5. Market basket Analysis and Visualization

Aim: To perform market basket analysis and visualization on the given dataset.

```
# This Python 3 environment comes with many helpful analytics libraries
installed
# It is defined by the kaggle/python docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list
all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/market-basket-optimization/Market_Basket_Optimisation.csv
```

Import Dataset

```
data = pd.read_csv('/kaggle/input/market-basket-
optimization/Market_Basket_Optimisation.csv', header=None)
```

Check Head of Dataset

```
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	1
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	sa
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

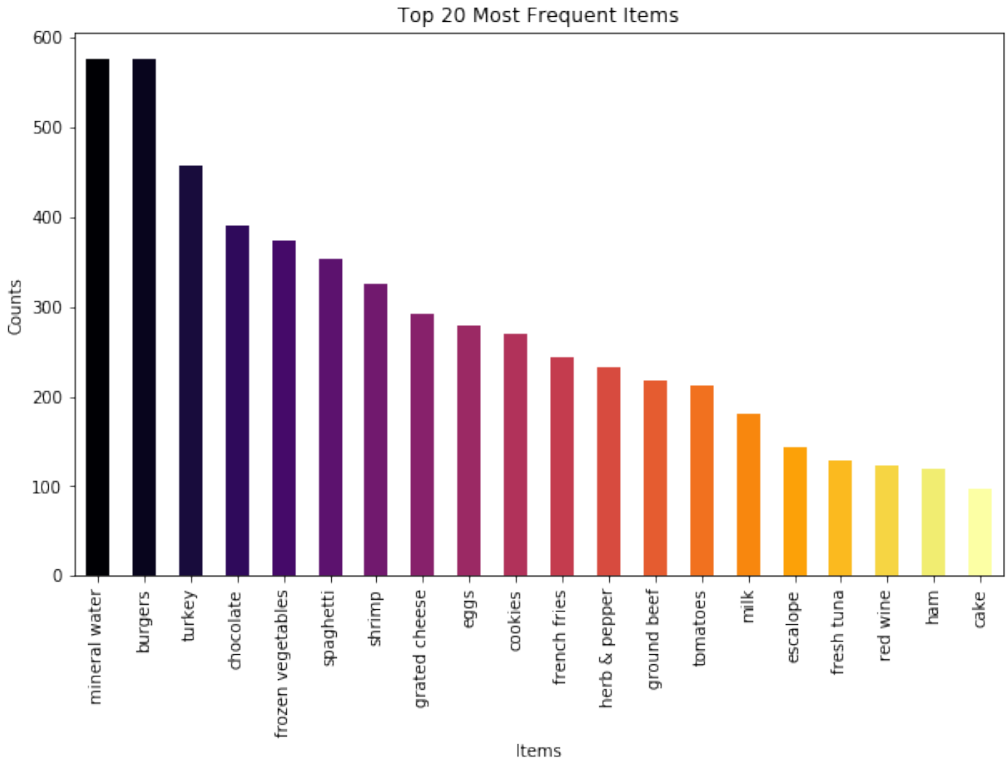
```
data.shape # Dataset Shape and Size
```

```
(7501, 20)
```

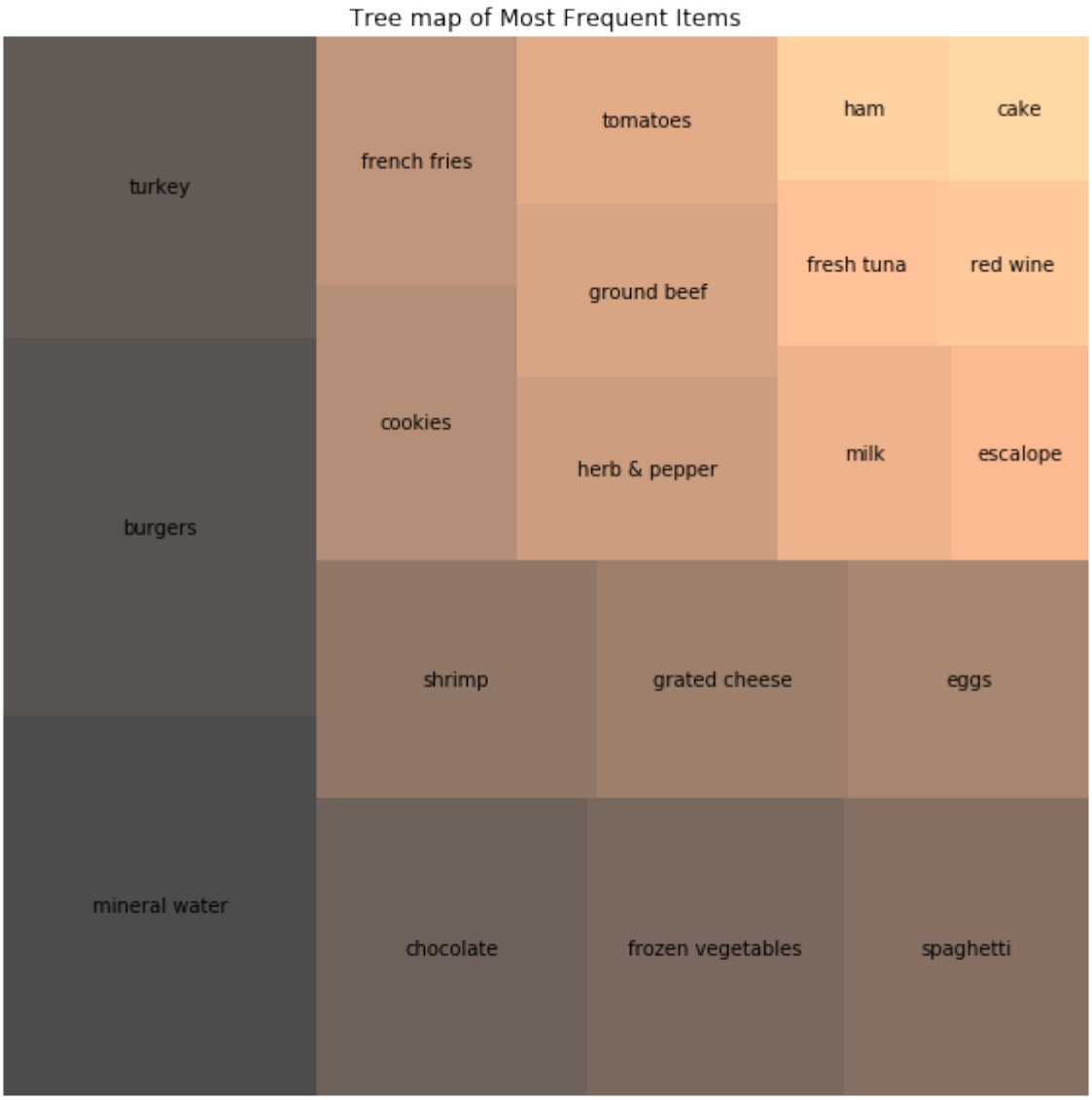
Visualization

```
# Most Frequent Items Bar plot
```

```
plt.rcParams['figure.figsize'] = (10,6)
color = plt.cm.inferno(np.linspace(0,1,20))
data[0].value_counts().head(20).plot.bar(color = color)
plt.title('Top 20 Most Frequent Items')
plt.ylabel('Counts')
plt.xlabel('Items')
plt.show()
```




```
# Tree Map of Most Frequent Items
import squarify
plt.rcParams['figure.figsize']=(10,10)
Items = data[0].value_counts().head(20).to_frame()
size = Items[0].values
lab = Items.index
color = plt.cm.copper(np.linspace(0,1,20))
squarify.plot(sizes=size, label=lab, alpha = 0.7, color=color)
plt.title('Tree map of Most Frequent Items')
plt.axis('off')
plt.show()
```




```
data['Items'] = 'items'
df = data.truncate(before=-1,after=15)

import networkx as nx

Items = nx.from_pandas_edgelist(df, source = 'Items', target = 0, edge_attr =
True)

plt.rcParams['figure.figsize'] = (20,20)
nx.draw_networkx_nodes(G=Items,pos=nx.spring_layout(Items),
node_size=15000,node_color='green')
nx.draw_networkx_edges(G=Items,pos=nx.spring_layout(Items), alpha=0.6,
width=3 ,edge_color='black')
nx.draw_networkx_labels(G=Items,pos=nx.spring_layout(Items),font_size=20,
font_family = 'sans-serif')
plt.axis('off')
plt.grid()
plt.title('Top 15 First Choices', fontsize = 20)
plt.show()



data.drop(columns='Items',axis=1, inplace=True)
```

```
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	1
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	sa
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Association Analysis And Apriori Algorithm

```
# list of list is needed as an input for transaction encoder
transactions = []
for i in range(0,7501):
    transactions.append([str(data.values[i,j]) for j in range(0,20)])

# TransactionEncoder learns the unique labels in the dataset, and via the
transform method,
# it transforms the input dataset (a Python list of lists) into a one-hot
encoded NumPy boolean array:

from mlxtend.preprocessing import TransactionEncoder
transac = TransactionEncoder()
dataset = transac.fit_transform(transactions)
dataset
```

```
array([[False,  True,  True, ...,  True, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False,  True, False]])
```

```
df = pd.DataFrame(dataset, columns= transac.columns_)
df.head()
```

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	bacon	barbecue sauce	black tea	blueberries	...	tu
0	False	True	True	False	True	False	False	False	False	False	...	Fa
1	False	False	False	False	False	False	False	False	False	False	...	Fa
2	False	False	False	False	False	False	False	False	False	False	...	Fa
3	False	False	False	False	True	False	False	False	False	False	...	Tr
4	False	False	False	False	False	False	False	False	False	False	...	Fa

5 rows × 121 columns

```
#Apriori Algorithm to find out most frequent itemset with min support of 0.003
```

```
from mlxtend.frequent_patterns import apriori, association_rules
frequent_itemsets = apriori(df, min_support=0.003, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x :
len(x))

frequent_itemsets.head(10)
```

	support	itemsets	length
0	0.020397	(almonds)	1
1	0.008932	(antioxydant juice)	1
2	0.004666	(asparagus)	1
3	0.033329	(avocado)	1
4	0.004533	(babies food)	1
5	0.008666	(bacon)	1
6	0.010799	(barbecue sauce)	1
7	0.014265	(black tea)	1
8	0.009199	(blueberries)	1
9	0.011465	(body spray)	1
10	0.033729	(brownies)	1
11	0.008666	(bug spray)	1
12	0.005866	(burger sauce)	1
13	0.087188	(burgers)	1
14	0.030129	(butter)	1
15	0.081056	(cake)	1
16	0.009732	(candy bars)	1
17	0.015331	(carrots)	1
18	0.004799	(cauliflower)	1
19	0.025730	(cereals)	1

```
#Filter Frequent itemset of minimum length 2

frequent_itemsets[frequent_itemsets['length'] >= 2].head(20)
```

	support	itemsets	length
116	0.005199	(burgers, almonds)	2
117	0.003066	(almonds, cake)	2
118	0.005999	(almonds, chocolate)	2
119	0.006532	(almonds, eggs)	2
120	0.004399	(french fries, almonds)	2
121	0.003066	(almonds, frozen vegetables)	2
122	0.005066	(almonds, green tea)	2
123	0.003866	(ground beef, almonds)	2
124	0.005199	(almonds, milk)	2
125	0.007599	(almonds, mineral water)	2
126	0.020264	(almonds, nan)	2
127	0.003066	(pancakes, almonds)	2

