Ion Iulia - Andreea Group 246

Price influence on Crypto-coins based on news using Sentiment Analysis

1. Problem Statement

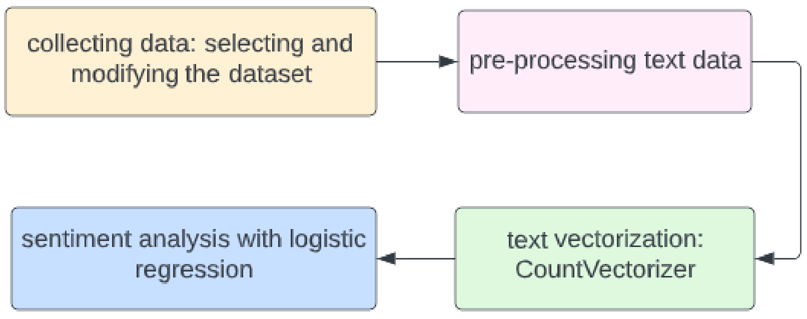
The price of cryptocurrencies fluctuates constantly, hourly. The cryptocurrency market is interconnected like many markets globally, but also by events happening around the world. So crisis situations make the cryptocurrency market unstable. In this project, we used news based to be able to evaluate the price fluctuation of bitcoin cryptocurrency in the market, based on sentiments expressed to drive sentiment analysis. In order to evaluate the model, we used the Kaggle dataset - Crypto News + , to which we added some modifications and sorted it to contain only the bitcoin cryptocurrency. Since it is a current topic, there are many articles based on Cryptocurrency price prediction using logistic regression, articles [1] [2] [3] were used as inspiration.

2. Proposed Solution

Theoretical Aspects

Unstructured data accounts for around 80% of the world's data. As a result, extracting information from unstructured data is a critical component of data analysis. Sentiment analysis is one application of text mining, which is the process of extracting useful information from unstructured text data. It understands and classifies subjective feelings from text input using natural language processing and machine learning techniques. Sentiment analysis is commonly used in corporate contexts to understand customer evaluations, detect spam from emails, and so on.

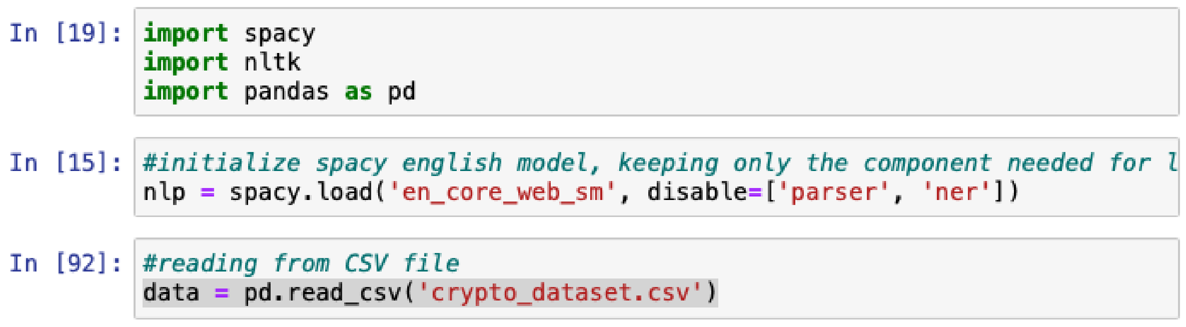
In this paper I tried to choose an appropriate dataset, which I later modified to meet my requirements. Pre-processing the text in the dataset is an extremely important step to have clearer data without unnecessary elements and incorrect figures. Next on the clean dataset we applied text vectorizations, where we used CountVectorizer, and finally we performed sentiment analysis with logistic regression. The figure below shows the application workflow:


collecting data: selecting and modifying the dataset
pre-processing text data
text vectorization: CountVectorizer
sentiment analysis with logistic regression

Application

- Preprocessing text data

I use tools from NLTK, Spacy, and some regular expressions to preprocess the news articles. The below code is used to import the libraries and the pre-build models in Spacy and the dataset is read using pandas.

