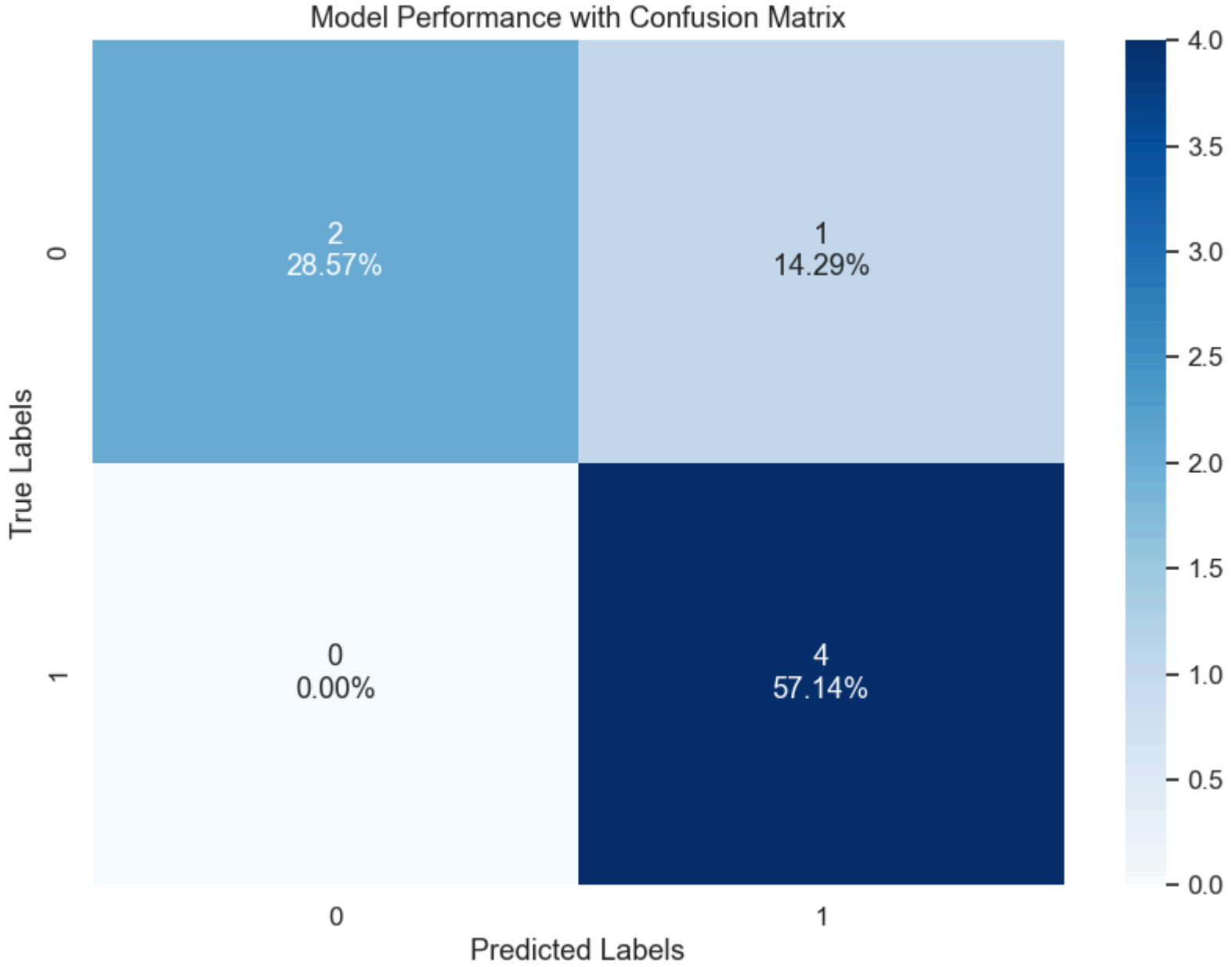
GaussianNB()

In [ ]:
```
# Print accuracy on training and test sets
print("Accuracy on training set: ", chosen_classifier_new.score(X_train, y_train))
print("Accuracy on test set: ", chosen_classifier_new.score(X_test, y_test))

# Print recall scores on training and test sets
get_recall_score(chosen_classifier_new)
```

```
Accuracy on training set:  0.8
Accuracy on test set:  0.8571428571428571
Recall on training set:  0.9090909090909091
Recall on test set:  1.0
```