	support	itemsets	length
128	0.005999	(almonds, spaghetti)	2
129	0.008799	(antioxydant juice, nan)	2
130	0.004666	(nan, asparagus)	2
131	0.004533	(burgers, avocado)	2
132	0.004266	(cake, avocado)	2
133	0.007066	(avocado, chocolate)	2
134	0.005599	(eggs, avocado)	2
135	0.007999	(french fries, avocado)	2

```
# Association Rules Mining to generate the rules with their coresponding support

rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
rules.head(50)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convictio
0	(burgers)	(almonds)	0.087188	0.020397	0.005199	0.059633	2.923577	3.420901e-03	1.041724
1	(almonds)	(burgers)	0.020397	0.087188	0.005199	0.254902	2.923577	3.420901e-03	1.225089
2	(almonds)	(cake)	0.020397	0.081056	0.003066	0.150327	1.854607	1.412939e-03	1.081527
3	(cake)	(almonds)	0.081056	0.020397	0.003066	0.037829	1.854607	1.412939e-03	1.018117
4	(almonds)	(chocolate)	0.020397	0.163845	0.005999	0.294118	1.795099	2.657211e-03	1.184553
5	(chocolate)	(almonds)	0.163845	0.020397	0.005999	0.036615	1.795099	2.657211e-03	1.016834
6	(almonds)	(eggs)	0.020397	0.179709	0.006532	0.320261	1.782108	2.866880e-03	1.206774
7	(eggs)	(almonds)	0.179709	0.020397	0.006532	0.036350	1.782108	2.866880e-03	1.016555
8	(french fries)	(almonds)	0.170911	0.020397	0.004399	0.025741	1.261983	9.133031e-04	1.005485
9	(almonds)	(french fries)	0.020397	0.170911	0.004399	0.215686	1.261983	9.133031e-04	1.057089
10	(almonds)	(frozen vegetables)	0.020397	0.095321	0.003066	0.150327	1.577065	1.121976e-03	1.064738
11	(frozen vegetables)	(almonds)	0.095321	0.020397	0.003066	0.032168	1.577065	1.121976e-03	1.012162
12	(almonds)	(green tea)	0.020397	0.132116	0.005066	0.248366	1.879913	2.371190e-03	1.154663
13	(green tea)	(almonds)	0.132116	0.020397	0.005066	0.038345	1.879913	2.371190e-03	1.018663
14	(ground beef)	(almonds)	0.098254	0.020397	0.003866	0.039349	1.929116	1.862046e-03	1.019728
15	(almonds)	(ground	0.020397	0.098254	0.003866	0.189542	1.929116	1.862046e-	1.112639

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convictio
		beef)						03	
16	(almonds)	(milk)	0.020397	0.129583	0.005199	0.254902	1.967098	2.556172e-03	1.168192
17	(milk)	(almonds)	0.129583	0.020397	0.005199	0.040123	1.967098	2.556172e-03	1.020551
18	(almonds)	(mineral water)	0.020397	0.238368	0.007599	0.372549	1.562914	2.736923e-03	1.213851
19	(mineral water)	(almonds)	0.238368	0.020397	0.007599	0.031879	1.562914	2.736923e-03	1.011860
20	(pancakes)	(almonds)	0.095054	0.020397	0.003066	0.032258	1.581489	1.127415e-03	1.012256
21	(almonds)	(pancakes)	0.020397	0.095054	0.003066	0.150327	1.581489	1.127415e-03	1.065052
22	(almonds)	(spaghetti)	0.020397	0.174110	0.005999	0.294118	1.689262	2.447827e-03	1.170011
23	(spaghetti)	(almonds)	0.174110	0.020397	0.005999	0.034456	1.689262	2.447827e-03	1.014561
24	(nan)	(asparagus)	0.999867	0.004666	0.004666	0.004667	1.000133	6.220563e-07	1.000001
25	(asparagus)	(nan)	0.004666	0.999867	0.004666	1.000000	1.000133	6.220563e-07	inf
26	(burgers)	(avocado)	0.087188	0.033329	0.004533	0.051988	1.559841	1.626837e-03	1.019682
27	(avocado)	(burgers)	0.033329	0.087188	0.004533	0.136000	1.559841	1.626837e-03	1.056495
28	(cake)	(avocado)	0.081056	0.033329	0.004266	0.052632	1.579158	1.564596e-03	1.020375
29	(avocado)	(cake)	0.033329	0.081056	0.004266	0.128000	1.579158	1.564596e-03	1.053835
30	(avocado)	(chocolate)	0.033329	0.163845	0.007066	0.212000	1.293907	1.604959e-03	1.061111
31	(chocolate)	(avocado)	0.163845	0.033329	0.007066	0.043124	1.293907	1.604959e-03	1.010237
32	(french fries)	(avocado)	0.170911	0.033329	0.007999	0.046802	1.404243	2.302675e-03	1.014135
33	(avocado)	(french fries)	0.033329	0.170911	0.007999	0.240000	1.404243	2.302675e-03	1.090907
34	(avocado)	(frozen smoothie)	0.033329	0.063325	0.005066	0.152000	2.400320	2.955443e-03	1.104570
35	(frozen smoothie)	(avocado)	0.063325	0.033329	0.005066	0.080000	2.400320	2.955443e-03	1.050729
36	(frozen vegetables)	(avocado)	0.095321	0.033329	0.004666	0.048951	1.468727	1.489114e-03	1.016426
37	(avocado)	(frozen vegetables)	0.033329	0.095321	0.004666	0.140000	1.468727	1.489114e-03	1.051953

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convictio
38	(green tea)	(avocado)	0.132116	0.033329	0.006266	0.047427	1.422995	1.862561e-03	1.014800
39	(avocado)	(green tea)	0.033329	0.132116	0.006266	0.188000	1.422995	1.862561e-03	1.068823
40	(ground beef)	(avocado)	0.098254	0.033329	0.003733	0.037992	1.139908	4.581534e-04	1.004847
41	(avocado)	(ground beef)	0.033329	0.098254	0.003733	0.112000	1.139908	4.581534e-04	1.015480
42	(avocado)	(honey)	0.033329	0.047460	0.003200	0.096000	2.022742	1.617773e-03	1.053694
43	(honey)	(avocado)	0.047460	0.033329	0.003200	0.067416	2.022742	1.617773e-03	1.036551
44	(milk)	(avocado)	0.129583	0.033329	0.007466	0.057613	1.728626	3.146823e-03	1.025769
45	(avocado)	(milk)	0.033329	0.129583	0.007466	0.224000	1.728626	3.146823e-03	1.121672
46	(avocado)	(mineral water)	0.033329	0.238368	0.011598	0.348000	1.459926	3.653906e-03	1.168147
47	(mineral water)	(avocado)	0.238368	0.033329	0.011598	0.048658	1.459926	3.653906e-03	1.016113
48	(olive oil)	(avocado)	0.065858	0.033329	0.003466	0.052632	1.579158	1.271234e-03	1.020375
49	(avocado)	(olive oil)	0.033329	0.065858	0.003466	0.104000	1.579158	1.271234e-03	1.042569

```
rules[(rules['lift'] >= 5) & (rules['confidence'] >= 0.4)]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convic
8192	(whole wheat pasta, mineral water)	(olive oil)	0.009599	0.065858	0.003866	0.402778	6.115863	0.003234	1.5641
15899	(nan, whole wheat pasta, mineral water)	(olive oil)	0.009599	0.065858	0.003866	0.402778	6.115863	0.003234	1.5641
15905	(whole wheat pasta, mineral water)	(olive oil, nan)	0.009599	0.065725	0.003866	0.402778	6.128268	0.003235	1.5643

Conclusion

Association analysis is easy to run and relatively easy to interpret.

References

https://www.kaggle.com/code/rgalkar/market-basket-analysis-and-visualization
https://www.kaggle.com/code/benroshan/market-basket-analysis

6. Visualization of Text Data

Aim: To visualize text data.

Visually representing the content of a text document is one of the most important tasks in the field of data visualization. As a data scientist, not only we explore the content of documents from different aspects and at different levels of details, but also we summarize a single document, show the words and topics, detect events, and create storylines.

we will use Womens Clothing E-Commerce Reviews data set, and try to explore and visualize. Not only we are going to explore text data, but also we will visualize numeric and categorical features.

Dataset

This dataset includes 23486 rows and 10 feature variables. Each row corresponds to a customer review, and includes the variables:

- **Clothing ID:** Integer Categorical variable that refers to the specific piece being reviewed.
- **Age:** Positive Integer variable of the reviewers age.
- **Title:** String variable for the title of the review.
- **Review Text:** String variable for the review body.
- **Rating:** Positive Ordinal Integer variable for the product score granted by the customer from 1 Worst, to 5 Best.
- **Recommended IND:** Binary variable stating where the customer recommends the product where 1 is recommended, 0 is not recommended.
- **Positive Feedback Count:** Positive Integer documenting the number of other customers who found this review positive.
- **Division Name:** Categorical name of the product high level division.
- **Department Name:** Categorical name of the product department name.
- **Class Name:** Categorical name of the product class name.

Import required modules

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

from textblob import TextBlob
from sklearn.feature_extraction.text import CountVectorizer
```

```
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('../input/womens-ecommerce-clothing-reviews/Womens Clothing E-Commerce Reviews.csv')
df.head()
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Departmen Name
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmates	Intimate
1	1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	Dresses
2	2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	Dresses
3	3	1049	50	My favorite buy!	I love, love, love this jumpsuit. it's fun, fl...	5	1	0	General Petite	Bottoms
4	4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	Tops

Let's remove null values and unnecessary columns

```
df.isnull().sum()
```

```
Unnamed: 0                    0
Clothing ID                   0
Age                           0
Title                      3810
Review Text                 845
Rating                        0
Recommended IND               0
Positive Feedback Count       0
Division Name                14
Department Name              14
Class Name                   14
dtype: int64
```

```
df.drop(['Unnamed: 0', 'Title'], axis=1, inplace=True)
df.dropna(inplace=True)
```

```
df.isnull().sum()
```

```
Clothing ID                0
Age                        0
Review Text                0
Rating                     0
Recommended IND            0
Positive Feedback Count    0
Division Name              0
Department Name            0
Class Name                 0
dtype: int64
```

Preprocess the data

```
def preprocess(ReviewText):
    ReviewText = ReviewText.str.replace("(<br/>)", "")
    ReviewText = ReviewText.str.replace('(<a).*(>).*(</a>)', '')
    ReviewText = ReviewText.str.replace('(&amp)', '')
    ReviewText = ReviewText.str.replace('(&gt)', '')
    ReviewText = ReviewText.str.replace('(&lt)', '')
    ReviewText = ReviewText.str.replace('(\xa0)', ' ')
    return ReviewText

df['Review Text'] = preprocess(df['Review Text'])
```

Using TextBlob to calculate sentiment polarity which lies in the range of [-1,1] where 1 means positive sentiment and -1 means a negative sentiment.

and also calculating word counts and review length.

```
df['Polarity'] = df['Review Text'].apply(lambda x:
TextBlob(x).sentiment.polarity)
df['word_count'] = df['Review Text'].apply(lambda x: len(str(x).split()))
df['review_len'] = df['Review Text'].apply(lambda x: len(str(x)))
```

Let's check most Postive, Neutral and Negative polarity reviews

Polarity == 1

```
cl = df.loc[df.Polarity == 1, ['Review Text']].sample(5).values
for c in cl:
    print(c[0])
```

```
Bought this dress for an indian wedding- it was perfect!
Fits perfect!
If there ever was the perfect feminine dress, this would be it .
So comfortable-so versatile-so perfect
I love it!!! i can wear out to dinner or just out to lunch with
friends!
```

Polarity == 0

```
cl = df.loc[df.Polarity == 0, ['Review Text']].sample(5).values
for c in cl:
    print(c[0])
```

```
Wish this site had "ask a question about the product" feature like amazon does.
just wondering if anyone has seen this product in person to verify color? it
```

```
looks orange but descriptor says red. would appreciate feedback. thank you for
your help.
I bought the mosaic pattern in a small. the pattern and colors look much better
in person. the design and quality are solid. the top runs small. it is difficult
to put on, but stays in place. i would recommend.
I have several of goodhyouman shirts and i get so many compliments on them.
especially the one that says forehead kisses are underrated. don't hesitate. buy
this shirt. you won't be sorry.....
Like the extra details on the neck and back - and it's versatile to wear
everywhere
Very sheer inexpensive material. even on sale i'm returning it
```

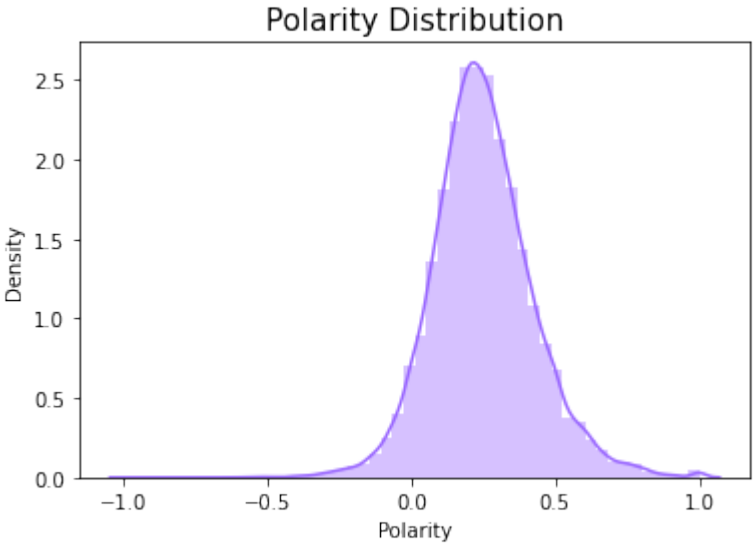
Polarity <= -0.7

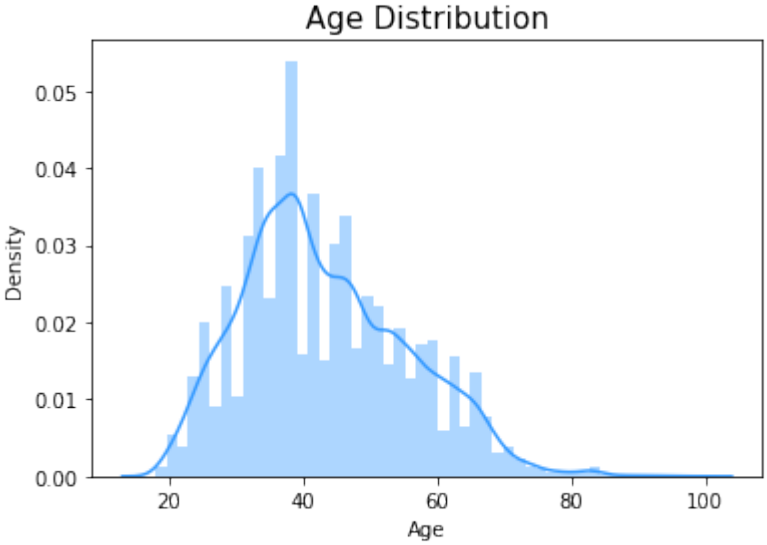
```
cl = df.loc[df.Polarity <= -0.7, ['Review Text']].sample(5).values
for c in cl:
    print(c[0])
```

