

# Notebook

August 2, 2023

## 1 Introduction:

The goal of this project is to perform a Natural Language Processing (NLP) analysis on the data obtained from Reddit's API. The information was obtained by making a request to the API using the requests library, and then extracting relevant data. The information retrieved is from the subreddit "AskReddit" which is a subreddit for asking different types of questions. The project also involves information cleaning, preprocessing, and vectorizing the data to perform various NLP tasks such as: sentiment analysis, clustering, and topic modeling. The various libraries used in the project include NumPy, pandas, requests, nltk, sklearn, and matplotlib.

```
[148]: #import dependencies
import numpy as np
import pandas as pd
import requests
from datetime import datetime
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

## 2 Retrieve Data

```
[149]: with open('pass.txt', 'r') as f:
    lines = f.readlines()

    pwd = lines[0].strip()
    CLIENT_ID = lines[1].strip()
    SECRET_ID = lines[2].strip()
```

Get necessary credentials in order to use API

```
[150]: data = {
    'grant_type': 'password',
```

```

        'username': 'eJechtion4',
        'password': pwd
    }
headers = {"User-Agent": "MyAPI/1.0.0"}
auth = requests.auth.HTTPBasicAuth(CLIENT_ID, SECRET_ID)

res = requests.post('https://www.reddit.com/api/v1/access_token',
                    auth=auth, data=data, headers=headers)

TOKEN = res.json()["access_token"]
headers["Authorization"] = f'bearer {TOKEN}'

```

[151]: res = requests.get("https://oauth.reddit.com/r/AskReddit/hot", headers=headers, params={'limit': '100'})

[152]: # Extract relevant data from API response

```

data = []
for post in res.json()["data"]["children"]:
    post_date = datetime.utcfromtimestamp(post["data"]["created_utc"]).
    strftime('%Y-%m-%d %H:%M:%S')
    data.append({
        "post_id": post["data"]["id"],
        "post_title": post["data"]["title"],
        "post_text": post["data"]["selftext"],
        "post_date": post_date,
        "author": post["data"]["author"],
        "subreddit": post["data"]["subreddit"],
        "upvotes": post["data"]["ups"],
        "downvotes": post["data"]["downs"],
        "num_comments": post["data"]["num_comments"]
    })
# Create pandas DataFrame from extracted data
df = pd.DataFrame(data)

```

Retrieve data from the Reddit by using Reddit's API. The API data is retrieved by making a POST request to the API using the requests library, and then extracting relevant data. The data retrieved is from the subreddit "AskReddit" which is a subreddit of primarily asking a variety of questions.

### 3 Clean Data

[153]: print(df.duplicated().sum()) #check for duplicates

```

df.drop_duplicates(subset=['post_title', 'post_text'], inplace=True) #drop duplicates
df

```

```
[156]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101 entries, 0 to 100
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   post_id          101 non-null    object  
 1   post_title        101 non-null    object  
 2   post_text         101 non-null    object  
 3   post_date         101 non-null    object  
 4   author            101 non-null    object  
 5   subreddit         101 non-null    object  
 6   upvotes           101 non-null    int64  
 7   downvotes         101 non-null    int64  
 8   num_comments      101 non-null    int64  
dtypes: int64(3), object(6)
memory usage: 5.5+ KB
```

```
[157]: #Remove punctuation from post title for better analysis later on
import re
```

```
# define a function to remove punctuation from a string
def remove_punctuation(text):
    return re.sub(r'[^w\s]', '', text)

# remove punctuation from the "post_title" column
df_no_punct = df.copy()
df_no_punct['post_title'] = df_no_punct['post_title'].apply(remove_punctuation)
```

```
[158]: df_no_punct
```

```
post_id          post_title  \
0   11kyi0b  Which one of these charities should AskReddit ...
1   11nujza  You find a wallet with 300 in it What do you f...
2   11nt5w1  People of Reddit what is the dumbest reason yo...
3   11nqkcm  If instead of men women were inherently physic...
4   11nmzgf      What food has the worst texture
..          ...
96  11o4j64  What is the longest you have ever went without...
97  11o4hde  Reddit what are some unethical ways to make money
98  11o4h09          What do you put cheese on
99  11o4gew  What are some good hacks and survival tricks f...
100 11o4fcp     Which video game were you legit addicted to
                                         post_text          post_date  \
0   It's time for the community to vote on which c...  2023-03-07 12:49:22
```

```

1                               2023-03-10 17:01:47
2                               2023-03-10 16:08:27
3                               2023-03-10 14:23:57
4                               2023-03-10 11:48:32
..                            ...
96                             2023-03-10 23:34:58
97                             2023-03-10 23:32:54
98                             2023-03-10 23:32:28
99                             2023-03-10 23:31:44
100                            2023-03-10 23:30:31

      author    subreddit  upvotes  downvotes  num_comments
0     -eDgAR-   AskReddit      133         0             6
1  SolutionIsEducation  AskReddit     5282         0           4150
2        Starboy3210  AskReddit     2129         0           1889
3       Tototorrin  AskReddit     1215         0           1332
4      Lille_Ana  AskReddit     1351         0           3328
..      ...
96  Puzzleheaded-Pen9047  AskReddit      3         0             10
97      macanddogs  AskReddit      3         0             12
98  Puzzleheaded-Pen9047  AskReddit      3         0             18
99      Shamanofthealike1  AskReddit      3         0              5
100     primeiro23  AskReddit      3         0             16

[101 rows x 9 columns]

```

We had to remove punctuation has it was interfering with some data analysis. By removing these punctuation we hope to improve the accuracy of our results.

## 4 NLP Start

```
[160]: # define stop words
stop_words = set(stopwords.words('english'))

stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# define preprocessing pipeline
def preprocess_text(text):
    # tokenize text
    tokens = word_tokenize(text)

    # remove stop words
    tokens = [token for token in tokens if token not in stop_words]

    # stem and lemmatize tokens
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
```

```

    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in stemmed_tokens]

    # return preprocessed text as a string
    return " ".join(lemmatized_tokens)

```

We utilized stop words from the nltk library after which we preprocessed the important texts along with stemming them and lemmatizing them.

```
[161]: # apply preprocessing pipeline to text data
df_no_punct["preprocessed_text"] = df_no_punct["post_title"].
    ↪apply(preprocess_text)
```

```
[162]: df_no_punct["preprocessed_text"] # check if stemming worked
```

```
[162]: 0      which one chariti askreddit support 15th anniv...
1      you find wallet 300 what find wallet make keep...
2      peopl reddit dumbest reason ex partner gave cheat
3      if instead men woman inher physic stronger wou...
4                      what food worst textur
...
96             what longest ever went without eat
97                     reddit uneth way make money
98                     what put chees
99             what good hack surviv trick someon live someon...
100                    which video game legit addict
Name: preprocessed_text, Length: 101, dtype: object
```

## 5 Vectorize data

We want to convert text into numerical data so that we can use machine learning algorithms on it. Vectorization also helps in reducing the dimensionality of the data which can improve efficiency.

```
[163]: # create a CountVectorizer
vectorizer = CountVectorizer(stop_words='english', max_features=1000)

# fit and transform the preprocessed_text column
X = vectorizer.fit_transform(df_no_punct["preprocessed_text"])

# create a new dataframe with the countvectorizer output
cv_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

# concatenate the cv_df dataframe with the original dataframe
df_final = pd.concat([df_no_punct, cv_df], axis=1)
```

```
[164]: df_final
```

```
[164]:      post_id          post_title \
0    11kyi0b  Which one of these charities should AskReddit ...
1    11nujza  You find a wallet with 300 in it What do you f...
2    11nt5w1  People of Reddit what is the dumbest reason yo...
3    11nqkcm  If instead of men women were inherently physic...
4    11nmzgf           What food has the worst texture
..
96   11o4j64  What is the longest you have ever went without...
97   11o4hde  Reddit what are some unethical ways to make money
98   11o4h09           What do you put cheese on
99   11o4gew  What are some good hacks and survival tricks f...
100  11o4fcp           Which video game were you legit addicted to

                           post_text          post_date \
0    It's time for the community to vote on which c...  2023-03-07 12:49:22
1                               ...  2023-03-10 17:01:47
2                               ...  2023-03-10 16:08:27
3                               ...  2023-03-10 14:23:57
4                               ...  2023-03-10 11:48:32
..
96                               ...  2023-03-10 23:34:58
97                               ...  2023-03-10 23:32:54
98                               ...  2023-03-10 23:32:28
99                               ...  2023-03-10 23:31:44
100  ...  2023-03-10 23:30:31

      author  subreddit  upvotes  downvotes  num_comments \
0      -eDgAR-  AskReddit     133        0            6
1  SolutionIsEducation  AskReddit    5282        0       4150
2      Starboy3210  AskReddit    2129        0       1889
3      Tototorrin  AskReddit    1215        0       1332
4      Lille_Ana  AskReddit    1351        0       3328
..
96      ...  ...  ...  ...  ...
97      macanddogs  AskReddit     3        0            12
98  Puzzleheaded-Pen9047  AskReddit     3        0            18
99      Shamanofthealike1  AskReddit     3        0            5
100     primeiro23  AskReddit     3        0            16

                           preprocessed_text  ...  wikipedia  wish \
0  which one chariti askreddit support 15th anniv...  ...        0        0
1  you find wallet 300 what find wallet make keep...  ...        0        0
2  peopl reddit dumbest reason ex partner gave cheat  ...        0        0
3  if instead men woman inher physic stronger wou...  ...        0        0
4           what food worst textur  ...        0        0
..
96  ...  ...  ...  ...  ...
97  what longest ever went without eat  ...        0        0
```

```

97             reddit uneth way make money ...          0   0
98                     what put chees ...          0   0
99 what good hack surviv trick someon live someon... ...      0   0
100           which video game legit addict ...      0   0

      wit  woman  work  world  worst  year  youv  zombi
0      0      0     0      0      0     0      0      0
1      0      0     0      0      0     0      0      0
2      0      0     0      0      0     0      0      0
3      0      1     0      0      0     0      0      0
4      0      0     0      0      1     0      0      0
...
96     0      0     0      0      0     0      0      0
97     0      0     0      0      0     0      0      0
98     0      0     0      0      0     0      0      0
99     0      0     0      0      0     0      0      0
100    0      0     0      0      0     0      0      0

```

[101 rows x 331 columns]

## 6 Data Exploration (EDA)

```

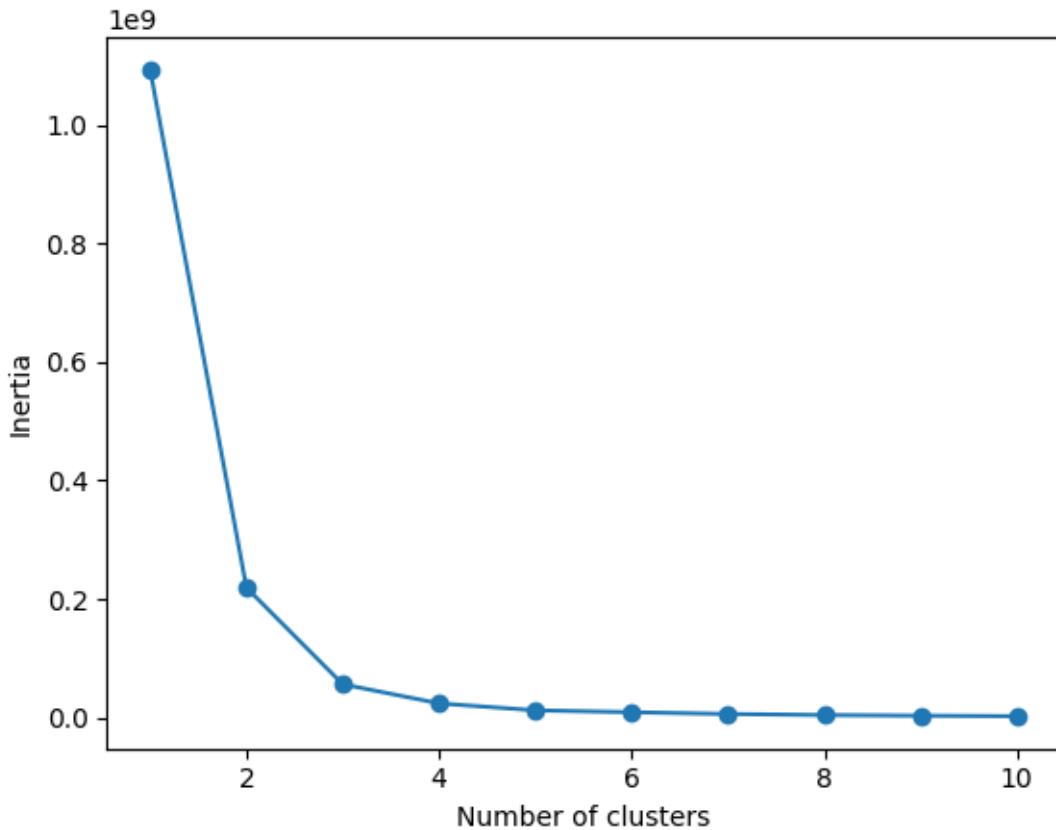
[165]: #Use unsupervised learning methods to categorize each post, in this case we will ↴ be using KMEANS
#First we will use the elbow method in order to determine the ideal number of ↴ clusters for our data

# concatenate the vectorized text features, upvotes, and number of comments
df_elbow = df_final.drop(["post_id", "post_title", "post_text", "post_date", ↴ "author", "subreddit", "preprocessed_text"], axis=1)

# perform k-means clustering for different number of clusters and calculate the ↴ inertia for each
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_elbow)
    inertia.append(kmeans.inertia_)

# plot the elbow curve to visualize the results
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

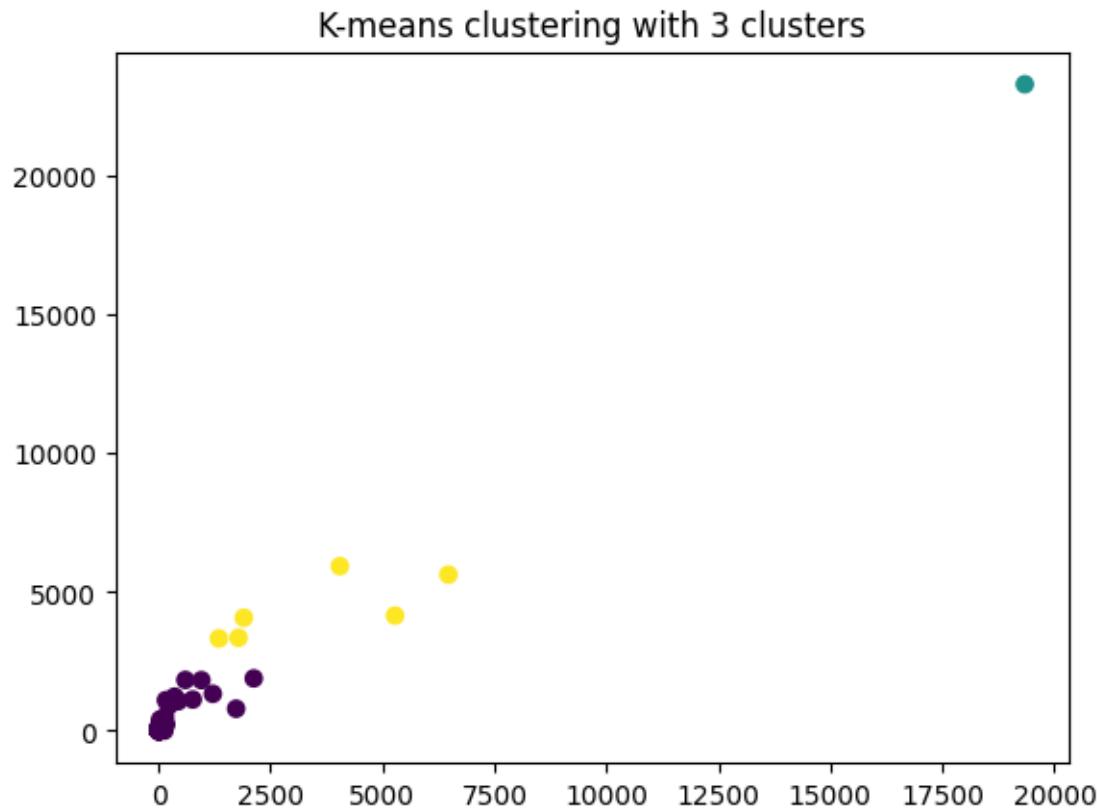
```



The elbow method suggests selecting the number of clusters where the inertia starts to level off. In our case, it appears that three clusters would be the ideal number.