4.17. Visualizing Model Performance with Confusion Matrix

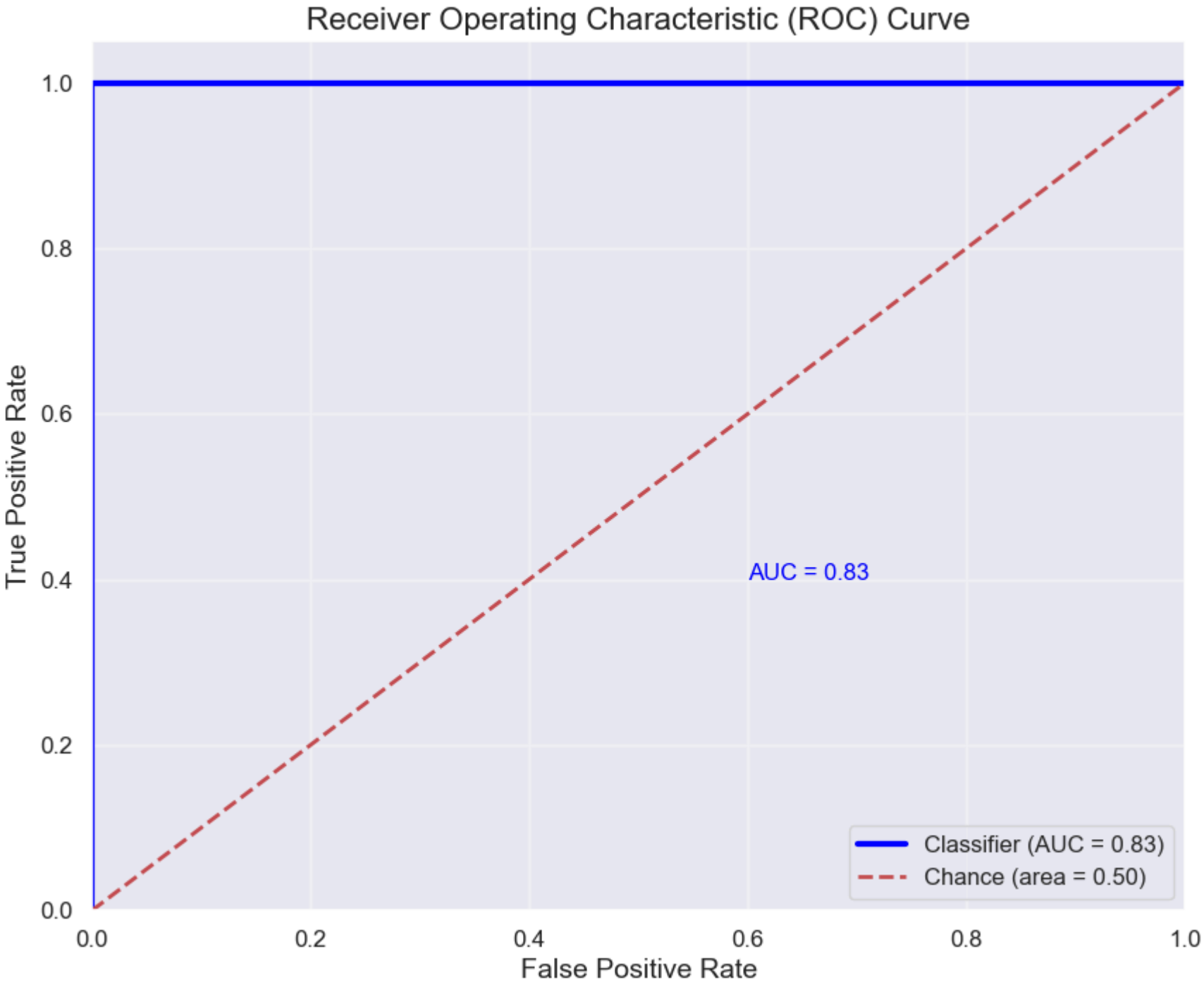
In [ ]:
```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Function to create and visualize a confusion matrix
def make_confusion_matrix(classifier, y_true):
    # Generate predictions using the classifier
    y_pred = classifier.predict(X_test)

    # Calculate the confusion matrix
    cf_matrix = metrics.confusion_matrix(y_true, y_pred)

    # Create labels for the matrix with counts and percentages
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten()/np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(cf_matrix.shape[0], cf_matrix.shape[1])

    # Set the size of the plot
    plt.figure(figsize=(10,7))

    # Create a heatmap with the confusion matrix, labels, and a color map
    sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

    # Add a title and axis labels
    plt.title('Model Performance with Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')

    # Display the plot
    plt.show()

# Call the function with the classifier and the test labels
make_confusion_matrix(chosen_classifier_new, y_test)
```



4.18. The ROC curve indicator

In [ ]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
```

```python
import matplotlib.pyplot as plt

# Calculate the AUC (Area Under the Curve) for the ROC curve
# This metric is used to evaluate the performance of a binary classifier.
rf_roc_auc = roc_auc_score(y_test, chosen_classifier_new.predict(X_test))

# Generate the data points for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, chosen_classifier_new.predict_proba(X_test)[:,1])

# Initialize the plot with larger figure size
plt.figure(figsize=(10, 8))

# Plot the ROC curve with thicker line
plt.plot(fpr, tpr, label=f'Classifier (AUC = {rf_roc_auc:.2f})', color='blue', lw=3)

# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'r--', label='Chance (area = 0.50)', lw=2)

# Display the AUC value on the plot
plt.text(0.6, 0.4, f'AUC = {rf_roc_auc:.2f}', fontsize=12, color='blue')

# Customize the plot
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=16)
plt.legend(loc="lower right", fontsize=12)
plt.grid(True, alpha=0.3)

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


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.83
Classifier (AUC = 0.83)
Chance (area = 0.50)

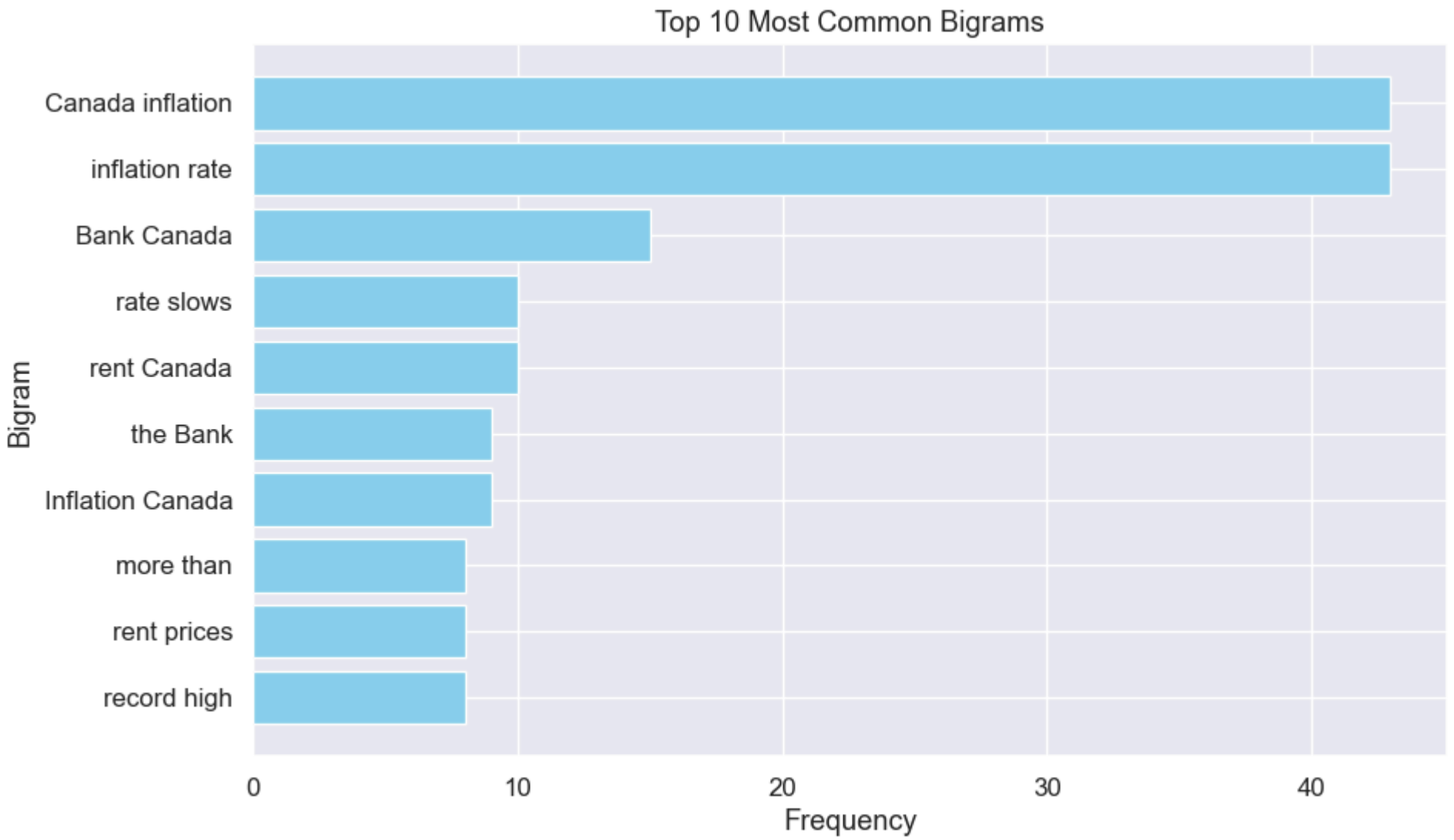
4.19. The 'Top 10' Bigrams in the "news2" file:

This is an indicator of the top 10 frequently occurring pairs of consecutive words in the "news2" file. These pairs, known as bigrams, provide insight into common word combinations or phrases found within the titles of news articles. Analyzing bigrams can offer valuable information about prevalent topics, themes, or language patterns present in the dataset.

In [ ]:
```
# install just once, after that comment the code for better performance

# %pip install nltk
# import nltk
# nltk.download('punkt')
```

In [ ]:
```
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.util import bigrams
import matplotlib.pyplot as plt
from collections import Counter

# Read the CSV file into a pandas DataFrame
df = pd.read_csv('news2.csv')

# Tokenize the text data into words
tokenized_text = df['Title'].apply(word_tokenize)

# Generate bigrams from the tokenized words, omitting bigrams with words less than 3 letters
bigram_list = []
for text in tokenized_text:
    filtered_text = [word for word in text if len(word) >= 3]  # Filter out words less than 3 letters
    bigram_list.extend(list(bigrams(filtered_text)))

# Count the frequency of each bigram
bigram_counts = Counter(bigram_list)

# Get the top 10 most common bigrams
top_10_bigrams = bigram_counts.most_common(10)

# Extract bigram words and their frequencies
bigram_words = [bigram[0][0] + ' ' + bigram[0][1] for bigram in top_10_bigrams]
bigram_frequencies = [freq for _, freq in top_10_bigrams]

# Plot the top 10 most common bigrams
plt.figure(figsize=(10, 6))
```

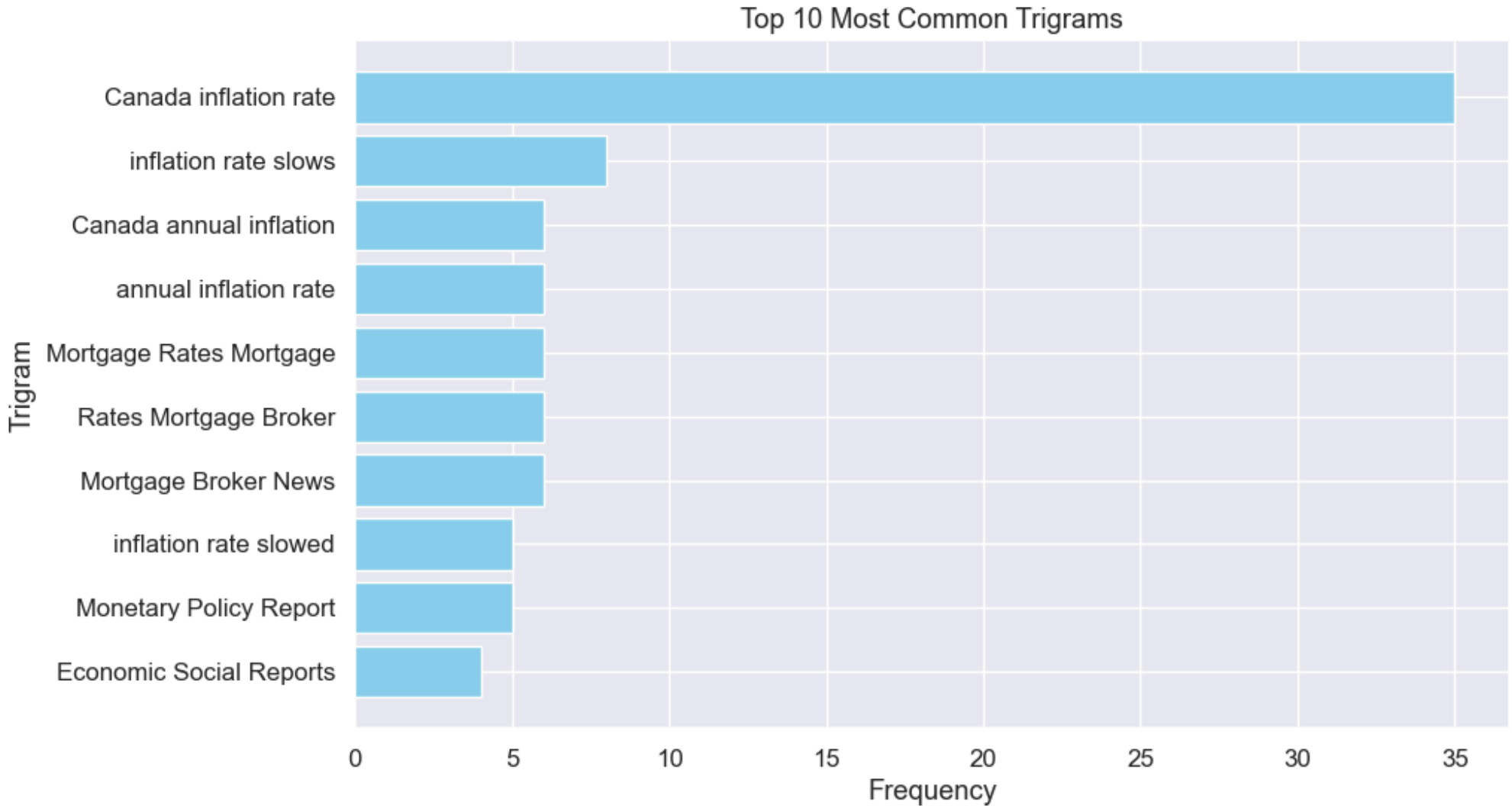
```
plt.barh(bigram_words, bigram_frequencies, color='skyblue')
plt.xlabel('Frequency')
plt.ylabel('Bigram')
plt.title('Top 10 Most Common Bigrams')
plt.gca().invert_yaxis()  # Invert y-axis to display the most common bigrams at the top
plt.show()
```



4.20. The 'Top 10' Trygrams in the "news2" file:

This highlights the top 10 frequently occurring triplets of consecutive words in the "news2" file. These triplets, referred to as trigrams, offer insights into recurring word sequences or phrases observed within the titles of news articles. Analyzing trigrams enables the identification of prevalent topics, recurring themes, or linguistic patterns prevalent in the dataset.