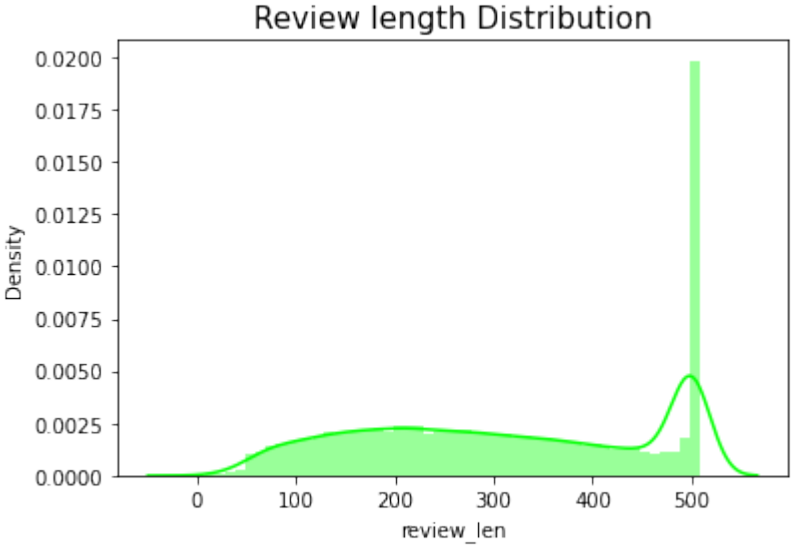
```
Awful color, horribly wrinkled and just a mess...so disappointed
Cut out design, no seems or hems.
 very disappointed in retailer
What a disappointment and for the price, it's outrageous!
Received this product with a gaping hole in it. very disappointed in the quality
and the quality control at the warehouse
The button fell off when i took it out of the bag, and i noticed that all of the
thread had unraveled. will be returning :-(
```

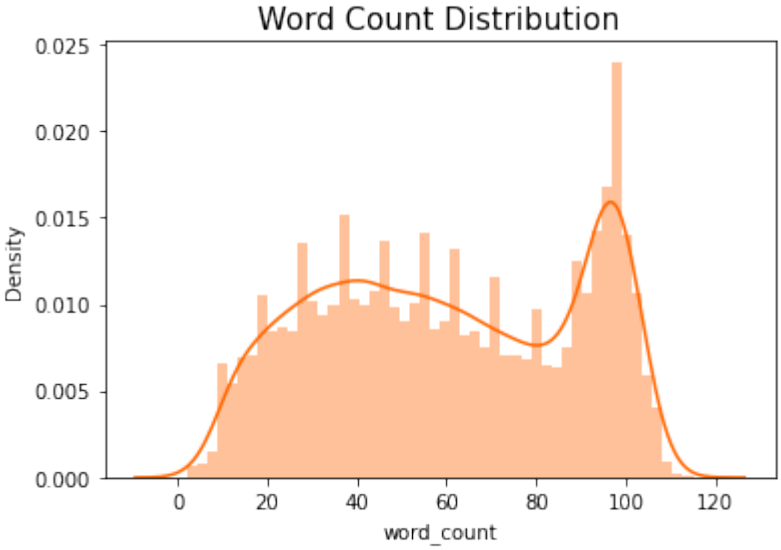
Distribution of review sentiment polarity score

```
features = ['Polarity', 'Age', 'review_len', 'word_count']
titles = ['Polarity Distribution', 'Age Distribution', 'Review length
Distribution', 'Word Count Distribution']
colors = ['#9966ff', '#3399ff', '#00ff00', '#ff6600']

for feature, title, color in zip(features, titles, colors):
    sns.distplot(x=df[feature], bins=50, color=color)
    plt.title(title, size=15)
    plt.xlabel(feature)
    plt.show()
```





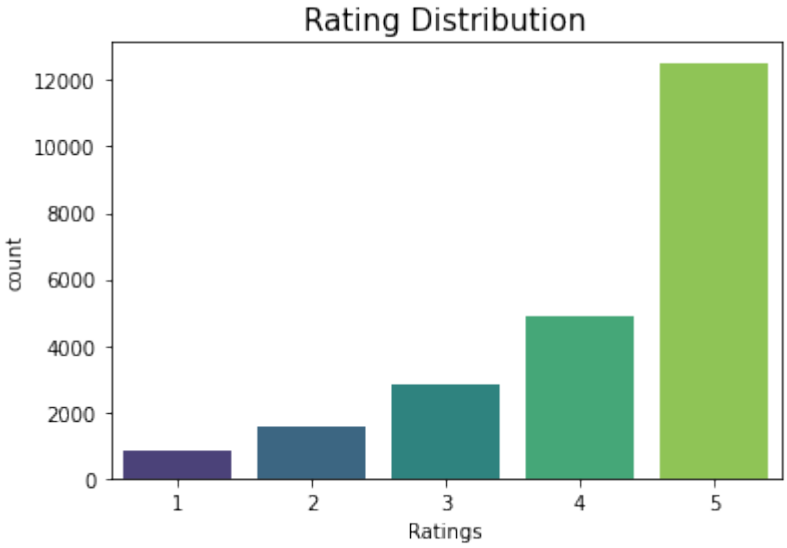




1. **Vast majority of the sentiment polarity scores are greater than zero, means most of them are pretty positive.**
2. **Most reviewers are in their 30s to 40s.**

Distribution of review ratings

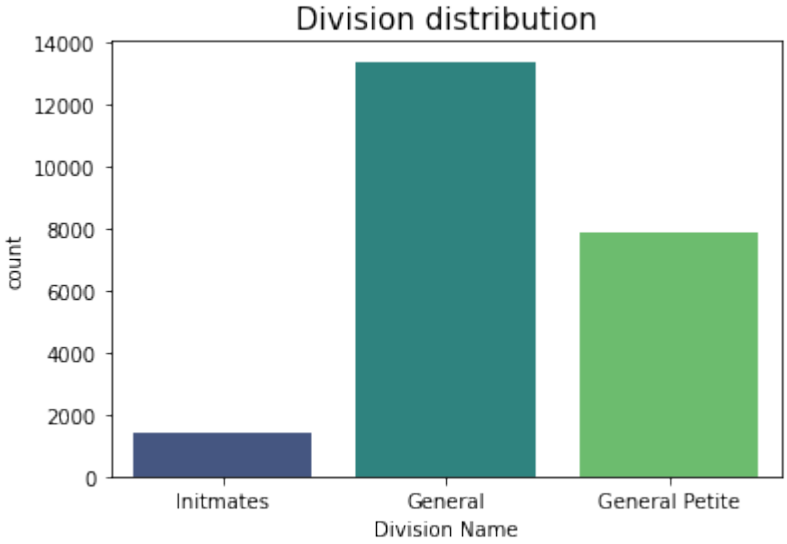
```
sns.countplot(x = 'Rating', palette='viridis', data=df)
plt.title('Rating Distribution', size=15)
plt.xlabel('Ratings')
plt.show()
```



The ratings are in align with the polarity score, that is, most of the ratings are pretty high at 4 or 5 ranges.

Distribution by Division Name

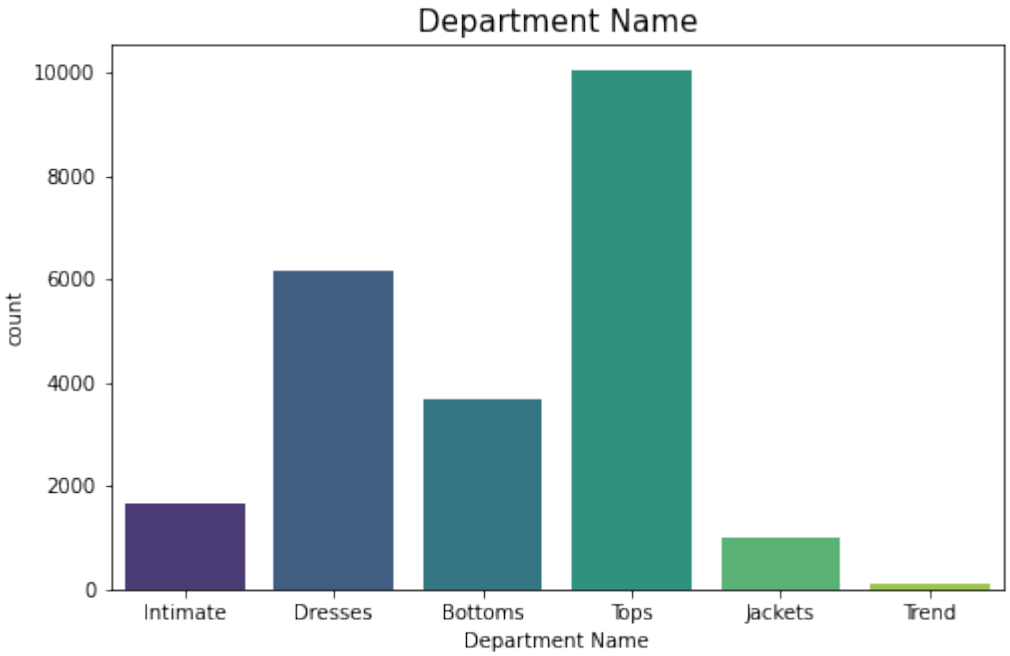
```
sns.countplot(x='Division Name', palette='viridis', data=df)
plt.title('Division distribution', size=15)
plt.show()
```



General division has the most number of reviews, and Initmates division has the least number of reviews.

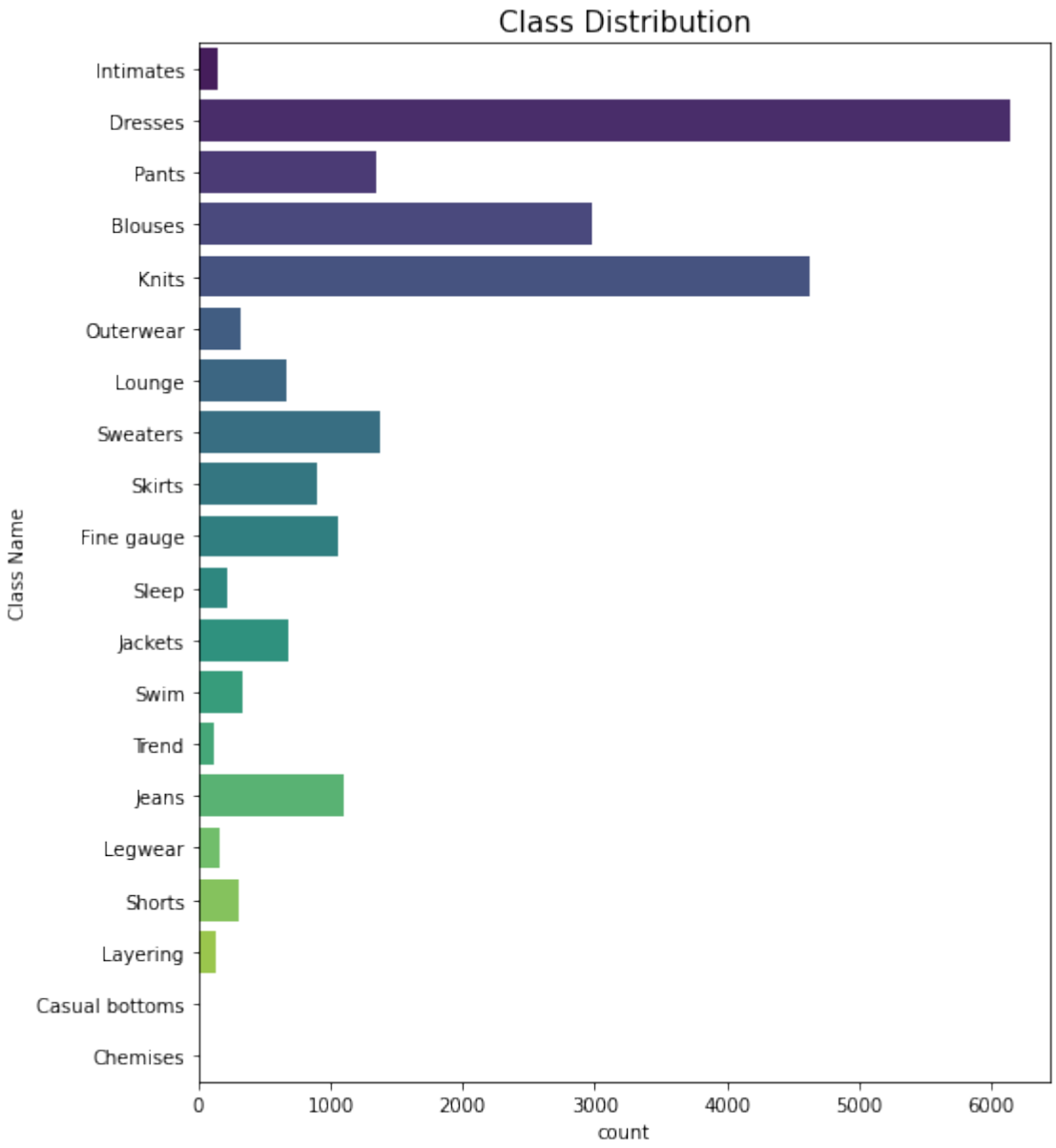
```
plt.figure(figsize=(8, 5))
sns.countplot(x='Department Name', palette='viridis', data=df)
plt.title('Department Name', size=15)
```

```
plt.show()
```



Distribution of Class

```
plt.figure(figsize=(8, 10))
sns.countplot(y='Class Name', palette='viridis', data=df)
plt.title('Class Distribution', size=15)
plt.show()
```



Unigrams, Bigrams and Trigrams

Now we come to “Review Text” feature, before explore this feature, we need to extract N-Gram features. N-grams are used to describe the number of words used as observation points, e.g., unigram means singly-worded, bigram means 2-worded phrase, and trigram means 3-worded phrase. In order to do this, we use scikit-learn’s CountVectorizer function.

