Iris Data

Content

Iris Dataset Report

Explanation of the Iris dataset attributes and details:

Context	The Iris flower data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in 1936. The data set consists of 50 samples from each of three species of Iris (Iris Setosa, Iris virginica, and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.
Content	The dataset contains a set of 150 records under 5 attributes.
Attributes	Sepal length Sepal width Petal length Petal width Class (Species)
Importance	This dataset became a typical test case for many statistical classification techniques in machine learning.
Link	https://www.kaggle.com/datasets/uciml/iris

Exploratory Data Analysis:

- **Importing the libraries and the data**

Importing the libraries and the data

In [66]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import seaborn as sns
plt.style.use('seaborn')
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import pairwise_distances
from sklearn.datasets import load_iris
```

Importing the data from .csv file

First we read the data from the dataset using `read_csv` from the pandas library.

In [31]:

```
data = pd.read_csv('data\iris.csv')
```

- **Viewing and describing the data**

In [40]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Describing the data as basic statistics using `describe()`

In [41]:
```
data.describe()
```

Out[41]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

- **Checking the data for inconsistencies and further cleaning the data if needed**

In [43]:
```
data.isnull().sum()
```

Out[43]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

The 'Id' column has no relevence therefore deleting it would be better.

Deleting 'customer_id' colummn using `drop()`.

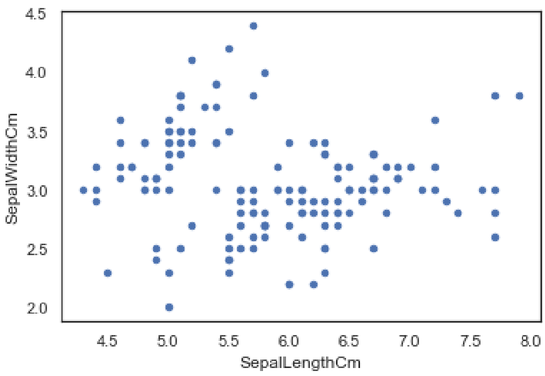
In [44]:
```
data["Species"].value_counts()
```

Out[44]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

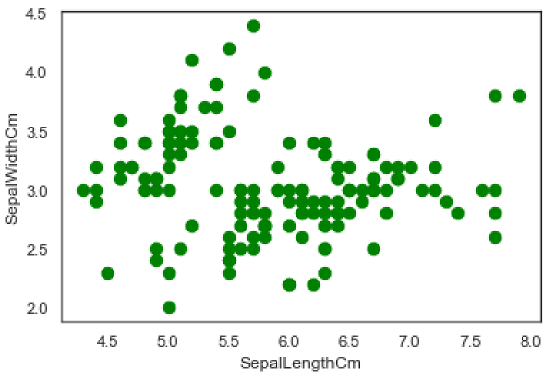
Data Visualization:

In [54]:
```
iris.plot(kind="scatter", x="SepalLengthCm", y="SepalWidthCm")
plt.show()
```

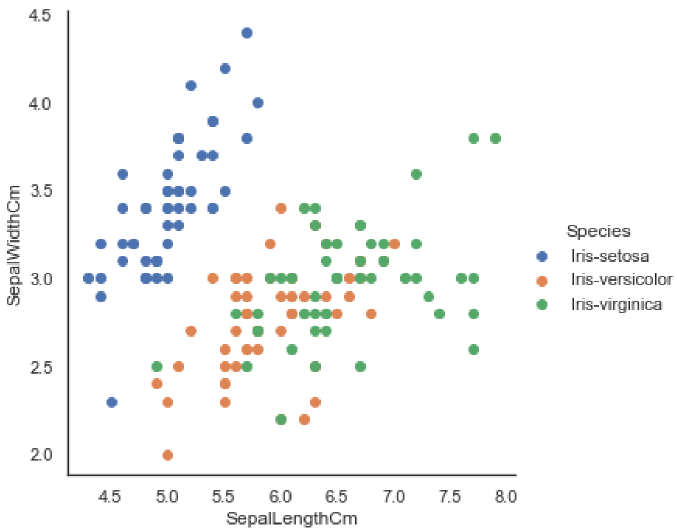
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoi
in case its length matches with *x* & *y*.  Please use the *color* keyword-argument
if you intend to specify the same RGB or RGBA value for all points.
```



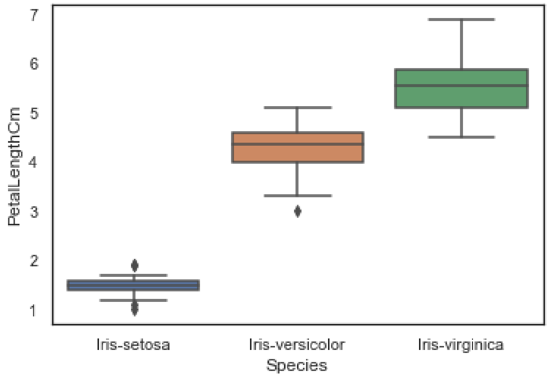
In [55]:
```
iris.plot(kind="scatter", x="SepalLengthCm", y="SepalWidthCm",color="green",s=70 )
plt.show()
```



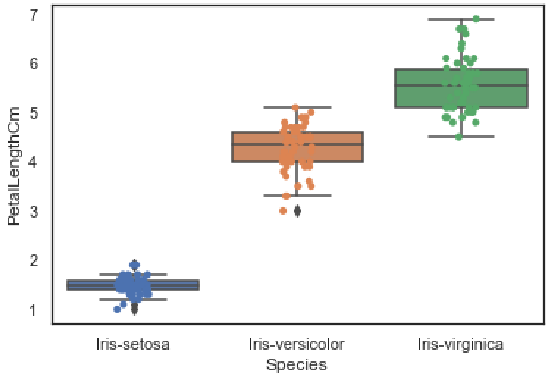
In [57]:
```
sns.FacetGrid(iris, hue="Species", size=5) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend()
plt.show()
```



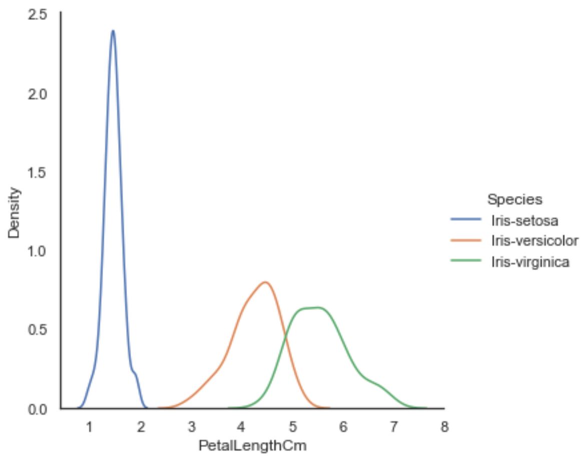
In [59]:
```python
sns.boxplot(x="Species", y="PetalLengthCm", data=iris )
plt.show()
```

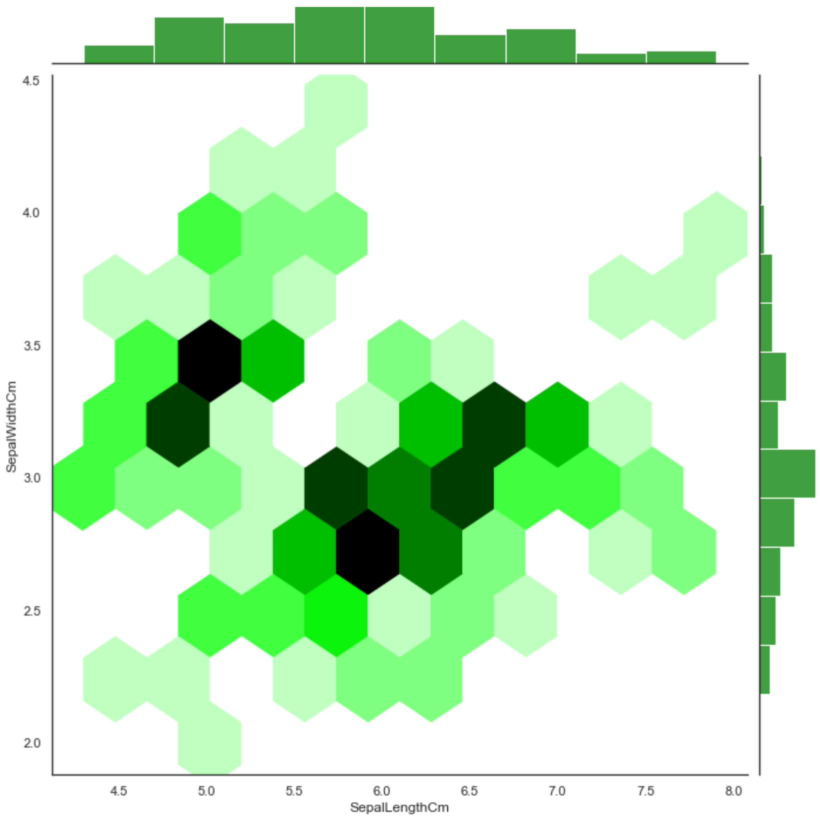


In [60]:
```python
ax= sns.boxplot(x="Species", y="PetalLengthCm", data=iris)
ax= sns.stripplot(x="Species", y="PetalLengthCm", data=iris, jitter=True, edgecolor="gray")
plt.show()
```