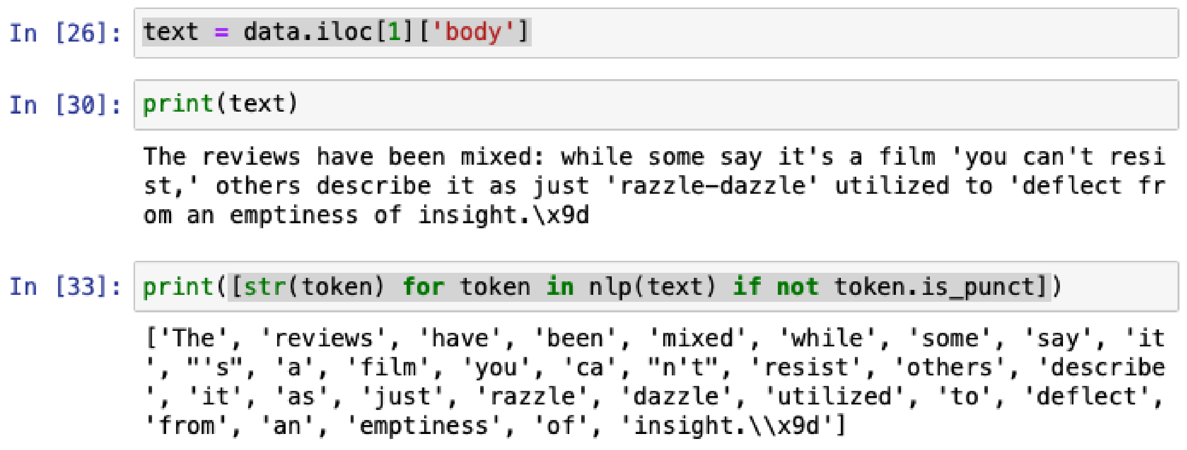


```
In [19]: import spacy
         import nltk
         import pandas as pd

In [15]: #initialize spacy english model, keeping only the component needed for l
         nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

In [92]: #reading from CSV file
         data = pd.read_csv('crypto_dataset.csv')
```

Text preparation processes will be applied to the columns "Sentiment," "Price," and "Body." To extract valuable elements from the news content, I preprocessed the articles using typical text mining processes such as tokenization, stopword removal, and lemmatization.