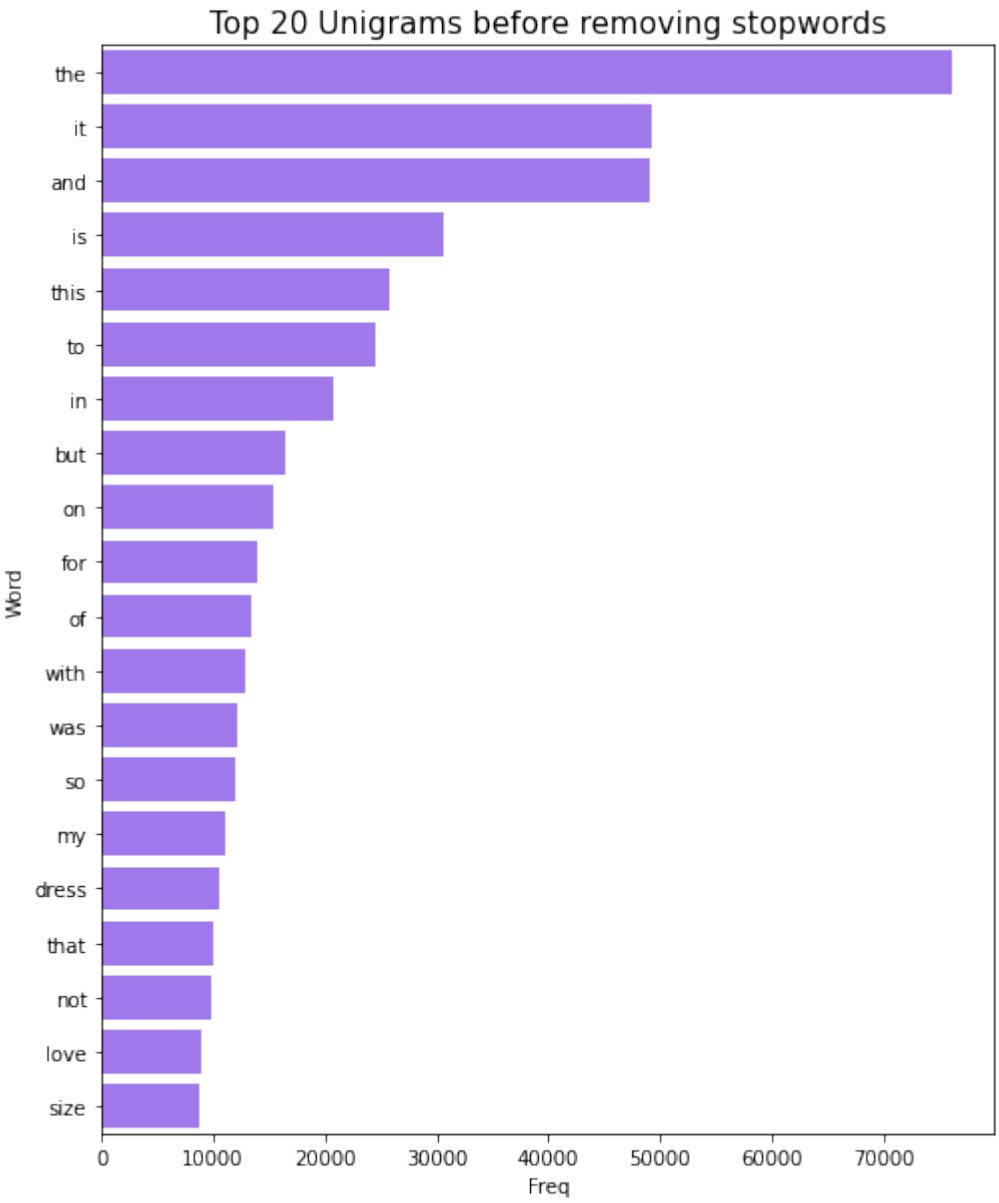
First, it would be interesting to compare unigrams before and after removing stop words.

```
def get_top_ngrams(corpus, ngram_range, stop_words=None, n=None):
    vec = CountVectorizer(stop_words=stop_words,
ngram_range=ngram_range).fit(corpus)
    bag_of_words = vec.transform(corpus)

    sum_words = bag_of_words.sum(axis=0)

    words_freq = [(word, sum_words[0, idx]) for word, idx in
vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
```

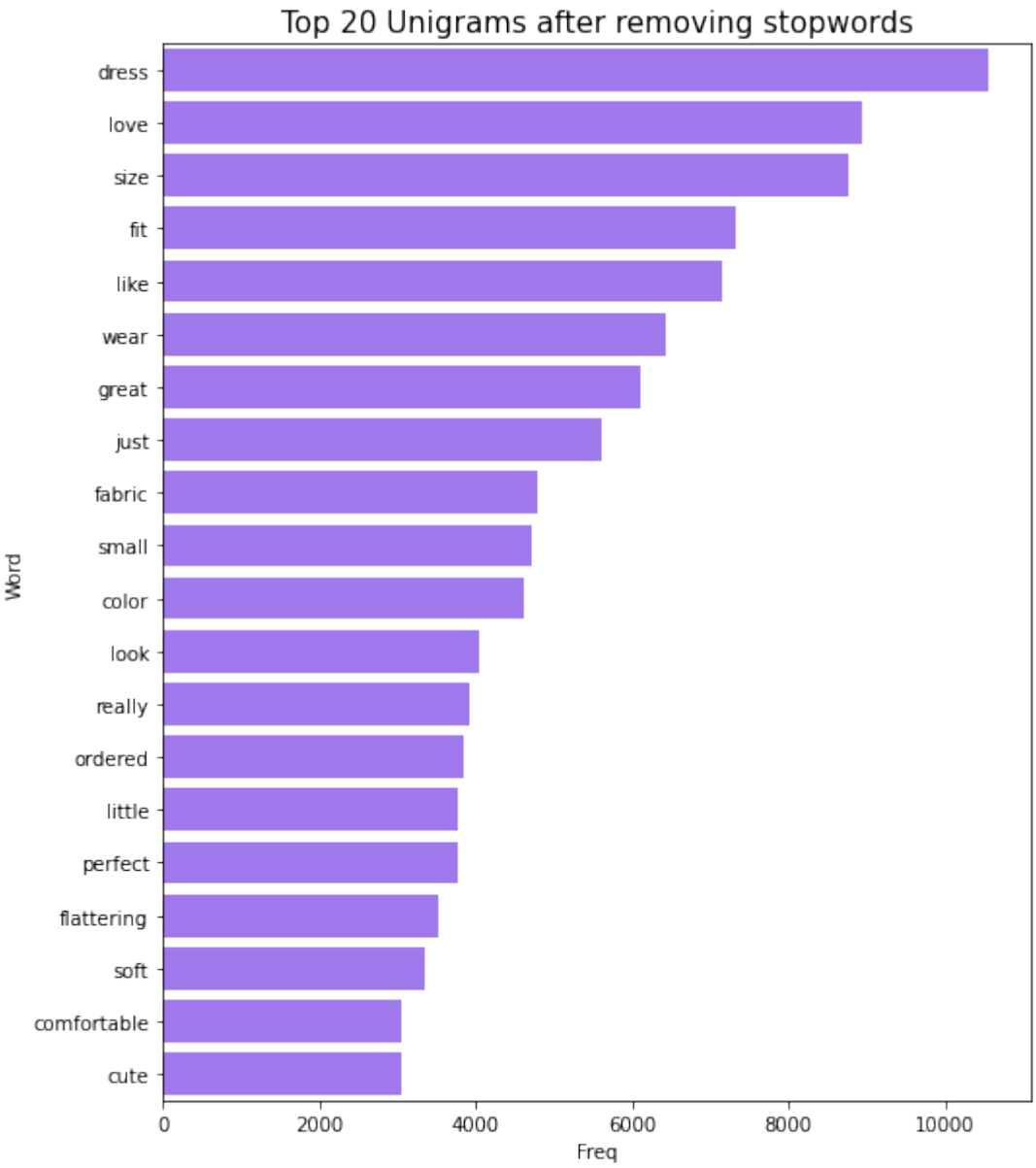
```
    common_words = words_freq[:n]
    words = []
    freqs = []
    for word, freq in common_words:
        words.append(word)
        freqs.append(freq)

    df = pd.DataFrame({'Word': words, 'Freq': freqs})
    return df

stop_words = None
n = 20
unigrams = get_top_ngrams(df['Review Text'], (1, 1), stop_words, n)
bigrams = get_top_ngrams(df['Review Text'], (2, 2), stop_words, n)
trigrams = get_top_ngrams(df['Review Text'], (3, 3), stop_words, n)

stop_words = 'english'
n = 20
unigrams_st = get_top_ngrams(df['Review Text'], (1, 1), stop_words, n)
bigrams_st = get_top_ngrams(df['Review Text'], (2, 2), stop_words, n)
trigrams_st = get_top_ngrams(df['Review Text'], (3, 3), stop_words, n)
```

Unigrams Distribution

```
plt.figure(figsize=(8, 10))
sns.barplot(x='Freq', y='Word', color=colors[0], data=unigrams)
plt.title('Top 20 Unigrams before removing stopwords', size=15)
plt.show()
```

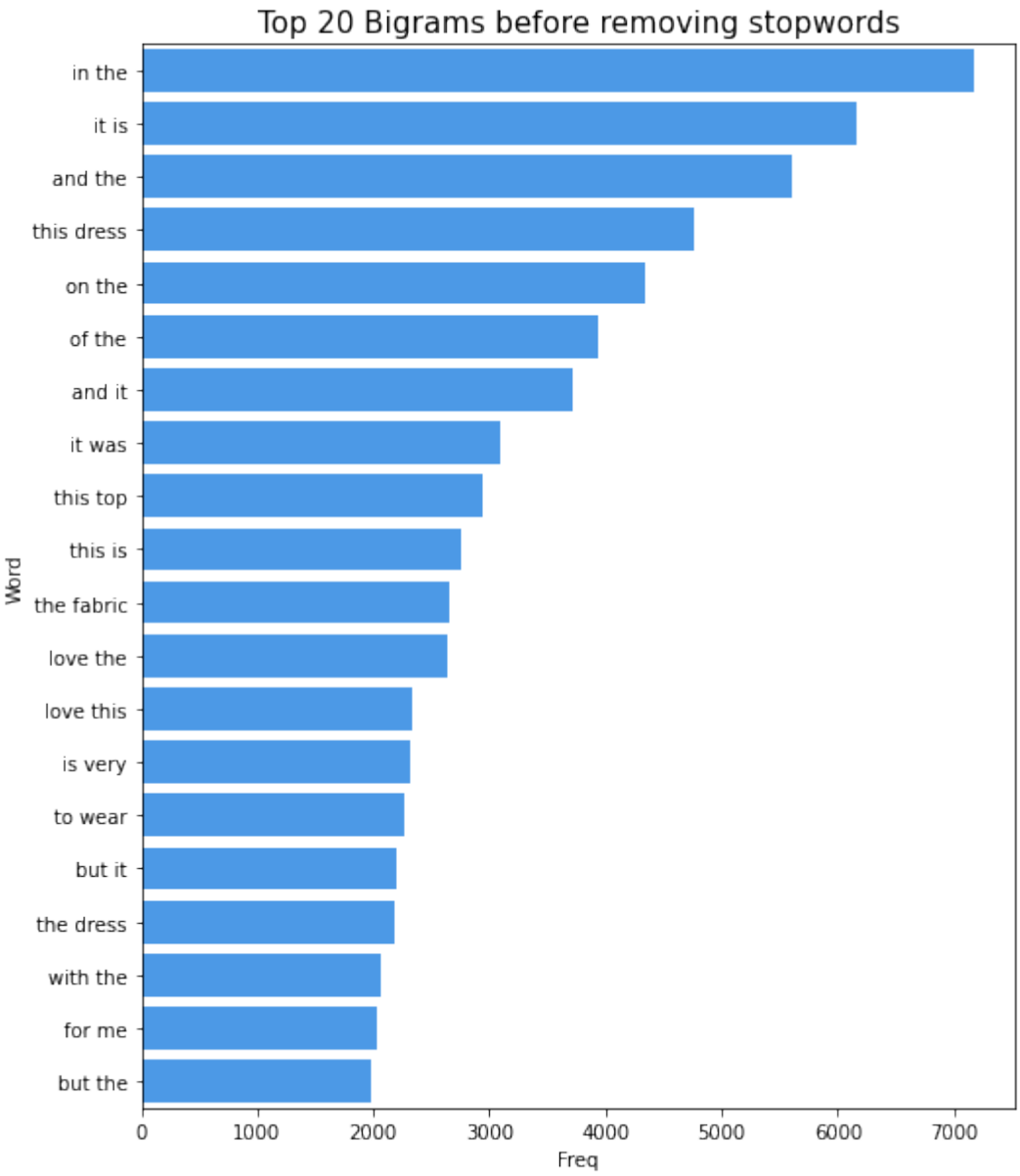



```
plt.figure(figsize=(8, 10))
sns.barplot(x='Freq', y='Word', color=colors[0], data=unigrams_st)
plt.title('Top 20 Unigrams after removing stopwords', size=15)
plt.show()
```