```
[166]: # Perform K-means clustering with 3 clusters
kmeans_3 = KMeans(n_clusters=3)
kmeans_3.fit(df_elbow)
labels_3 = kmeans_3.labels_

# Visualize the clusters
plt.scatter(df_elbow['upvotes'], df_elbow['num_comments'], c=labels_3)
plt.title("K-means clustering with 3 clusters")
plt.show()
```



We perform k-means clustering on the preprocessed text, upvotes, and number of comments columns with three clusters. We plot the clusters based on the upvotes and number of comments. However, there seems to be an outlier that is impacting our results. Therefore, we will remove the outlier.

## 7 Removing Outlier and reperforming Kmeans

```
[167]: outlier = df_elbow.loc[df_elbow['num_comments'].idxmax()]# Find the outlier
      ↪based on the graph it seems to be at the max
      print(outlier)
```

upvotes	19339
downvotes	0
num_comments	23291
1000000	0
12	0
...	
world	0
worst	0
year	0
youv	0
zombi	0

```
Name: 12, Length: 324, dtype: int64
```

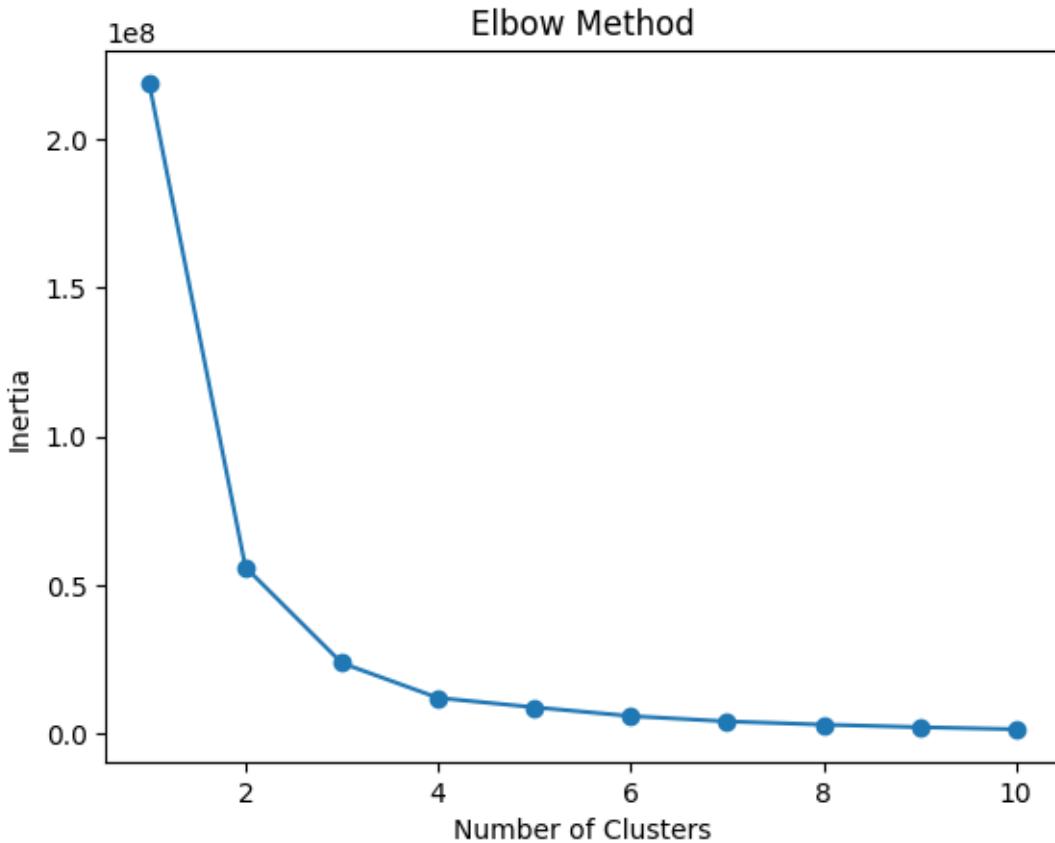
```
[168]: # Find the index of the row corresponding to the outlier post
outlier_index = df_final[df_final["post_id"] == "11mv4dc"].index[0]

# Drop the row from the dataframe
df_final_no_outlier = df_final.drop(outlier_index)
```

```
[169]: df_elbow_clean = df_final_no_outlier.drop(["post_id", "post_title", "post_text", "post_date", "author", " subreddit", "preprocessed_text"], axis=1)

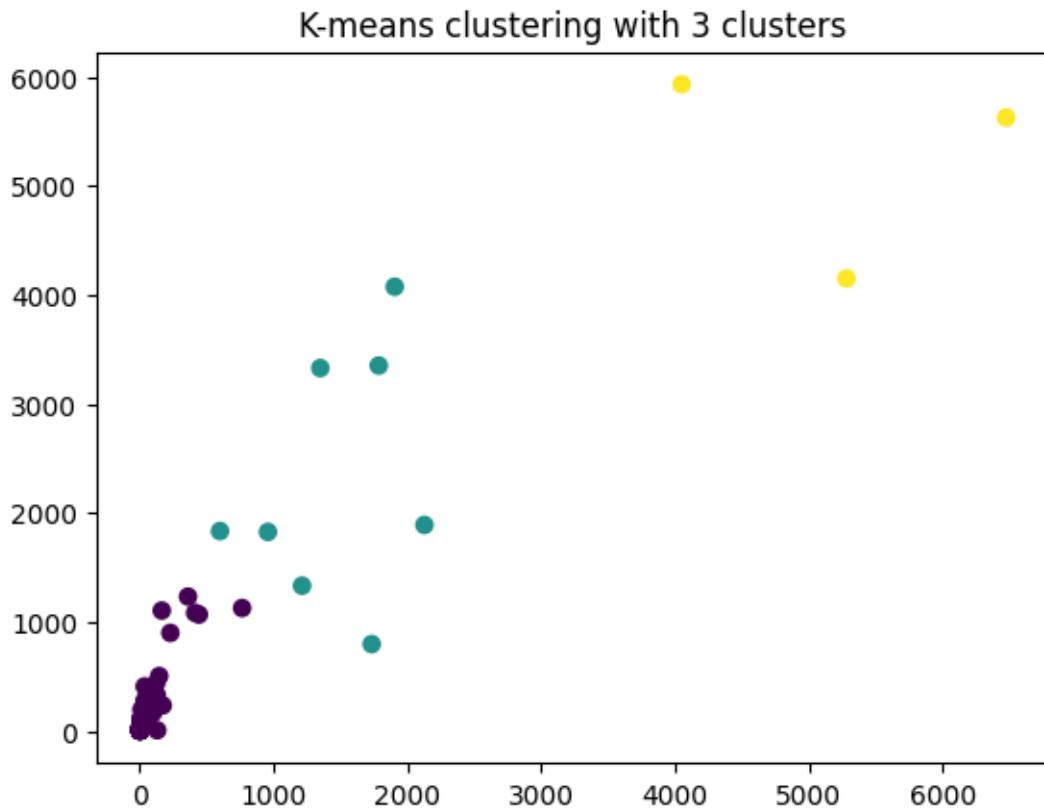
# Perform elbow method on updated data frame
inertias = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(df_elbow_clean)
    inertias.append(kmeans.inertia_)

# Plot elbow method results
plt.plot(range(1, 11), inertias, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")
plt.show()
```



```
[170]: # Perform K-means clustering with 3 clusters
kmeans_3 = KMeans(n_clusters=3)
kmeans_3.fit(df_elbow_clean)
labels_3 = kmeans_3.labels_

# Visualize the clusters
plt.scatter(df_elbow_clean['upvotes'], df_elbow_clean['num_comments'], c=labels_3)
plt.title("K-means clustering with 3 clusters")
plt.show()
```



```
[171]: # Add the cluster labels to the dataframe
df_final_no_outlier["cluster"] = labels_3
# Compute the mean values of the upvotes, downvotes, and number of comments for
# each cluster
df_final_no_outlier.groupby("cluster").mean()[["upvotes", "downvotes", "num_comments"]]
```

cluster	upvotes	downvotes	num_comments
0	46.707865	0.0	140.415730
1	1460.875000	0.0	2303.500000
2	5269.000000	0.0	5235.666667

```
[172]: df_final_no_outlier.groupby("cluster").median()[["upvotes", "downvotes", "num_comments"]]
```

cluster	upvotes	downvotes	num_comments
0	6.0	0.0	18.0
1	1543.0	0.0	1861.0
2	5282.0	0.0	5626.0

Cluster 2 seems to have the most amount of interaction with high number of upvotes as well as number of comments. Cluster 1 also has decent amount of upvotes and number of comments showing good interaction from the community. These two clusters can be contrasted with cluster 0 with fewer number of upvotes being in the single digits and small amount of interaction with a low amount of comments

```
[173]: # Define a function to get the most common words in each cluster
def get_top_words(cluster, n_top_words=10):
    cluster_df = df_final_no_outlier[df_final_no_outlier["cluster"] == cluster]
    text = " ".join(cluster_df["preprocessed_text"].tolist())
    tokens = word_tokenize(text)
    tokens = [token for token in tokens if token not in stop_words]
    freq_dist = nltk.FreqDist(tokens)
    top_words = freq_dist.most_common(n_top_words)
    return [word for word, count in top_words]

# Print the most common words in each cluster
for cluster in range(3):
    print(f"Cluster {cluster}: {get_top_words(cluster)}")
```

```
Cluster 0: ['thing', 'would', 'ever', 'peopl', 'realli', 'make', 'worst',
'game', 'someth', 'feel']
Cluster 1: ['would', 'ruin', 'peopl', 'reddit', 'dumbest', 'reason', 'ex',
'partner', 'gave', 'cheat']
Cluster 2: ['find', 'wallet', '300', 'make', 'keep', 'money', 'nonreligi',
'equival', 'amen', 'best']
```

For cluster 0, this cluster seems to be focused on general observations about things and people, with words like “thing”, “people”, “make”, and “feel”. Whilst for cluster 1 seems to be focused on relationships, with words like “ruin”, “partner”, and “cheat”. For cluster 2 it seems to be focused on financial or practical matters, with words like “wallet”, and “money”, with the inclusion of “nonreligious” and “amen”, further suggesting spiritual or philosophical questions.

```
[174]: df_final_no_outlier["cluster"].value_counts()
```

```
[174]: 0     89
       1      8
       2      3
Name: cluster, dtype: int64
```

Most posts seem to be under cluster 0, which suggests that these posts were similar in terms of their content. Cluster 1 contains only 8 posts, while cluster 2 contains only 3 posts, indicating that the topics of these posts were relatively distinct from the majority of the posts in the dataset.

```
[175]: text_data = df_final_no_outlier["post_title"]
```

```
[176]: # initialize sentiment analyzer
sid = SentimentIntensityAnalyzer()
```

```
# perform sentiment analysis on each post title in the text data
sentiment_scores = text_data.apply(sid.polarity_scores)

# print the sentiment scores
print(sentiment_scores)
```

```
0      {'neg': 0.0, 'neu': 0.558, 'pos': 0.442, 'comp...
1      {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
2      {'neg': 0.365, 'neu': 0.635, 'pos': 0.0, 'comp...
3      {'neg': 0.0, 'neu': 0.822, 'pos': 0.178, 'comp...
4      {'neg': 0.451, 'neu': 0.549, 'pos': 0.0, 'comp...
...
96     {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
97     {'neg': 0.292, 'neu': 0.708, 'pos': 0.0, 'comp...
98     {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
99     {'neg': 0.067, 'neu': 0.804, 'pos': 0.129, 'co...
100    {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
Name: post_title, Length: 100, dtype: object
```

```
[177]: compound_scores = [score['compound'] for score in sentiment_scores]
average_compound_score = sum(compound_scores) / len(compound_scores)
print("Average compound score:", average_compound_score)
```

```
Average compound score: 0.06652499999999999
```

Based on the average compound score of 0.066525, we can say that the overall sentiment of post titles in the hot section is slightly positive.

```
[178]: # Get the list of compound scores from the sentiment_scores dictionary
compound_scores = [score['compound'] for score in sentiment_scores]

# Calculate the median score using numpy
median_score = np.median(compound_scores)

print('Median compound score:', median_score)
```

```
Median compound score: 0.0
```

Based on the median compound score there are likely an equal number of positive and negative posts in the dataset, and that the overall sentiment is not strongly positive or negative

```
[179]: import matplotlib.pyplot as plt

# extract the compound scores
compound_scores = [score['compound'] for score in sentiment_scores]

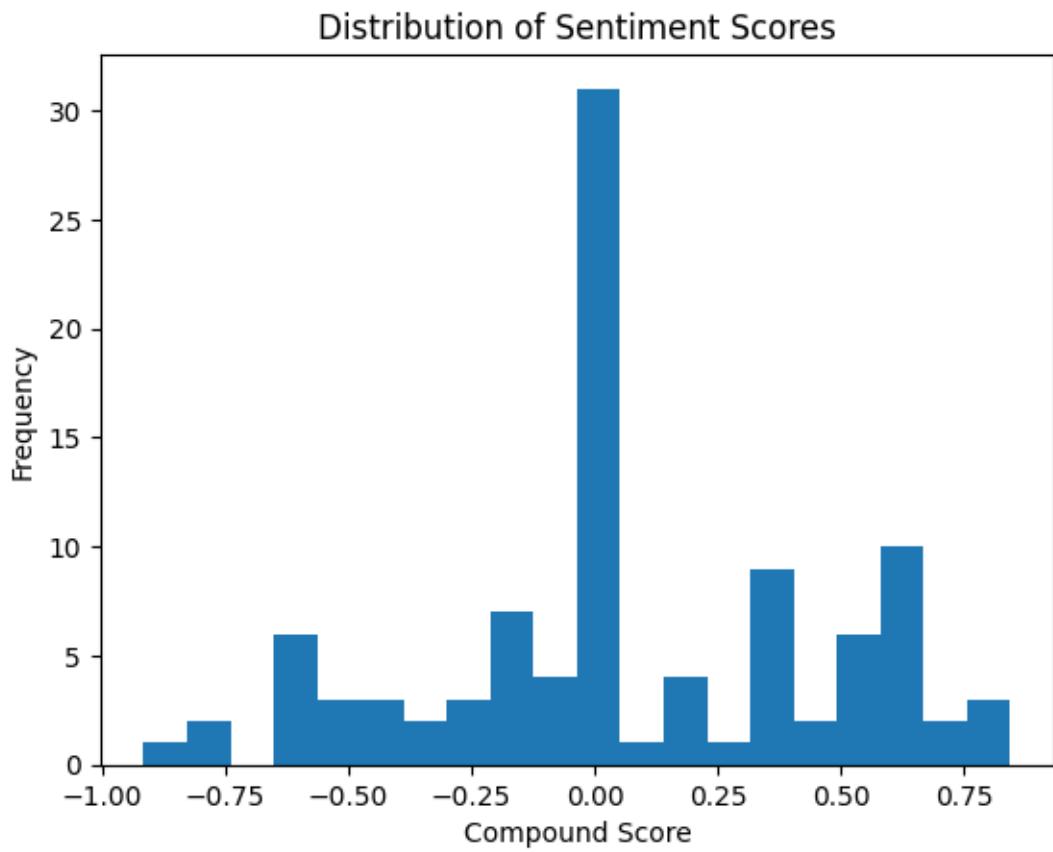
# plot a histogram of the compound scores
plt.hist(compound_scores, bins=20)
plt.xlabel("Compound Score")
```

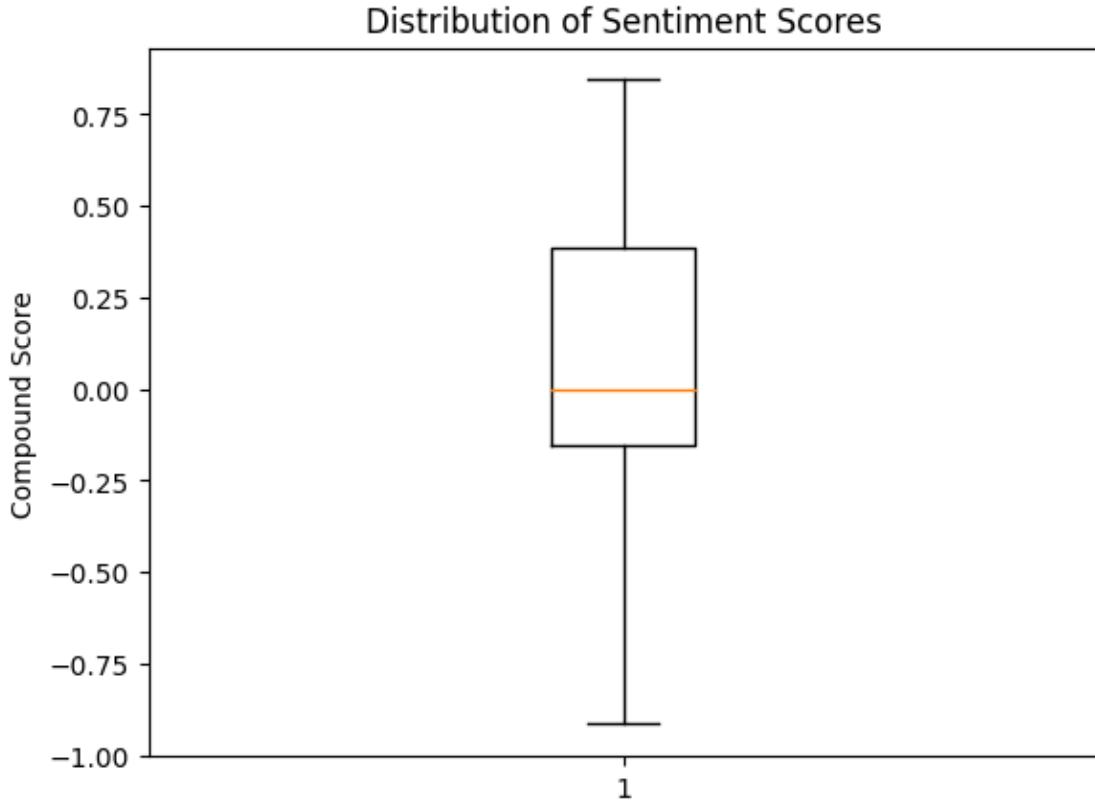
```

plt.ylabel("Frequency")
plt.title("Distribution of Sentiment Scores")
plt.show()