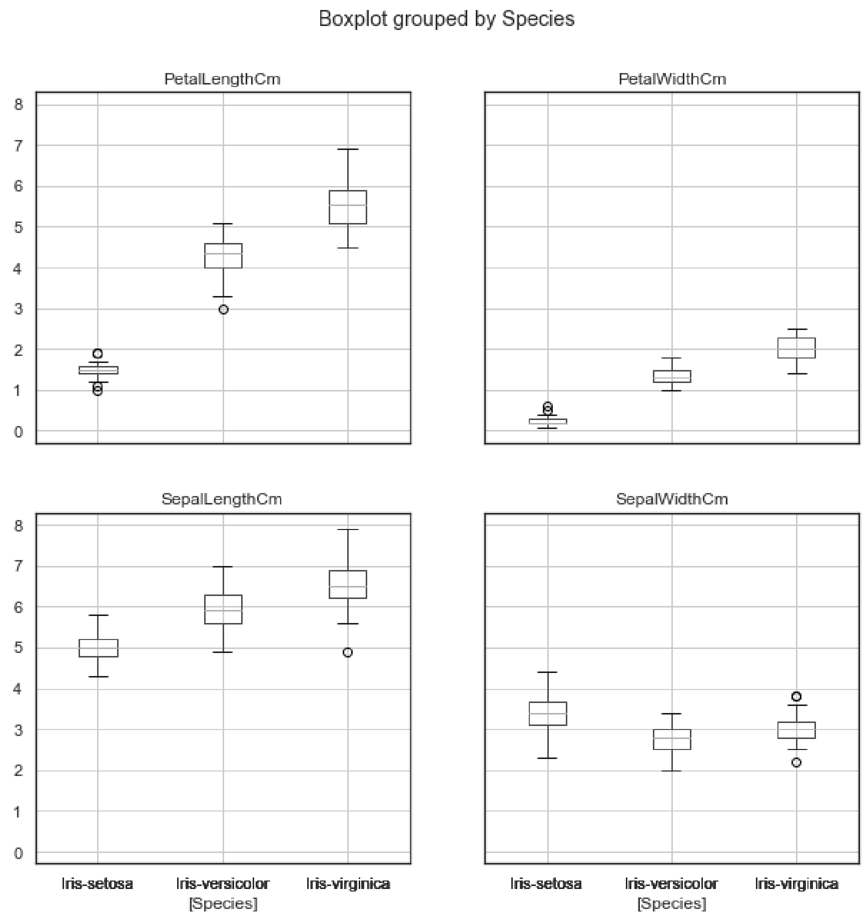


In [63]:
```python
sns.FacetGrid(iris, hue="Species", size=5) \
   .map(sns.kdeplot, "PetalLengthCm") \
   .add_legend()
plt.show()
```

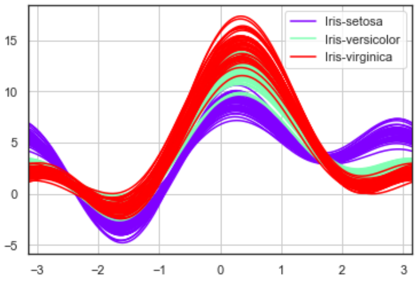




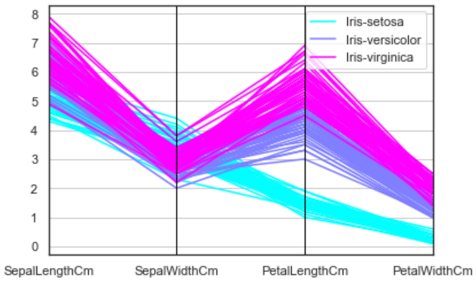

```
iris.drop("Id", axis=1).boxplot(by="Species", figsize=(10, 10))
plt.show()
```



```
In [68]:  from pandas.plotting import andrews_curves
andrews_curves(iris.drop("Id", axis=1), "Species",colormap='rainbow')
plt.show()
```




```
In [69]:  from pandas.plotting import parallel_coordinates
parallel_coordinates(iris.drop("Id", axis=1), "Species",colormap='cool')
plt.show()
```



Classification:

We have done 4 different classification models

Train-test split

```
X=data.drop(["Species"],axis=1)
y=data["Species"]
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=25)
```

KNeighborsClassifier

```
KNNClassifier = KNeighborsClassifier()
KNNClassifier.fit(X_train, y_train)

y_pred_KNN = KNNClassifier.predict(X_test)
```

```
C:\Users\Tony\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction fu
nctions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.
0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is take
n will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
Accuracy Score: 0.921
Confusion Matrix:
 [[11  0  0]
 [ 0 13  3]
 [ 0  0 11]]
```

DecisionTreeClassifier

In [94]:
```
DTCclassifier = DecisionTreeClassifier(random_state = 0, splitter = 'random')

DTCclassifier.fit(X_train, y_train)
y_pred_DTC = DTCclassifier.predict(X_test)
```

In [95]:
```
# Calculate accuracy score
accuracy_DTC = round(accuracy_score(y_test, y_pred_DTC), 3)

# Calculate confusion matrix
cm_DTC = confusion_matrix(y_test, y_pred_DTC)

# Append results to lists
accuracy_scores.append(accuracy_DTC)
confusion_matrices.append(cm_DTC)

# Store model names
model_names.append("DecisionTreeClassifier")

# Print results
print("Accuracy Score:", accuracy_DTC)
print("Confusion Matrix:\n", cm_DTC)
```

```
Accuracy Score: 0.895
Confusion Matrix:
 [[11  0  0]
 [ 0 13  3]
 [ 0  1 10]]
```

LogisticRegression

```
LRclassifier = LogisticRegression(solver='liblinear')
LRclassifier.fit(X_train, y_train)

y_pred_LR = LRclassifier.predict(X_test)
```

```
# Calculate accuracy score
accuracy_LR = round(accuracy_score(y_test, y_pred_LR), 3)

# Calculate confusion matrix
cm_LR = confusion_matrix(y_test, y_pred_LR)

# Append results to lists
accuracy_scores.append(accuracy_LR)
confusion_matrices.append(cm_LR)

# Store model names
model_names.append("LogisticRegression")

# Print results
print("Accuracy Score:", accuracy_LR)
print("Confusion Matrix:\n", cm_LR)
```

```
Accuracy Score: 0.921
Confusion Matrix:
 [[11  0  0]
 [ 0 13  3]
 [ 0  0 11]]
```

RandomForestClassifier

In [98]:
```
RFclassifier = RandomForestClassifier(random_state = 0)

RFclassifier.fit(X_train, y_train)
y_pred_RF = RFclassifier.predict(X_test)
```

In [99]:
```
# Calculate accuracy score
accuracy_RF = round(accuracy_score(y_test, y_pred_RF), 3)

# Calculate confusion matrix
cm_RF = confusion_matrix(y_test, y_pred_RF)

# Append results to lists
accuracy_scores.append(accuracy_RF)
confusion_matrices.append(cm_RF)

# Store model names
model_names.append("RandomForestClassifier")

# Print results
print("Accuracy Score:", accuracy_RF)
print("Confusion Matrix:\n", cm_RF)
```

```
Accuracy Score: 0.947
Confusion Matrix:
 [[11  0  0]
 [ 0 15  1]
 [ 0  1 10]]
```

Results

In [102]:
```
results_df = pd.DataFrame({'Model': model_names, 'Accuracy Score': accuracy_scores})
results_df
```

Out[102]:

	Model	Accuracy Score
0	KNeighborsClassifier	0.921
1	DecisionTreeClassifier	0.895
2	LogisticRegression	0.921
3	RandomForestClassifier	0.947

Clustering (K-means):

Now that we have the clusters created, we will enter them into a different column

```
clusters = clustering_data.copy()
clusters['Cluster_Prediction'] = kms.fit_predict(clustering_data)
clusters.head()
```

46]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Cluster_Prediction
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