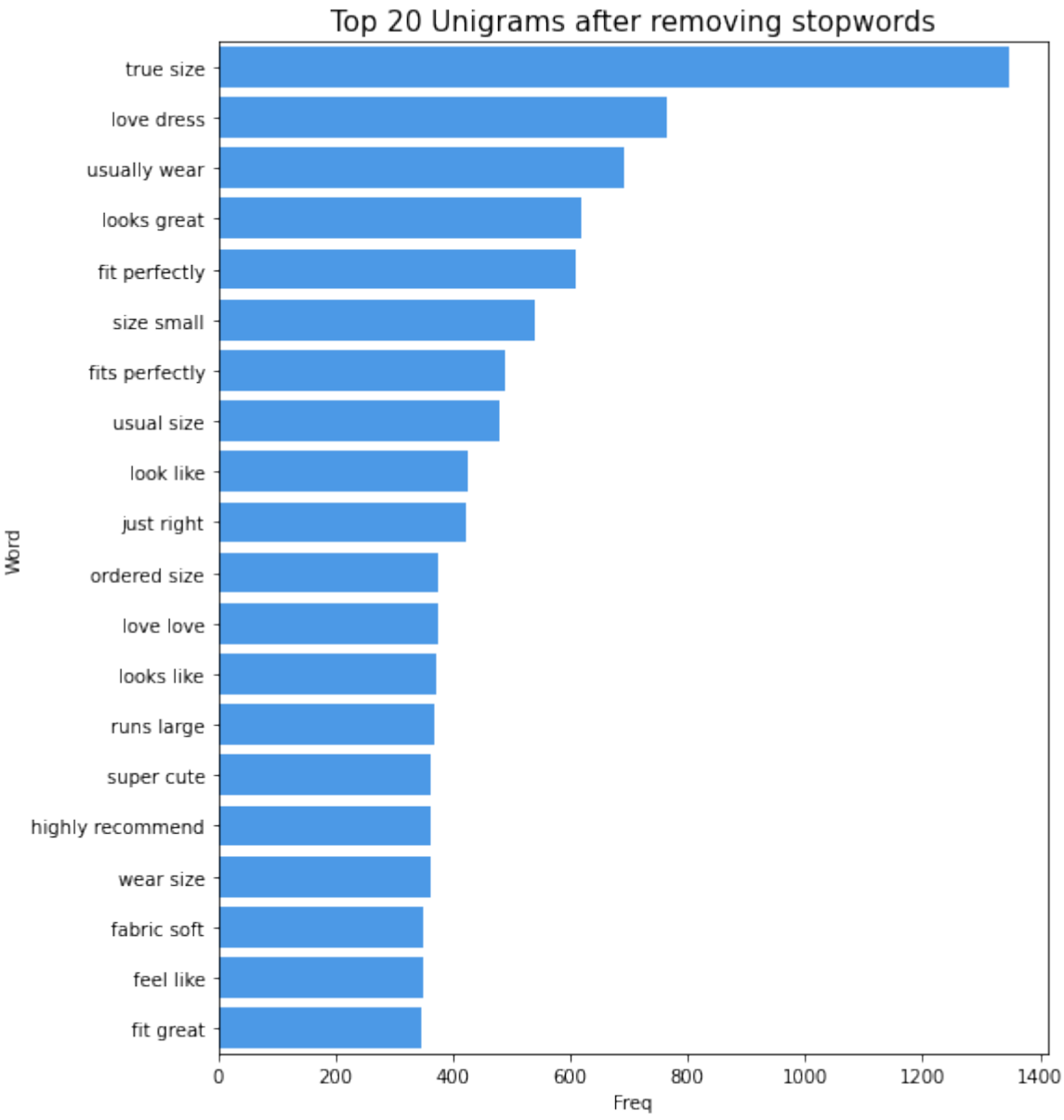


Bigrams Distribution

```
plt.figure(figsize=(8, 10))
sns.barplot(x='Freq', y='Word', color=colors[1], data=bigrams)
plt.title('Top 20 Bigrams before removing stopwords', size=15)
plt.show()
```




```
plt.figure(figsize=(8, 10))
sns.barplot(x='Freq', y='Word', color=colors[1], data=bigrams_st)
plt.title('Top 20 Unigrams after removing stopwords', size=15)
plt.show()
```



References

1. https://www.kaggle.com/code/bhaveshkumar2806/complete-eda-and-visualization-of-text-data

2. https://medium.com/@frankonyango.w/nlp-3-ways-to-easily-visualize-your-text-data-using-texthero-c31ce633aeff

3. https://www.kaggle.com/code/harshsingh2209/complete-guide-to-eda-on-text-data

4. https://www.kaggle.com/code/nilimajauhari/eda-for-textual-data-text-visualization

7. Visualization on streaming dataset

Aim: to visualize streaming dataset for weather forecasting.

Description

Weather forecasting is the application of science and technology to predict the conditions of the atmosphere for a given location and time. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere, land, and ocean and using meteorology to project how the atmosphere will change at a given place. Weather forecasting now relies on computer-based models that take many atmospheric factors into account. Forecasts based on temperature and precipitation are important to agriculture, and therefore to traders within commodity markets. Temperature forecasts are used by utility companies to estimate demand over coming days.

Features

```
- station - used weather station number: 1 to 25
- Date - Present day: yyyy-mm-dd ('2013-06-30' to '2017-08-30')
- Present_Tmax - Maximum air temperature between 0 and 21 h on the present day
(Â°C): 20 to 37.6
- Present_Tmin - Minimum air temperature between 0 and 21 h on the present day
(Â°C): 11.3 to 29.9
- LDAPS_RHmin - LDAPS model forecast of next-day minimum relative humidity (%):
19.8 to 98.5
- LDAPS_RHmax - LDAPS model forecast of next-day maximum relative humidity (%):
58.9 to 100
- LDAPSTmaxlapse - LDAPS model forecast of next-day maximum air temperature
applied lapse rate (Â°C): 17.6 to 38.5
- LDAPSTminlapse - LDAPS model forecast of next-day minimum air temperature
applied lapse rate (Â°C): 14.3 to 29.6
- LDAPS_WS - LDAPS model forecast of next-day average wind speed (m/s): 2.9 to
21.9
- LDAPS_LH - LDAPS model forecast of next-day average latent heat flux (W/m2):
-13.6 to 213.4
- LDAPS_CC1 - LDAPS model forecast of next-day 1st 6-hour split average cloud
cover (0-5 h) (%): 0 to 0.97
- LDAPS_CC2 - LDAPS model forecast of next-day 2nd 6-hour split average cloud
cover (6-11 h) (%): 0 to 0.97
- LDAPS_CC3 - LDAPS model forecast of next-day 3rd 6-hour split average cloud
cover (12-17 h) (%): 0 to 0.98
- LDAPS_CC4 - LDAPS model forecast of next-day 4th 6-hour split average cloud
cover (18-23 h) (%): 0 to 0.97
- LDAPS_PPT1 - LDAPS model forecast of next-day 1st 6-hour split average
precipitation (0-5 h) (%): 0 to 23.7
- LDAPS_PPT2 - LDAPS model forecast of next-day 2nd 6-hour split average
precipitation (6-11 h) (%): 0 to 21.6
- LDAPS_PPT3 - LDAPS model forecast of next-day 3rd 6-hour split average
precipitation (12-17 h) (%): 0 to 15.8
- LDAPS_PPT4 - LDAPS model forecast of next-day 4th 6-hour split average
precipitation (18-23 h) (%): 0 to 16.7
- lat - Latitude (Â°): 37.456 to 37.645
- lon - Longitude (Â°): 126.826 to 127.135
- DEM - Elevation (m): 12.4 to 212.3
- Slope - Slope (Â°): 0.1 to 5.2
- Solar radiation - Daily incoming solar radiation (wh/m2): 4329.5 to 5992.9
- Next_Tmax - The next-day maximum air temperature (Â°C): 17.4 to 38.9
- Next_Tmin - The next-day minimum air temperature (Â°C): 11.3 to 29.8T
```

Shape analysis :

- **target(s)** : Next_Tmax & Next_Tmin
- **rows and columns** : 7752 , 25
- **features types** : qualitatives : 1 (la date du relevé) , quantitatives : 24 (tout le reste)
- **NaN analysis** :
    - vraiment pas beaucoup de NaN (moitié des variables = 1% de NaN)

Features analysis :

- **Target visualization** :
    - Next_Tmax = 30.27° +/- 3.12°
    - Next_Tmin = 22.930 +/- 2.49°

Code :

```
# This Python 3 environment comes with many helpful analytics libraries
installed

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list
all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

```
/kaggle/input/temperature-forecast-project-using-ml/temp.csv
```

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Importing the data set
df = pd.read_csv('/kaggle/input/temperature-forecast-project-using-ml/temp.csv')

pd.set_option('display.max_row',25) #Affiche au plus 25 éléments dans les
résultats de pandas
pd.set_option('display.max_column',25) #Affiche au plus 25 éléments dans les
résultats de pandas
df.head()
```

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	LD C
0	1.0	30-06-	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	0.23

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	LD
		2013									
1	2.0	30-06-2013	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	0.22
2	3.0	30-06-2013	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	0.20
3	4.0	30-06-2013	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	0.21
4	5.0	30-06-2013	31.4	21.9	56.174095	90.155128	29.113934	23.486480	5.735004	107.965535	0.15

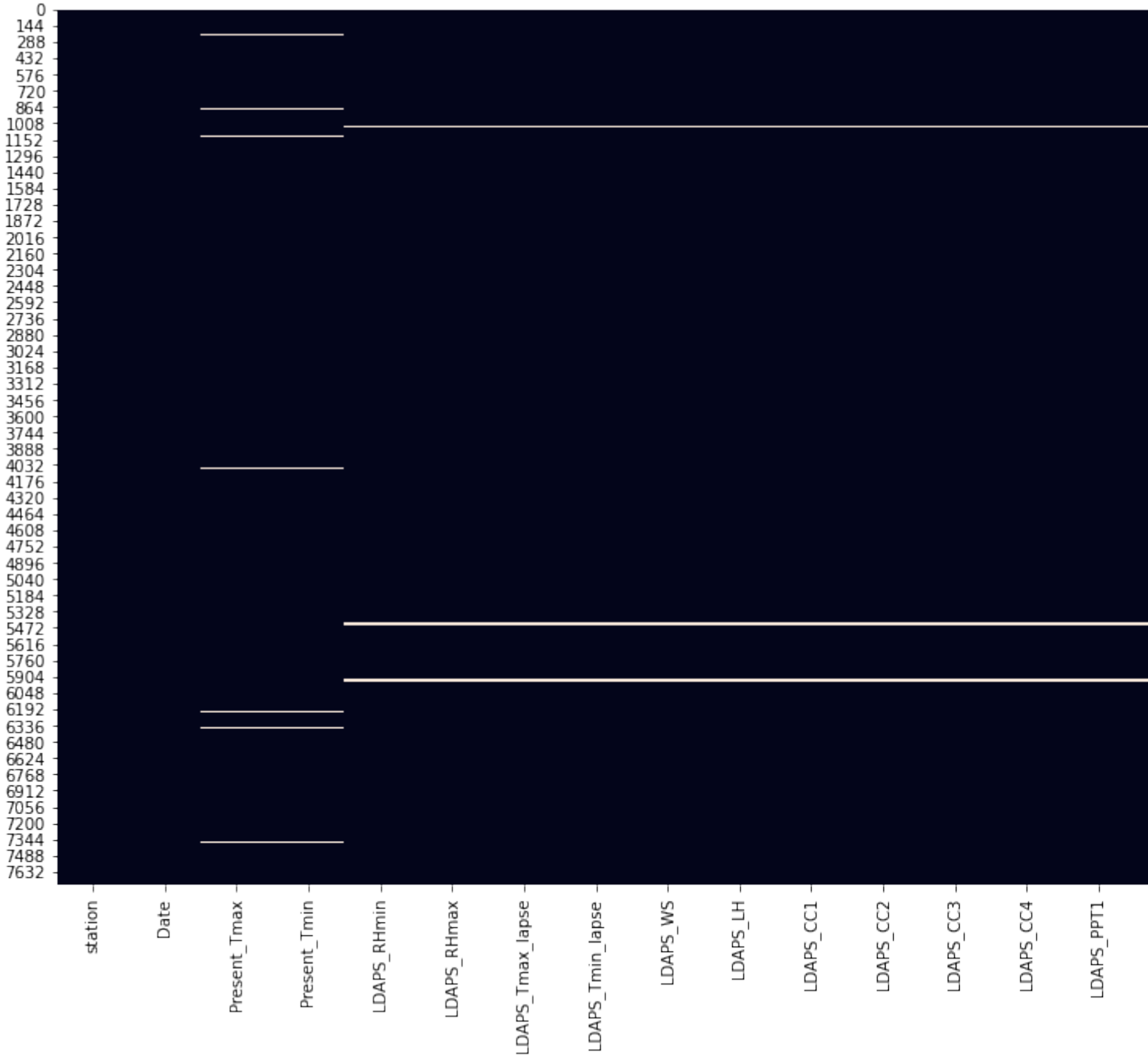
```
df.dtypes.value_counts() # Compte les nombre de types de variables
```

```
float64    24
object      1
dtype: int64
```

```
df.shape
```

```
(7752, 25)
```