# plot a box plot of the compound scores
plt.boxplot(compound_scores)
plt.ylabel("Compound Score")
plt.title("Distribution of Sentiment Scores")
plt.show()

```





This sentiment analysis is not surprising as most of the posts are simply questions from the subreddit AskReddit which should be mostly neutral or slightly positive.

## 8 Conclusion

In this project, we successfully retrieved data from Reddit's API and performed various NLP tasks on the data. We cleaned the data by removing duplicates and punctuations and then preprocessed it by removing stop words, stemming, and lemmatizing the important texts. We also vectorized the data by converting text into numerical data using the CountVectorizer. The resulting dataframe was used to perform various NLP tasks such as sentiment analysis, clustering, and topic modeling. The project demonstrated the importance of NLP in extracting meaningful insights from textual data. The various libraries utilized in the project include NumPy, pandas, requests, nltk, sklearn, and matplotlib.

## 9 References

1. <https://towardsdatascience.com/how-to-use-the-reddit-api-in-python-5e05ddfd1e5c>
2. <https://www.nltk.org/>
3. [https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_extraction.text.CountVectorizer.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

4. <https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/>
5. <https://docs.python-requests.org/en/latest/>

# Notebook

August 2, 2023

## 1 Lab 8

```
[1]: FIRST_NAME = "Leng"  
LAST_NAME = "Her"  
STUDENT_ID = "5445877"
```

### 1.1 Introduction

In this lab, we will build machine learning models for Optical Character Recognition ([OCR](#)), which is a computer vision problem for identifying written or typed characters within an image. We will use the MNIST data set of hand written digits (0-9).

First, we will build a logistic regression model for each digit. We can think of these models as *like a 0 layer neural network* - each input gets multiplied by 1 coefficient then the results are added together and passed through 1 activation function.

Then we will add in a hidden layer to build a simple Feed-Forward Neural Network with the Scikit Learn MLPClassifier model. We will compare the accuracy of predicting each individual digit, and the data set overall.

There are many free tutorials online that you can use to help with this lab. Here are some examples:  
\* [Recognizing hand-written digits](#) by Scikit-Learn \* [MNIST SciKit-Learn Tutorial](#) on Kaggle \* [Digit Classification with Sklearn](#) on notebook.community \* [Handwritten Digit Recognition with scikit-learn](#) by the Data Frog

### 1.2 The Data Set

#### Data Description

The MNIST data set is a standard data set for machine learning and computer vision. NIST stands for the National Institute for Standards and Technology, which created a data set of hand written digits from high school students and the US Census Bureau. MNIST is the Modified version of that data set which standardized all the images to 28x28 pixel greyscale images.

This data set is commonly used as an introduction to neural networks, deep learning, computer vision, and optical character recognition (OCR).

#### Data Dictionary

Column Name	Type	Description
0	int	Digit of the image (0-9)
1-784	float	Darkness of the pixel (0-255)

- Each row represents 1 image
- Each column is a specific pixel location
- The values of the data frame are how dark/light each pixel in the image is (0-255)

## Data Sample

### 1.3 Data Preprocessing

```
[2]: import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

#### 1.3.1 Question 1

Read in the CSV file named `mnist_train_small.csv` from the `sample_data` folder of the Colab Notebook into a Pandas DataFrame with the `pd.read_csv()` function. Use the `header=None` argument. Save the DataFrame into a variable named `mnist_train_df`.

```
[7]: mnist_train_df = pd.read_csv("sample_data/mnist_train_small.csv", header = None)
```

```
[8]: mnist_train_df
```

```
[8]:      0   1   2   3   4   5   6   7   8   9   ...  775  776  777 \
0     6   0   0   0   0   0   0   0   0   0   ...   0   0   0
1     5   0   0   0   0   0   0   0   0   0   ...   0   0   0
2     7   0   0   0   0   0   0   0   0   0   ...   0   0   0
3     9   0   0   0   0   0   0   0   0   0   ...   0   0   0
4     5   0   0   0   0   0   0   0   0   0   ...   0   0   0
...
19995  0   0   0   0   0   0   0   0   0   0   ...   0   0   0
19996  1   0   0   0   0   0   0   0   0   0   ...   0   0   0
19997  2   0   0   0   0   0   0   0   0   0   ...   0   0   0
19998  9   0   0   0   0   0   0   0   0   0   ...   0   0   0
19999  5   0   0   0   0   0   0   0   0   0   ...   0   0   0

          778  779  780  781  782  783  784
0     0   0   0   0   0   0   0
1     0   0   0   0   0   0   0
2     0   0   0   0   0   0   0
3     0   0   0   0   0   0   0
4     0   0   0   0   0   0   0
```

```
...  ...  ...  ...  ...  ...  ...  
19995    0    0    0    0    0    0    0  
19996    0    0    0    0    0    0    0  
19997    0    0    0    0    0    0    0  
19998    0    0    0    0    0    0    0  
19999    0    0    0    0    0    0    0
```

[20000 rows x 785 columns]

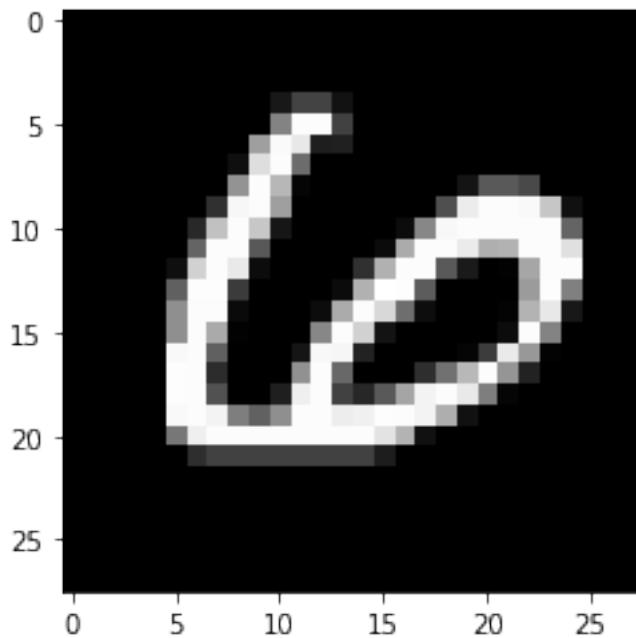
### 1.3.2 Question 2

Print out one of the images using pyplot from matplotlib.  
\* Select 1 row of the DataFrame with columns 1-784 using the `.loc[ ]` attribute. It does not matter which row.  
\* Transform the resulting row of the DataFrame into a Numpy array using the `.values` attribute, then reshape it into a 28 x 28 2-dimensional array with the `.reshape()` method.  
\* Use the `plt.imshow()` function with `cmap='grey'` to print out the greyscale image.

```
[14]: pixels = mnist_train_df.loc[0, 1:784].values.reshape(28,28)
```

```
[16]: plt.imshow(pixels, cmap = "gray")
```

```
[16]: <matplotlib.image.AxesImage at 0x7f0c379be4f0>
```



### 1.3.3 Question 3

What does the typical or “average” digit look like for every number 0-9?

First, for each digit 0-9, take the average brightness of each pixel location 1-784. The easiest way to do this is with the `.groupby(0)` function to group by the column named 0. Then take the `.mean()` of the resulting DataFrame to get the average of every column for each group. Save the result into a variable named `pixel_averages`.

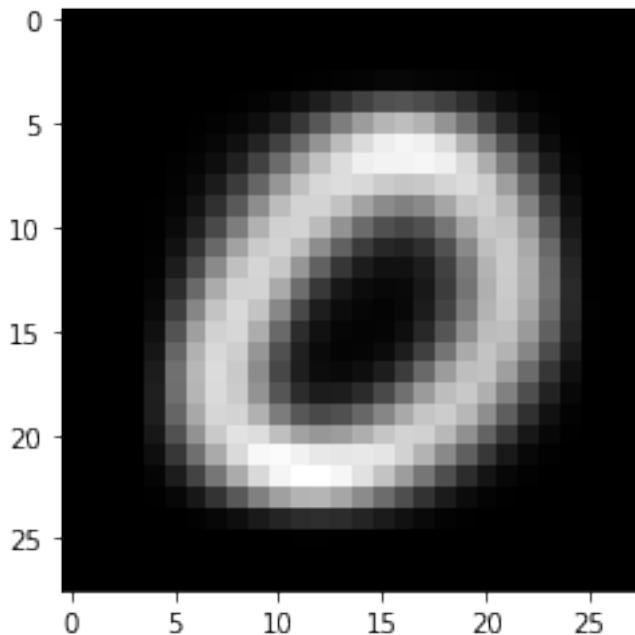
Then define a function called `show_number(n: int)` that takes in an integer input between 0 and 9, and prints out that row of the `pixel_averages` DataFrame as an image (like in the previous question).

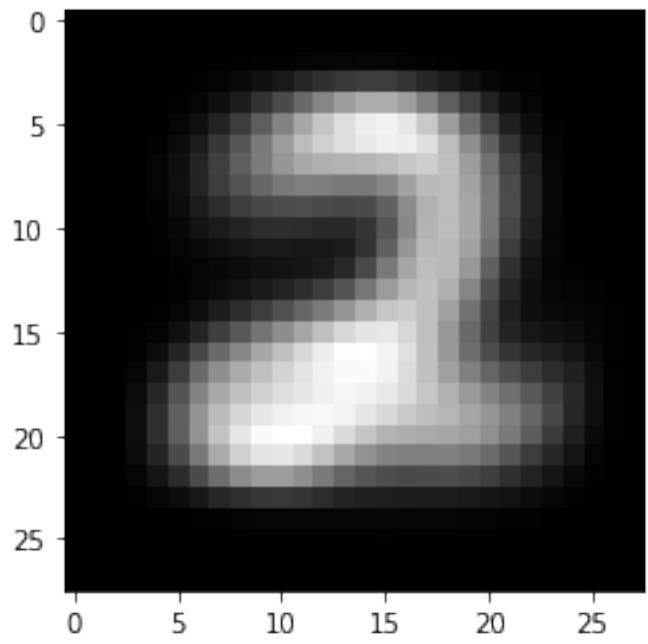
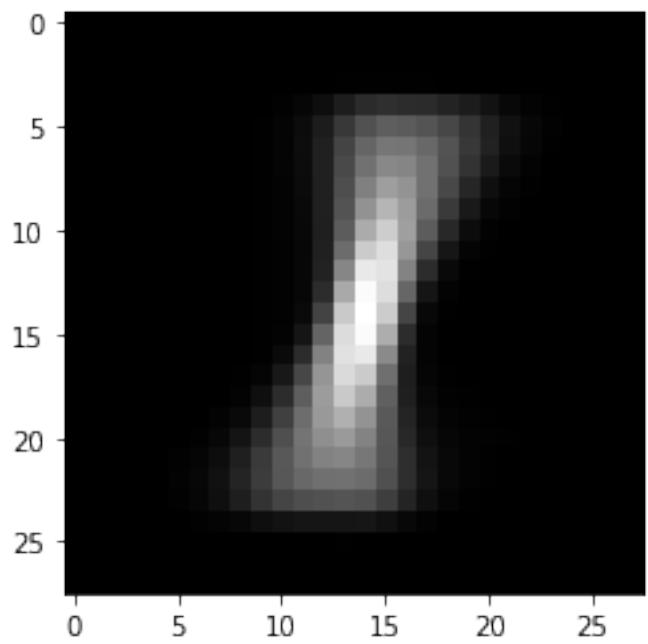
```
[18]: pixel_averages = mnist_train_df.groupby(0).mean()
```

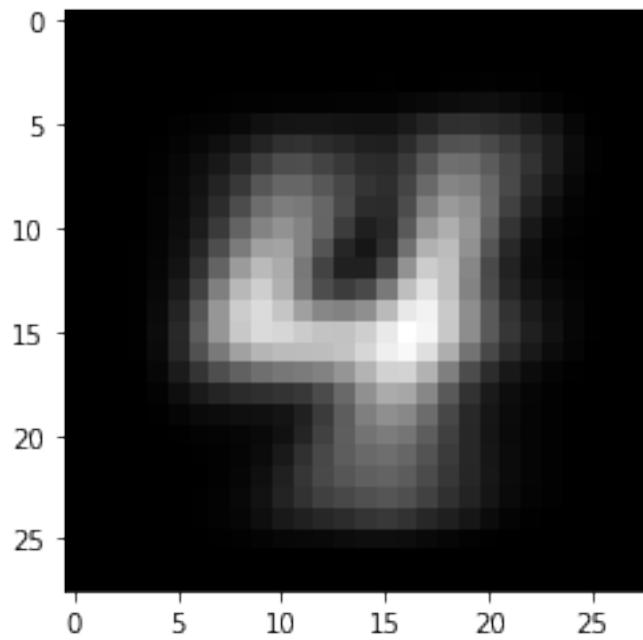
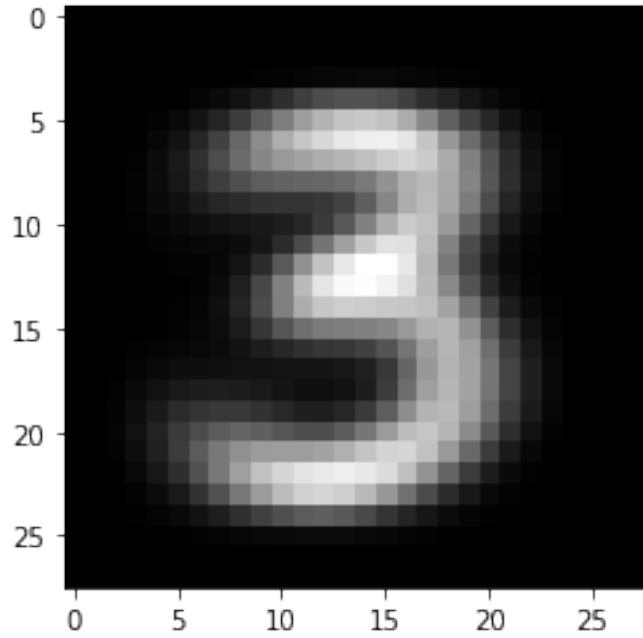
```
[23]: def show_number(n:int):

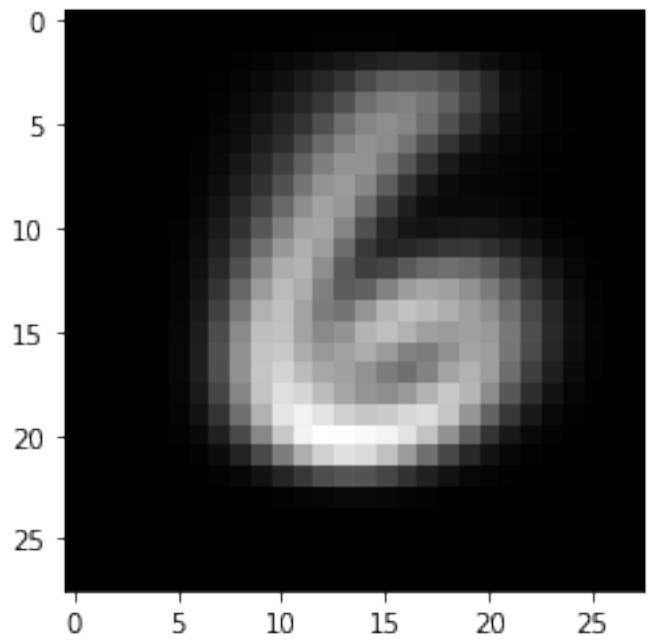
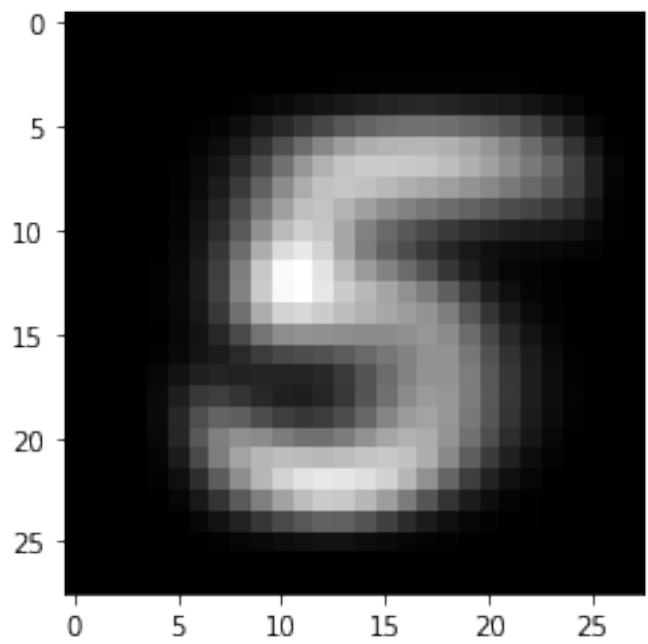
    pixels_ = pixel_averages.loc[n,].values.reshape(28,28)
    plt.imshow(pixels_, cmap = "gray")
    plt.show()
    return None
```