In [ ]:
```
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.util import ngrams
import matplotlib.pyplot as plt
from collections import Counter

# Read the CSV file into a pandas DataFrame
df = pd.read_csv('news2.csv')

# Tokenize the text data into words
tokenized_text = df['Title'].apply(word_tokenize)

# Generate trigrams from the tokenized words, omitting words less than 4 letters
trigram_list = []
for text in tokenized_text:
    filtered_text = [word for word in text if len(word) >= 4]  # Filter out words less than 4 letters
    trigram_list.extend(list(ngrams(filtered_text, 3)))

# Count the frequency of each trigram
trigram_counts = Counter(trigram_list)

# Get the top 10 most common trigrams
top_10_trigrams = trigram_counts.most_common(10)

# Extract trigram words and their frequencies
trigram_words = [' '.join(trigram[0]) for trigram in top_10_trigrams]
trigram_frequencies = [freq for _, freq in top_10_trigrams]

# Plot the top 10 most common trigrams
plt.figure(figsize=(10, 6))
plt.barh(trigram_words, trigram_frequencies, color='skyblue')
plt.xlabel('Frequency')
plt.ylabel('Trigram')
plt.title('Top 10 Most Common Trigrams')
plt.gca().invert_yaxis()  # Invert y-axis to display the most common trigrams at the top
plt.show()
```



4.21. The 'Top 10' most common words in the "news2" file:

This analysis reveals the top 10 most frequently occurring words found in the titles of news articles within the "news2" file. By tokenizing the text data and counting the frequency of each word, this approach identifies the words that appear most often across the dataset. Understanding the prevalence of these words provides insights into the primary topics, subjects, or language usage within the news articles.

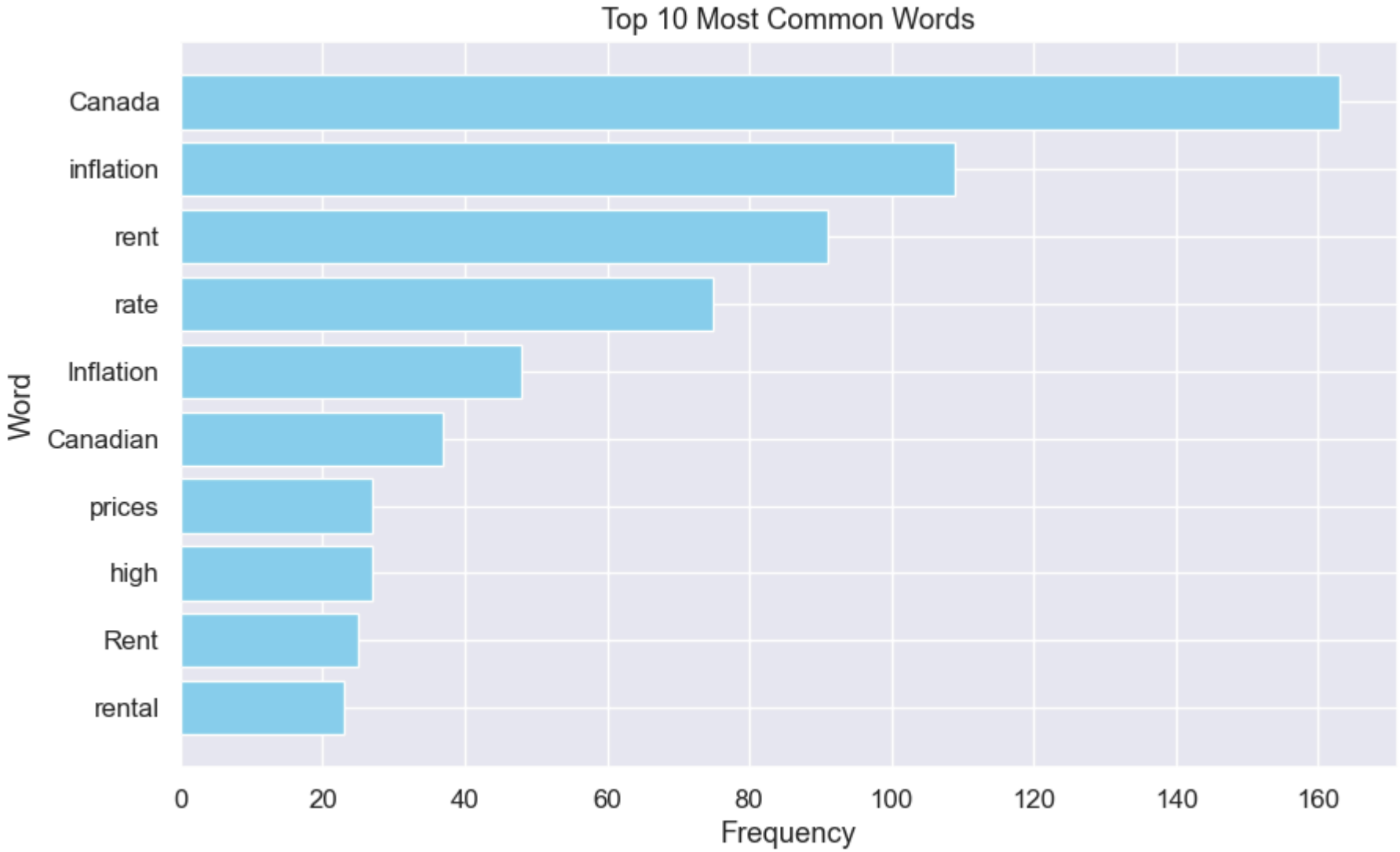
In [ ]:
```
from collections import Counter
from nltk.tokenize import word_tokenize
```

```
# Tokenize the text data into words
tokenized_text = df['Title'].apply(word_tokenize)

# Flatten the list of tokenized words and filter out words less than four letters
words = [word for sublist in tokenized_text for word in sublist if len(word) >= 4]

# Count the frequency of each word
word_freq = Counter(words)

# Get the top 10 most common words
top_10_words = word_freq.most_common(10)

# Sort the top 10 words from major to minor in counting
top_10_words_sorted = sorted(top_10_words, key=lambda x: x[1], reverse=True)

# Display the sorted top 10 most common words
print("Top 10 Most Common Words:")
for word, freq in top_10_words_sorted:
    print(f"{word}: {freq}")
```

```
Top 10 Most Common Words:
Canada: 163
inflation: 109
rent: 91
rate: 75
Inflation: 48
Canadian: 37
prices: 27
high: 27
Rent: 25
rental: 23
```