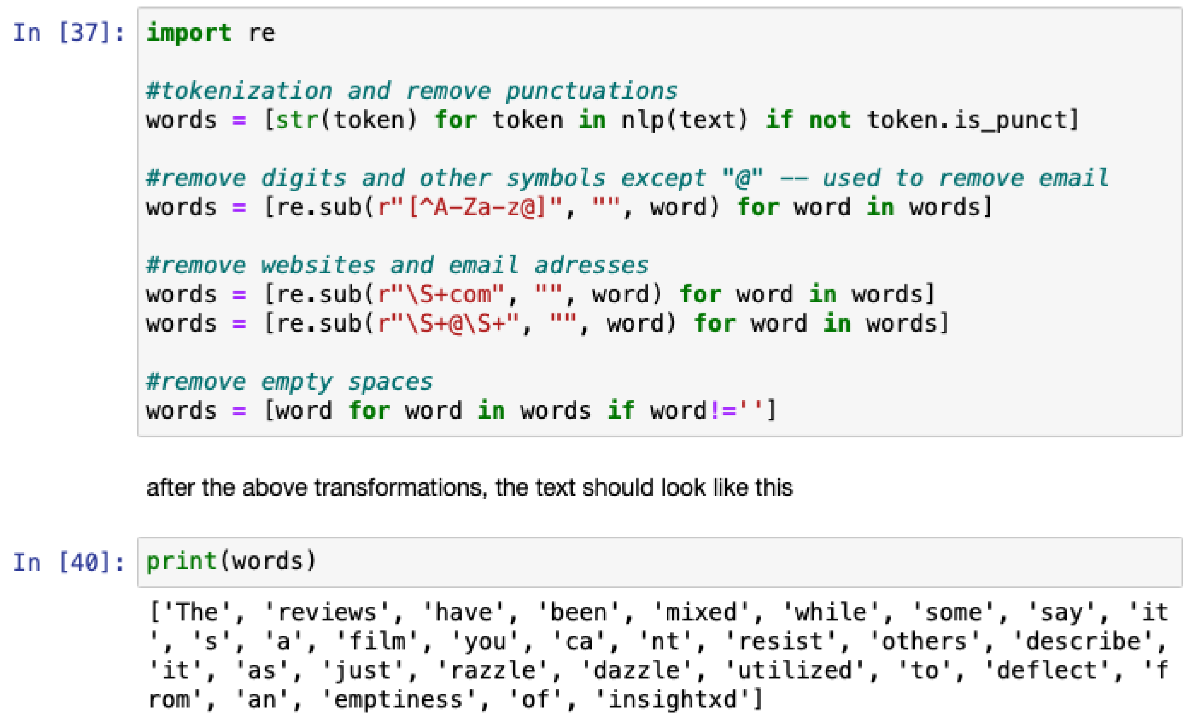
- Tokenization

Tokenization is the initial step in preparing text data, which involves breaking each sentence down into individual words. Individual words, rather than entire phrases, are used to break down the relationships between them. It is, nonetheless, a standard tool for analyzing enormous amounts of text data. It is efficient and convenient for computers to evaluate text data by looking at what words appear in an article and how many times they appear, and this is sufficient to get meaningful conclusions. Consider the first news article in my dataset and apply the nlp model described above to it.



```
In [26]: text = data.iloc[1]['body']

In [30]: print(text)

         The reviews have been mixed: while some say it's a film 'you can't resi
         st,' others describe it as just 'razzle-dazzle' utilized to 'deflect fr
         om an emptiness of insight.\x9d

In [33]: print([str(token) for token in nlp(text) if not token.is_punct])

         ['The', 'reviews', 'have', 'been', 'mixed', 'while', 'some', 'say', 'it
         ', "'s", 'a', 'film', 'you', 'ca', "n't", 'resist', 'others', 'describe
         ', 'it', 'as', 'just', 'razzle', 'dazzle', 'utilized', 'to', 'deflect',
         'from', 'an', 'emptiness', 'of', 'insight.\\x9d']
```

Each news story will be transformed into a collection of words, symbols, numerals, and punctuation after tokenization. You can also select whether or not every word should be converted to lowercase. The next step is to get rid of any unnecessary data. Symbols, numerals, and punctuation, for example. To get rid of them, I'll utilize spacy and regex.



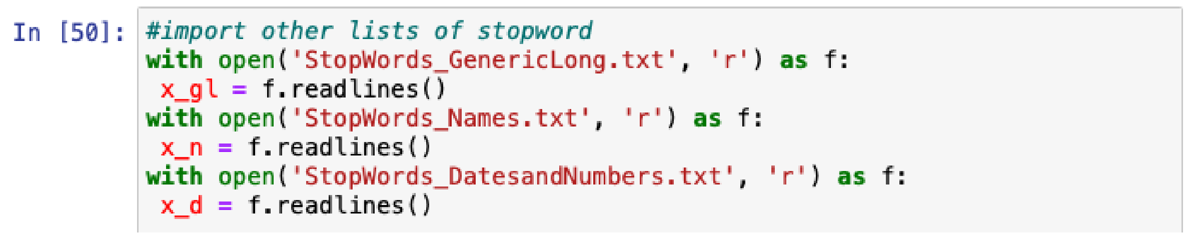
```
In [37]: import re

         #tokenization and remove punctuations
         words = [str(token) for token in nlp(text) if not token.is_punct]

         #remove digits and other symbols except "@" -- used to remove email
         words = [re.sub(r"[^A-Za-z@]", "", word) for word in words]

         #remove websites and email adresses
         words = [re.sub(r"\S+com", "", word) for word in words]
         words = [re.sub(r"\S+@\S+", "", word) for word in words]

         #remove empty spaces
         words = [word for word in words if word!='']
```