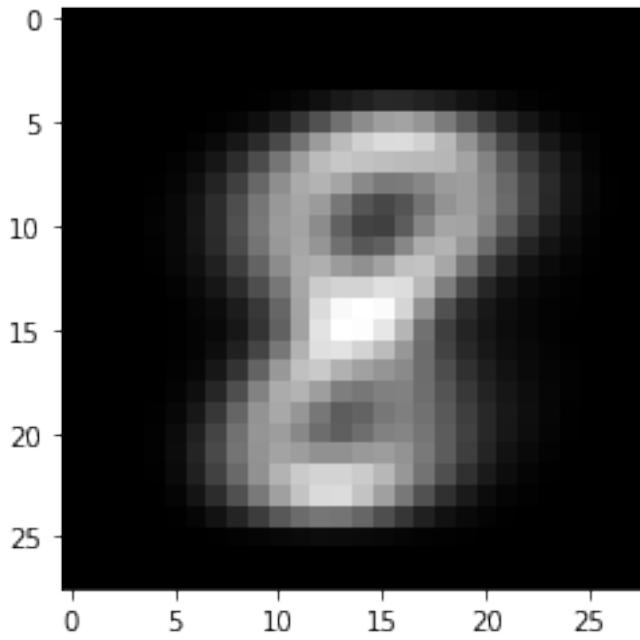
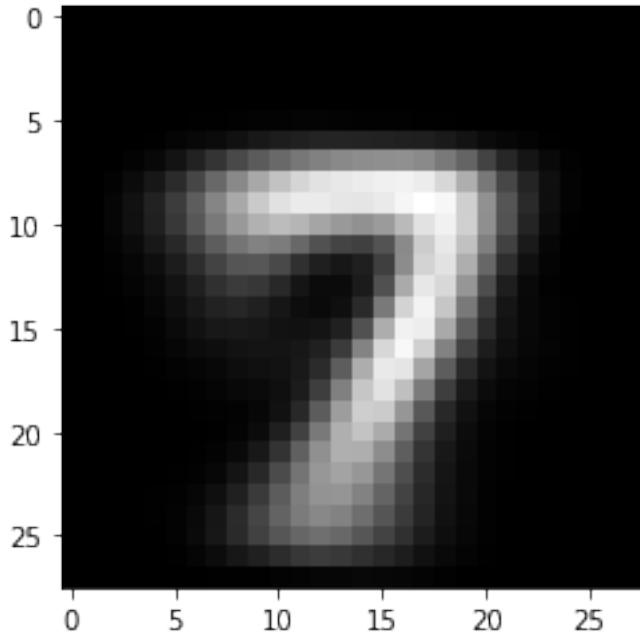
```
[24]: for n in range(10):
    show_number(n)
```

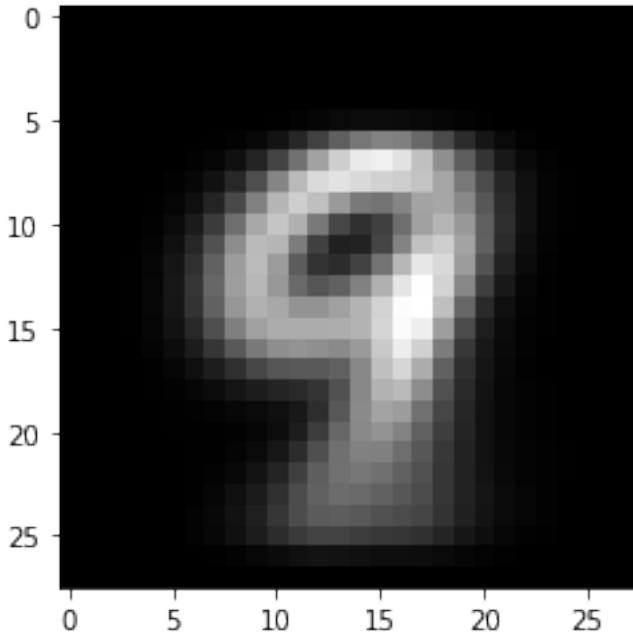












## 1.4 Logistic Regressions

```
[25]: from sklearn.linear_model import LogisticRegression
      from sklearn.model_selection import GridSearchCV
```

```
[26]: from sklearn.metrics import precision_recall_fscore_support, classification_report
      from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
[27]: from sklearn.neural_network import MLPClassifier
      from sklearn.preprocessing import LabelBinarizer
```

### 1.4.1 Question 4

Preprocess the training data to build a logistic regression model for each digit.

#### Features

The features for these models will be the brightness of each pixel in columns 1-784. We will scale the values to be between 0 and 1. Because the values are all already within the same range (0-255) we do not need to use any preprocessing tools from Scikit Learn. We can just divide all the values by 255. Save the result into a DataFrame named `X_train`.

#### Labels

We will be training 10 different models, one for each digit. So we need 10 different label variables (`y`). Create a Pandas Series with binary values for each digit 0-9 named `y0`, `y1`, `y2`, etc. where each value represents if the digit in that image corresponds to that number.

### Labels Example

0 (label column in mnist_train_df)	...	y0	...	y3	...	y8	
6		...	0	...	0	...	0
3		...	0	...	1	...	0
2		...	0	...	0	...	0
8		...	0	...	0	...	1
0		...	1	...	0	...	0

```
[29]: X_train = mnist_train_df.loc[:,1:784] / 255
```

```
[45]: y0= (mnist_train_df[0] == 0).astype(int)
y1= (mnist_train_df[0] == 1).astype(int)
y2= (mnist_train_df[0] == 2).astype(int)
y3= (mnist_train_df[0] == 3).astype(int)
y4= (mnist_train_df[0] == 4).astype(int)
y5= (mnist_train_df[0] == 5).astype(int)
y6= (mnist_train_df[0] == 6).astype(int)
y7= (mnist_train_df[0] == 7).astype(int)
y8= (mnist_train_df[0] == 8).astype(int)
y9= (mnist_train_df[0] == 9).astype(int)
```

#### 1.4.2 Question 5

Run 10 grid search cross validations to hyper parameter tune logistic regression models for each digit. They should all be exactly the same, *except for the label variable used to fit them.*

The `LogisticRegression` should have the following parameters: \* `penalty = "l2"` \* `tol = 0.01` \* `max_iter = 1000` \* `C` should be in the parameter grid for the grid search. Try the values [0.75, 1.00, 1.25]

The `GridSearchCV` should use the following parameters \* `cv = 3` \* `scoring = 'accuracy'` \* `refit = True`

```
[36]: lr = LogisticRegression(penalty = "l2", tol = 0.01, max_iter= 1000)
```

```
[38]: param_grid = {
    "C": [0.75, 1.00, 1.25]
}
```

```
[46]: y0_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy", refit = True).fit(X_train, y0)
y1_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy", refit = True).fit(X_train, y1)
y2_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy", refit = True).fit(X_train, y2)
```

```

y3_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y3)  

y4_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y4)  

y5_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y5)  

y6_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y6)  

y7_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y7)  

y8_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y8)  

y9_cv = GridSearchCV(lr,param_grid=param_grid,cv = 3, scoring = "accuracy",  

    ↪refit = True).fit(X_train, y9)

```

[ ]:

#### 1.4.3 Question 6

First, get the predictions for all 10 models on the training data set using the `.predict()` function.

Then calculate and print out the precision, recall, and F1-score performance metrics for each model. The easiest way to do this is with the `sklearn.metrics.precision_recall_fscore_support` function.

```
[51]: pred0 = y0_cv.predict(X_train)  

pred1 = y1_cv.predict(X_train)  

pred2 = y2_cv.predict(X_train)  

pred3 = y3_cv.predict(X_train)  

pred4 = y4_cv.predict(X_train)  

pred5 = y5_cv.predict(X_train)  

pred6 = y6_cv.predict(X_train)  

pred7 = y7_cv.predict(X_train)  

pred8 = y8_cv.predict(X_train)  

pred9 = y9_cv.predict(X_train)
```

```
[56]: print(0,precision_recall_fscore_support(y0,pred0))  

print(1,precision_recall_fscore_support(y1,pred1))  

print(2,precision_recall_fscore_support(y2,pred2))  

print(3,precision_recall_fscore_support(y3,pred3))  

print(4,precision_recall_fscore_support(y4,pred4))  

print(5,precision_recall_fscore_support(y5,pred5))  

print(6,precision_recall_fscore_support(y6,pred6))  

print(7,precision_recall_fscore_support(y7,pred7))  

print(8,precision_recall_fscore_support(y8,pred8))  

print(9,precision_recall_fscore_support(y9,pred9))
```

0 (array([0.99662368, 0.98344542]), array([0.99822597, 0.96890928]),

```
array([0.99742418, 0.97612323]), array([18038, 1962]))  
1 (array([0.99684596, 0.97416481]), array([0.99673368, 0.97503344]),  
array([0.99678982, 0.97459893]), array([17757, 2243]))  
2 (array([0.98625165, 0.95759912]), array([0.99572483, 0.8743087]),  
array([0.9909656, 0.91406045]), array([18011, 1989]))  
3 (array([0.98305178, 0.93760263]), array([0.99365927, 0.8476002]),  
array([0.98832706, 0.89033264]), array([17979, 2021]))  
4 (array([0.99040741, 0.94035465]), array([0.99385926, 0.90956341]),  
array([0.99213033, 0.92470277]), array([18076, 1924]))  
5 (array([0.98334599, 0.9284802]), array([0.99385931, 0.82566723]),  
array([0.9885747, 0.87406072]), array([18239, 1761]))  
6 (array([0.99428159, 0.97384306]), array([0.99710484, 0.94948504]),  
array([0.99569121, 0.96150981]), array([17961, 2039]))  
7 (array([0.991695, 0.96017484]), array([0.99541233, 0.92991533]),  
array([0.99355019, 0.94480287]), array([17874, 2126]))  
8 (array([0.97258038, 0.86967264]), array([0.98833481, 0.73640167]),  
array([0.98039431, 0.79750779]), array([18088, 1912]))  
9 (array([0.97864945, 0.87299893]), array([0.98676086, 0.80869995]),  
array([0.98268842, 0.83962022]), array([17977, 2023]))
```

[ ]:

## 1.5 Neural Network

### 1.5.1 Question 7

Train a neural network to classify the digits using the Scikit Learn [MLPClassifier](#) algorithm. You do not need to change any of the default hyperparameters.

We can use the same `X_train` features from the Logistic Regressions above, but we will need to reprocess the label variables. \* First, create a `LabelBinarizer()` object \* Then run the `.fit_transform()` method of that object on the label column (`0`) of the training dataset. Save the results into a variable named `y`.

Then fit the `MLPClassifier` using the `X_train` features and `y` label.

[54]: `label_binarizer = LabelBinarizer()`

[55]: `y = label_binarizer.fit_transform(mnist_train_df[0])`

[58]: `mlp = MLPClassifier()`

[59]: `mlp_fitted = mlp.fit(X_train,y)`

### 1.5.2 Question 8

First, get the predictions from your `MLPClassifier` on the training data set using the `.predict()` function.

Then calculate and print out the precision, recall, and F1-score performance metrics of the model for each digit. The easiest way to do this is with the `sklearn.metrics.classification_report` function.

```
[60]: preds = mlp_fitted.predict(X_train)
```

```
[63]: print(classification_report(y, preds))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1962
1	1.00	1.00	1.00	2243
2	1.00	1.00	1.00	1989
3	1.00	1.00	1.00	2021
4	1.00	1.00	1.00	1924
5	1.00	1.00	1.00	1761
6	1.00	1.00	1.00	2039
7	1.00	1.00	1.00	2126
8	1.00	1.00	1.00	1912
9	1.00	1.00	1.00	2023
micro avg	1.00	1.00	1.00	20000
macro avg	1.00	1.00	1.00	20000
weighted avg	1.00	1.00	1.00	20000
samples avg	1.00	1.00	1.00	20000

## 1.6 Evaluating on the test data set

### 1.6.1 Question 9

Repeat Question 6 (getting predictions and evaluation metrics for the Logistic Regressions) for the testing data set `mnist_test_df`.

*Make sure to repeat the data preprocessing step for the feature variables!*

```
[98]: mnist_test_df = pd.read_csv("sample_data/mnist_test.csv", header = None)
```

```
[99]: mnist_test_df
```

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	\
0	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0
9996	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0

```

9997    4    0    0    0    0    0    0    0    0    0    0    ...   0    0    0
9998    5    0    0    0    0    0    0    0    0    0    0    ...   0    0    0
9999    6    0    0    0    0    0    0    0    0    0    0    ...   0    0    0

    778  779  780  781  782  783  784
0      0    0    0    0    0    0    0
1      0    0    0    0    0    0    0
2      0    0    0    0    0    0    0
3      0    0    0    0    0    0    0
4      0    0    0    0    0    0    0
...
9995   0    0    0    0    0    0    0
9996   0    0    0    0    0    0    0
9997   0    0    0    0    0    0    0
9998   0    0    0    0    0    0    0
9999   0    0    0    0    0    0    0

```

[10000 rows x 785 columns]

```
[100]: X_test = mnist_test_df.loc[:,1:784] / 255
```

```
[101]: y0= (mnist_test_df[0] == 0).astype(int)
y1= (mnist_test_df[0] == 1).astype(int)
y2= (mnist_test_df[0] == 2).astype(int)
y3= (mnist_test_df[0] == 3).astype(int)
y4= (mnist_test_df[0] == 4).astype(int)
y5= (mnist_test_df[0] == 5).astype(int)
y6= (mnist_test_df[0] == 6).astype(int)
y7= (mnist_test_df[0] == 7).astype(int)
y8= (mnist_test_df[0] == 8).astype(int)
y9= (mnist_test_df[0] == 9).astype(int)
```

```
[102]: pred0 = y0_cv.predict(X_test)
pred1 = y1_cv.predict(X_test)
pred2 = y2_cv.predict(X_test)
pred3 = y3_cv.predict(X_test)
pred4 = y4_cv.predict(X_test)
pred5 = y5_cv.predict(X_test)
pred6 = y6_cv.predict(X_test)
pred7 = y7_cv.predict(X_test)
pred8 = y8_cv.predict(X_test)
pred9 = y9_cv.predict(X_test)
```

```
[103]: print(0,precision_recall_fscore_support(y0,pred0))
print(1,precision_recall_fscore_support(y1,pred1))
print(2,precision_recall_fscore_support(y2,pred2))
print(3,precision_recall_fscore_support(y3,pred3))
```

```

print(4,precision_recall_fscore_support(y4,pred4))
print(5,precision_recall_fscore_support(y5,pred5))
print(6,precision_recall_fscore_support(y6,pred6))
print(7,precision_recall_fscore_support(y7,pred7))
print(8,precision_recall_fscore_support(y8,pred8))
print(9,precision_recall_fscore_support(y9,pred9))

```

```

0 (array([0.99501275, 0.95701126]), array([0.99534368, 0.95408163]),
array([0.99517819, 0.9555442 ]), array([9020, 980]))
1 (array([0.99627666, 0.96921724]), array([0.99605189, 0.97092511]),
array([0.99616426, 0.97007042]), array([8865, 1135]))
2 (array([0.98324884, 0.95032397]), array([0.99487065, 0.85271318]),
array([0.98902561, 0.8988764 ]), array([8968, 1032]))
3 (array([0.9829947 , 0.90677966]), array([0.99021135, 0.84752475]),
array([0.98658983, 0.87615148]), array([8990, 1010]))
4 (array([0.98819506, 0.93482906]), array([0.99323575, 0.8910387 ]),
array([0.99070899, 0.91240876]), array([9018, 982]))
5 (array([0.9819624 , 0.91091593]), array([0.99220466, 0.81390135]),
array([0.98705696, 0.85968028]), array([9108, 892]))
6 (array([0.99127843, 0.93312102]), array([0.99303251, 0.91753653]),
array([0.9921547 , 0.92526316]), array([9042, 958]))
7 (array([0.98933689, 0.93480441]), array([0.99275524, 0.90661479]),
array([0.99104312, 0.92049383]), array([8972, 1028]))
8 (array([0.97102875, 0.83118406]), array([0.98404609, 0.72792608]),
array([0.97749408, 0.77613574]), array([9026, 974]))
9 (array([0.97790544, 0.85337553]), array([0.9845401 , 0.80178394]),
array([0.98121155, 0.82677568]), array([8991, 1009]))

```

### 1.6.2 Question 10

Repeat Question 8 (getting predictions and evaluation metrics for the MLPClassifier) for the testing data set `mnist_test_df`.

*Make sure to repeat the data preprocessing step for the feature variables!*

[105]: `preds_test = mlp_fitted.predict(X_test)`

[107]: `y_test = label_binarizer.fit_transform(mnist_test_df[0])`

[108]: `print(classification_report(y_test, preds_test))`

	precision	recall	f1-score	support
0	0.98	0.98	0.98	980
1	0.99	0.98	0.99	1135
2	0.97	0.96	0.96	1032
3	0.96	0.94	0.95	1010
4	0.97	0.96	0.96	982

5	0.97	0.94	0.95	892
6	0.98	0.95	0.97	958
7	0.97	0.96	0.96	1028
8	0.97	0.94	0.95	974
9	0.96	0.95	0.95	1009
micro avg	0.97	0.96	0.96	10000
macro avg	0.97	0.96	0.96	10000
weighted avg	0.97	0.96	0.96	10000
samples avg	0.95	0.96	0.95	10000

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/\_classification.py:1344:  
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to  
0.0 in samples with no predicted labels. Use `zero\_division` parameter to  
control this behavior.