In [ ]:

```
import matplotlib.pyplot as plt

# Extract words and frequencies from the sorted list
words_sorted = [word for word, _ in top_10_words_sorted]
frequencies_sorted = [freq for _, freq in top_10_words_sorted]

# Plot the top 10 most common words
plt.figure(figsize=(10, 6))
plt.barh(words_sorted, frequencies_sorted, color='skyblue')
plt.xlabel('Frequency')
plt.ylabel('Word')
plt.title('Top 10 Most Common Words')
plt.gca().invert_yaxis()  # Invert y-axis to display the most common words at the top
plt.show()
```

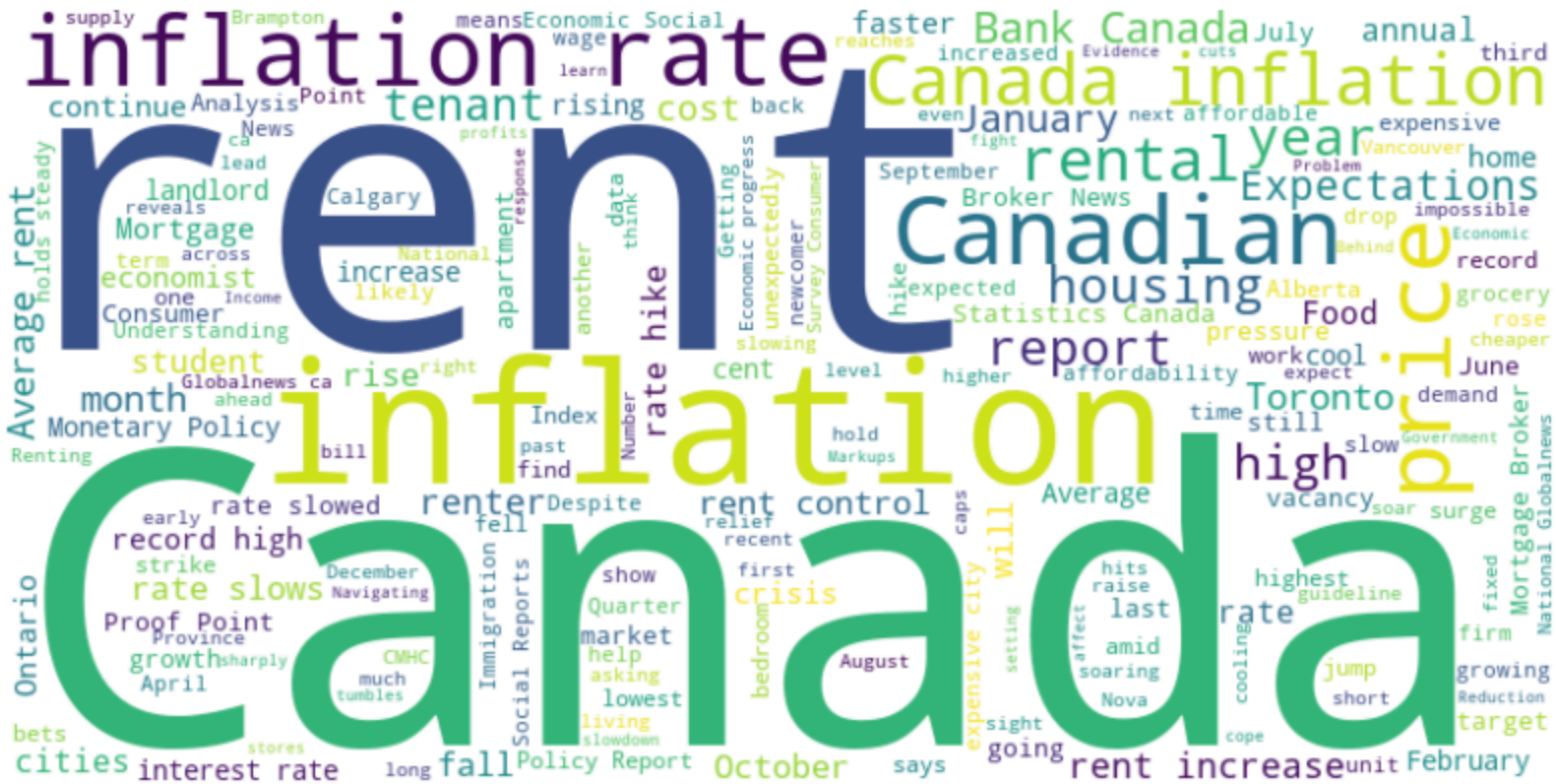


4.22. The word cloud in the "news2" file:

This visualization presents a word cloud generated from the titles of news articles within the "news2" dataset. The word cloud visually represents the frequency of words, with larger words indicating higher frequency. By aggregating and displaying the most common words in a visually appealing manner, the word cloud offers a quick overview of the prominent themes or topics present in the news articles.

In [ ]:
```
# Word Cloud Visualization
from wordcloud import WordCloud
```

```
# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(words))

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



4.23. Named Entity Recognition (NER) Analysis for the "news2" file:

This analysis utilizes spaCy's NER model to identify named entities (such as persons, organizations, locations, etc.) within the titles of news articles in the "news2" dataset. The code extracts and displays these named entities along with their corresponding labels, providing insights into the specific entities mentioned and their respective categories or types.

In [ ]:
```
# install just once, after that comment the code for better performance
# !python -m spacy download en_core_web_sm
```

In [ ]:
```
# Named Entity Recognition (NER)
import spacy
import re

# Load the English language model for spaCy
nlp = spacy.load('en_core_web_sm')

# Define a regular expression pattern to match numbers and percentages
number_pattern = re.compile(r'^-?\d*\.?\d+%?$')

# Apply named entity recognition to each title
ner_results = []
for title in df['Title']:
    doc = nlp(title)
    for ent in doc.ents:
        # Filter out numbers, percentages, and words with less than 4 letters
        if not number_pattern.match(ent.text) and len(ent.text) >= 4:
            ner_results.append((ent.text, ent.label_))