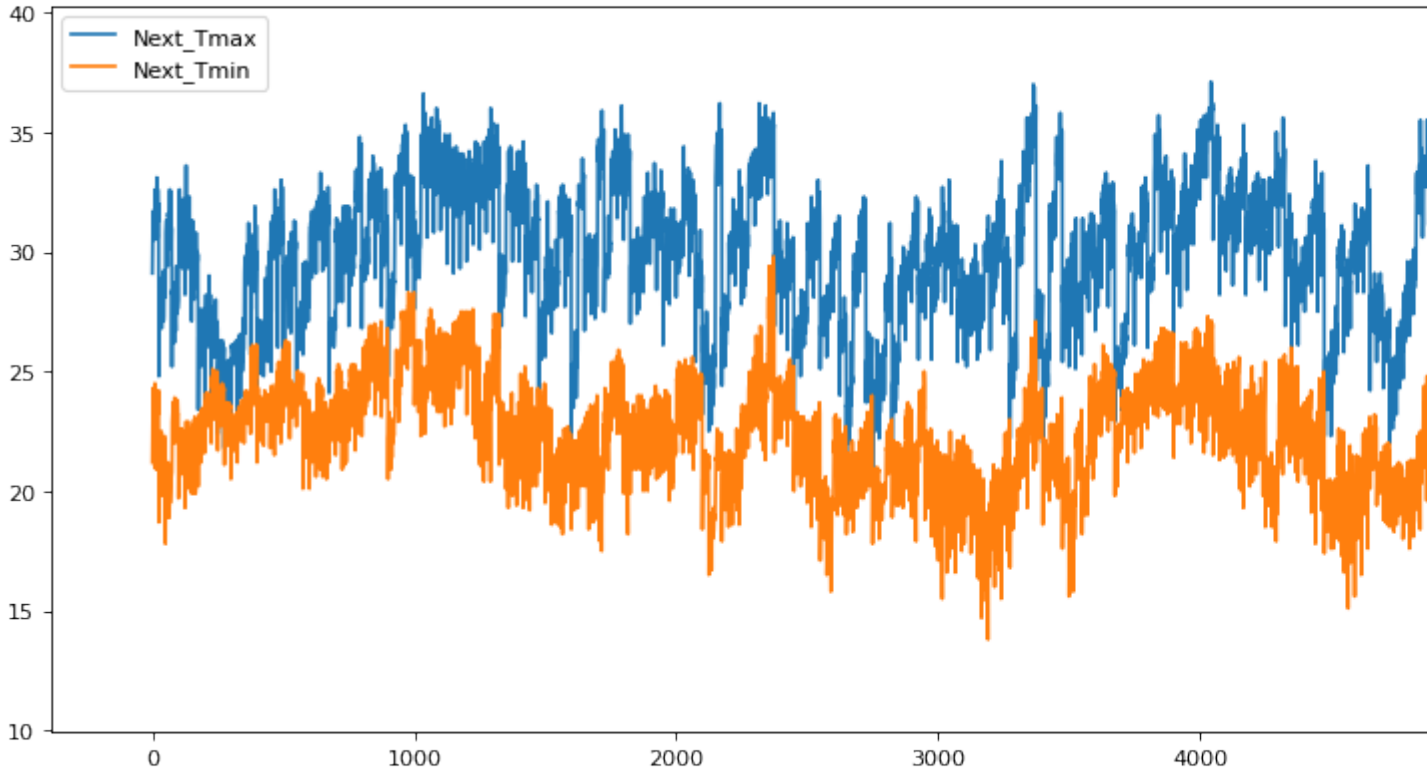
```
plt.figure(figsize=(20,10))
sns.heatmap(df.isna(),cbar=False)
plt.show()
print((df.isna().sum()/df.shape[0]*100).sort_values(ascending=False))
```



LDAPS_CC3	0.967492
LDAPS_PPT4	0.967492
LDAPS_PPT2	0.967492
LDAPS_PPT1	0.967492
LDAPS_CC4	0.967492
LDAPS_CC2	0.967492
LDAPS_CC1	0.967492
LDAPS_LH	0.967492
LDAPS_WS	0.967492
LDAPS_Tmin_lapse	0.967492
LDAPS_Tmax_lapse	0.967492
LDAPS_RHmax	0.967492
LDAPS_RHmin	0.967492
LDAPS_PPT3	0.967492
Present_Tmin	0.902993
Present_Tmax	0.902993
Next_Tmax	0.348297
Next_Tmin	0.348297
Date	0.025800

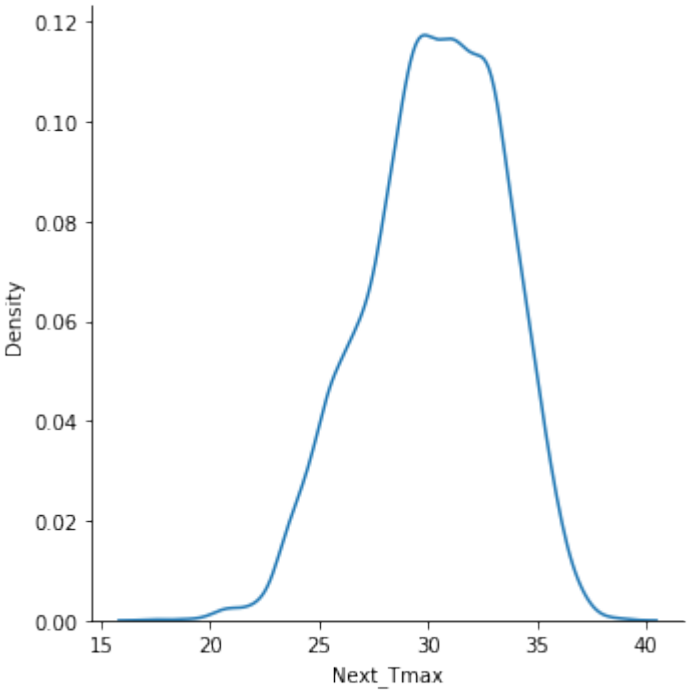
```
station             0.025800
lat                 0.000000
lon                 0.000000
DEM                 0.000000
Slope               0.000000
Solar radiation     0.000000
dtype: float64
```

```
plt.figure(figsize=(18, 6), dpi=80)
plt.plot(df["Next_Tmax"],label="Next_Tmax")
plt.plot(df["Next_Tmin"],label="Next_Tmin")
plt.legend()
plt.show()
```

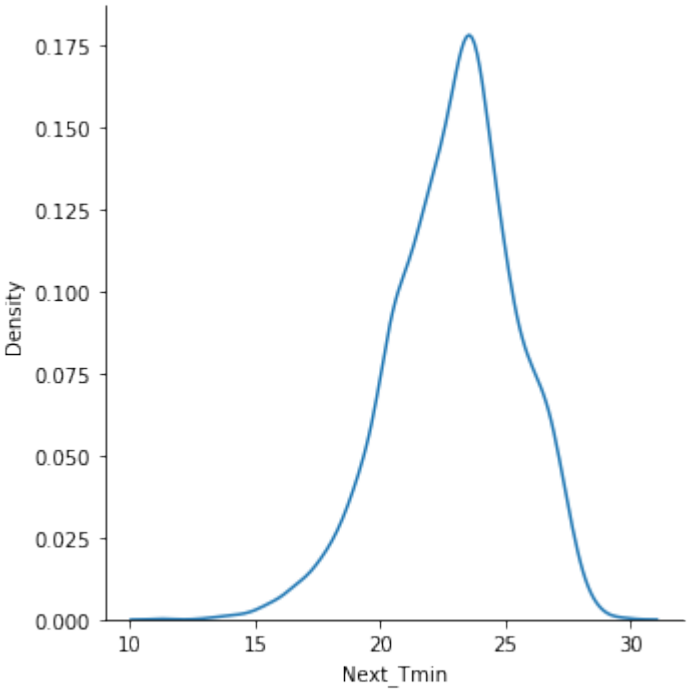



```
for col in ["Next_Tmax","Next_Tmin"]:
    plt.figure()
    sns.displot(df[col],kind='kde')
    plt.show()
print(df["Next_Tmax"].mean())
print(df["Next_Tmax"].std())
print(df["Next_Tmin"].mean())
print(df["Next_Tmin"].std())
```

```
<Figure size 432x288 with 0 Axes>
```

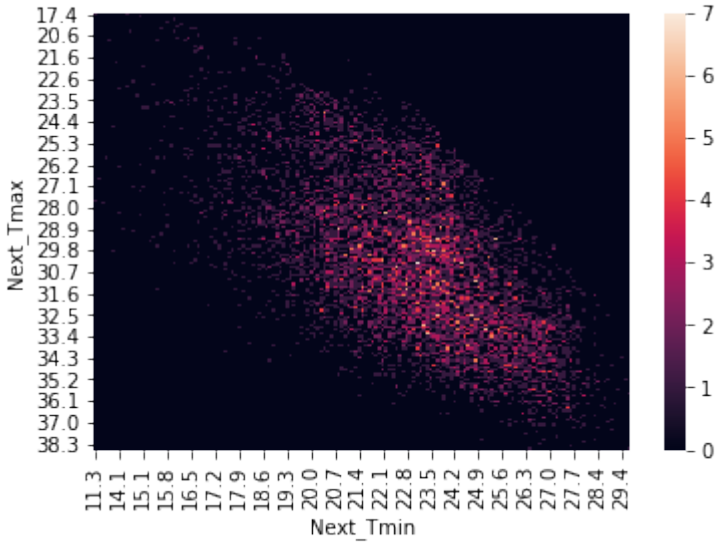


<Figure size 432x288 with 0 Axes>



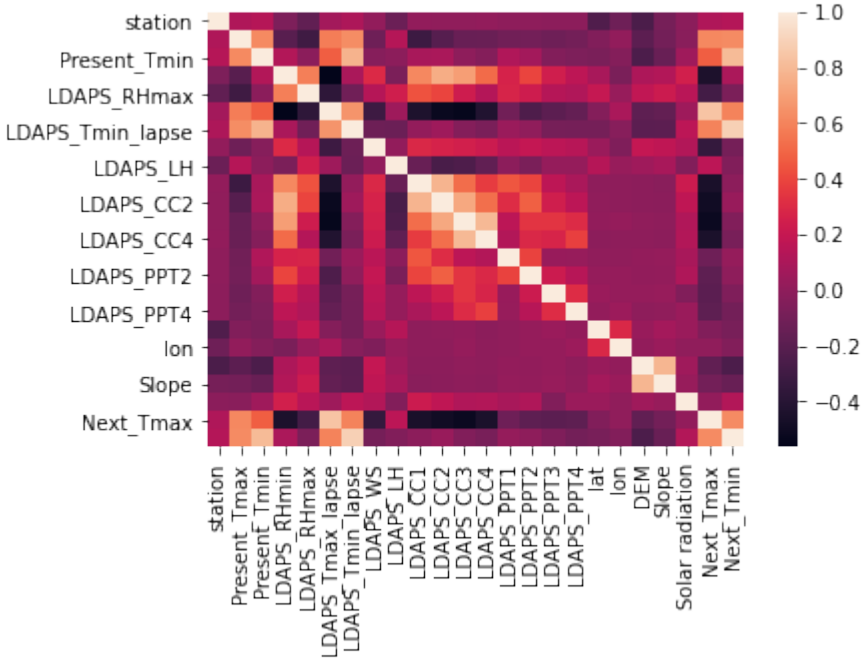
30.274886731391593
3.128010057855773
22.93222006472492
2.487612771331068

```
plt.figure()
sns.heatmap(pd.crosstab(df['Next_Tmax'],df['Next_Tmin']))
plt.show()
```




```
sns.heatmap(df.corr())
```

```
<AxesSubplot:>
```



Data preprocessing

- Deal with NaN
- Encode qualitative features
- Develop our first modelling strategy

```
# Importing the data set
df = pd.read_csv('/kaggle/input/temperature-forecast-project-using-ml/temp.csv')
Save = df.copy()

def feature_engineering(df):
    df = df.drop(["Date"],axis=1)
    print(df.dtypes.value_counts()) # Compte les nombre de types de variables
    return(df)
```

```
def imputation(df):
    #df = df.fillna(-999)
    df = df.dropna(axis=0)
    return df

def encodage(df):
    return df

def preprocessing(df):
    df = imputation(df)
    df = encodage(df)
    df = feature_engineering(df)

    X = df.drop(['Next_Tmax','Next_Tmin'],axis=1)
    y_max = df["Next_Tmax"]
    y_min = df["Next_Tmin"]

    print(X.shape)
    print(y_max.shape)

    return X,y_max,y_min

from sklearn.model_selection import train_test_split
trainset, testset = train_test_split(df, test_size=0.2, random_state=0)

X_train, y_min_train, y_max_train = preprocessing(trainset)
X_test, y_min_test, y_max_test = preprocessing(testset)

float64    24
dtype: int64
(6068, 22)
(6068,)
float64    24
dtype: int64
(1520, 22)
(1520,)
```