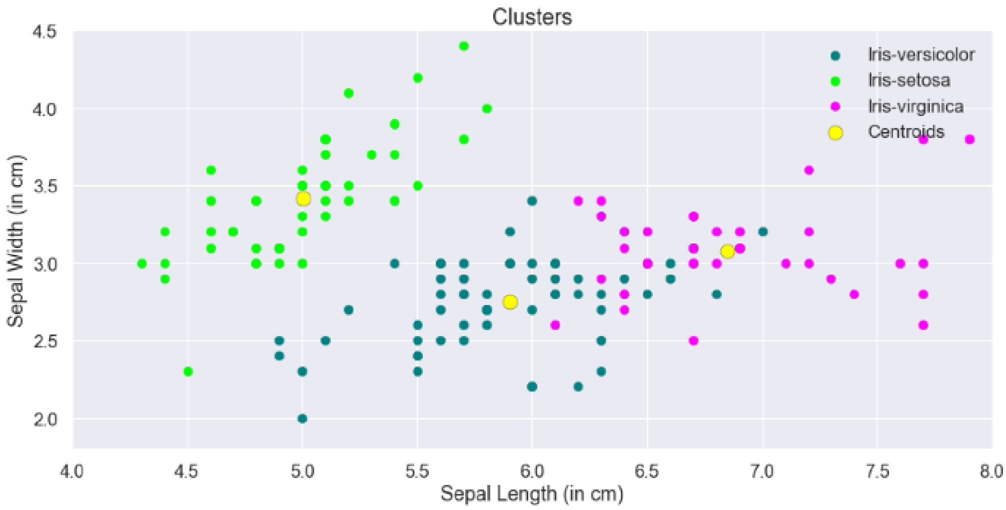
We can also get the centroids of the clusters by the `cluster_centers_` attribute of KMeans algorithm.

```
kms.cluster_centers_
```

47]:
```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

In [48]:

```
fig, ax = plt.subplots(figsize=(15,7))
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 0]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 0]['SepalWidthCm'],
            s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-versicolor')


plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 1]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 1]['SepalWidthCm'],
            s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-setosa')


plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 2]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 2]['SepalWidthCm'],
            s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-virginica')

plt.scatter(x=kms.cluster_centers_[:, 0], y=kms.cluster_centers_[:, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor=
plt.legend(loc='upper right')
plt.xlim(4,8)
plt.ylim(1.8,4.5)
ax.set_ylabel('Sepal Width (in cm)')
ax.set_xlabel('Sepal Length (in cm)')
plt.title('Clusters', fontsize = 20)
plt.show()
```



Analysis

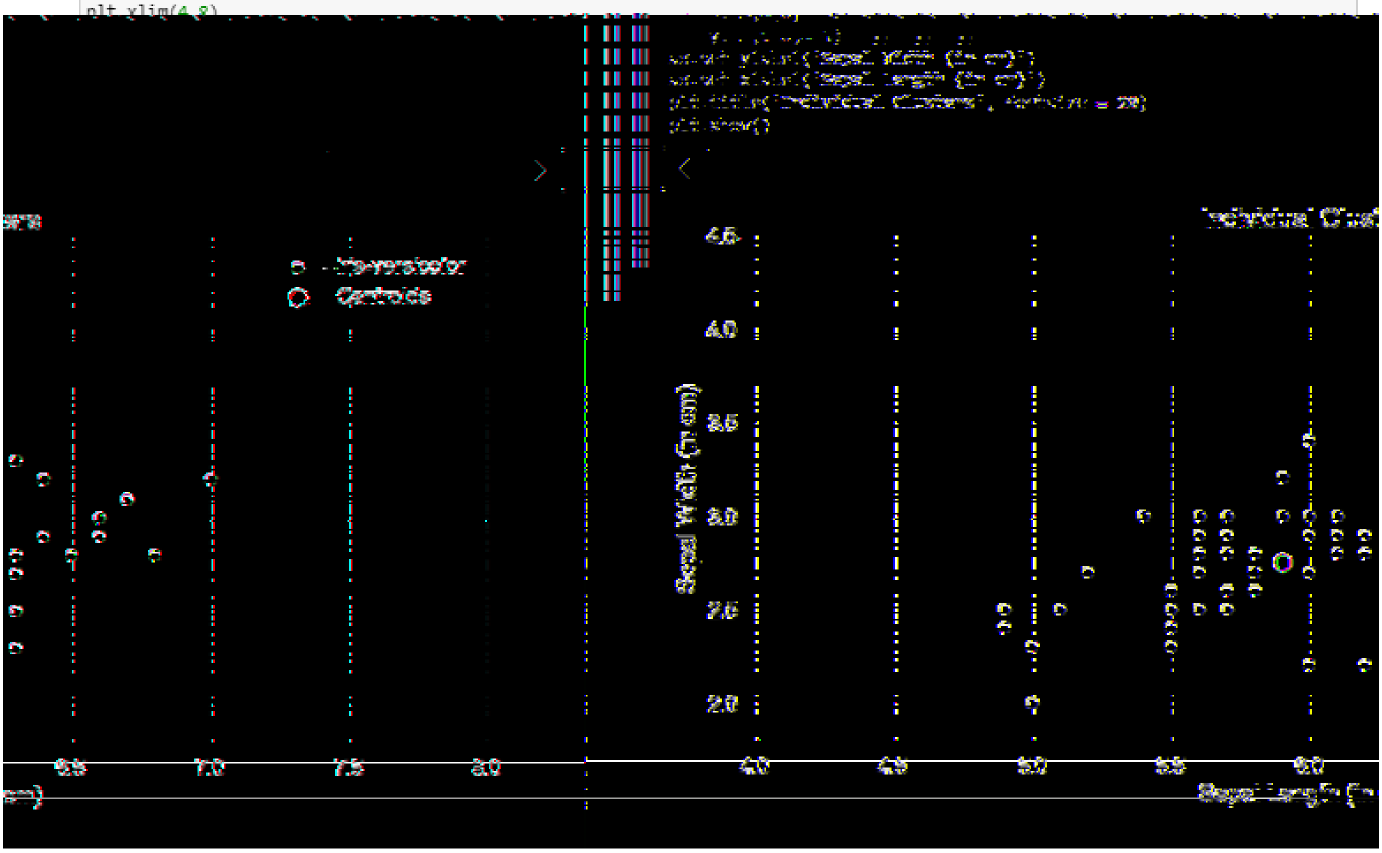
Analyzing Data using the above graph becomes much more easier as it gives us a visual aid for better understanding of the data. Kmeans has divided the dataset into 3 clusters based on Annual income and the spending scores of the individual customers. The following clusters are created by the model,

1. Iris-versicolor
2. Iris-setosa
3. Iris-virginica

1. Iris-versicolor

Iris versicolor is a flowering herbaceous perennial plant, growing 10–80 cm (4–31 in) high. It tends to form large clumps from thick, creeping rhizomes. The unwinged, erect stems generally have basal leaves that are more than 1 cm (½ in) wide. Leaves are folded on the midribs so that they form an overlapping flat fan. The well developed blue flower has 6 petals and sepals spread out nearly flat and have two forms. The longer sepals are hairless and have a greenish-yellow blotch at their base. The inferior ovary is bluntly angled. Flowers are usually light to deep blue (purple and violet are not uncommon) and bloom during May to July. Fruit is a 3-celled, bluntly angled capsule. The large seeds can be observed floating in fall.

9]:
```
fig, ax = plt.subplots(figsize=(15,7))
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 0]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 0]['SepalWidthCm'],
            s=70,edgecolor='teal', linewidth=0.3, c='teal', label='Iris-versicolor')


plt.scatter(x=kms.cluster_centers_[0, 0], y=kms.cluster_centers_[0, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor=
plt.legend(loc='upper right')
plt.xlim(4,8)
```

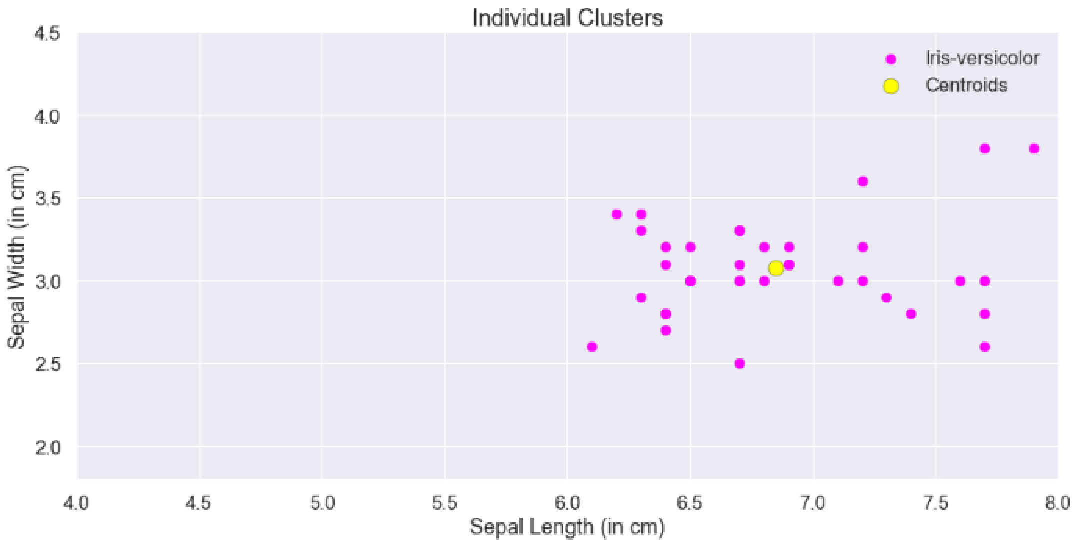


3. Iris-virginica

Iris virginica is a perennial plant. The plant has 2 to 4 erect or arching, bright green, lance-shaped leaves that are flattened into one plane at the base. Leaves are 1–3 cm (½–1 ¼ in) wide and are sometimes longer than the flower stalk. The fleshy roots (1–2 cm or ½–¾ in in diameter) are rhizomes that spread underground. Pale brown, variably shaped seeds are born in three-part fruit capsules (3–6 cm or 1 ¼–2 ¼ in long, 1–2 cm or ½–¾ in wide). The slightly fragrant flowers (4 cm or 1 ½ in long, 7 cm or 2 ¾ in across) consist of 3 horizontal sepals, or "falls", and 3 erect petals. The petals and sepals can vary in color from dark-violet to pinkish-white. The sepals have a splash of yellow to yellow-orange at the crest. Each plant has 2 to 6 flowers that bloom from April to May upon a single, erect, 30–90 cm (12–35 in) tall stalk. The stalk is sometimes branched and has a slight zigzag appearance.

]:

```
fig, ax = plt.subplots(figsize=(15,7))
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 2]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 2]['SepalWidthCm'],
            s=70,edgecolor='magenta', linewidth=0.3, c='magenta', label='Iris-versicolor')


plt.scatter(x=kms.cluster_centers_[2, 0], y=kms.cluster_centers_[2, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor=
plt.legend(loc='upper right')
plt.xlim(4,8)
plt.ylim(1.8,4.5)
ax.set_ylabel('Sepal Width (in cm)')
ax.set_xlabel('Sepal Length (in cm)')
plt.title('Individual Clusters', fontsize = 20)
plt.show()
```



2. Iris-setosa

Iris setosa is similar in form to a miniature Japanese iris, or a dwarf version of Iris sibirica but a shorter lived version. The shallowly rooted, large, branching rhizomes spread over time to create large clumps. The rhizomes are grey-brown, thick, and are covered with old (maroon-brown) fibrous leaf remains (of last seasons leaves). It has branched stems, which are very variable in height, ranging from 10 cm (5 inches) up to 1 m (3 ft) tall. The larger plants can grow beyond the height of the leaves. The roundish stems are between 1.5–9 cm in diameter with 1 to 3 branches. Iris setosa has mid-green leaves, which are grass-like, and lanceolate (sword-shaped). They have a purplish tinged base and the leaves can measure 30–60 cm (12–24 in) long by 0.8–2.5 cm wide. The plant has 3–4 flowers per stem (between 6 and 13 for the whole plant, in groups of 3,) and it blooms between June and July. The large flowers are between 5–8 cm (3–6 in) across, usually 7–8 cm, and come in a range of shades of blue, which can depend on the location. and range from violet, purple-blue, violet-blue, blue, to lavender. Very occasionally, there are pink or white forms.

]:

```
fig, ax = plt.subplots(figsize=(15,7))
plt.scatter(x=clusters[clusters['Cluster_Prediction'] == 1]['SepalLengthCm'],
            y=clusters[clusters['Cluster_Prediction'] == 1]['SepalWidthCm'],
            s=70,edgecolor='lime', linewidth=0.3, c='lime', label='Iris-versicolor')


plt.scatter(x=kms.cluster_centers_[1, 0], y=kms.cluster_centers_[1, 1], s = 170, c = 'yellow', label = 'Centroids',edgecolor=
plt.legend(loc='upper right')
plt.xlim(4,8)
plt.ylim(1.8,4.5)
ax.set_ylabel('Sepal Width (in cm)')
ax.set_xlabel('Sepal Length (in cm)')
plt.title('Individual Clusters', fontsize = 20)
plt.show()
```

Dissimilarity Matrix:

In [65]:

```
iris = load_iris()
data = iris.data
dissimilarity_matrix = pairwise_distances(data, metric='euclidean')
dissimilarity_df = pd.DataFrame(dissimilarity_matrix)
dissimilarity_df.to_csv("iris_dissimilarity_matrix.csv", index=False)
dissimilarity_df
```

Out[65]:

	0	1	2	3	4	5	6	7	8	9	...	140	141	142	143	
0	0.000000	0.538516	0.509902	0.648074	0.141421	0.616441	0.519615	0.173205	0.921954	0.469042	...	5.019960	4.636809	4.208325	5.257376	5.136
1	0.538516	0.000000	0.300000	0.331662	0.608276	1.090871	0.509902	0.424264	0.509902	0.173205	...	5.072475	4.702127	4.180909	5.320714	5.206
2	0.509902	0.300000	0.000000	0.244949	0.509902	1.086278	0.264575	0.412311	0.435890	0.316228	...	5.228767	4.868265	4.334743	5.475400	5.353
3	0.648074	0.331662	0.244949	0.000000	0.648074	1.166190	0.331662	0.500000	0.300000	0.316228	...	5.104900	4.760252	4.177320	5.349766	5.232
4	0.141421	0.608276	0.509902	0.648074	0.000000	0.616441	0.458258	0.223607	0.921954	0.529150	...	5.061620	4.686150	4.246175	5.297169	5.173
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
145	4.654031	4.700000	4.864155	4.745524	4.701064	4.284857	4.796874	4.598913	4.914265	4.666905	...	0.424264	0.244949	1.034408	0.734847	0.616
146	4.276681	4.249706	4.430576	4.288356	4.330127	3.988734	4.384062	4.200000	4.429447	4.220190	...	1.063015	0.943398	0.547723	1.307670	1.284
147	4.459821	4.498889	4.661545	4.533211	4.504442	4.102438	4.593474	4.397727	4.701064	4.457578	...	0.608276	0.519615	0.774597	0.842615	0.793
148	4.650806	4.718050	4.848711	4.719110	4.678675	4.264974	4.749737	4.589118	4.888763	4.672259	...	0.624500	0.818535	0.948683	0.806226	0.624
149	4.140048	4.153312	4.298837	4.149699	4.173727	3.818377	4.217819	4.060788	4.302325	4.106093	...	1.122497	1.122497	0.331662	1.319091	1.256

Reuters Dataset Report

About the dataset:

- It is a collection of documents with news articles. The original corpus has 10,369 documents and a vocabulary of 29,930 words.
- It is a benchmark dataset for document classification.
- It has 90 classes, 7769 training documents and 3019 testing documents .

About the attributes:

- Reuters: it contains boolean values which indicates whether the news document is sourced from Reuters or not. A “yes” value would mean that the article is sourced from Reuters, indicating that Reuters publisher or originator of the document. This means that the information in the article comes from Reuters news agency. On the other hand, a “No” value would mean that the article is not sourced from reuters. In this case, the document may have been sourced from a different news agency .