# Display the filtered named entities and their labels
print("Named Entities and Labels:")
for ent, label in ner_results:
    print(f"{ent}: {label}")
```

Named Entities and Labels:
Canadians - Statistics Canada: ORG
Canadian: NORP
Canada: GPE
January: DATE
Canada: GPE
2.8 per cent: MONEY
Canada: GPE
February: DATE
2nd month in a row: DATE
Canada: GPE
2017 to 2022: DATE
Canada: GPE
Canada: GPE
31-year: DATE
Shelter Inflation Problem: ORG
Canadian: NORP
The Bank of Canada's: ORG
the Inflation Reduction Act: LAW
Canada: GPE
Canada: GPE
Canada: GPE
October: DATE
Canada: GPE
the 1980s: DATE
1990s: DATE
Canada: GPE
Canada: GPE
January: DATE
Canada: GPE
30-year: DATE
Canada: GPE
last month: DATE
39-year: DATE
Understanding Inflation Dynamics: ORG
The Role of Government Expenditures: ORG
the Bank of Canada’s: ORG
Canada: GPE
almost 2 years: DATE
Canada: GPE
October 2023: DATE
Canada: GPE
June: DATE
April: DATE
up to 4.4%: PERCENT
September: DATE

the Bank of Canada: ORG
Canada: GPE
40 years ago: DATE
today: DATE
Globalnews.ca: PRODUCT
Canadian: NORP
the Bank of Canada: ORG
Canadian: NORP
COVID-19: ORG
January 2024: DATE
Canada: GPE
April: DATE
Canadians' Affordability Concerns Remain Acute: ORG
Canada: GPE
annual: DATE
December: DATE
StatCan: ORG
Canada: GPE
January: DATE
the 1980s: DATE
1990s: DATE
Bellemare: PERSON
Canada: GPE
the Bank of Canada’s: ORG
Canadian: NORP
Canada: GPE
March: DATE
Canada: GPE
December: DATE
Canada: GPE
November: DATE
Canada: GPE
Canada: GPE
Canada: GPE
Canadian: NORP
Canadian: NORP
Third Quarter of 2023: DATE
Canadians: NORP
Canada: GPE
February: DATE
June: DATE
Canada: GPE
Canada: GPE
annual: DATE
3.8 per cent: MONEY
September: DATE

Canada: GPE
Canada: GPE
annual: DATE
January: DATE
April 2023: DATE
Canadian: NORP
Canadians: NORP
Canada: GPE
Canada: GPE
January: DATE
Canada: GPE
Canada: GPE
four per cent: MONEY
August: DATE
StatCan: ORG
Canadian: NORP
October: DATE
Monetary Policy Report Press Conference Opening Statement: ORG
Canada: GPE
31-year: DATE
Canada: GPE
first: ORDINAL
June 2021: DATE
VIEW Canada's: ORG
annual: DATE
January: DATE
October 2023: DATE
VIEW Canada's: ORG
Canada: GPE
March: DATE
July 2023: DATE
Canada: GPE
3.4 per cent: MONEY
December: DATE
the Bank of Canada: ORG
Canadian: NORP
Canada: GPE
27-month: DATE
Canada: GPE
Governing Council: ORG
October 25, 2023: DATE
Canada: GPE
the Bank of Canada: ORG
Canada: GPE
February: DATE
Canada: GPE

January: DATE
Canada: GPE
September: DATE
Canada: GPE
January: DATE
Canadian: NORP
July: DATE
Bank of Canada: ORG
Canada: GPE
Bank of Canada: ORG
the Bank of Canada's: ORG
February: DATE
Canada: GPE
Canada: GPE
February: DATE
Globalnews.ca: PRODUCT
Canada: GPE
Food Price Report: ORG
Canadians: NORP
Canada: GPE
October: DATE
Alberta: GPE
Calgary: GPE
Statistics Canada: ORG
Canada: GPE
Bank of Canada: ORG
June: DATE
Canadian: NORP
Canada: GPE
Bank of Canada: ORG
Canadians: NORP
November: DATE
The Daily Chase: ORG
Canada: GPE
Canada: GPE
October: DATE
U.S.: GPE
Canada: GPE
October: DATE
Canada: GPE
Canada: GPE
annual: DATE
September: DATE
Bank of Canada: ORG
Canada: GPE
Canada: GPE

1987-2028: DATE
Canada: GPE
30-year: DATE
Canada: GPE
Canada: GPE
third quarter: DATE
Statistics Canada: ORG
Canada: GPE
Canada: GPE
January: DATE
One-Year: DATE
Years: DATE
the Bank of Canada: ORG
Canada: GPE
the Bank of Canada: ORG
February: DATE
Canada: GPE