Modeling

- Standardise features
- Define a regression model
- Compute score and RMSE (in °C)

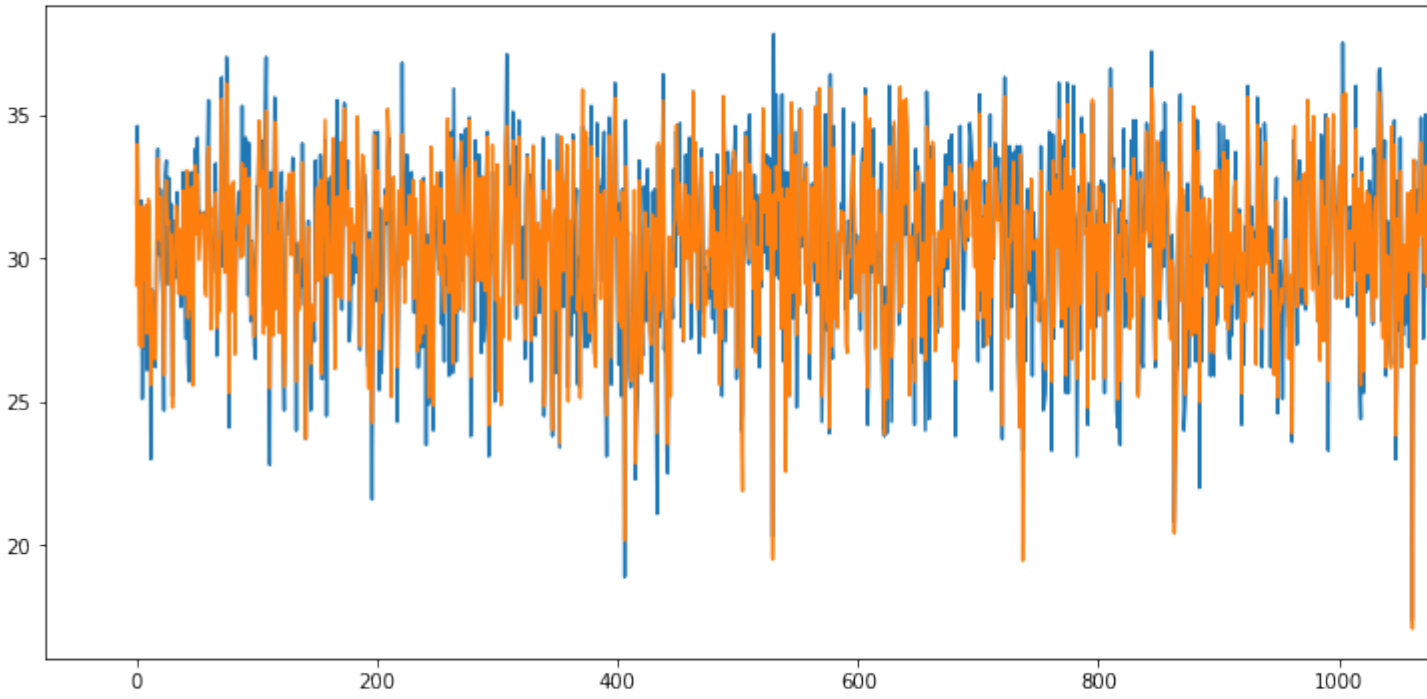
Part 1 - SGDRegressor

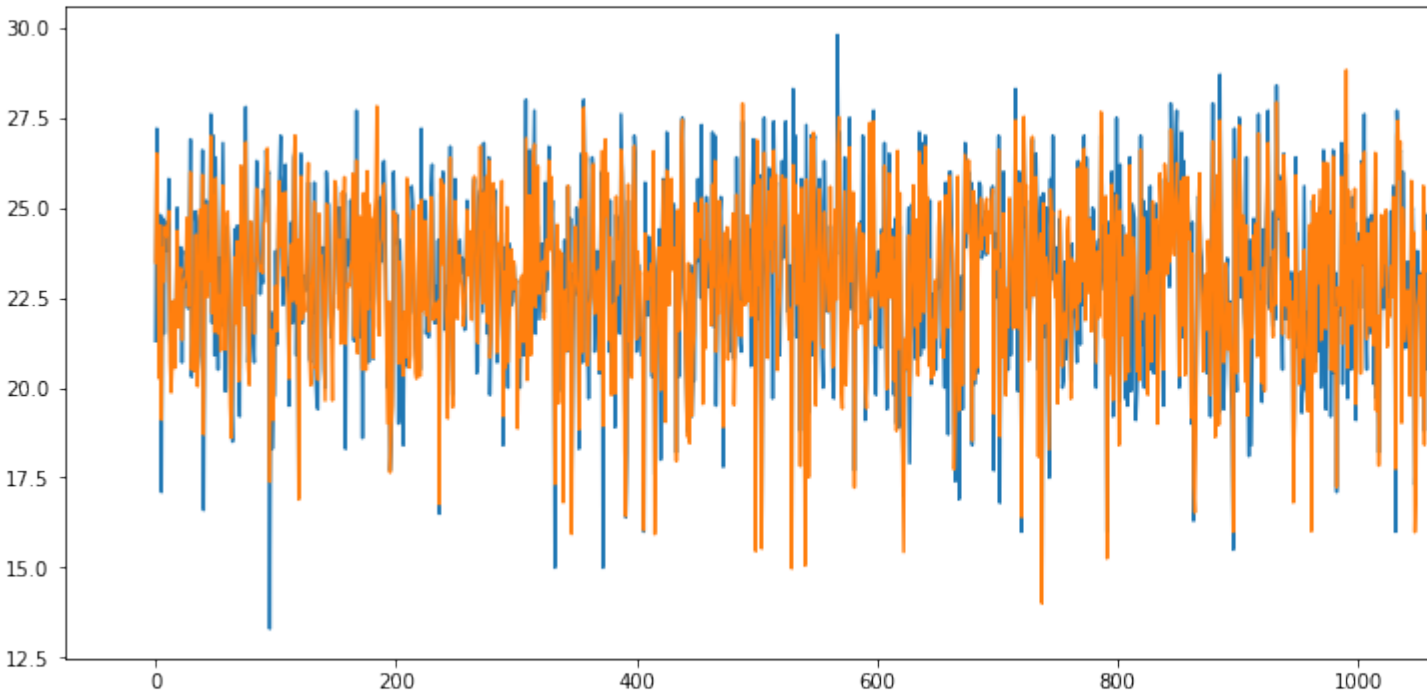
```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import cross_validate

reg_max = make_pipeline(StandardScaler(),
                    SGDRegressor(loss='squared_loss', penalty='l2',
max_iter=1000, tol=1e-3))
reg_max.fit(X_train, y_max_train)

reg_min = make_pipeline(StandardScaler(),
                    SGDRegressor(loss='squared_loss', penalty='l2',
max_iter=1000, tol=1e-3))
reg_min.fit(X_train, y_min_train)
```

```
cv_results_min = cross_validate(reg_min, X_train, y_min_train, cv=5,
scoring=('r2', "neg_root_mean_squared_error"), return_train_score=True)
cv_results_max = cross_validate(reg_max, X_train, y_max_train, cv=5,
scoring=('r2', "neg_root_mean_squared_error"), return_train_score=True)

print('Pour le Next_Tmin :')
print('Test RMSE :' ,
-cv_results_min['test_neg_root_mean_squared_error'].mean())
print('Test r2 :' , cv_results_min['test_r2'].mean())
print("Train RMSE :" ,
-cv_results_min['train_neg_root_mean_squared_error'].mean())
print("Train r2 :" , cv_results_min['train_r2'].mean())
print("*----------------------------------------*")
print('Pour le Next_Tmax :')
print('Test RMSE :' ,
-cv_results_max['test_neg_root_mean_squared_error'].mean())
print('Test r2 :' , cv_results_max['test_r2'].mean())
print("Train RMSE :" ,
-cv_results_max['train_neg_root_mean_squared_error'].mean())
print("Train r2 :" , cv_results_max['train_r2'].mean())
```

```
Pour le Next_Tmin :
Test RMSE : 1.4710491957080583
Test r2 : 0.7754581562198303
Train RMSE : 1.4675815658054128
Train r2 : 0.7767996956102122
*----------------------------------------*
Pour le Next_Tmax :
Test RMSE : 1.010211731765659
Test r2 : 0.8335435910922501
Train RMSE : 1.0072290803213872
Train r2 : 0.8350208191227744
```

```
Next_Tmin_predict = reg_min.predict(X_test)
Next_Tmax_predict = reg_max.predict(X_test)

plt.figure(figsize=(18,6))
plt.plot(y_min_test.to_numpy(),label="Next_Tmin")
plt.plot(Next_Tmin_predict,label="Next_Tmin_predict")
plt.legend()
plt.show()

plt.figure(figsize=(18,6))
plt.plot(y_max_test.to_numpy(),label="Next_Tmax")
plt.plot(Next_Tmax_predict,label="Next_Tmax_predict")
plt.legend()
plt.show()
```





References

1. https://www.kaggle.com/code/dorianvoydie/weather-forecasting-98-acc
2. https://www.kaggle.com/code/subinium/weather-dashboard-eda-visualization

8. Visualizing Geospatial Data

Aim: to visualize geospatial data.

Introduction

This is to create maps for different objectives. We use Matplotlib and work with another Python visualization library, namely **Folium**. **Folium** was developed for the sole purpose of visualizing geospatial data. While other libraries are available to visualize geospatial data, such as **plotly**, they might have a cap on how many API calls you can make within a defined time frame. **Folium**, on the other hand, is completely free.

Exploring Datasets with *pandas* and Matplotlib

Toolkits: This lab heavily relies on *pandas* and **Numpy** for data wrangling, analysis, and visualization. The primary plotting library we will explore in this lab is **Folium**.

Datasets:

1. San Francisco Police Department Incidents for the year 2016 - Police Department Incidents from San Francisco public data portal. Incidents derived from San Francisco Police Department (SFPD) Crime Incident Reporting system. Updated daily, showing data for the entire year of 2016. Address and location has been anonymized by moving to mid-block or to an intersection.
2. Immigration to Canada from 1980 to 2013 - International migration flows to and from selected countries - The 2015 revision from United Nation's website. The dataset contains annual data on the flows of international migrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. For this lesson, we will focus on the Canadian Immigration data

Downloading and Prepping Data

Import Primary Modules:

```
import numpy as np  # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

Introduction to Folium

Folium is a powerful Python library that helps you create several types of Leaflet maps. The fact that the Folium results are interactive makes this library very useful for dashboard building.

From the official Folium documentation page:

> Folium builds on the data wrangling strengths of the Python ecosystem and the mapping strengths of the Leaflet.js library. Manipulate your data in Python, then visualize it in on a Leaflet map via Folium.

> Folium makes it easy to visualize data that's been manipulated in Python on an interactive Leaflet map. It enables both the binding of data to a map for choropleth visualizations as well as passing Vincent/Vega visualizations as markers on the map.
>
> The library has a number of built-in tilesets from OpenStreetMap, Mapbox, and Stamen, and supports custom tilesets with Mapbox or Cloudmade API keys. Folium supports both GeoJSON and TopoJSON overlays, as well as the binding of data to those overlays to create choropleth maps with color-brewer color schemes.

Let's install Folium

Folium is not available by default. So, we first need to install it before we are able to import it.

```
import folium

print('Folium installed and imported!')
```

```
Folium installed and imported!
```

Generating the world map is straigtforward in **Folium**. You simply create a **Folium** *Map* object and then you display it. What is attactive about **Folium** maps is that they are interactive, so you can zoom into any region of interest despite the initial zoom level.

```
# define the world map
world_map = folium.Map()

# display world map
world_map
```

Go ahead. Try zooming in and out of the rendered map above.

You can customize this default definition of the world map by specifying the centre of your map and the intial zoom level.

All locations on a map are defined by their respective *Latitude* and *Longitude* values. So you can create a map and pass in a center of *Latitude* and *Longitude* values of **[0, 0]**.

For a defined center, you can also define the intial zoom level into that location when the map is rendered. **The higher the zoom level the more the map is zoomed into the center**.

Let's create a map centered around Canada and play with the zoom level to see how it affects the rendered map.

```
# define the world map centered around Canada with a low zoom level
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4)

# display world map
world_map
```

Let's create the map again with a higher zoom level

As you can see, the higher the zoom level the more the map is zoomed into the given center.

Question: Create a map of Mexico with a zoom level of 4.

```
mexic_maps=folium.Map(location=[19.432608, -99.133209],zoom_start=4)
mexic_maps
```

```
 # define Mexico's geolocation coordinates
mexico_latitude = 23.6345
mexico_longitude = -102.5528

 # define the world map centered around Canada with a higher zoom level
mexico_map = folium.Map(location=[mexico_latitude, mexico_longitude],
zoom_start=4)


 # display world map
mexico_map
```

Another cool feature of **Folium** is that you can generate different map styles.

A. Stamen Toner Maps

These are high-contrast B+W (black and white) maps. They are perfect for data mashups and exploring river meanders and coastal zones.

Let's create a Stamen Toner map of canada with a zoom level of 4.