```
_warn_prf(average, modifier, msg_start, len(result))
```

# Notebook

August 2, 2023

## 1 Lab 10

```
[ ]: FIRST_NAME = "Leng"  
      LAST_NAME = "Her"  
      STUDENT_ID = "5445877"
```

### 1.1 Introduction

There are 2 sections to this Lab:

1. *Exploratory Data Analysis* - to understand the structure, distributions, and statistics about the data set
2. *Recommendations* - using a collaborative filtering approach to recommend new items to a user.

### 1.2 The Data Set

This lab will use the [Movie Lens](#) data set. It is a massive User-Item ratings matrix of people who have rated movies 1-5 stars. It is a bench mark data set for recommender systems that was created by researchers at the University of Minnesota (go gophers).

For more information, check out the [README](#) file for the data set.

### 1.3 Setup

Run the following command to download the data set zip file.

```
[1]: ! curl https://files.grouplens.org/datasets/movielens/ml-100k.zip --output  
      ↵movielens.zip
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current	
			Dload	Upload	Total	Spent	Left	Speed
100	4808k	100	4808k	0	0	5786k	0	--:--:-- --:--:-- --:--:-- 5779k

Run the following command to unzip the data.

```
[2]: ! unzip movielens.zip
```

```
Archive: movielens.zip  
creating: ml-100k/
```

```
inflating: ml-100k/allbut.pl
inflating: ml-100k/mku.sh
inflating: ml-100k/README
inflating: ml-100k/u.data
inflating: ml-100k/u.genre
inflating: ml-100k/u.info
inflating: ml-100k/u.item
inflating: ml-100k/u.occupation
inflating: ml-100k/u.user
inflating: ml-100k/u1.base
inflating: ml-100k/u1.test
inflating: ml-100k/u2.base
inflating: ml-100k/u2.test
inflating: ml-100k/u3.base
inflating: ml-100k/u3.test
inflating: ml-100k/u4.base
inflating: ml-100k/u4.test
inflating: ml-100k/u5.base
inflating: ml-100k/u5.test
inflating: ml-100k/ua.base
inflating: ml-100k/ua.test
inflating: ml-100k/ub.base
inflating: ml-100k/ub.test
```

Run the following blocks of code to read in the data into Pandas [Data Frames](#).

```
[3]: import pandas as pd
      import numpy as np
```

```
[4]: column_names = ['userId', 'itemId', 'rating', 'timestamp']
      ratings_df = pd.read_table('ml-100k/u.data', sep='\t', names=column_names)
      userIds = ratings_df.sort_values('userId').userId.unique()
      itemIds = ratings_df.sort_values('itemId').itemId.unique()
```

```
[5]: ratings_df.head()
```

```
[5]:   userId  itemId  rating  timestamp
 0      196     242      3  881250949
 1      186     302      3  891717742
 2      22      377      1  878887116
 3      244      51      2  880606923
 4      166     346      1  886397596
```

## 1.4 Exploratory Data Analysis

Print out the answers to the following questions. You can use Pandas, PySpark SQL, or any other python functions or packages to calculate the answers.

**Question 1** How many users and how many items are in the data set?

```
[6]: num_users = len(ratings_df['userId'].unique())
num_items = len(ratings_df['itemId'].unique())

print(f"There are {num_users} unique users and {num_items} unique items in the
Movie Lens data set.")
```

There are 943 unique users and 1682 unique items in the Movie Lens data set.

**Question 2** What is the overall mean and standard deviation of all the ratings?

```
[7]: mean_rating = ratings_df['rating'].mean()
std_rating = ratings_df['rating'].std()

print(f"The overall mean rating is {mean_rating:.2f} with a standard deviation
of {std_rating:.2f}.)")
```

The overall mean rating is 3.53 with a standard deviation of 1.13.

**Question 3** What is the distribution of the ratings? (how many 1s, 2s, 3s, 4s, and 5s are there)

```
[8]: rating_counts = ratings_df['rating'].value_counts().sort_index()

print("Distribution of ratings:")
print(rating_counts)
```

Distribution of ratings:

```
1      6110
2     11370
3     27145
4     34174
5     21201
Name: rating, dtype: int64
```

**Question 4** List the top 5 items based on their mean rating. Only consider items that have been rated at least 50 times.

```
[9]: # Group the ratings data frame by item ID and calculate the mean rating and
# number of ratings for each item
item_stats = ratings_df.groupby('itemId').agg({'rating': [np.mean, np.size]})

# Filter out items with less than 50 ratings
item_stats = item_stats[item_stats['rating']['size'] >= 50]

# Sort the resulting data frame by mean rating in descending order
item_stats = item_stats.sort_values([('rating', 'mean')], ascending=False)
```

```
# Print out the top 5 items
print("Top 5 items by mean rating (with at least 50 ratings):")
print(item_stats['rating']['mean'].head())
```

Top 5 items by mean rating (with at least 50 ratings):  
 itemId  
 408 4.491071  
 318 4.466443  
 169 4.466102  
 483 4.456790  
 114 4.447761  
 Name: mean, dtype: float64

**Question 5** List the top 5 users based on their total number of items they have rated.

```
[10]: # Group the ratings data frame by user ID and count the number of items each user has rated
user_ratings_count = ratings_df.groupby('userId').size()

# Sort the resulting data frame by number of ratings in descending order and print out the top 5 users
print("Top 5 users by number of items rated:")
print(user_ratings_count.sort_values(ascending=False).head())
```

Top 5 users by number of items rated:  
 userId  
 405 737  
 655 685  
 13 636  
 450 540  
 276 518  
 dtype: int64

## 1.5 Recommendations

The following questions are all going to calculate components of this equation to predict the ratings of movies of the user with UserId=1. Then we will print out the top 5 highest predicted movies which that user does not already have a rating for.

$$pred(user_1, i) = \bar{r}_1 + \frac{\sum_{u \in neighbors} sim(user_1, user_u) \cdot (r_{ui} - \bar{r}_u)}{\sum_{u \in neighbors} sim(user_1, user_u)} \quad \forall i \in Items$$

First, run the following code to transform the raw data into a user-item matrix named `user_item`.

```
[11]: # Create user item matrix
user_item = pd.pivot_table(ratings_df, index='userId', values='rating',
                           columns='itemId', fill_value=np.nan)
```

```
[12]: # Set the userId to find recommendations for
userId = 1
```

**Question 6** Find the average rating of every user. Save the results in a Pandas Series named `user_averages`. The index should be the `userId`, and the values should be that user's average rating.

This is calculating the  $\bar{r}_1$  and  $\bar{r}_u$  values from the equation above.

The DataFrame `groupby` method is the easiest way to do this.

- Groupby the 'userId' column of `ratings_df`.
- Take the `.mean()` of the 'rating' column.

```
[13]: # Group the ratings data frame by user ID and calculate the mean rating for
      ↪each user
user_averages = ratings_df.groupby('userId')['rating'].mean()

print("Average raitngs for each user:")
print(user_averages)
```

```
Average ratings for each user:
userId
1      3.610294
2      3.709677
3      2.796296
4      4.333333
5      2.874286
...
939    4.265306
940    3.457944
941    4.045455
942    4.265823
943    3.410714
Name: rating, Length: 943, dtype: float64
```

**Question 7** Subtract each user's average rating from all their other ratings. Then fill all the missing values with 0. Save the result in a DataFrame named `centered_by_user`.

This is calculating the  $(r_{ui} - \bar{r}_u)$  part of the equation above.

The DataFrame `subtract` and `fillna` methods are the easiest way to do this.

- Run the `subtract` method on the `user_item` DataFrame created above
- Subtract the `user_averages` Series created in the previous question
  - *NOTE: use the argument `axis='index'` in the `subtract` method to subtract by row instead of by column*
- Then use the `fillna()` method to replace the missing values with 0's.

```
[14]: # Subtract each user's average rating from all their other ratings
centered_by_user = user_item.subtract(user_averages, axis='index')

# Fill all the missing values with 0
centered_by_user = centered_by_user.fillna(0)

print("User-item matrix with ratings centered by user:")
print(centered_by_user)
```

User-item matrix with ratings centered by user:

itemId	1	2	3	4	5	6	7	\			
userId											
1	1.389706	-0.610294	0.389706	-0.610294	-0.610294	1.389706	0.389706				
2	0.290323	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000				
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000				
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000				
5	1.125714	0.125714	0.000000	0.000000	0.000000	0.000000	0.000000				
...	...	...	...	...	...	...	...				
939	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000			
940	0.000000	0.000000	0.000000	-1.457944	0.000000	0.000000	0.000000	0.542056			
941	0.954545	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.045455			
942	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000				
943	0.000000	1.589286	0.000000	0.000000	0.000000	0.000000	0.000000				
itemId	8	9	10	...	1673	1674	1675	1676	1677	1678	\
userId				...							
1	-2.610294	1.389706	-0.610294	...	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.000000	0.000000	-1.709677	...	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
5	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	
939	0.000000	0.734694	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
940	1.542056	-0.457944	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
941	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
942	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
943	0.000000	-0.410714	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0	
itemId	1679	1680	1681	1682							
userId											
1	0.0	0.0	0.0	0.0							
2	0.0	0.0	0.0	0.0							
3	0.0	0.0	0.0	0.0							
4	0.0	0.0	0.0	0.0							
5	0.0	0.0	0.0	0.0							
...	...	...	...	...							
939	0.0	0.0	0.0	0.0							

```
940      0.0    0.0    0.0    0.0  
941      0.0    0.0    0.0    0.0  
942      0.0    0.0    0.0    0.0  
943      0.0    0.0    0.0    0.0
```

```
[943 rows x 1682 columns]
```

**Question 8** Calculate the cosine similarity between every pair of users. Save the result into an array named `user_similarity_matrix`.

This is calculating the  $\text{sim}(\text{user}_1, \text{user}_u)$  parts of the equation above.

The easiest way to do this is with the `pairwise.cosine_similarity()` function from Scikit-Learn on the `centered_by_user` DataFrame created in the previous question.

```
[15]: from sklearn.metrics import pairwise
```

```
[16]: # Calculate the cosine similarity between every pair of users  
user_similarity_matrix = pairwise.cosine_similarity(centered_by_user)  
  
print("User similarity matrix:")  
print(user_similarity_matrix)
```

```
User similarity matrix:  
[[ 1.0000000e+00  4.34108342e-02  1.10508910e-02 ...  2.87902189e-02  
  -3.12704963e-02  3.21234686e-02]  
 [ 4.34108342e-02  1.00000000e+00  1.36577422e-02 ... -1.73443333e-02  
   1.20678821e-02  3.91732571e-02]  
 [ 1.10508910e-02  1.36577422e-02  1.00000000e+00 ...  3.44144054e-02  
  -9.18699063e-03  1.48879883e-03]  
 ...  
 [ 2.87902189e-02 -1.73443333e-02  3.44144054e-02 ...  1.00000000e+00  
  -1.90550456e-02  6.66444773e-04]  
 [-3.12704963e-02  1.20678821e-02 -9.18699063e-03 ... -1.90550456e-02  
   1.00000000e+00  4.03538399e-02]  
 [ 3.21234686e-02  3.91732571e-02  1.48879883e-03 ...  6.66444773e-04  
  4.03538399e-02  1.00000000e+00]]
```

**Question 9** Run the following blocks of code to transform the user similarity matrix into a long format DataFrame.

```
[17]: # Unstack the similarity matrix into a long format  
user_sim_df = pd.DataFrame(user_similarity_matrix).unstack().reset_index()  
user_sim_df.columns = ['user1', 'user2', 'similarity']
```

```
[18]: # Correct for matrix being indexed at 0, and userId's starting at 1  
user_sim_df['user1'] = user_sim_df['user1'] + 1  
user_sim_df['user2'] = user_sim_df['user2'] + 1
```

```
[19]: # Remove rows comparing a user to themselves
user_sim_df = user_sim_df[user_sim_df.user1 != user_sim_df.user2]
```

```
[20]: # View the results
user_sim_df.head()
```

```
[20]:   user1  user2  similarity
1         1      2    0.043411
2         1      3    0.011051
3         1      4    0.059303
4         1      5    0.134514
5         1      6    0.103373
```

**Question 10** From the `user_sim_df` created in the previous question, find the top 20 users who are most closely related to `userId=1` based on cosine similarity. Save the results in a DataFrame named `top20`.

This is filtering the  $\text{sim}(\text{user}_1, \text{user}_u)$  for only the relevant neighbors in the equation above.

*Hints:* \* Filter `user_sim_df` by where the '`user1`' column = 1 \* Use the `.sort_values()` method to sort the remaining values by the `similarity` column with `ascending=False` \* Take the top 20 rows with the `.head()` method

```
[22]: # Filter the user similarity data frame by where the 'user1' column = 1
top_similar_users = user_sim_df[user_sim_df['user1'] == 1]

# Sort the resulting data frame by similarity in descending order
top_similar_users = top_similar_users.sort_values('similarity', ascending=False)

# Take the top 20 rows
top20 = top_similar_users.head(20)
```

```
[23]: print("Top 20 users most closely related to userId=1:")
print(top20)
```

Top 20 users most closely related to `userId=1`:

```
   user1  user2  similarity
772      1    773    0.204792
867      1    868    0.202321
591      1    592    0.196592
879      1    880    0.195801
428      1    429    0.190661
275      1    276    0.187476
915      1    916    0.186358
221      1    222    0.182415
456      1    457    0.182253
7        1      8    0.180891
659      1    660    0.179206
```

343	1	344	0.176834
756	1	757	0.176318
362	1	363	0.175709
478	1	479	0.174467
302	1	303	0.173798
565	1	566	0.173653
549	1	550	0.173542
12	1	13	0.171911
660	1	661	0.171152

**Question 11** Filter the `centered_by_user` DataFrame to only include the ratings from the top 20 neighbors. Save the resulting DataFrame into a variable named `regularized_ratings`.

This is filtering the  $(r_{ui} - \bar{r}_u)$  for only the relevant neighbors in the equation above.