after the above transformations, the text should look like this

```
In [40]: print(words)

         ['The', 'reviews', 'have', 'been', 'mixed', 'while', 'some', 'say', 'it
         ', 's', 'a', 'film', 'you', 'ca', 'nt', 'resist', 'others', 'describe',
         'it', 'as', 'just', 'razzle', 'dazzle', 'utilized', 'to', 'deflect', 'f
         rom', 'an', 'emptiness', 'of', 'insightxd']
```

- Stopwords

The news item is considerably cleaner now, but there are still certain words we don't want to see, such as "and," "we," and so on. The next stage is to eliminate the unnecessary words, also known as stopwords. Stopwords are words that appear often in many articles but have no obvious significance. Stopwords include 'I,' 'the,' 'a,' and 'of.' These are the terms that, if omitted, will not interfere with the interpretation of articles. We can import the stopwords from the NLTK library to remove them. In addition, I give other lists of stopwords that are commonly used in economic research, such as dates and times, more generic words that are not economically significant, and so on. This is how I put together my stopwords list:



```
In [50]: #import other lists of stopword
         with open('StopWords_GenericLong.txt', 'r') as f:
          x_gl = f.readlines()
         with open('StopWords_Names.txt', 'r') as f:
          x_n = f.readlines()
         with open('StopWords_DatesandNumbers.txt', 'r') as f:
          x_d = f.readlines()
```

```
#import nltk stopwords
stopwords = nltk.corpus.stopwords.words('english')

#combine all stopwords
[stopwords.append(x.rstrip()) for x in x_gl]
[stopwords.append(x.rstrip()) for x in x_n]
[stopwords.append(x.rstrip()) for x in x_d]

#change all stopwords into lowercase
stopwords_lower = [s.lower() for s in stopwords]
```

stopwords will be removed from all the articles from the dataset and applying to the above specific example, in order to see if the the stopwords are remove successfully:

```
In [51]: words = [word.lower() for word in words if word.lower() not in stopwords

In [52]: print(words)

         ['reviews', 'mixed', 'film', 'ca', 'nt', 'resist', 'describe', 'razzle'
         , 'dazzle', 'utilized', 'deflect', 'emptiness', 'insightxd']
```

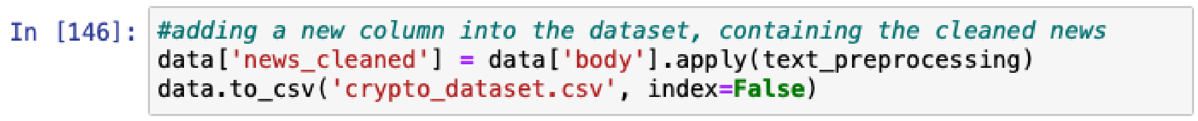
After independently verifying that the preprocessing works, I added everything into a function that returns a string parameter containing the text of all the above preprocessing

```
In [56]: def text_preprocessing(str_input):
              #tokenization, remove punctuation, lemmatization
              words=[token.lemma_ for token in nlp(str_input) if not token.is_pun

              # remove symbols, websites, email addresses
              words = [re.sub(r"[^A-Za-z@]", "", word) for word in words]
              words = [re.sub(r"\S+com", "", word) for word in words]
              words = [re.sub(r"\S+@\S+", "", word) for word in words]
              words = [word for word in words if word!=' ']
              words = [word for word in words if len(word)!=0]

              #remove stopwords
              words=[word.lower() for word in words if word.lower() not in stopwo
              #combine a list into one string
              string = " ".join(words)
              return string
```

I then created a function that selects random items from the dataset to test the functionality

```
In [67]: import random

         index = random.randint(0, data.shape[0])

         random_article = data.iloc[index]['body']
         print('Article before pre-processing:\n'+ random_article)
         print('\n')
         print('Article after pre-processing:\n' + text_preprocessing(random_arti
```