- Topics: this column contains information about the topics or subjects covered in the news article. It typically includes one or more keywords or labels that describe the main themes or categories of the documents . this column helps classify and organize the articles based on their content.

- Title: this column contains the title or headline of the news article. The title is a concise and informative summary that aims to capture the main idea or focus of the document. It is usually a brief sentence or phrase that is designed to attract readers’ attention and provide them with a preview of the article’s content.

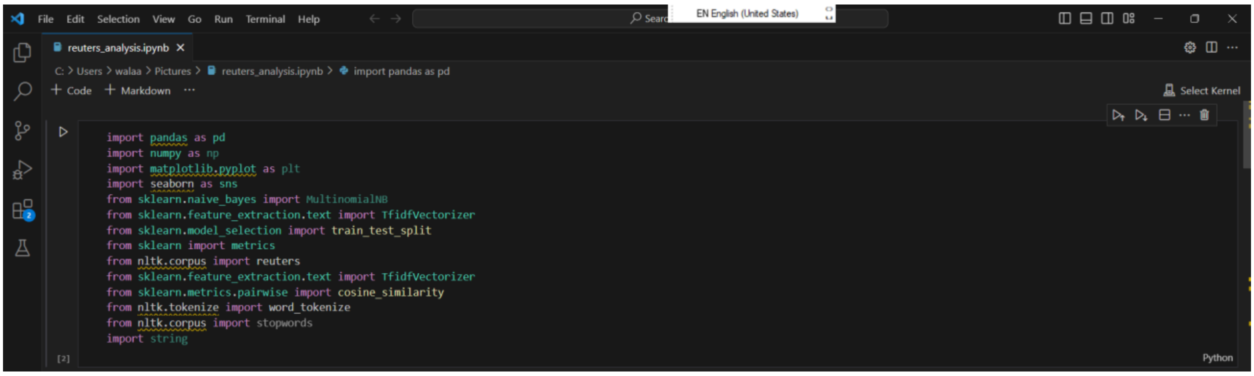
- Body: this column contains the main body of the news article. It consists of the complete text that provides detailed information, context, and analysis of the news story. The body typically includes paragraphs or sections that elaborate on the topics mentioned in the title and may contain quotes, statistics, opinions, and other relevant information related to the article’s subject matter.

○ The combination of the four columns provides comprehensive information about the Reuters dataset, including the source, topics, and content of the news articles, allowing for analysis, categorization and exploration of the dataset based on various criteria.

Code:

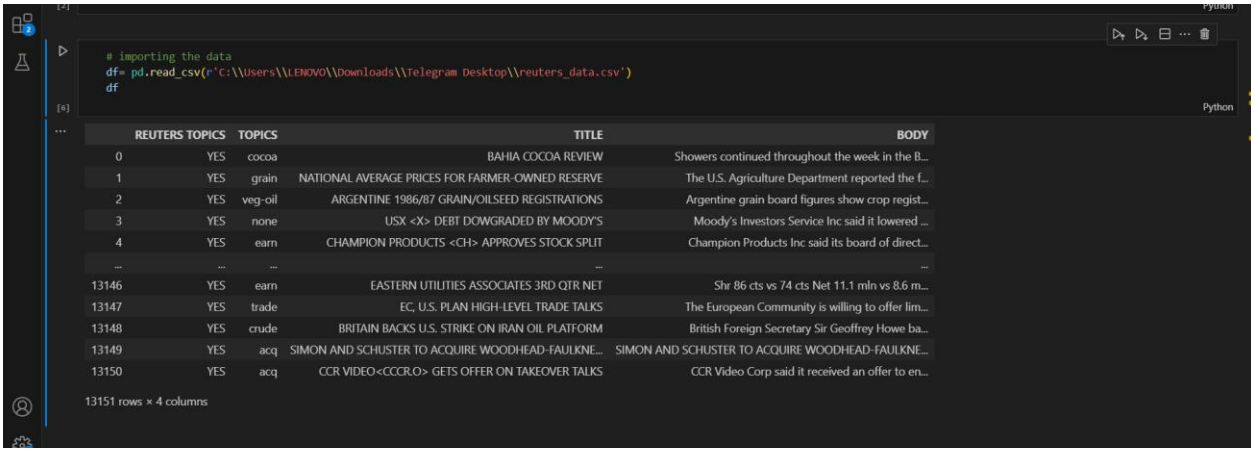
1. Importing libraries

● First, we imported libraries that we will use:

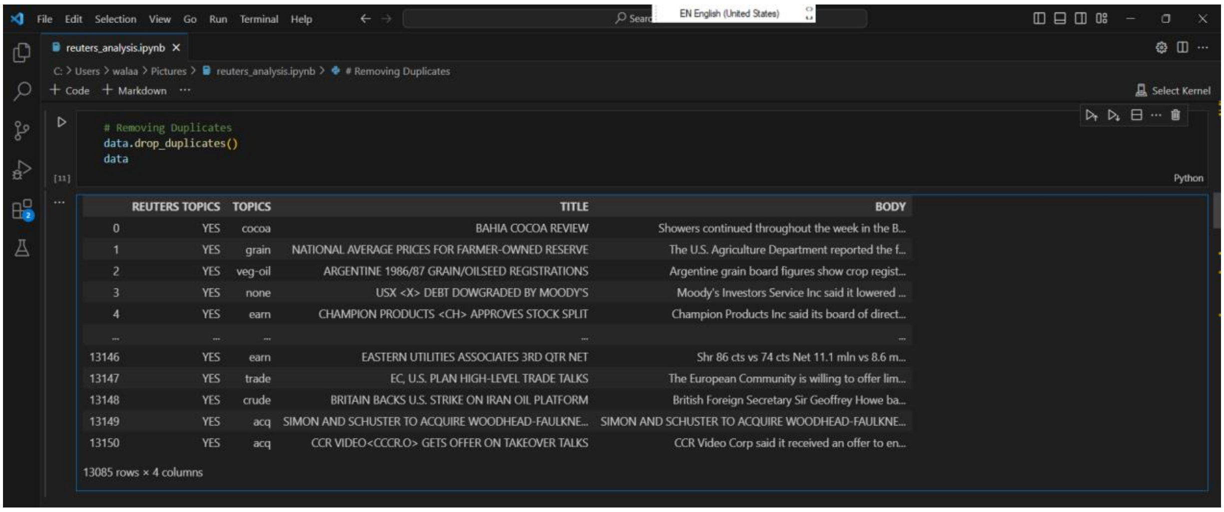


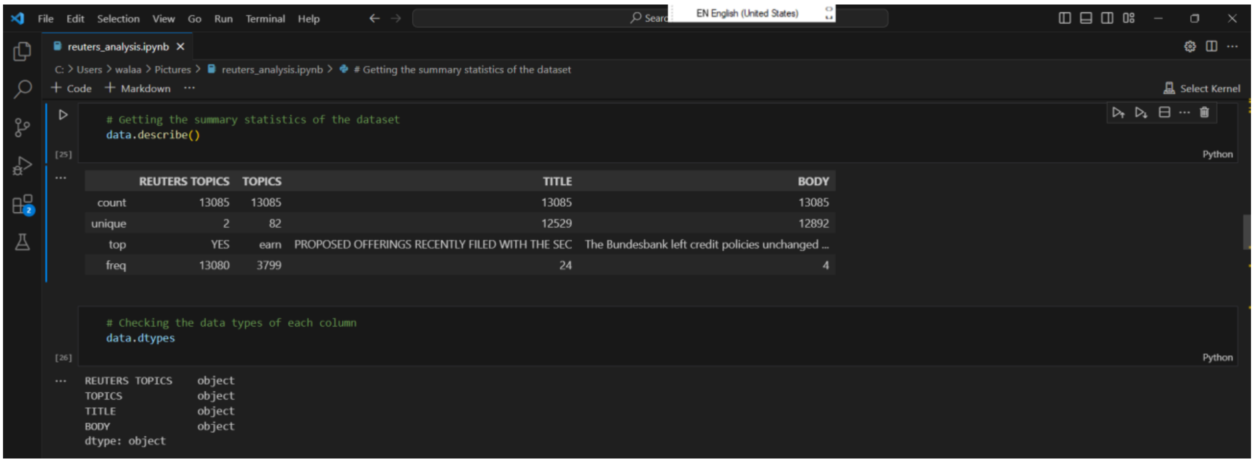
2. Cleaning and Exploring the Data

● Next, we imported our dataset and printed it:



- After that we made cleaning and exploratory analysis on our data:

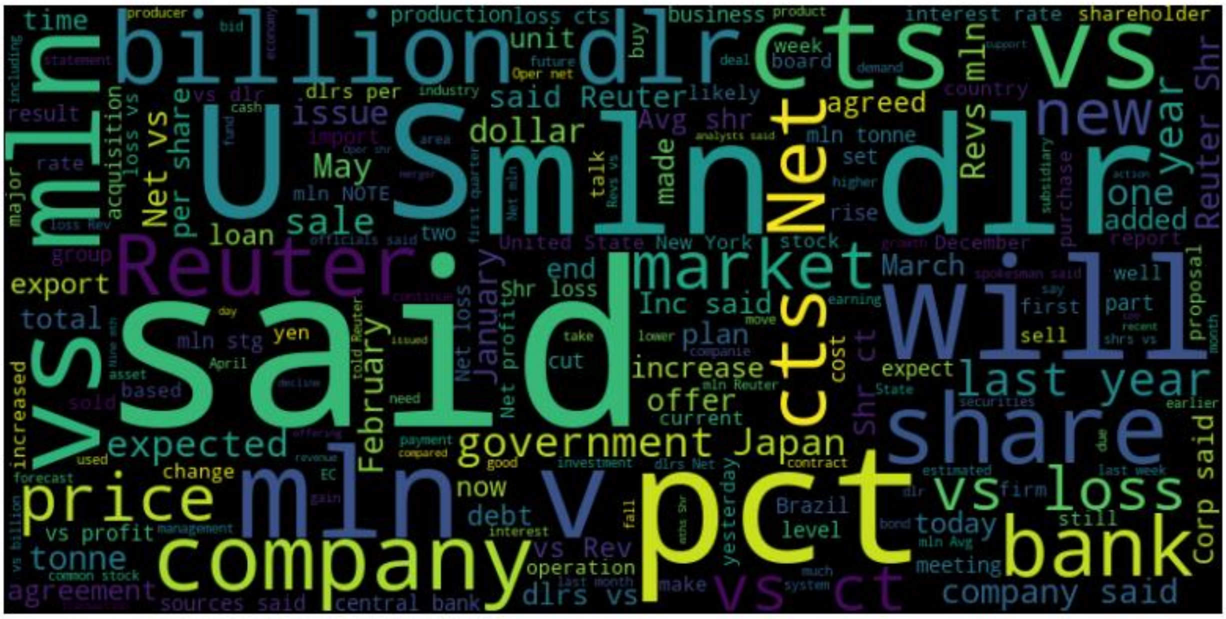




3. Visualization

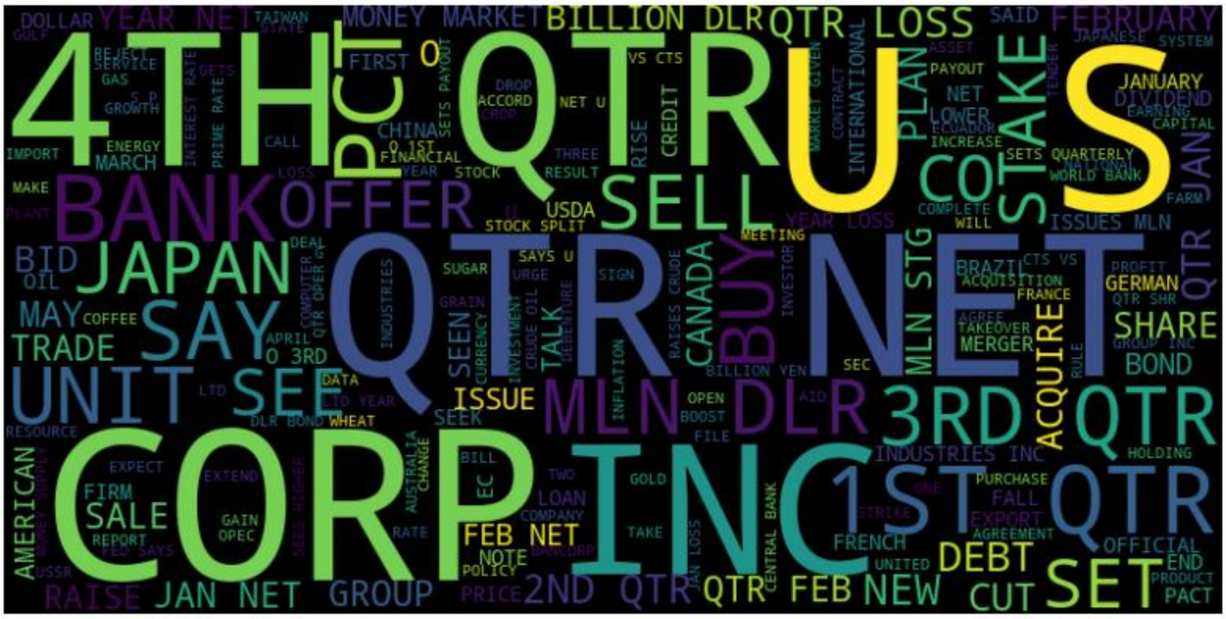
- Then we created a word cloud of the “BODY” column

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

# Combine all text data into a single string
text = ' '.join(data['BODY'])

# Create a word cloud( to identify the most frequent words in the text)
wordcloud = WordCloud(width=800, height=400, max_font_size=150).generate(text)

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

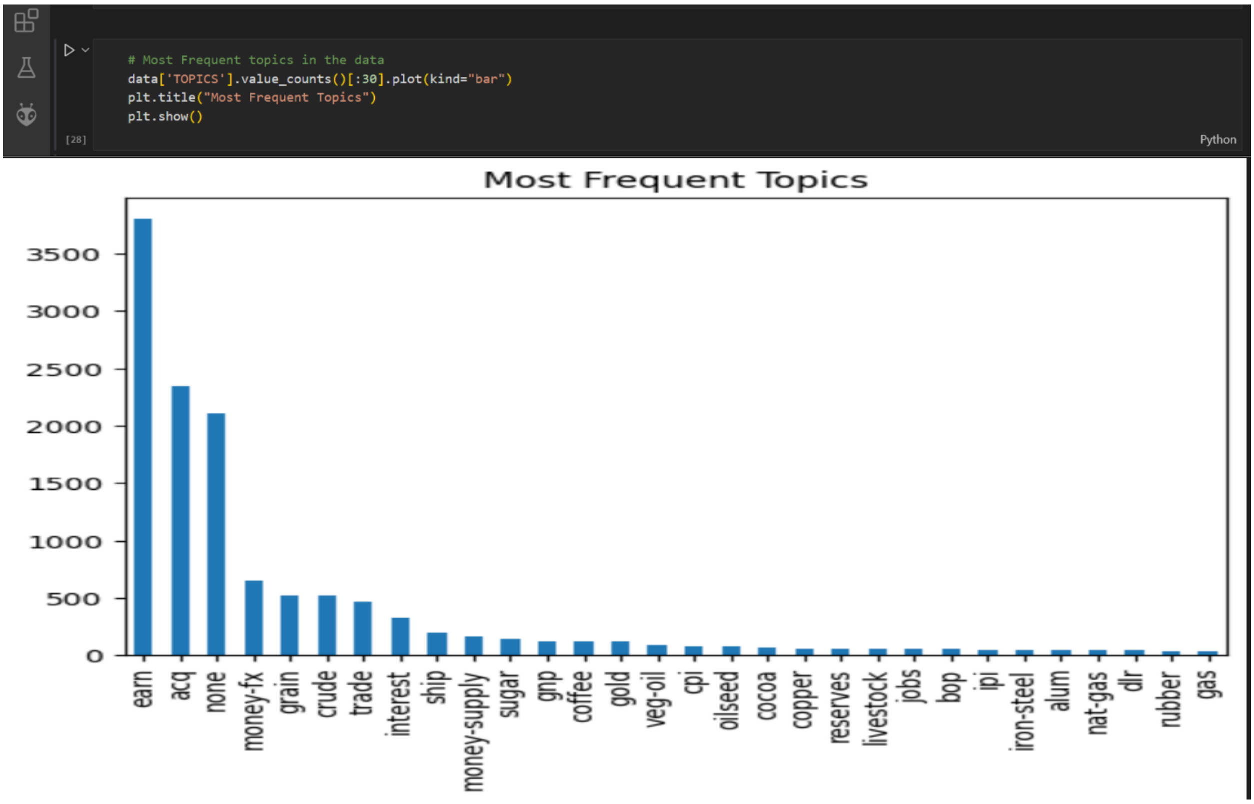


- Then we created a word cloud of the “Title” column

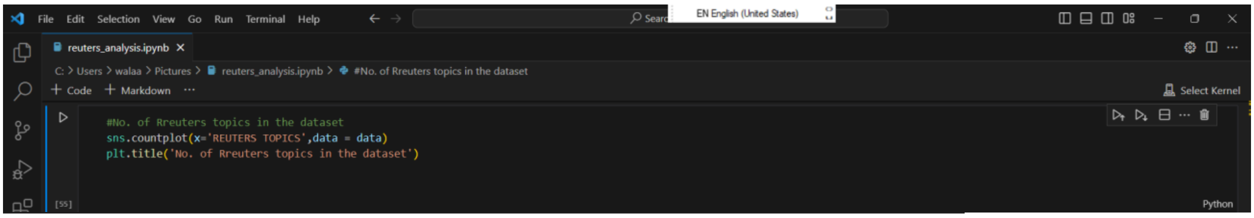
```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all text data into a single string
text = ' '.join(data['TITLE'])