Use the `.loc[]` attribute of the DataFrame. You can filter out all the rows you want by passing an array or list of the userId's that you want in the resulting DataFrame.

```
[24]: # Extract the userIds of the top 20 most similar users to userId=1
top20_userIds = top20['user2'].values

# Filter the centered_by_user DataFrame to only include the ratings from the top 20 neighbors
regularized_ratings = centered_by_user.loc[top20_userIds]

print("Centered by user ratings of the top 20 neighbors:")
print(regularized_ratings)
```

Centered by user ratings of the top 20 neighbors:

itemId	1	2	3	4	5	6	7	\
userId								
773	-0.279503	-0.279503	0.000000	0.000000	0.000000	-0.279503	-1.279503	
868	1.048077	-0.951923	0.000000	0.000000	0.000000	0.000000	2.048077	
592	0.186111	0.000000	0.186111	0.186111	0.000000	0.000000	1.186111	
880	0.573370	-0.426630	-2.426630	0.573370	-0.426630	0.000000	-0.426630	
429	-0.393720	-0.393720	-1.393720	0.606280	0.000000	0.000000	-1.393720	
276	1.534749	0.534749	-0.465251	0.534749	-0.465251	0.000000	1.534749	
916	0.634069	-0.365931	-0.365931	0.634069	-0.365931	0.000000	0.634069	
222	0.950904	-0.049096	0.000000	-0.049096	0.000000	0.000000	1.950904	
457	-0.025271	0.000000	0.000000	-0.025271	0.000000	0.000000	-0.025271	
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-0.796610	
660	0.415179	-0.584821	-1.584821	0.000000	0.000000	0.000000	0.415179	
344	-0.668421	0.000000	0.000000	0.331579	-0.668421	0.000000	0.331579	
757	0.632530	-0.367470	0.000000	1.632530	0.000000	0.000000	0.632530	
363	-1.054662	0.945338	0.000000	1.945338	-2.054662	0.000000	-0.054662	
479	1.579208	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
303	1.634298	-0.365702	-0.365702	0.634298	-1.365702	0.000000	0.634298	
566	0.000000	1.557047	0.000000	0.000000	0.000000	0.000000	0.557047	
550	-0.675000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

13	-0.097484	-0.097484	0.000000	1.902516	-2.097484	0.000000	-1.097484
661	1.082645	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

itemId	8	9	10	...	1673	1674	1675	1676	1677	1678	\
userId				...							
773	0.000000	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
868	0.000000	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
592	1.186111	1.186111	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
880	0.573370	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
429	-0.393720	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
276	0.534749	1.534749	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
916	0.000000	1.634069	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
222	-2.049096	1.950904	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
457	0.974729	0.974729	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.000000	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
660	-0.584821	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
344	1.331579	1.331579	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
757	0.000000	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
363	1.945338	-0.054662	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
479	1.579208	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
303	1.634298	1.634298	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
566	0.557047	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
550	0.000000	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	0.902516	-0.097484	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
661	1.082645	0.000000	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0

itemId	1679	1680	1681	1682
userId				
773	0.0	0.0	0.0	0.000000
868	0.0	0.0	0.0	0.000000
592	0.0	0.0	0.0	0.000000
880	0.0	0.0	0.0	0.000000
429	0.0	0.0	0.0	0.000000
276	0.0	0.0	0.0	0.000000
916	0.0	0.0	0.0	-0.365931
222	0.0	0.0	0.0	0.000000
457	0.0	0.0	0.0	0.000000
8	0.0	0.0	0.0	0.000000
660	0.0	0.0	0.0	0.000000
344	0.0	0.0	0.0	0.000000
757	0.0	0.0	0.0	0.000000
363	0.0	0.0	0.0	0.000000
479	0.0	0.0	0.0	0.000000
303	0.0	0.0	0.0	0.000000
566	0.0	0.0	0.0	0.000000
550	0.0	0.0	0.0	0.000000
13	0.0	0.0	0.0	0.000000
661	0.0	0.0	0.0	0.000000

```
[20 rows x 1682 columns]
```

**Question 12** Get the weighted average ratings for `userId=1` by multiplying each users movie ratings by their similarity to user 1, then taking the sum of all the scaled ratings for each movie.

The result should be a Pandas Series with length equal to 1682, one value for each movie. Save the resulting Series into a variable named `numerator`.

This is calculating a vector as the numerator for the above equation:

$$\sum_{u \in \text{neighbors}} \text{sim}(\text{user}_1, \text{user}_u) \cdot (r_{ui} - \bar{r}_u).$$

Use the `np.matmul()` function to matrix multiply the `top20` similarity values by the `regularized_ratings` DataFrame.

```
[25]: # Calculate the numerator of the weighted average equation
numerator = np.matmul(top20['similarity'].values, regularized_ratings)

print("Weighted average numerator:")
print(numerator)
```

Weighted average numerator:

```
itemId
1      1.296710
2     -0.198754
3     -1.207261
4      1.594897
5     -1.316110
...
1678    0.000000
1679    0.000000
1680    0.000000
1681    0.000000
1682   -0.068194
Length: 1682, dtype: float64
```

**Question 13** Calculate the sum of all the similarity values for the `top20` users. Save the result into a variable named `denominator`.

This is calculating the denominator of the equation above.

$$\sum_{u \in \text{neighbors}} \text{sim}(\text{user}_1, \text{user}_u)$$

```
[27]: # Calculate the sum of all the similarity values for the top20 users
denominator = top20['similarity'].abs().sum()
```

**Question 14** Calculate the projected ratings for every movie for `user1`. Save the results into a Pandas Series named `ratings`. The index should be the `itemId` of each movie, and the values should be user1's projected rating of that movie.

First, divide the numerator and denominator. This gives the projected difference from user1's average rating.

Then, add in user1's average rating from the `user_averages` Series calculated in Question 6.

```
[28]: # Calculate the projected ratings for every movie for user1
projected_diff = numerator / denominator
ratings = user_averages[1] + projected_diff
ratings.index.name = 'itemId'
ratings.name = 'predicted_rating'
```

**Question 15** Find the movies with the top 5 highest predicted ratings that `userId=1 has not yet rated`.

```
[29]: # Find the movies with the top 5 highest predicted ratings that userId=1 has not yet rated
rated_items = user_item.loc[1].dropna().index
unrated_items = user_item.columns.difference(rated_items)
top5_unrated_movies = ratings[ratings.index.isin(unrated_items)].nlargest(5)
```

```
[30]: print("Top 5 highest predicted ratings for user 1:")
print(top5_unrated_movies)
```

```
Top 5 highest predicted ratings for user 1:
itemId
318    4.385344
651    4.291911
408    4.276558
483    4.180320
357    4.143762
Name: predicted_rating, dtype: float64
```

# Notebook

August 2, 2023

## 1 Lab 3

```
[111]: FIRST_NAME = "Leng"  
LAST_NAME = "Her"  
STUDENT_ID = "5445877"
```

### 1.1 Introduction

In this lab, we will rework some of the machine learning tasks from Lab 2 to use k-fold cross validation and the Scikit Learn Pipelines framework. Then we will log the cross validation results to ML Flow.

### 1.2 The Data Set

#### Data Description

This is a simulated data set of students performance in the INET 4062 class. *None of these are actual students.*

#### Data Dictionary

Column Name	Type	Description
studentId	int	Unique Id of student
gpa	float	Current cumulative GPA
labHours	float	Number of hours spent per week on labs
studyHours	float	Number of hours spent studying for each exam
took4061	int	Binary if student took INET 4061 (0=No, 1=Yes)
pythonExp	int	A High, Medium, or Low rating from student on previous python experience (0=Low, 1=Medium, 2=High)
statsRating	int	A 0-5 rating of ability on statistics
height	float	Height of student in inches
eyeColor	str	Eye color of student
followers	int	Number of followers on all social media accounts
grade	float	Percentage grade in INET 4062 out of 100

Column Name	Type	Description
letterGrade	str	Letter grade derived from the percentage

## Data Sample

studentId	gpa	labHours	studyHours	took4061	pythonExp	statsRating	height	eyeColor	followers	grade	letterGrade
0	3.6206	3.0089	4.36066	1	2	3	69.4933	brown	632	92.31	A-
1	3.1939	12.524	4.88687	0	2	2	67.3275	blue	44	85.59	B
2	3.1945	0.903686	2.0478	1	1	5	69.3401	green	181	88.39	B+
3	3.2779	4.88015	0.822806	1	2	4	67.8951	blue	347	90.91	A-
4	2.5	1.47281	7.51036	0	2	4	67.708	brown	1070	84.14	B
5	2.5616	2.53166	5.50934	0	1	5	69.6897	hazel	18	84.37	B
6	3.1558	2.60646	1.56167	0	1	5	69.045	hazel	7007	84.2	B
7	3.7340	2.41052	2.99812	1	2	5	66.5794	brown	5599	93.6	A
8	2.9845	2.68131	1.73898	1	2	4	69.1432	hazel	1206	88.73	B+
9	3.8450	0.43147	4.9316	1	1	1	68.034	blue	40097	92.12	A-

## Simulate Data

```
[112]: import numpy as np
import pandas as pd

n = 10000 # number of records to simulate
np.random.seed(40) # set the seed of the random number generator

# Current GPA
gpa = 0.4 * np.random.randn(n) + 3.25
gpa = np.clip(gpa, 2.5, 4.0)

# Average hours per week on Labs
labHours = 5.5/np.exp(2*np.random.rand(n))

# Number of hours studying for exam
studyHours = np.power(2*np.random.rand(n) + 0.75, 2)

# Junior or Senior
isSenior = np.random.binomial(size=n, n=1, p=0.67)

# Took 4061
took4061 = np.random.binomial(size=n, n=1, p=0.75)

# Previous Python Experience
pythonExp = np.random.binomial(size=n, n=2, p=0.70)

# Ability in statistics
```

```

statsRating = np.random.binomial(size=n, n=5, p=0.75)

# Height
height = 4 * np.random.rand(n) + 66.5

# Eye Color
eyeColor = np.random.choice(["blue", "green", "brown", "hazel"], n)

# Social media followers
followers = (10 ** (1+5*np.random.beta(3, 7, size=n))).round()

# simulate grades
grade = 72 + (((gpa**2)/3 + np.sqrt(statsRating+1)) *
               np.sqrt(labHours/3 + studyHours/6 + pythonExp + 3*took4061)) + \
        (1+3*np.random.rand())

# Compile columns into a DataFrame
students_df = pd.DataFrame({
    'gpa' : gpa,
    'labHours' : labHours,
    'studyHours' : studyHours,
    'took4061' : took4061,
    'pythonExp' : pythonExp,
    'statsRating' : statsRating,
    'height' : height,
    'eyeColor' : eyeColor,
    'followers' : followers,
    'grade' : grade.round(2)
})

# Define a function to calculate the letter grade based
# on the percentage in the class
def getLetterGrade(x):
    if x < 76.67:
        return("C")
    elif x < 80:
        return("C+")
    elif x < 83.33:
        return("B-")
    elif x < 86.67:
        return("B")
    elif x < 90:
        return("B+")
    elif x < 93.33:
        return("A-")
    else:
        return("A")

```

```

# Add the letter grade column to the DataFrame
students_df['letterGrade'] = students_df['grade'].apply(lambda row: ↴
    ↪getLetterGrade(row))

# Rename the index of the DataFrame to be `studentId`  

# because that index uniquely identifies 1 student
students_df.index.rename("studentId", inplace=True)
students_df.reset_index(drop=False, inplace=True)

```

[113]: students\_df

	studentId	gpa	labHours	studyHours	took4061	pythonExp	\
0	0	3.006981	4.312076	6.154981	1	1	
1	1	3.199545	4.468924	6.607994	1	1	
2	2	2.976157	2.074952	1.599081	1	2	
3	3	3.621486	0.994602	7.210698	1	2	
4	4	2.512240	0.819105	2.258261	1	1	
...	...	...	...	...	...	...	
9995	9995	3.265625	0.915808	7.422218	1	2	
9996	9996	3.197060	2.061146	4.550661	1	2	
9997	9997	2.513514	2.893784	3.403938	1	2	
9998	9998	3.115089	3.472824	6.973135	1	2	
9999	9999	2.500000	2.506367	7.262839	0	2	
	statsRating	height	eyeColor	followers	grade	letterGrade	
0	4	68.968833	hazel	150.0	88.10	B+	
1	3	68.210087	hazel	499.0	88.65	B+	
2	5	69.524292	green	117.0	87.94	B+	
3	5	70.242933	brown	1049.0	92.19	A-	
4	1	66.546452	brown	72.0	82.34	B-	
...	...	...	...	...	...	...	
9995	5	68.925192	hazel	149.0	90.12	A-	
9996	4	67.969562	green	863.0	89.08	B+	
9997	4	67.587846	green	334.0	85.85	B	
9998	3	67.842580	green	266.0	88.92	B+	
9999	4	69.570120	brown	1655.0	83.45	B	

[10000 rows x 12 columns]

### 1.3 Question 1

Define a Scikit-Learn [ColumnTransformer](#) do the data preprocessing steps before building machine learning models. Save this to a variable named `preprocessor`.

It must do the following 2 steps:

- \* Scale 3 numeric features (`gpa`, `labHours`, `studyHours`) to have values between 0-1 with [MinMaxScaler](#).
- \* One hot encode 3 categorical features (`took4061`, `pythonExp`, `statsRating`) using [OneHotEncoder](#).

### 1.3.1 Resources

- Scikit Learn User Guides: [ColumnTransformer](#), [MinMaxScaler](#), [OneHotEncoder](#)
- Class examples: [ML Pipelines Notebook](#), [ML Flow Example Notebook](#)
- Other examples: [Analytics Vidhya](#), [UBC SPSC 330 Notes](#), [Medium - MLearning.ai](#)

### 1.3.2 Answer

```
[114]: from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

[115]: encoder = OneHotEncoder()

# define numeric and categorical feature columns
numeric_cols = ['gpa', 'labHours', 'studyHours']
categorical_cols = ['took4061', 'pythonExp', 'statsRating']

# create the transformer
preprocessor = ColumnTransformer(transformers=[
    ('num', MinMaxScaler(), numeric_cols),
    ('cat', encoder, categorical_cols)])

[116]: preprocessor
```

```
[116]: ColumnTransformer(transformers=[('num', MinMaxScaler(),
                                         ['gpa', 'labHours', 'studyHours']),
                                         ('cat', OneHotEncoder(),
                                         ['took4061', 'pythonExp', 'statsRating'])])
```

## 1.4 Question 2

Define Scikit-Learn Pipeline objects for 3 Regression alforithms:  
\* [Lasso](#) - name this variable `lm_pipeline` for “*linear model*”  
\* [KNeighborsRegressor](#) - name this variable `knn_pipeline` for “*k nearest neighbor*”  
\* [RandomForestRegressor](#) - name this variable `rf_pipeline` for “*random forest*”

Each pipeline should have 2 steps: 1. The `preprocessor` from Question 1 2. A regression algorithm mentioned above

### 1.4.1 Resources

- Scikit Learn User Guides: [Lasso](#), [KNN Regression](#), [Random Forest](#)
- Class examples: [ML Pipelines Notebook](#), [ML Flow Example Notebook](#)
- Other exmaples: [Machine Learning Mastery](#), [UBC SPSC 330 Notes](#), [Medium - MLearning.ai](#) (same article as Q1)

### 1.4.2 Answer

```
[117]: from sklearn.pipeline import Pipeline  
  
from sklearn.linear_model import Lasso  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.ensemble import RandomForestRegressor  
  
[118]: # Define the pipelines  
lm_pipeline = Pipeline(steps=[('preprocessor', preprocessor),  
                           ('regressor', Lasso())])  
  
[119]: knn_pipeline = Pipeline(steps=[('preprocessor', preprocessor),  
                           ('regressor', KNeighborsRegressor())])  
  
[120]: rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),  
                           ('regressor', RandomForestRegressor())])
```

## 1.5 Question 3

Define Python dictionaries to hold the hyperparameters for each ML algroithm. These dictionaries will be used by Scikit-Learn's `ParameterGrid` to do a full grid search of all combinations of hyperparameters.

The names of the dictionaries and the hyperparameters to search through for each algorithm should be as follows:

### Lasso

```
lm_params * alpha : [0.05, 0.10, 0.25, 0.50, 0.75, 1]
```

### K nearest neighbors

```
knn_params * n_neighbors : [5, 10, 20] * weights : ["uniform", "distance"]
```

### Random Forest

```
rf_params * n_estimators : [25, 100] * max_depth : [3, 10]
```

#### 1.5.1 Resources

Note: Because the we are running these algroithms from within a Scikit-Learn pipeline, we need to *prefix the hyperparameter names with the name of the pipeline step and 2 underscores* that runs the algorithm.