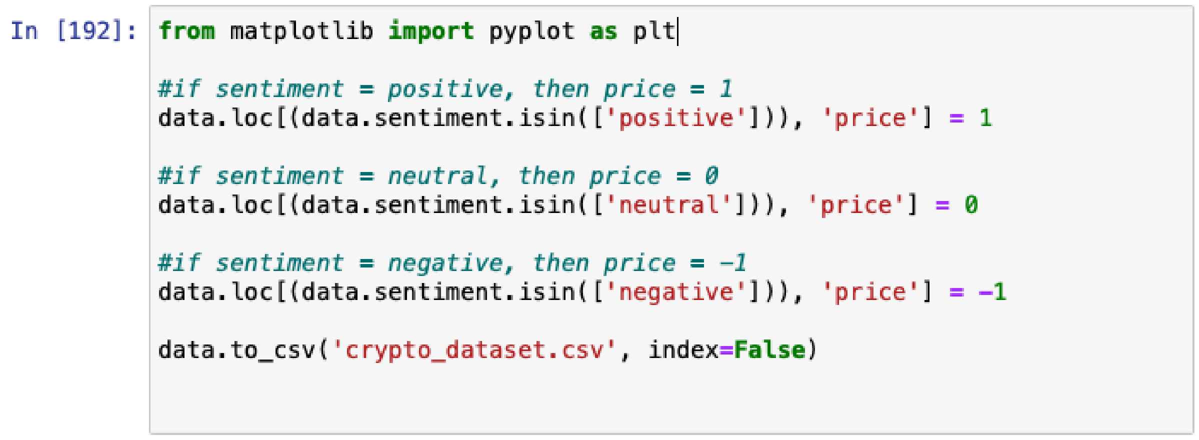
Finally we added in a new column, the articles resulting after preprocessing - named ‘news_cleaned’



```
In [146]: #adding a new column into the dataset, containing the cleaned news
          data['news_cleaned'] = data['body'].apply(text_preprocessing)
          data.to_csv('crypto_dataset.csv', index=False)
```

In order to see the price fluctuation of the bitcoin cryptocurrency, it was necessary to change the price value, initially set to 0. Based on the sentiment associated with the item, the price can take the following values:

- positive - price will increase
- neutral - price will remain the same
- negative - price will decrease



```
In [192]: from matplotlib import pyplot as plt

          #if sentiment = positive, then price = 1
          data.loc[(data.sentiment.isin(['positive'])), 'price'] = 1

          #if sentiment = neutral, then price = 0
          data.loc[(data.sentiment.isin(['neutral'])), 'price'] = 0

          #if sentiment = negative, then price = -1
          data.loc[(data.sentiment.isin(['negative'])), 'price'] = -1

          data.to_csv('crypto_dataset.csv', index=False)
```

- CountVectorizer

The Bag of Words technique is used by the CountVectorizer, which ignores text structures and solely extracts information from word counts. Every document will be converted to a vector format. The occurrence count of each unique term in this document is one of the vector's inputs. The CountVectorizer will turn the text data into a m*n sparse matrix if the corpus has m documents and each document contains n unique words.

The CountVectorizer uses two phases to create a sparse matrix from a list of documents: fit and transform. The vectorizer reads the list of documents, counts the number of unique words in the corpus, and assigns an index to each word throughout the fitting process. The fitted data must now be transformed. CountVectorizer counts the number of times each unique word appears in each document.

```
#train and test split
X_train, X_test, y_train, y_test = train_test_split(data['news_cleaned']
                                                     data['price'],
                                                     random_state=0)

vect = CountVectorizer(min_df=2, max_df=0.1).fit(X_train)
X_train_vectorized = vect.transform(X_train)
print('Number of features: {}'. format(len(vect.get_feature_names())))
```

- Setting Target Value

Take, for example, my project, which aims to forecast changes in bitcoin future values based on recently published news stories. I define good news as that which predicts an increase in price, while negative news predicts a reduction in price. I now need to assign the goal values for my dataset because I've already collected and converted the text data that will be used as features.