# Create a word cloud
wordcloud = WordCloud(width=800, height=400, max_font_size=150).generate(text)

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

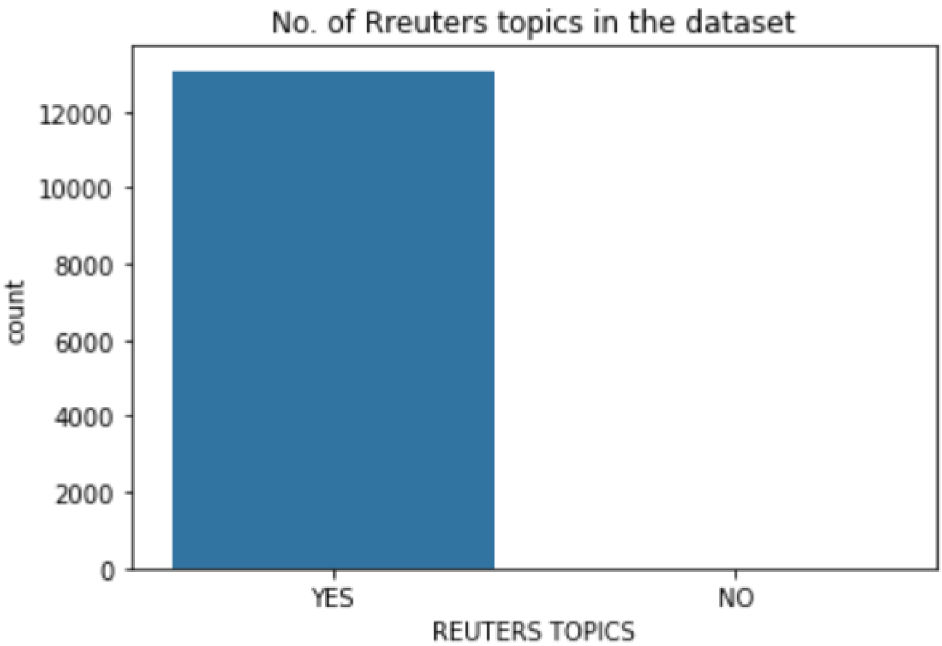


- Then we created a graph of the most frequent Topics

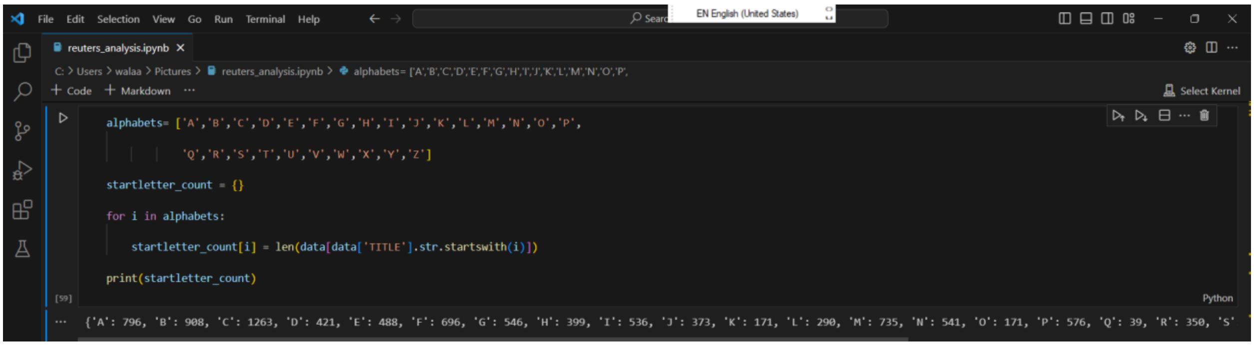


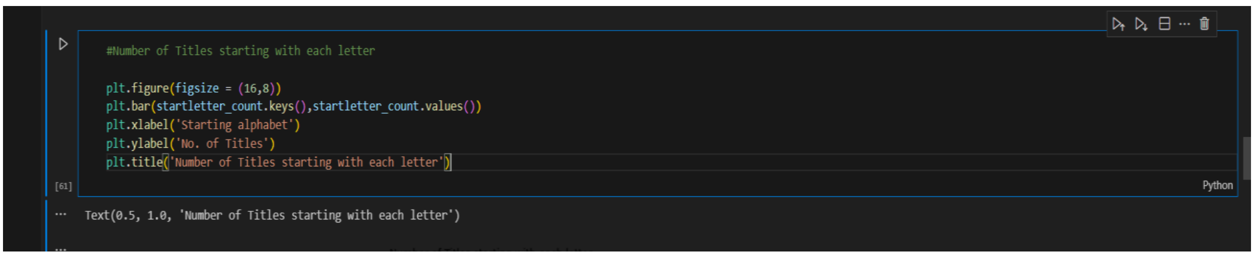
- we made a visualization that shows no. of Reuters topics in the dataset:

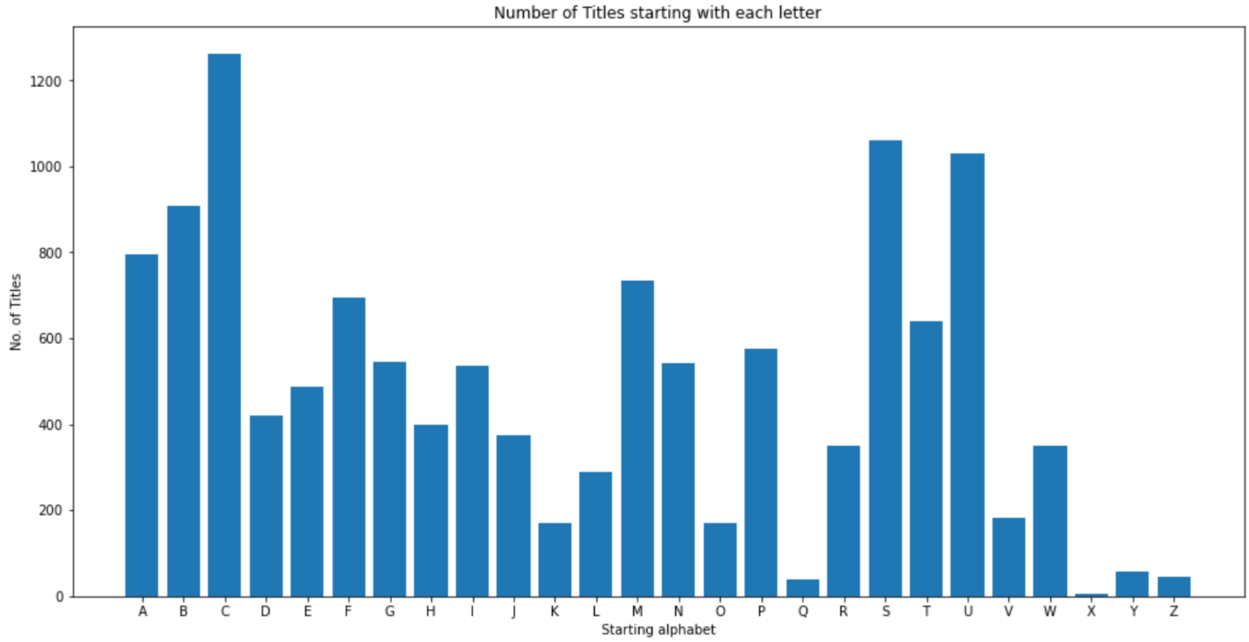




- We created a dictionary that contained each letter in the alphabet and how many titles start with that letter:







- We created a pandas series. its index contains all the words in the "BODY" column and its values is how many times this word appeared in the "BODY" column:

```python
# frequency of each word in the "BODY" column
ser = pd.Series(' '.join(data['BODY']).split()).value_counts()
ser
```

```
the            74852
of             46923
to             43448
and            33608
in             32750
               ...
99.78              1
19,186,000         1
12,438,000         1
157.9              1
Intercep           1
Length: 83811, dtype: int64
```

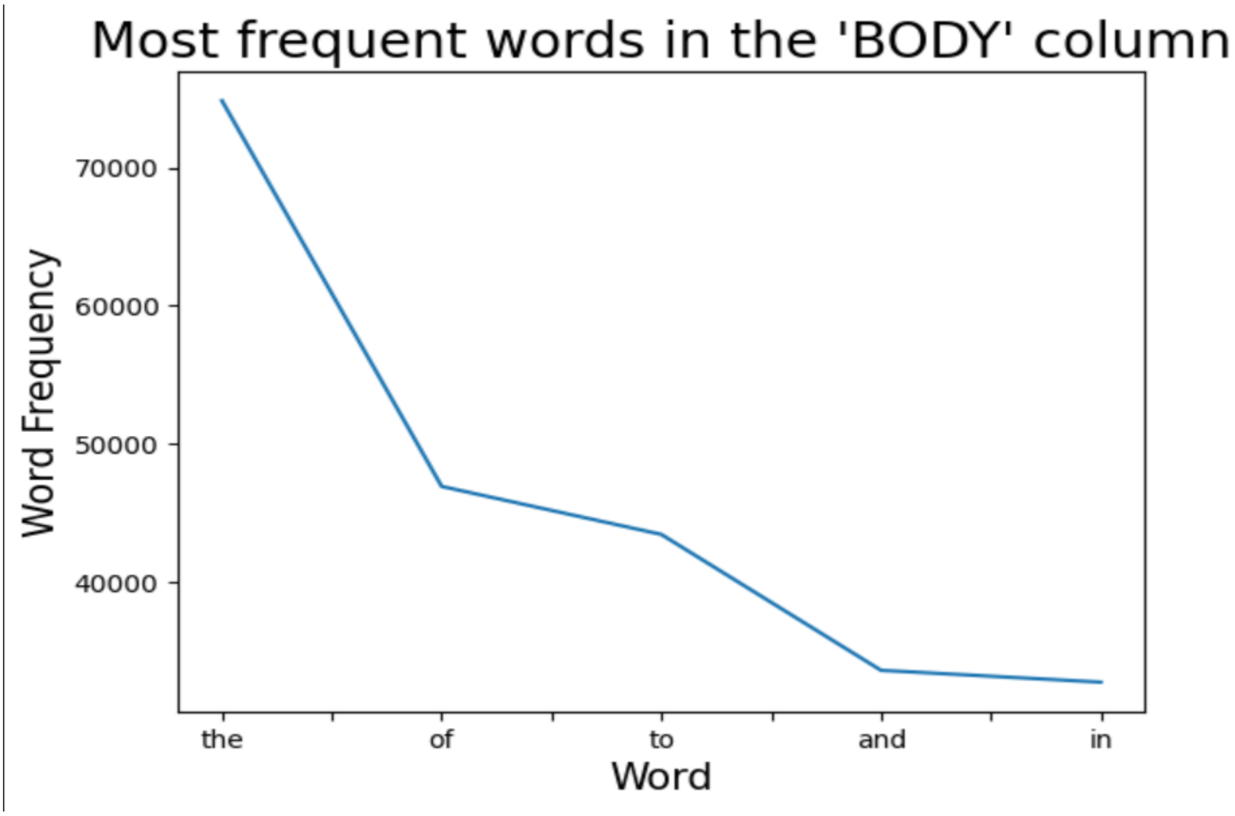
- We then selected the first five elements in the series:

```python
#Most frequent words in the 'BODY' column
topFiveWords = (ser[:5])
topFiveWords
```

```
the    74852
of     46923
to     43448
and    33608
in     32750
dtype: int64
```