July: DATE
Canadian: NORP
Expectations Continue: ORG
Canadian: NORP
Canadian: NORP
Canadian: NORP
Second Quarter of 2023: DATE
Canada: GPE
February: DATE
Canada: GPE
annual: DATE
Canada: GPE
35-year: DATE
Canada: GPE
Canadian Rent Growth to Cool: ORG
Canada: GPE
Globalnews.ca: PRODUCT
Canada: GPE
July: DATE
The Average Cost to Rent: ORG
Canada: GPE
Canadians: NORP
Canada: GPE
Rental Landscape: ORG
Canada: GPE
2,193: MONEY
Canada: GPE
18-year-old: DATE
Canada: GPE

Canada: GPE
Canada: GPE
Canada: GPE
next month: DATE
January: DATE
2,196: MONEY
Canada: GPE
first: ORDINAL
Ottawa: GPE
Canada Housing Benefit: ORG
Freeland: GPE
Canada: GPE
Calgary: GPE
Canada: GPE
Canada: GPE
decades: DATE
Canada: GPE
Edmonton: PERSON
Canada: GPE
more than 18%: PERCENT
last year: DATE
Ontario: PERSON
third: ORDINAL
Canada: GPE
Brampton: GPE
Brampton: GPE
Canadian: NORP
Canada: GPE
Canada: GPE
New Aviva Canada: ORG
Macklem: GPE
Vancouver: PERSON
Canadian: NORP
6th month in a row: DATE
Toronto: GPE
Canada: GPE
Canada: GPE
Canadian: NORP
Toronto: GPE
Vancouver: GPE
Ontario Caps 2023 Rent Increase Guideline Below Inflation: ORG
2.5 Per Cent: MONEY
Ontario Newsroom: PERSON
Canada: GPE
B.C.: GPE
Canada: GPE

years ago: DATE
Canada: GPE
Canadian: NORP
monthly: DATE
Canada: GPE
Canadian: NORP
April 1: DATE
another 11%: PERCENT
past year: DATE
Canada: GPE
Canada: GPE
2,078: MONEY
July: DATE
Canadian Renters': ORG
Canadian: NORP
sixth consecutive month: DATE
year: DATE
Winnipeg: LOC
Canadian: NORP
Visa Information: ORG
Canadian Immigration Services: ORG
Free Online Evaluation: ORG
Canada: GPE
North York: GPE
hundreds: CARDINAL
Torontonians: NORP
Halifax: ORG
year-over-year: DATE
Canadian: NORP
Canada: GPE
2,117: MONEY
August: DATE
Canada: GPE
Nova Scotia: FAC
the 1970s: DATE
P.E.I.: ORG
New Brunswick: GPE
another 9 per cent: MONEY
past year: DATE
Annual: DATE
Canada: GPE
Canada: GPE
July: DATE
Ontario: GPE
Canadian: NORP
June: DATE

Toronto: GPE
3rd month: DATE
FIRST: ORDINAL
Canadian: NORP
Toronto: GPE
Canada: GPE
Montreal: GPE
this fall: DATE
N.S.: GPE
Canada: GPE
N.B.: GPE
Canada: GPE
Toronto: GPE
Alberta: GPE
B.C.: GPE
more than $100: MONEY
Canada: GPE
Ontario: GPE
Edmonton: PERSON
Canada: GPE
N.B.: ORG
Maritimes: GPE
Canada: GPE
January: DATE
Canada: GPE
Canadian: NORP
two years: DATE
- Mortgage Rates & Mortgage Broker News: ORG
One-third: CARDINAL
Canadian: NORP
Canada: GPE
Scotians: NORP
Ontario: GPE
The Housing Crisis: ORG
Canada: GPE
Canada: GPE
third month: DATE
Airbnbs: PERSON
Sweden: GPE
Canada: GPE
Kelowna: PERSON
fourth: ORDINAL
Canada: GPE
Vancouver: GPE
Canada: GPE
Richmond: GPE

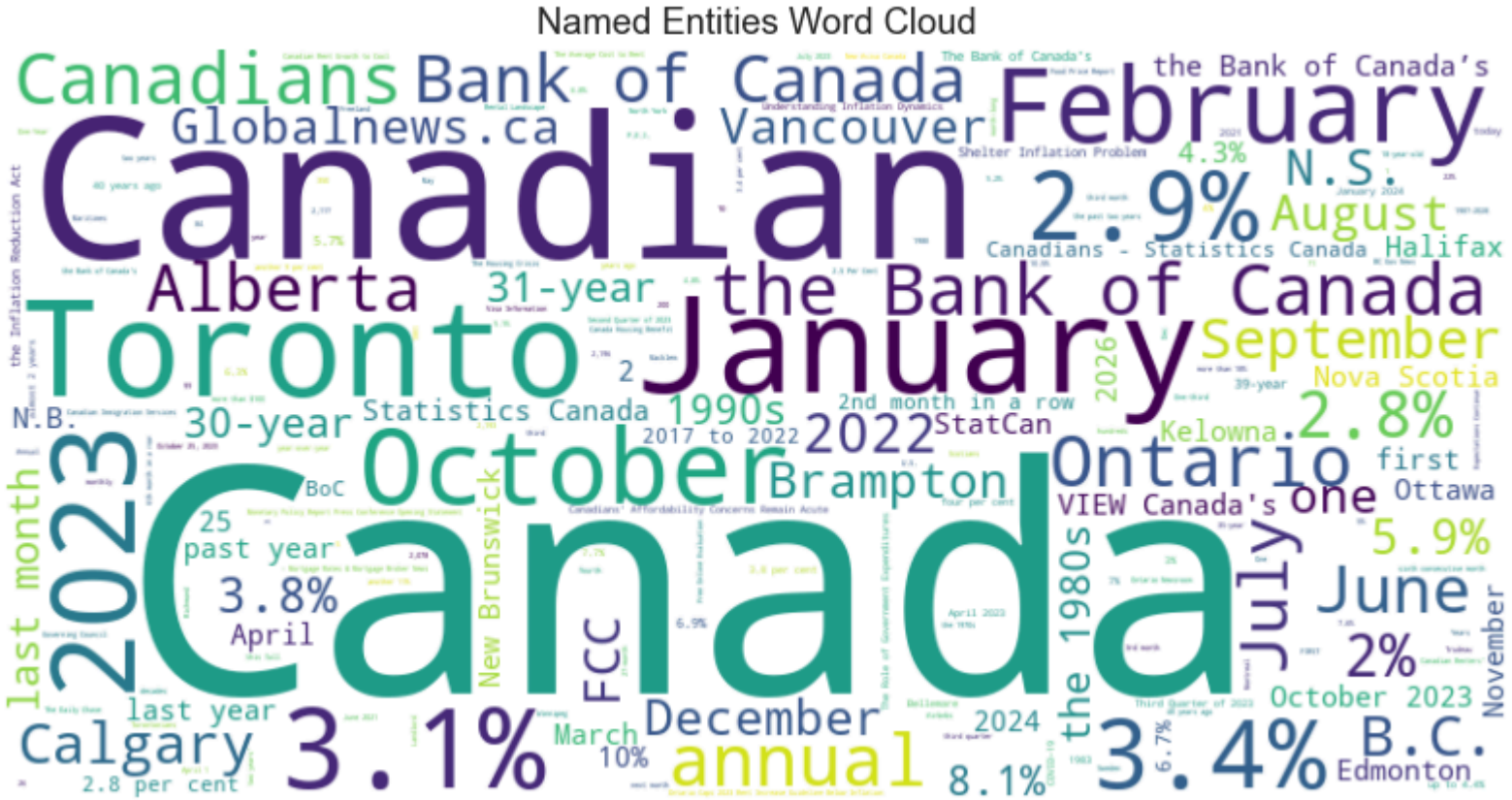
Canadian: NORP
last month: DATE
Canadian: NORP
Trudeau: PERSON
Alberta: GPE
48 years ago: DATE
Canada: GPE
Alberta: GPE
Ontario: PERSON
two years': DATE
Calgary: GPE
the past two years: DATE
Toronto: GPE
month-long: DATE
Canada: GPE
Landlord: PERSON
Toronto: GPE
BC Gov News: ORG
Canada: GPE
Ontario: GPE
MPP Michael Ford's: PERSON
Canada: GPE
Toronto: GPE
Brampton: GPE
Canada: GPE
Canada: GPE
Canada: GPE
Canadian: NORP
Canada: GPE
the last two years: DATE
Rentals.ca - Mortgage Rates & Mortgage Broker News: ORG
Nova Scotia: FAC
Canada: GPE
300,000: CARDINAL
Toronto: GPE
Canadian: NORP
June: DATE
Canada: GPE
Canadian: NORP
Canada: GPE
Canada: GPE
Canadians: NORP
Monday: DATE
Hundreds: CARDINAL
N.S.: GPE
New Brunswick: GPE