```
# create a Stamen Toner map of the world centered around Canada
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4, tiles='Stamen
Toner')

# display map
world_map
```

Feel free to zoom in and out to see how this style compares to the default one.

B. Stamen Terrain Maps

These are maps that feature hill shading and natural vegetation colors. They showcase advanced labeling and linework generalization of dual-carriageway roads.

Let's create a Stamen Terrain map of Canada with zoom level 4.

```
# create a Stamen Toner map of the world centered around Canada
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4, tiles='Stamen
Terrain')

# display map
world_map
```

Feel free to zoom in and out to see how this style compares to Stamen Toner and the default style.

C. Mapbox Bright Maps

These are maps that quite similar to the default style, except that the borders are not visible with a low zoom level. Furthermore, unlike the default style where country names are displayed in each country's native language, *Mapbox Bright* style displays all country names in English.

Let's create a world map with this style.

```
# create a world map with a Mapbox Bright style.
world_map = folium.Map(tiles='Mapbox Bright')
```

```
# display the map
world_map
```

Zoom in and notice how the borders start showing as you zoom in, and the displayed country names are in English.

Question: Create a map of Mexico to visualize its hill shading and natural vegetation. Use a zoom level of 6.

```
mexic_maps=folium.Map(location=[19.432608,
-99.133209],zoom_start=4,tiles='Stamen Terrain')
mexic_maps
```

Maps with Markers

Let's download and import the data on police department incidents using *pandas* `read_csv()` method.

Download the dataset and read it into a *pandas* dataframe:

```
df_incidents = pd.read_csv('https://s3-api.us-
geo.objectstorage.softlayer.net/cf-courses-
data/CognitiveClass/DV0101EN/labs/Data_Files/Police_Department_Incidents_-
_Previous_Year__2016_.csv')

print('Dataset downloaded and read into a pandas dataframe!')

Dataset downloaded and read into a pandas dataframe!

Let's take a look at the first five items in our dataset.

  File "<ipython-input-12-7a9b878e53fe>", line 1
    Let's take a look at the first five items in our dataset.
                                                            ^
SyntaxError: EOL while scanning string literal
```

Let's take a look at the first five items in our dataset.

```
df_incidents.head()
```

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution
0	120058272	WEAPON LAWS	POSS OF PROHIBITED WEAPON	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED
1	120058272	WEAPON LAWS	FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	ARREST, BOOKED
2	141059263	WARRANTS	WARRANT ARREST	Monday	04/25/2016 12:00:00 AM	14:59	BAYVIEW	ARREST, BOOKED
3	160013662	NON-CRIMINAL	LOST	Tuesday	01/05/20	23:50	TENDERLOIN	NONE

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resolution
			PROPERTY		16 12:00:00 AM			
4	160002740	NON-CRIMINAL	LOST PROPERTY	Friday	01/01/2016 12:00:00 AM	00:30	MISSION	NONE

So each row consists of 13 features:

1. **IncidntNum**: Incident Number
2. **Category**: Category of crime or incident
3. **Descript**: Description of the crime or incident
4. **DayOfWeek**: The day of week on which the incident occurred
5. **Date**: The Date on which the incident occurred
6. **Time**: The time of day on which the incident occurred
7. **PdDistrict**: The police department district
8. **Resolution**: The resolution of the crime in terms whether the perpetrator was arrested or not
9. **Address**: The closest address to where the incident took place
10. **X**: The longitude value of the crime location
11. **Y**: The latitude value of the crime location
12. **Location**: A tuple of the latitude and the longitude values
13. **PdId**: The police department ID

Let's find out how many entries there are in our dataset.

```
df_incidents.shape
```

```
(150500, 13)
```

So the dataframe consists of 150,500 crimes, which took place in the year 2016. In order to reduce computational cost, let's just work with the first 100 incidents in this dataset.

```
# get the first 100 crimes in the df_incidents dataframe
limit = 100
df_incidents = df_incidents.iloc[0:limit, :]
```

Let's confirm that our dataframe now consists only of 100 crimes.

```
df_incidents.shape
```

```
(100, 13)
```

Now that we reduced the data a little bit, let's visualize where these crimes took place in the city of San Francisco. We will use the default style and we will initialize the zoom level to 12.

```
# San Francisco latitude and longitude values
latitude = 37.77
longitude = -122.42

# create map and display it
sanfran_map = folium.Map(location=[latitude, longitude], zoom_start=12)
```

```
# display the map of San Francisco
sanfran_map
```

Now let's superimpose the locations of the crimes onto the map. The way to do that in **Folium** is to create a *feature group* with its own features and style and then add it to the sanfran_map.

```
# instantiate a feature group for the incidents in the dataframe
incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_incidents.Y, df_incidents.X):
    incidents.add_child(
        folium.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add incidents to map
sanfran_map.add_child(incidents)
```

You can also add some pop-up text that would get displayed when you hover over a marker. Let's make each marker display the category of the crime when hovered over.

```
# instantiate a feature group for the incidents in the dataframe
incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_incidents.Y, df_incidents.X):
    incidents.add_child(
        folium.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add pop-up text to each marker on the map
latitudes = list(df_incidents.Y)
longitudes = list(df_incidents.X)
labels = list(df_incidents.Category)

for lat, lng, label in zip(latitudes, longitudes, labels):
    folium.Marker([lat, lng], popup=label).add_to(sanfran_map)

# add incidents to map
sanfran_map.add_child(incidents)
```

Isn't this really cool? Now you are able to know what crime category occurred at each marker.

If you find the map to be so congested will all these markers, there are two remedies to this problem. The simpler solution is to remove these location markers and just add the text to the circle markers themselves as follows:

```
# create map and display it
sanfran_map = folium.Map(location=[latitude, longitude], zoom_start=12)

# loop through the 100 crimes and add each to the map
for lat, lng, label in zip(df_incidents.Y, df_incidents.X,
df_incidents.Category):
    folium.CircleMarker(
        [lat, lng],
        radius=5, # define how big you want the circle markers to be
        color='yellow',
        fill=True,
        popup=label,
        fill_color='blue',
        fill_opacity=0.6
    ).add_to(sanfran_map)

# show map
sanfran_map
```

The other proper remedy is to group the markers into different clusters. Each cluster is then represented by the number of crimes in each neighborhood. These clusters can be thought of as pockets of San Francisco which you can then analyze separately.

To implement this, we start off by instantiating a *MarkerCluster* object and adding all the data points in the dataframe to this object.

```
from folium import plugins

# let's start again with a clean copy of the map of San Francisco
sanfran_map = folium.Map(location = [latitude, longitude], zoom_start = 12)

# instantiate a mark cluster object for the incidents in the dataframe
incidents = plugins.MarkerCluster().add_to(sanfran_map)

# loop through the dataframe and add each data point to the mark cluster
for lat, lng, label, in zip(df_incidents.Y, df_incidents.X,
df_incidents.Category):
    folium.Marker(
        location=[lat, lng],
        icon=None,
        popup=label,
    ).add_to(incidents)

# display map
sanfran_map
```

Notice how when you zoom out all the way, all markers are grouped into one cluster, *the global cluster*, of 100 markers or crimes, which is the total number of crimes in our dataframe. Once you start zooming in, the *global cluster* will start breaking up into smaller clusters. Zooming in all the way will result in individual markers.

Choropleth Maps

A `Choropleth` map is a thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map, such as population density or per-capita income. The choropleth map provides an easy way to visualize how a measurement

varies across a geographic area or it shows the level of variability within a region. Below is a `Choropleth` map of the US depicting the population by square mile per state.

<img src = "https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/coursera/Images/2000_census_population_density_map_by_state.png" width = 600>

```
df_can = pd.read_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data_Files/Canada.xlsx',
                       sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2)

print('Data downloaded and read into a dataframe!')
```

```
Data downloaded and read into a dataframe!
```

Let's take a look at the first five items in our dataset.

```
df_can.head()
```

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	1981	1
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	39	3
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	0	0
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	67	7
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	1	0
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	0	0

Let's find out how many entries there are in our dataset.

```
# print the dimensions of the dataframe
print(df_can.shape)
```

```
(195, 43)
```

Clean up data. We will make some modifications to the original dataset to make it easier to create our visualizations. Refer to *Introduction to Matplotlib and Line Plots* and *Area Plots*, *Histograms*, *and Bar Plots* notebooks for a detailed description of this preprocessing.

```
# clean up the dataset to remove unnecessary columns (eg. REG)
df_can.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)

# let's rename the columns so that they make sense
df_can.rename(columns={'OdName':'Country',
'AreaName':'Continent','RegName':'Region'}, inplace=True)

# for sake of consistency, let's also make all column labels of type string
df_can.columns = list(map(str, df_can.columns))

# add total column
df_can['Total'] = df_can.sum(axis=1)
```

```
# years that we will be using in this lesson - useful for plotting later on
years = list(map(str, range(1980, 2014)))
print ('data dimensions:', df_can.shape)
```

```
data dimensions: (195, 39)
```

Let's take a look at the first five items of our cleaned dataframe.

```
df_can.head()
```

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	496	741	828	1076
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	1	2	2	3
2	Algeria	Africa	Northern Africa	Developing regions	80	67	71	69	63	44	69	132	242	434
3	American Samoa	Oceania	Polynesia	Developing regions	0	1	0	0	0	0	0	1	0	1
4	Andorra	Europe	Southern Europe	Developed regions	0	0	0	0	0	0	2	0	0	0

In order to create a `Choropleth` map, we need a GeoJSON file that defines the areas/boundaries of the state, county, or country that we are interested in. In our case, since we are endeavoring to create a world map, we want a GeoJSON that defines the boundaries of all world countries. For your convenience, we will be providing you with this file, so let's go ahead and download it. Let's name it **world_countries.json**.

```
# download countries geojson file
!wget --quiet https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data_Files/world_countries.json -O world_countries.json

print('GeoJSON file downloaded!')
```

```
GeoJSON file downloaded!
```

Now that we have the GeoJSON file, let's create a world map, centered around **[0, 0]** *latitude* and *longitude* values, with an intial zoom level of 2, and using *Mapbox Bright* style.

```
world_geo = r'world_countries.json' # geojson file

# create a plain world map
world_map = folium.Map(location=[0, 0], zoom_start=2, tiles='Mapbox Bright')
```

And now to create a `Choropleth` map, we will use the *choropleth* method with the following main parameters:

1. geo_data, which is the GeoJSON file.
2. data, which is the dataframe containing the data.
3. columns, which represents the columns in the dataframe that will be used to create the `Choropleth` map.

4. key_on, which is the key or variable in the GeoJSON file that contains the name of the variable of interest. To determine that, you will need to open the GeoJSON file using any text editor and note the name of the key or variable that contains the name of the countries, since the countries are our variable of interest. In this case, **name** is the key in the GeoJSON file that contains the name of the countries. Note that this key is case_sensitive, so you need to pass exactly as it exists in the GeoJSON file.

```
# generate choropleth map using the total immigration of each country to Canada
from 1980 to 2013
world_map.choropleth(
    geo_data=world_geo,
    data=df_can,
    columns=['Country', 'Total'],
    key_on='feature.properties.name',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Immigration to Canada'
)

# display map
world_map
```

```
/opt/conda/lib/python3.6/site-packages/folium/folium.py:426: FutureWarning: The
choropleth  method has been deprecated. Instead use the new Choropleth class,
which has the same arguments. See the example notebook 'GeoJSON_and_choropleth'
for how to do this.
  FutureWarning
```

As per our `Choropleth` map legend, the darker the color of a country and the closer the color to red, the higher the number of immigrants from that country. Accordingly, the highest immigration over the course of 33 years (from 1980 to 2013) was from China, India, and the Philippines, followed by Poland, Pakistan, and interestingly, the US.

Notice how the legend is displaying a negative boundary or threshold. Let's fix that by defining our own thresholds and starting with 0 instead of -6,918!

```
world_geo = r'world_countries.json'

# create a numpy array of length 6 and has linear spacing from the minium total
immigration to the maximum total immigration
threshold_scale = np.linspace(df_can['Total'].min(),
                              df_can['Total'].max(),
                              6, dtype=int)
threshold_scale = threshold_scale.tolist() # change the numpy array to a list
threshold_scale[-1] = threshold_scale[-1] + 1 # make sure that the last value of
the list is greater than the maximum immigration

# let Folium determine the scale.
world_map = folium.Map(location=[0, 0], zoom_start=2, tiles='Mapbox Bright')
world_map.choropleth(
    geo_data=world_geo,
    data=df_can,
    columns=['Country', 'Total'],
    key_on='feature.properties.name',
    threshold_scale=threshold_scale,
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.2,
```

```
    legend_name='Immigration to Canada',
    reset=True
)
world_map
```

```
/opt/conda/lib/python3.6/site-packages/folium/folium.py:426: FutureWarning: The
choropleth  method has been deprecated. Instead use the new Choropleth class,
which has the same arguments. See the example notebook 'GeoJSON_and_choropleth'
for how to do this.
  FutureWarning
```

Much better now! Feel free to play around with the data and perhaps create `Choropleth` maps for individuals years, or perhaps decades, and see how they compare with the entire period from 1980 to 2013.

References

1. https://www.kaggle.com/code/muhammetcepi/creating-maps-and-visualizing-geospatial-data
2. https://www.kaggle.com/code/umangsharma9533/visualizing-geospatial-data-in-python
3. https://www.linkedin.com/pulse/visualizing-geospatial-data-python-willy-simons
4. https://towardsdatascience.com/visualizing-geospatial-data-in-python-e070374fe621