- Then we plotted the most frequent words:

```python
topFiveWords.plot(x="words", y="frequency")
plt.xlabel("Word",  size = 15)
plt.ylabel("Word Frequency", size = 15)
plt.title("Most frequent words in the 'BODY' column", size = 20)
```

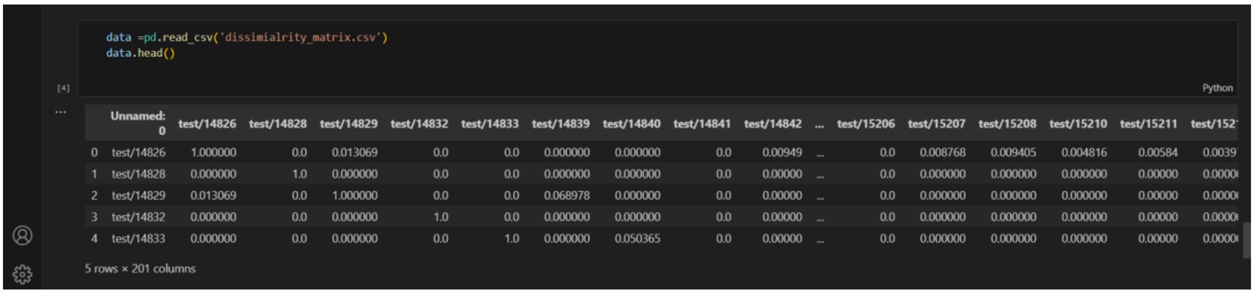


4. Dissimilarity Matrix

- Then we made a subset of our dataset to make the dissimilarity matrix:

```python
subset_doc =reuters.fileids()[:200] # Choosing 200 documents as a subset
subset_corpus=[''.join(word_tokenize(reuters.raw(doc_id).lower())) for doc_id in subset_doc]

# Use TF-IDF Vectorizer for the subset
tfidf_vectorizer =TfidfVectorizer(stop_words='english')
tfidf_matrix =tfidf_vectorizer.fit_transform(subset_corpus)

# Compute the cosine similarity matrix for the subset
similarity_matrix = cosine_similarity(tfidf_matrix , tfidf_matrix)

# Convert similarity matrix to a DataFrame
dataFrame_similarity =pd.DataFrame(similarity_matrix , index= subset_doc ,columns =subset_doc)

dataFrame_similarity.to_csv('dissimialrity_matrix.csv')
```

- Then we printed the dissimilarity matrix:



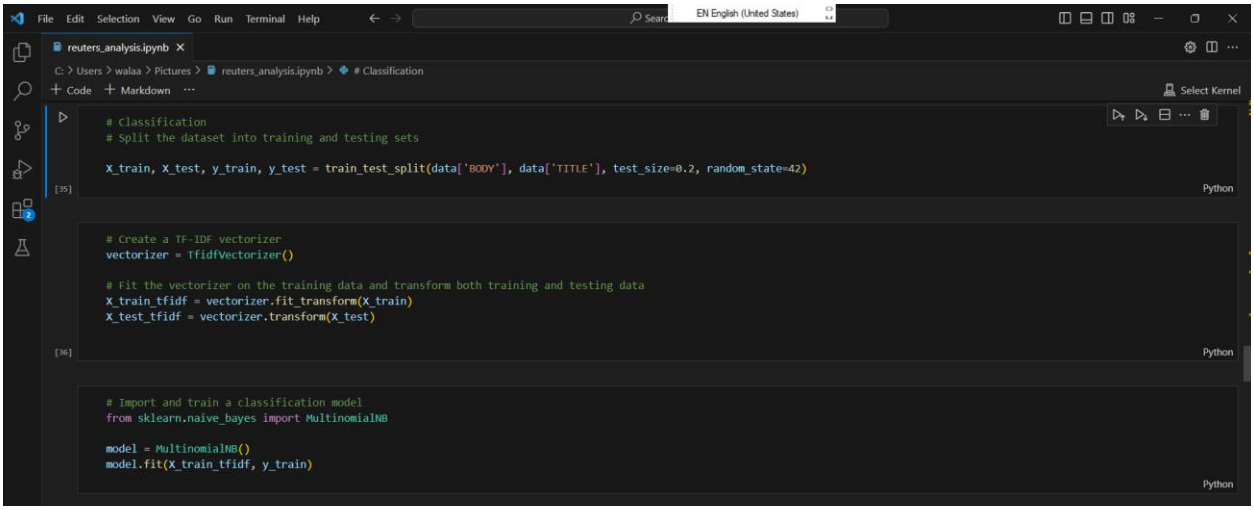
```
data =pd.read_csv('dissimialrity_matrix.csv')
data.head()
```

	Unnamed: 0	test/14826	test/14828	test/14829	test/14832	test/14833	test/14839	test/14840	test/14841	test/14842	...	test/15206	test/15207	test/15208	test/15210	test/15211	test/152
0	test/14826	1.000000	0.0	0.013069	0.0	0.0	0.000000	0.000000	0.0	0.00949	...	0.0	0.008768	0.009405	0.004816	0.00584	0.0039
1	test/14828	0.000000	1.0	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.00000	...	0.0	0.000000	0.000000	0.000000	0.00000	0.0000
2	test/14829	0.013069	0.0	1.000000	0.0	0.0	0.068978	0.000000	0.0	0.00000	...	0.0	0.000000	0.000000	0.000000	0.00000	0.0000
3	test/14832	0.000000	0.0	0.000000	1.0	0.0	0.000000	0.000000	0.0	0.00000	...	0.0	0.000000	0.000000	0.000000	0.00000	0.0000
4	test/14833	0.000000	0.0	0.000000	0.0	1.0	0.000000	0.050365	0.0	0.00000	...	0.0	0.000000	0.000000	0.000000	0.00000	0.0000

5 rows × 201 columns

5. Classification Model

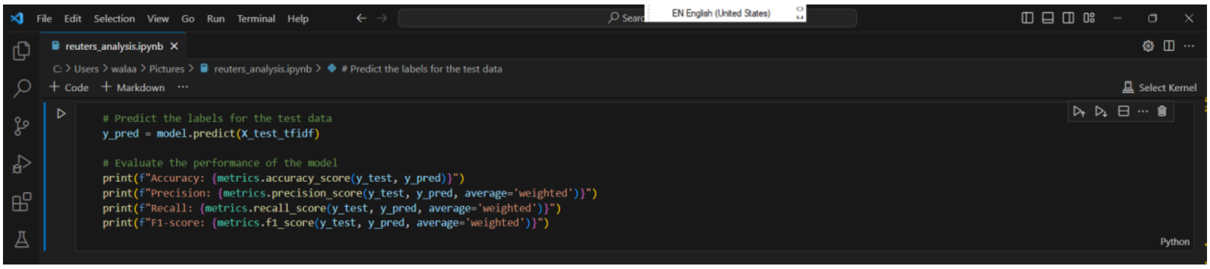
- We made a classification model, we split the data into training data and test data and made our classification model based on this partitioning:



```
# Classification
# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(data['BODY'], data['TITLE'], test_size=0.2, random_state=42)
```

```
# Create a TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit the vectorizer on the training data and transform both training and testing data
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

```
# Import and train a classification model
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train_tfidf, y_train)
```



```
# Predict the labels for the test data
y_pred = model.predict(X_test_tfidf)

# Evaluate the performance of the model
print(f"Accuracy: {metrics.accuracy_score(y_test, y_pred)}")
print(f"Precision: {metrics.precision_score(y_test, y_pred, average='weighted')}")
print(f"Recall: {metrics.recall_score(y_test, y_pred, average='weighted')}")
print(f"F1-score: {metrics.f1_score(y_test, y_pred, average='weighted')}")
```