The directions of price change with respect to various news stories are the project's target value.

- Introducing Logistic Regression

To develop the machine learning model, I now need to use an estimator. The logistic regression model is one of many that can be used to address the binary classification problem. A linear classifier, logistic regression is a transition from a linear function:

$$f(x) = b_0 + b_1 * x_1 + \ldots + b_n * x_n$$

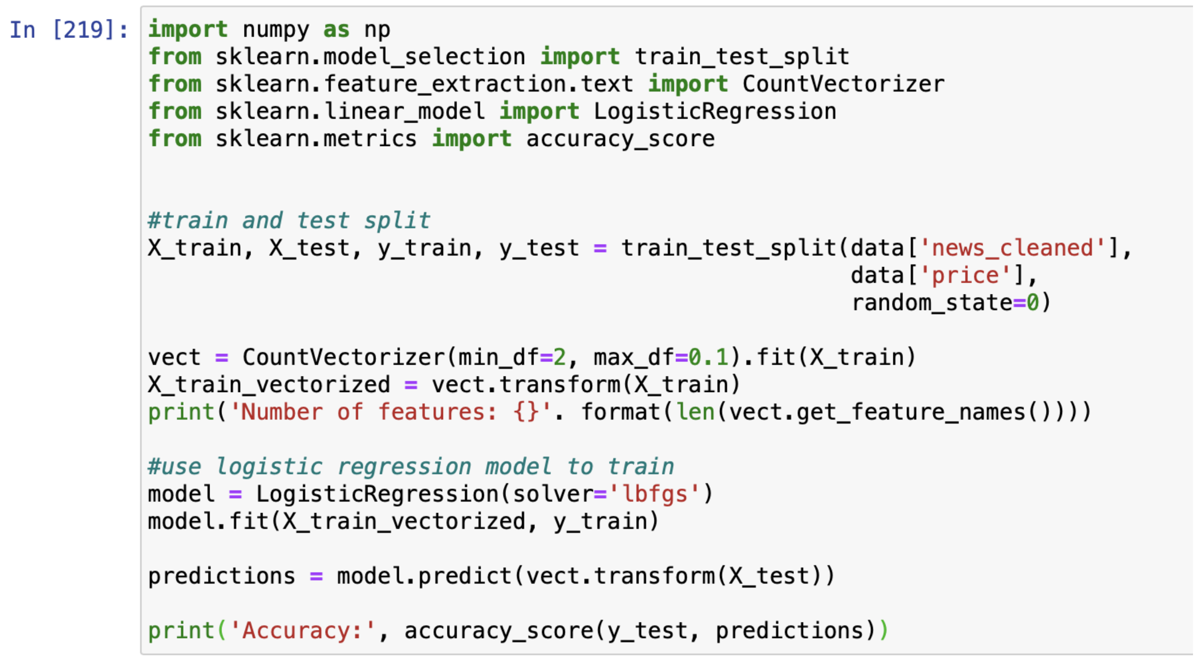
where b0, b1...bn are the regression coefficient estimators for a set of independent variables x=(x1,x2,...xn). The sigmoid function of f(x) is the logistic regression function p(x):

$$p(x) = \sigma(f(x)) = \frac{1}{1 + exp^{-f(x)}}$$

The value of p(x) after transformation will be between [0,1], which can be regarded as a probability. In general, 1-p(x) is the chance that x is in the negative class, and p(x) is the expected probability that x is in the positive class.