For example, if were going to run a grid search over the following parameters for a Support Vector Regression (SVR):

```
svr_params = {  
    'C' : [1, 2, 3],    'kernel' : ["linear", "rbf"]  
}
```

And we were running the `SVR` in the following pipeline:

```
svr_pipeline = Pipeline(
```

```

        steps=[  

            ("preprocessing", preprocessor),  

            ("the_SVR_step", SVR())  

        ]  

    )

```

Then we would define the parameter grid dictionary as follows:

```

svr_params = {  

    'the_SVR_step__C' : [1, 2, 3],  

    'the_SVR_step__kernel' : ["linear", "rbf"]  

}

```

Class Examples: \* [ML Pipelines Notebook](#) \* [ML Flow Example Notebook](#)

### 1.5.2 Answer

```
[121]: lm_params = {  

    'regressor__alpha': [0.05, 0.10, 0.25, 0.50, 0.75, 1]  

}
```

```
[122]: knn_params = {  

    'regressor__n_neighbors': [5, 10, 20],  

    'regressor__weights': ["uniform", "distance"]  

}
```

```
[123]: rf_params = {  

    'regressor__n_estimators': [25, 100],  

    'regressor__max_depth': [3, 10]  

}
```

## 1.6 Question 4

### Part a

Split the `students_df` data frame into 2 parts, `X` and `y`. \* `X` - contains only the feature columns of the data frame that are used in `preprocessor`. \* `y` contains only the label columns “`grade`”

### Part b

Then do a train/test split on the `X` and `y` datasets where 75% of the rows are in the train set, and 25% are in the test set. Set the random seed to 4062 for the train/test split.

*Note: Parts a & b are the same as Question 1 from Lab 2*

### Part C

Define 3 `GridSearchCV` objects, 1 for each algorithm. \* The `estimator` for each `GridSearchCV` is the corresponding `**_pipeline` object containing the ML algorithm from Question 2. \* The `param_grid` for each `GridSearchCV` is the corresponding `**_params` dictionary from Question 3. \* The `scoring` parameter for each `GridSearchCV` should be set to “`r2`”

### Part D

Fit the `GridSearchCV` objects to the training data (`X_train`, `y_train`) using the `.fit()` method.

### 1.6.1 Resources

- Scikit Learn User Guides: [train\\_test\\_split](#), [GridSearchCV](#)
- Class examples: [ML Pipelines Notebook](#), [ML Flow Example Notebook](#)
- Other examples: [Python Simplified](#) (gold star means highly recommended example)

### 1.6.2 Answer

```
[124]: from sklearn.model_selection import GridSearchCV, train_test_split

[125]: #Part A
X = students_df[['gpa', 'labHours', 'studyHours', 'took4061', 'pythonExp', ↴
    'statsRating']]
y = students_df['grade']

[126]: #Part B
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ↴
    random_state=4062)
#Part C
lm_grid = GridSearchCV(lm_pipeline, lm_params, scoring='r2')
knn_grid = GridSearchCV(knn_pipeline, knn_params, scoring='r2')
rf_grid = GridSearchCV(rf_pipeline, rf_params, scoring='r2')

[127]: #Part D
lm_grid.fit(X_train, y_train)
knn_grid.fit(X_train, y_train)
rf_grid.fit(X_train, y_train)

[127]: GridSearchCV(estimator=Pipeline(steps=[('preprocessor',
                                              ColumnTransformer(transformers=[('num',
                                                                 MinMaxScaler(),
                                                                 ['gpa',
                                                                 'labHours',
                                                                 'studyHours']),
                                             ('cat',
                                              OneHotEncoder(),
                                              ['took4061',
                                               'pythonExp',
                                               'statsRating'])])),
                                              ('regressor', RandomForestRegressor()))]),
param_grid={'regressor__max_depth': [3, 10],
            'regressor__n_estimators': [25, 100]},
scoring='r2')
```

## 1.7 Question 5

Create 3 ML flow experiments called “Lasso”, “KNN”, and “Random Forest”.

For each of these experiments, log the grid search cross validation results from the associated algorithm. The full results for each set of parameters can be found in the `.cv_results_` attribute of the `GridSearchCV` objects.

Each run should:

- \* Log the hyperparameters for the model with `mlflow.log_param()`.
- \* Log the `mean_test_score` of all the folds with `mlflow.log_metric()`. Name the metric “`r2`”
- \* Set a tag for the run as “*algorithm*” using `mlflow.set_tag()`. The value be the corresponding Scikit Learn model object used: *Lasso*, *KNeighborsRegressor*, or *RandomForestRegressor*.

After logging, you should be able to click the link for the ML Flow UI below and see all the experiment results. It should look similar to this.

### 1.7.1 Resources

- ML Flow Python API Documentation: [log\\_param](#), [log\\_metric](#), [set\\_tag](#)
- Class example: [ML Flow Example Notebook](#)
- Other example: [MLflow Sklearn Example GitHub](#)

### 1.7.2 ML Flow Setup

```
[128]: # Install the mlflow python package
%pip install mlflow --quiet
```

```
[129]: import mlflow
```

```
[130]: # Run the UI as a background process
get_ipython().system_raw("mlflow ui --port 5000 &")
```

```
[131]: # Print out the URL to access the port the UI is running on
from google.colab.output import eval_js
print(eval_js("google.colab.kernel.proxyPort(5000)"))
```

<https://pl14v99h89-496ff2e9c6d22116-5000-colab.googleusercontent.com/>

Note: Couldn't get link to work even trying different ports, so I ran it on local machine to access MLflow. Ended up working easier

### 1.7.3 Answer

```
[132]: # create the three experiments
with mlflow.start_run(run_name="Lasso") as run:
    mlflow.log_param("alpha", lm_grid.best_params_['regressor__alpha'])
    mlflow.log_metric("r2", lm_grid.best_score_)
    mlflow.set_tag("algorithm", "Lasso")
```

```
[133]: with mlflow.start_run(run_name="KNN") as run:  
    mlflow.log_param("n_neighbors", knn_grid.  
    ↪best_params_['regressor__n_neighbors'])  
    mlflow.log_param("weights", knn_grid.best_params_['regressor__weights'])  
    mlflow.log_metric("r2", knn_grid.best_score_)  
    mlflow.set_tag("algorithm", "KNeighborsRegressor")
```

```
[134]: with mlflow.start_run(run_name="Random Forest") as run:  
    mlflow.log_param("n_estimators", rf_grid.  
    ↪best_params_['regressor__n_estimators'])  
    mlflow.log_param("max_depth", rf_grid.best_params_['regressor__max_depth'])  
    mlflow.log_metric("r2", rf_grid.best_score_)  
    mlflow.set_tag("algorithm", "RandomForestRegressor")
```

Heres some attributes shown with 2 runtimes of each model

# Notebook

August 2, 2023

## 1 Lab 4

```
[ ]: FIRST_NAME = "Leng"  
      LAST_NAME = "Her"  
      STUDENT_ID = "5445877"
```

### 1.1 Introduction

This lab is all about reading and writing files from AWS S3. We will build on top of the models and functions that were created in Labs 2 and 3. The machine learning part is already completed below. Each question will contribute 1 part to a system that can help track machine learning models and predictions with S3.

### 1.2 The Data Set

#### Data Description

This is a simulated data set of students performance in the INET 4062 class. *None of these are actual students.*

#### Data Dictionary

Column Name	Type	Description
studentId	int	Unique Id of student
gpa	float	Current cumulative GPA
labHours	float	Number of hours spent per week on labs
studyHours	float	Number of hours spent studying for each exam
took4061	int	Binary if student took INET 4061 (0=No, 1=Yes)
pythonExp	int	A High, Medium, or Low rating from student on previous python experience (0=Low, 1=Medium, 2=High)
statsRating	int	A 0-5 rating of ability on statistics
height	float	Height of student in inches
eyeColor	str	Eye color of student
followers	int	Number of followers on all social media accounts

Column Name	Type	Description
grade	float	Percentage grade in INET 4062 out of 100
letterGrade	str	Letter grade derived from the percentage

## Data Sample

studentId	gpa	labHours	studyHours	took4061	pythonExp	statsRatio	height	eyeColor	followers	grade	letterGrade
0	3.6206	3.0089	4.36066	1	2	3	69.4933	brown	632	92.31	A-
1	3.1939	12.524	4.88687	0	2	2	67.3275	blue	44	85.59	B
2	3.1945	0.9036	862.0478	1	1	5	69.3401	green	181	88.39	B+
3	3.2779	4.88015	0.822806	1	2	4	67.8951	blue	347	90.91	A-
4	2.5	1.47281	7.51036	0	2	4	67.708	brown	1070	84.14	B
5	2.5616	2.53166	5.50934	0	1	5	69.6897	hazel	18	84.37	B
6	3.1558	2.60646	1.56167	0	1	5	69.045	hazel	7007	84.2	B
7	3.7340	2.41052	2.99812	1	2	5	66.5794	brown	5599	93.6	A
8	2.9845	2.68131	1.73898	1	2	4	69.1432	hazel	1206	88.73	B+
9	3.8450	0.43147	4.9316	1	1	1	68.034	blue	40097	92.12	A-

## Simulate Data

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

n = 10000 # number of records to simulate
np.random.seed(40) # set the seed of the random number generator

# Current GPA
gpa = 0.4 * np.random.randn(n) + 3.25
gpa = np.clip(gpa, 2.5, 4.0)

# Average hours per week on Labs
labHours = 5.5/np.exp(2*np.random.rand(n))

# Number of hours studying for exam
studyHours = np.power(2*np.random.rand(n) + 0.75, 2)

# Junior or Senior
isSenior = np.random.binomial(size=n, n=1, p=0.67)

# Took 4061
took4061 = np.random.binomial(size=n, n=1, p=0.75)

# Previous Python Experience
pythonExp = np.random.binomial(size=n, n=2, p=0.70)
```

```

# Ability in statistics
statsRating = np.random.binomial(size=n, n=5, p=0.75)

# Height
height = 4 * np.random.rand(n) + 66.5

# Eye Color
eyeColor = np.random.choice(["blue", "green", "brown", "hazel"], n)

# Social media followers
followers = (10 ** (1+5*np.random.beta(3, 7, size=n))).round()

# simulate grades
grade = 72 + (((gpa**2)/3 + np.sqrt(statsRating+1)) *
               np.sqrt(labHours/3 + studyHours/6 + pythonExp + 3*took4061)) + \
        (1+3*np.random.rand())

# Compile columns into a DataFrame
students_df = pd.DataFrame({
    'gpa' : gpa,
    'labHours' : labHours,
    'studyHours' : studyHours,
    'took4061' : took4061,
    'pythonExp' : pythonExp,
    'statsRating' : statsRating,
    'height' : height,
    'eyeColor' : eyeColor,
    'followers' : followers,
    'grade' : grade.round(2)
})

# Define a function to calculate the letter grade based
# on the percentage in the class
def getLetterGrade(x):
    if x < 76.67:
        return("C")
    elif x < 80:
        return("C+")
    elif x < 83.33:
        return("B-")
    elif x < 86.67:
        return("B")
    elif x < 90:
        return("B+")
    elif x < 93.33:
        return("A-")

```

```

    else:
        return("A")

# Add the letter grade column to the DataFrame
students_df['letterGrade'] = students_df['grade'].apply(lambda row:_
    ↪getLetterGrade(row))

# Rename the index of the DataFrame to be `studentId`  

# because that index uniquely identifies 1 student
students_df.index.rename("studentId", inplace=True)
students_df.reset_index(drop=False, inplace=True)

```

```
/tmp/ipykernel_8297/3537645836.py:12: RuntimeWarning: divide by zero encountered  

in true_divide
    labHours = 5.5/np.exp(2*np.random.rand(n))
```

[ ]: students\_df

	studentId	gpa	labHours	studyHours	took4061	pythonExp	\
0	0	3.006981	4.312076	6.154981	1	1	
1	1	3.199545	4.468924	6.607994	1	1	
2	2	2.976157	2.074952	1.599081	1	2	
3	3	3.621486	0.994602	7.210698	1	2	
4	4	2.512240	0.819105	2.258261	1	1	
...	...	...	...	...	...	...	
9995	9995	3.265625	0.915808	7.422218	1	2	
9996	9996	3.197060	2.061146	4.550661	1	2	
9997	9997	2.513514	2.893784	3.403938	1	2	
9998	9998	3.115089	3.472824	6.973135	1	2	
9999	9999	2.500000	2.506367	7.262839	0	2	
	statsRating	height	eyeColor	followers	grade	letterGrade	
0	4	68.968833	hazel	150.0	88.10	B+	
1	3	68.210087	hazel	499.0	88.65	B+	
2	5	69.524292	green	117.0	87.94	B+	
3	5	70.242933	brown	1049.0	92.19	A-	
4	1	66.546452	brown	72.0	82.34	B-	
...	...	...	...	...	...	...	
9995	5	68.925192	hazel	149.0	90.12	A-	
9996	4	67.969562	green	863.0	89.08	B+	
9997	4	67.587846	green	334.0	85.85	B	
9998	3	67.842580	green	266.0	88.92	B+	
9999	4	69.570120	brown	1655.0	83.45	B	

[10000 rows x 12 columns]

### 1.3 Machine Learning

```
[ ]: from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor

[ ]: # Define feature variables
numeric_columns = ["gpa", "labHours", "studyHours"]
categorical_columns = ["took4061", "pythonExp", "statsRating"]

# Column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('numerical', MinMaxScaler(), numeric_columns),
        ('categorical', OneHotEncoder(), categorical_columns)
    ]
)

[ ]: # K Nearest Neighbor Pipeline
knn_pipeline_model = Pipeline(
    steps=[
        ("preprocessing", preprocessor),
        ("knn", KNeighborsRegressor(n_neighbors=10, weights='uniform'))
    ]
)

# Random Forest Pipeline
rf_pipeline_model = Pipeline(
    steps=[
        ("preprocessing", preprocessor),
        ("random_forest", RandomForestRegressor(n_estimators=100, max_depth=10))
    ]
)

[ ]: # Split features and label variable
X = students_df[numeric_columns+categorical_columns]
y = students_df["grade"]

# Train models
knn_pipeline_model = knn_pipeline_model.fit(X, y)
rf_pipeline_model = rf_pipeline_model.fit(X, y)
```

### 1.4 Question 1

- First create an S3 Service Resource object using the `boto3.resource()` function. Name the variable `s3`.

- Then use the `.create_bucket()` method of the S3 Service Resource to create a new S3 bucket.
  - Save the output of this function into a variable named `bucket`. This is a Python representation of the Bucket.
  - The bucket name should be globally unique within the AWS Region.

```
[ ]: import boto3
```

```
[ ]: s3 = boto3.resource('s3')
bucket_name = 'lab-4-leng'
bucket = s3.create_bucket(Bucket=bucket_name)
```

#### 1.4.1 Q1 Test

The following code should print out the name of the S3 bucket. You can also view the S3 buckets in the [S3 Management Console](#).

```
[ ]: # Print out the name of all the S3 buckets
print([bucket.name for bucket in s3.buckets.all()])
```

['lab-4-leng']

#### 1.5 Question 2

The above code under the heading “Machine Learning” trains two ML pipelines with different algorithms named `knn_model_pipeline` and `rf_model_pipeline`.

Save these 2 Scikit-Learn objects to pickle files in the S3 bucket created in Question 1 with the keys "models/v1/model.pkl" and "models/v2/model.pkl" respectively.

```
[ ]: import pickle
import pandas as pd
```

```
[ ]: # names for the pickled models
knn_model_file = 'models/v1/model.pkl'
rf_model_file = 'models/v2/model.pkl'

# Save KNN pipeline model
knn_model_bytes = pickle.dumps(knn_pipeline_model)
s3.Object(bucket_name, knn_model_file).put(Body=knn_model_bytes)

# Save Random Forest pipeline model
rf_model_bytes = pickle.dumps(rf_pipeline_model)
s3.Object(bucket_name, rf_model_file).put(Body=rf_model_bytes)
```

```
[ ]: {'ResponseMetadata': {'RequestId': 'H3XS367ZQ2RSG5XK',
 'HostId': 'fSm2eHeoher/TJ0n8s1rC1QANzdxI1eEMB0BUy/ZmN5hg5NZY10VXBZCC2D3CzBIjQAoAwjb+s=',
 'HTTPStatusCode': 200,
```

```
'HTTPHeaders': {'x-amz-id-2': 'fSm2eHeoher/TJOn8s1rC1QANzdxI1eEMB0BUy/ZmN5hg5NZY10VXBZCC2D3CzBIjQAoAwjb+s=',
  'x-amz-request-id': 'H3XS367ZQ2RSG5XK',
  'date': 'Tue, 21 Feb 2023 23:07:17 GMT',
  'etag': '"c43394d3fc875a25d55e0673a3c6968d"',
  'server': 'AmazonS3',
  'content-length': '0'},
  'RetryAttempts': 0},
'ETag': '"c43394d3fc875a25d55e0673a3c6968d"'}

```

### 1.5.1 Q2 Test

The following code should print out `['models/v1/model.pkl', 'models/v2/model.pkl']`.

You should also be able to view the objects in the S3 Management Console.