```
Canada: GPE
Canada: GPE
Canadian: NORP
Canada Struggle: ORG
Rental Housing Crisis: ORG
Canada: GPE
B.C.: GPE
Toronto: GPE
Ford: ORG
Manitoba Shared Health: ORG
more than $1M: MONEY
Canada: GPE
a decade: DATE
August: DATE
Canada: GPE
Ottawa: GPE
January: DATE
Canada: GPE
October: DATE
Calgary: GPE
Halifax: ORG
20th: ORDINAL
Canada: GPE
N.S.: GPE
more than 13%: PERCENT
October: DATE
last year: DATE
Toronto: GPE
Toronto: GPE
Kelowna: ORG
B.C.: GPE
10th: ORDINAL
Canada: GPE
Alberta: GPE
Calgarians: NORP
Canada: GPE
```

In [ ]:

```python
from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import spacy

# Load the English language model for spaCy
nlp = spacy.load('en_core_web_sm')

# Apply named entity recognition to each title and extract named entities
```

```
named_entities = []
for title in df['Title']:
    doc = nlp(title)
    for ent in doc.ents:
        named_entities.append(ent.text)

# Count the frequency of each named entity
entity_freq = Counter(named_entities)

# Create a word cloud from the frequencies
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(entity_freq)

# Display the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Named Entities Word Cloud')
plt.axis('off')  # Remove axis
plt.show()
```



Chapter 5 - Conclusions

Here are some conclusions about the development of the models and sentiment analysis in this project:

1. **Model Development:**
    - Several machine learning classifiers were evaluated for predicting inflation changes based on the features extracted from the dataset.
    - The models were trained and evaluated using techniques such as cross-validation and performance metrics like accuracy, recall, and ROC AUC score.
    - Different classifiers such as Naive Bayes, Logistic Regression, Random Forest, Decision Tree, and Support Vector Machine (SVM) were tested to identify the best-performing model.
2. **Feature Selection:**
    - Feature selection techniques like pipeline with SelectFromModel were employed to identify the most relevant features for the models.
    - Features were selected based on their importance in predicting inflation changes, enhancing model efficiency and reducing computational complexity.
3. **Sentiment Analysis:**
    - Sentiment analysis was conducted on news article titles to gauge the overall sentiment surrounding inflation-related topics.
    - TextBlob was used to analyze sentiment polarity and subjectivity, providing insights into the general sentiment trend over time.
    - Visualization techniques such as line plots were employed to visualize sentiment analysis results, enabling stakeholders to identify sentiment patterns and trends.
4. **Model Evaluation and Performance:**
    - Model performance was assessed using various evaluation metrics, including accuracy, recall, confusion matrix, and ROC AUC score.
    - The best-performing model was selected based on its ability to accurately predict inflation changes and its generalization capability on unseen data.
    - Performance metrics were used to quantify the effectiveness of the models in capturing inflation dynamics and providing actionable insights.
5. **Integration of Data Analysis Techniques:**
    - Data preprocessing, feature engineering, model development, and sentiment analysis were integrated into a cohesive workflow to derive meaningful insights from the dataset.
    - By combining multiple data analysis techniques, the project aimed to provide a comprehensive understanding of inflation trends and sentiment dynamics.
6. **Future Considerations:**

- Further refinement of models and sentiment analysis techniques could enhance predictive accuracy and provide more nuanced insights into inflation-related dynamics.
- Continuous monitoring and updating of models with new data can improve their performance and adaptability to changing economic conditions.
- Exploring advanced machine learning algorithms and natural language processing techniques may offer additional opportunities for improving model accuracy and sentiment analysis capabilities.

Overall, the development of models and sentiment analysis in this project aimed to provide valuable insights into inflation dynamics and sentiment trends, aiding stakeholders in making informed decisions in the financial and economic domain.

References

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95.

Python Software Foundation. (n.d.). Matplotlib: Visualization with Python. Retrieved from https://matplotlib.org/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825-2830.

Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd.

Websites:

Statistics Canada. (n.d.). A snapshot of how inflation is affecting Canadians - Statistics Canada. Retrieved from https://news.google.com/articles/CBMiVGh0dHBzOi8vd3d3LnN0YXRjYW4uZ2MuY2EvbzEvZW4vcGx1cy8zMDk2LXNuYXBzaG90LWhvdy1pbmZsYXRpb24tYWZmZW
hl=en-CA&gl=CA&ceid=CA%3Aen

Bank of Canada. (n.d.). What's happening to inflation and why it matters. Retrieved from https://news.google.com/articles/CBMiVGh0dHBzOi8vd3d3LmJhbmtvZmNhbmFkYS5jYS8yMDIyLzEwL3doYXRzLWhhcHBlbmluZy10by1pbmZsYXRpb24tYW5kLXdo
hl=en-CA&gl=CA&ceid=CA%3Aen

TD Economics. (n.d.). Canadian Inflation: A New Vintage. Retrieved from https://news.google.com/articles/CBMiMWh0dHBzOi8vZWNvbm9taWNzLnRkLmNvbS9jYS1pbmZsYXRpb24tbmV3LXZpbnRhZ2XSAQA?hl=en-CA&gl=CA&ceid=CA%3Aen

CBC News. (n.d.). Canada's inflation rate slowed to 2.9% in January as gas prices fell. Retrieved from https://news.google.com/articles/CBMiQWh0dHBzOi8vd3d3LmNiYy5jYS9uZXdzL2J1c2luZXNzL2luZmxhdGlvbi1qYW51YXJ5LTIwMjQtMS43MTE5Nzk20gEgaHR0cH hl=en-CA&gl=CA&ceid=CA%3Aen

CP24. (n.d.). Canada inflation rate ticks down to 2.8 per cent. Retrieved from https://news.google.com/articles/CBMidWh0dHBzOi8vd3d3LmNwMjQuY29tL25ld3MvdW5hbWJpZ3VvdXNseS1nb29kLWluZmxhdGlvbi1zbG93cy1pbi1mZWJydWF hl=en-CA&gl=CA&ceid=CA%3Aen