- Implementation of Logistic Regression

I divide my dataset into training and test sets before using logistic regression and training the model. "df['sentiment"]" refers to "cleaned" news stories, whereas "df['price']" refers to the target value of price dummies. I design a machine learning pipeline and use GridSearchCV to identify the best hyper-parameters to find the optimal transformer and estimator. For your convenience, I've included the code below:



```
In [219]: import numpy as np
          from sklearn.model_selection import train_test_split
          from sklearn.feature_extraction.text import CountVectorizer
          from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import accuracy_score


          #train and test split
          X_train, X_test, y_train, y_test = train_test_split(data['news_cleaned'],
                                                              data['price'],
                                                              random_state=0)

          vect = CountVectorizer(min_df=2, max_df=0.1).fit(X_train)
          X_train_vectorized = vect.transform(X_train)
          print('Number of features: {}'. format(len(vect.get_feature_names())))

          #use logistic regression model to train
          model = LogisticRegression(solver='lbfgs')
          model.fit(X_train_vectorized, y_train)

          predictions = model.predict(vect.transform(X_test))

          print('Accuracy:', accuracy_score(y_test, predictions))
```

GridSearchCV will look for the hyper-parameters that produce the maximum accuracy for model evaluation if none are specified. We may use different metrics if accuracy isn't the optimum metric for evaluating the model. You can specify it by using the GridSearchCV function's "scoring" argument.

In order to evaluate the accuracy of the model I used the accuracy_score method, which gives us the accuracy, more specifically, for my article, in which I evaluated a dataset containing 1500 articles, it resulted in an accuracy of 0.53.

3. Implementation

- Libraries

This section presents the libraries used throughout the project.

- **spaCy** - is a free, open-source library for advanced Natural Language Processing (NLP) in Python. spaCy is a production-ready tool that allows you to create apps that analyze and "understand" massive amounts of text. It can be used to create data extraction and natural language understanding systems, as well as to pre-process text for deep learning.

- **usage in code** - initializing spacy english model, keeping only the component needed for lemmatization and creating the engine

- **nltk** - NLTK is a popular Python programming language for working with human language data. It includes a set of text processing tools for categorization, tokenization, stemming, tagging, parsing, and semantic reasoning, as well as wrappers for industrial-strength NLP libraries and easy-to-use interfaces to over 50 corpora and lexical resources like WordNet.
    - **usage in code** - used in order to import stop words, to eliminate them from the dataset

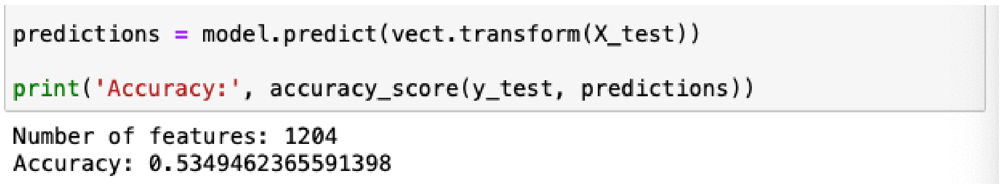
- there are also used the following libraries - pandas, re, random, matplotlib, sklearn, numpy

● Functions

The only function created in this program is the one for text preprocessing, which includes tokenization, removal of symbols, sites and emails, and stopwords, combining a list into a single string. The rest of the functions are the default ones from the imported libraries.

4. Experimental Results

The code was run on the entire dataset containing about 1400 entries. The method that was applied to evaluate the result is accuracy_score().



```
predictions = model.predict(vect.transform(X_test))

print('Accuracy:', accuracy_score(y_test, predictions))
```

```
Number of features: 1204
Accuracy: 0.5349462365591398
```

In my opinion, the low accuracy is due to the fact that most of the sentiments associated with the articles are neutral, which makes the whole process much harder.

REFERENCES

[1] - Abraham, Jethin, et al. "Cryptocurrency price prediction using tweet volumes and sentiment analysis." *SMU Data Science Review* 1.3 (2018): 1.

[2] - Ibrahim, Ahmed. "Forecasting the Early Market Movement in Bitcoin Using Twitter's Sentiment Analysis: An Ensemble-based Prediction Model." *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*. IEEE, 2021.

[3] - Lamon, Connor, Eric Nielsen, and Eric Redondo. "Cryptocurrency price prediction using news and social media sentiment." *SMU Data Sci. Rev* 1.3 (2017): 1-22.