```
[ ]: print([obj.key for obj in bucket.objects.all()])
```

```
['models/v1/model.pkl', 'models/v2/model.pkl']
```

## 1.6 Question 3

The function below is an answer to Question 8 of Lab 2 that reads in a saved machine learning model from a pickle file to make predictions on new data.

Modify this question to do the following:

- \* Read in both machine learning pickle files from S3
- \* Get predictions from *both* models. Save both predictions in the same dictionary to return at the end of the function (example below).
- \* Create a unique identifier for this set of predictions with the **UUID** package. Use the `uuid.uuid4().hex` function.
- \* Save both the input data and the model predictions to S3 objects with the following keys:
- \* `new_students : "invocations/{uuid}/features.json"`
- \* `predictions : "invocations/{uuid}/predictions.json"`

### Function example from Lab 2 Q8

```
def predict(new_students: dict) -> dict:

    # Read in the ML model object
    with open('model.pkl', 'rb') as f:
        model = pickle.load(f)

    # Convert the new data into a dataframe
    new_students_df = pd.DataFrame(new_students)

    # Get predictions
    new_predictions = model.predict(new_students_df).tolist()

    # Convert into a dictionary
    predictions = {'predictions' : new_predictions}
```

```
    return(predictions)
```

### Example output dictionary

*Note: the predicted values do not need to match*

```
{'model1': [87.324], 'model2': [87.02621619272111]}
```

```
[ ]: import json  
import uuid
```

```
[ ]: new_students = [{  
    'took4061' : 1,  
    'pythonExp' : 1,  
    'statsRating' : 3,  
    'gpa' : 3.1,  
    'labHours' : 2.2,  
    'studyHours' : 4.7  
}]
```

```
[ ]: def predict(new_students: dict) -> dict:  
  
    # Read in the ML model objects  
    bucket_name = 'lab-4-leng'  
    model1 = pickle.loads(s3.Bucket(bucket_name).Object('models/v1/model.pkl').  
    ↪get()['Body'].read())  
    model2 = pickle.loads(s3.Bucket(bucket_name).Object('models/v2/model.pkl').  
    ↪get()['Body'].read())  
  
    # Convert the new data into a dataframe  
    new_students_df = pd.DataFrame(new_students)  
  
    # Get predictions  
    new_predictions1 = model1.predict(new_students_df).tolist()  
    new_predictions2 = model2.predict(new_students_df).tolist()  
  
    # Convert into a dictionary  
    uuid_val = uuid.uuid4().hex  
    predictions = {'model1': new_predictions1, 'model2': new_predictions2}  
  
    # Savedata and model predictions  
    s3.Bucket(bucket_name).Object(f'invocations/{uuid_val}/features.json').  
    ↪put(Body=json.dumps(new_students).encode())  
    s3.Bucket(bucket_name).Object(f'invocations/{uuid_val}/predictions.json').  
    ↪put(Body=json.dumps(predictions).encode())  
  
    return(predictions)
```

### 1.6.1 Q3 test

The following block of code should create three separate invocations saved in S3 and print out the results of the function. The 3 invocations should be visible in the S3 Management Console.

```
[ ]: predict(new_students)  
predict(new_students)  
predict(new_students)
```

```
[ ]: {'model1': [87.32400000000001], 'model2': [87.02621619272111]}
```

## 1.7 Question 4

Define a function named `list_invocations()` that returns a list of *distinct* (no duplicates) unique invocation ids (the UUIDs created by the previous step) that are in the S3 bucket.

```
[ ]: def list_invocations():  
    s3 = boto3.client('s3')  
    invocations = set()  
    prefix = 'invocations/'  
    response = s3.list_objects_v2(Bucket=bucket_name, Prefix=prefix)  
    while True:  
        if 'Contents' not in response:  
            break  
        for obj in response['Contents']:  
            key = obj['Key']  
            uuid = key.split('/')[-2]  
            invocations.add(uuid)  
        if not response['IsTruncated']:  
            break  
        response = s3.list_objects_v2(Bucket=bucket_name, Prefix=prefix,  
        ↪ContinuationToken=response['NextContinuationToken'])  
    return list(invocations)
```

### 1.7.1 Q4 Test

The following code should print out a list of 3 UUID's.

```
[ ]: invocations = list_invocations()  
invocations
```

```
[ ]: ['3538d8e3bbfb41558efda1cf099166d4',  
      '2acca5f72cce4cfcadc53cfa7a8a58b5',  
      '528d2c59064441529888f742e14f7f28']
```

## 1.8 Question 5

Define a function named `get_invocation()` that does the following:

- \* Takes in 1 argument named `invocation_id`, which would represent the UUIDs created by the function in Question 3.
- \* Gets the `features.json` and `predictions.json` files saved for that invocation from S3.
- \* Returns the features and predictions in a dictionary like in the example below.

— **Example return dictionary**

```
{  
    'features': '[{"took4061": 1, "pythonExp": 1, "statsRating": 3, "gpa": 3.1, "labHours": 2.2, "studyHours": 4.7}]',  
    'predictions': '{"model1": [87.324], "model2": [87.02621619272111]}'  
}
```

```
[ ]: def get_invocation(invocation_id: str) -> dict:  
    features_key = f"invocations/{invocation_id}/features.json"  
    predictions_key = f"invocations/{invocation_id}/predictions.json"  
  
    # Read the features  
    obj = s3.Object(bucket_name, features_key)  
    features = obj.get()['Body'].read().decode('utf-8')  
  
    # Read the predictions  
    obj = s3.Object(bucket_name, predictions_key)  
    predictions = obj.get()['Body'].read().decode('utf-8')  
  
    return {'features': features, 'predictions': predictions}
```

### 1.8.1 Q5 Test

The following code should print out the dictionary including the features and predictions similar to the example above.

```
[ ]: invocation_id = invocations[0]  
get_invocation(invocation_id)  
  
[ ]: {'features': '[{"took4061": 1, "pythonExp": 1, "statsRating": 3, "gpa": 3.1, "labHours": 2.2, "studyHours": 4.7}]',  
      'predictions': '{"model1": [87.32400000000001], "model2": [87.02621619272111]}'}
```

## 1.9 Question 6

Define a function named `delete_invocation()` that does the following:

- \* Takes in 1 argument named `invocation_id`, which would represent the UUIDs created by the function in Question 3.
- \* Uses the `get_invocation()` from Question 5 to get the features and predictions for the `invocation_id`
- \* Uploads the dictionary returned by `get_invocation()` to S3 with the object key as “`trash/{invocation_id}/invocation.json`”
- \* Deletes the `features.json` and `predictions.json` objects from S3 for the `invocation_id`.
- \* At the end of the function print “Successfully deleted” and return `None`

```
[ ]: def delete_invocation(invocation_id):
    # Get features and predictions
    invocation_data = get_invocation(invocation_id)
    features = invocation_data['features']
    predictions = invocation_data['predictions']

    # Send invocation to trash
    s3.Object(bucket_name, f"trash/{invocation_id}/invocation.json") .
    ↪put(Body=json.dumps(invocation_data))

    # Delete features and predictions
    s3.Object(bucket_name, f"invocations/{invocation_id}/features.json") .
    ↪delete()
    s3.Object(bucket_name, f"invocations/{invocation_id}/predictions.json") .
    ↪delete()

    print("Successfully deleted")
    return None
```

### 1.9.1 Q6 Test

The following code should print out “*Successfully deleted*” and show the S3 object that has been “moved into the trash”.

```
[ ]: delete_invocation(invocation_id)
[obj.key for obj in bucket.objects.filter(Prefix="trash")]
```

Successfully deleted

```
[ ]: ['trash/3538d8e3bbfb41558efda1cf099166d4/invocation.json']
```

[ ]:

[ ]:

# Notebook

August 2, 2023

## 1 Lab 9

```
[ ]: FIRST_NAME = "Leng"  
      LAST_NAME = "Her"  
      STUDENT_ID = "5445877"
```

### 1.1 Introduction

This lab will be an introduction to using the 2 most popular deep learning frameworks: Pytorch and Tensorflow. We will use the same MNIST data set from the previous lab, except this time we will build convolutional neural networks.

### 1.2 The Data Set

#### Data Description

The MNIST data set is a standard data set for machine learning and computer vision. NIST stands for the National Institute for Standards and Technology, which created a data set of hand written digits from high school students and US Census Bureau. MNIST is the Modified version of that data set that standardizes all the images to 28x28 pixel greyscale images.

This data set is commonly used as an introduction to neural networks, deep learning, computer vision, and optical character recognition (OCR).

#### Data Dictionary

- Each row represents 1 image
- Each column is a specific pixel location
- The values of the data frame are how dark/light each pixel in the image is (0-255)

Column Name	Type	Description
0	int	Digit of the image
1-784	float	Darkness of the pixel

#### Data Sample

## 1.3 Tensorflow & Keras

The first 5 questions will be related to writing code that trains and evaluates a convolutional neural network with Keras and Tensorflow. Use this Keras MNIST [tutorial](#) as a reference.

```
[1]: import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

### 1.3.1 Question 1

Load in the MNIST data into Numpy arrays named `x_train`, `y_train`, `x_test`, and `y_test` using the `keras.datasets` module.

Resources \* TF Keras datasets [documentation](#) \* Keras MNIST [tutorial](#)

```
[2]: # Load the MNIST data
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize the pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>  
11490434/11490434 [=====] - 0s 0us/step

```
[3]: print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("First y_train value: ", y_train[0])
```

```
x_train shape: (60000, 28, 28)
y_train shape: (60000,)
First y_train value: 5
```

### 1.3.2 Question 2

Preprocess the Numpy arrays containing the MNIST data to be inputs into a Keras neural network with the following 3 steps:

- i. Scale all the values of the pixels (`x_train`, `x_test`) to be between 0 and 1 by dividing the values by 255.
- ii. Add a dimension to the `x_train` and `x_test` arrays using the `np.expand_dims()` function. So instead of `x_train` having a shape of (60000, 28, 28), change it to have the shape equal to (60000, 28, 28, 1).
- iii. Use the `keras.utils.to_categorical()` function to create a one hot encoding of the label variable (similar to `LabelBinarizer` in Scikit Learn)

```
[23]: from keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale pixel values to be between 0 and 1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Add a channel dimension to the images
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)

# One-hot encode the labels
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
[24]: print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("First y_train value: ", y_train[0])
```

```
x_train shape: (60000, 28, 28, 1)
y_train shape: (60000, 10)
First y_train value: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

### 1.3.3 Question 3

Define the convolutional neural network model using the `keras.Sequential` object. Use the same network architecture as in the Keras MNIST tutorial. Save the result into a variable named `tf_model`.

**Network layers** \* Input (shape=28,28,1) \* Conv2D (filters=32, kernel\_size=3,3, activation=relu)  
 \* MaxPooling2D (pool\_size=2,2) \* Conv2D (filters=64, kernel\_size=3,3, activation=relu) \*  
 MaxPooling2D (pool\_size=2,2) \* Flatten \* Dropout (rate=0.5) \* Dense (units=10, activation=softmax)

```
[25]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, ↴
    Dense

# Define the model architecture
tf_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
```

```
        Dense(10, activation='softmax')
    ])
```

```
[26]: tf_model.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_4 (MaxPooling 2D)	(None, 13, 13, 32)	0
conv2d_5 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dropout_2 (Dropout)	(None, 1600)	0
dense_2 (Dense)	(None, 10)	16010
<hr/>		
Total params: 34,826		
Trainable params: 34,826		
Non-trainable params: 0		

---

#### 1.3.4 Question 4

Compile and fit the model to the training data.

##### Compile step

Run the `.compile()` method of the `tf_model` object that you created with the following arguments:

- \* `loss = "categorical_crossentropy"`
- \* `optimizer = "adam"`
- \* `metrics = ["accuracy"]`

##### Fit step

Run the `.fit()` method of the `tf_model` with the `x` and `y` training sets. Set the following input arguments:

- \* `batch_size = 128`
- \* `epochs = 5`
- \* `validation_split = 0.1`

```
[27]: # Compile the model
tf_model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

# Fit the model to the training data
```

```
history = tf_model.fit(x_train, y_train,
                       batch_size=128,
                       epochs=5,
                       validation_split=0.1)
```

```
Epoch 1/5
422/422 [=====] - 35s 81ms/step - loss: 0.3624 -
accuracy: 0.8916 - val_loss: 0.0873 - val_accuracy: 0.9748
Epoch 2/5
422/422 [=====] - 33s 78ms/step - loss: 0.1167 -
accuracy: 0.9648 - val_loss: 0.0556 - val_accuracy: 0.9848
Epoch 3/5
422/422 [=====] - 34s 80ms/step - loss: 0.0873 -
accuracy: 0.9724 - val_loss: 0.0472 - val_accuracy: 0.9867
Epoch 4/5
422/422 [=====] - 34s 81ms/step - loss: 0.0722 -
accuracy: 0.9774 - val_loss: 0.0431 - val_accuracy: 0.9878
Epoch 5/5
422/422 [=====] - 34s 79ms/step - loss: 0.0621 -
accuracy: 0.9812 - val_loss: 0.0365 - val_accuracy: 0.9910
```

### 1.3.5 Question 5

Evaluate the accuracy of the model on the training and testing sets. Use the `.evaluate()` method of the `tf_model` object. Then print out the loss and accuracy of both the training and testing sets.

```
[28]: # Evaluate the model on the training set
train_loss, train_acc = tf_model.evaluate(x_train, y_train, verbose=0)
print('Training loss: {}, Training accuracy: {}'.format(train_loss, train_acc))
```

```
Training loss: 0.034619104117155075, Training accuracy: 0.9898166656494141
```

```
[29]: # Evaluate the model on the test set
test_loss, test_acc = tf_model.evaluate(x_test, y_test, verbose=0)
print('Test loss: {}, Test accuracy: {}'.format(test_loss, test_acc))
```

```
Test loss: 0.035138096660375595, Test accuracy: 0.9889000058174133
```

## 1.4 Pytorch & Torchvision

Questions 6-10 for this lab will be replicating the same model with Pytorch (building a CNN for the MNIST dataset). This time, the code is already written but it does not have any comments. Your job for answering the questions will be to describe how the various Pytorch functions and modules work, what they do, and why they are important for training and evaluating a CNN.

```
[31]: import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

#### 1.4.1 Question 6

First, read this article: [Datasets & DataLoaders](#).

Then write a short description of Pytorch DataLoaders and Datasets. What do they do and how are they realted?

```
[32]: training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
data/MNIST/raw/train-images-idx3-ubyte.gz
```

```
0%|           | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
data/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
0%|           | 0/28881 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw
```

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
data/MNIST/raw/t10k-images-idx3-ubyte.gz
```

```
0%|           | 0/1648877 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw
```

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
```

```
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
data/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
0%|          | 0/4542 [00:00<?, ?it/s]
```

```
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

```
[33]: train_loader = DataLoader(training_data, batch_size=128)
test_loader = DataLoader(test_data, batch_size=128)
```

## 1.4.2 Question 7

Read this article giving an introduction to Pytorch called “[What is torch.nn really?](#)”.

Then write a summary about each of the different parts of the `torch.nn` module and what they are used for.

```
[35]: class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout = nn.Dropout(0.5)
        self.fullyconnected = nn.Linear(1600, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout(x)
        x = torch.flatten(x, 1)
        x = self.fullyconnected(x)
        output = F.log_softmax(x, dim=1)
        return output
```

## 1.4.3 Question 8

For each of the lines of code below, describe what they do and why it is needed in the model training process. What would go wrong if that step was forgotten or removed?

```
output = model(data)
```

*What does it do?*

It calculates the output of the neural network given some input data.

*Why is it needed?* It is needed to make predictions on the input data, which is used to calculate the loss and update the model weights. —

```
loss = F.cross_entropy(output, target)
```

*What does it do?*

It calculates the cross-entropy loss between the output and the target.

*Why is it needed?* It is needed to measure how well the model is performing on the task. The goal of training is to minimize this loss. —

```
loss.backward()
```

*What does it do?*

It computes the gradients of the loss with respect to all of the model parameters using backpropagation.

*Why is it needed?*

- 1.5 It is needed to calculate the gradient of the loss with respect to the model weights. This is necessary to update the weights in the right direction during optimization.**

```
optimizer.step()
```

*What does it do?*

It updates the parameters of the model using the computed gradients. > *Why is it needed?* If this step is not performed, the model weights will not be updated and the loss will not improve over time. —

```
[34]: def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

### 1.5.1 Question 9

For each of the lines of code below, describe what they do and why it is needed in the model evaluation process. What would go wrong if that step was forgotten or removed?

```
with torch.no_grad()
```

*What does it do?*

It temporarily disables the gradient calculation context, which makes the forward pass computations more memory-efficient

*Why is it needed?* During evaluation, we don't need to compute gradients since we're not going to backpropagate and update the model's parameters. If we forget to use this — memory usage during evaluation will be unnecessarily high and the evaluation may run slower. —

```
[36]: def test(model, device, test_loader):
    test_loss = 0
    correct = 0
    model.eval()
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.
         format(
            test_loss, correct, len(test_loader.dataset),
            100. * correct / len(test_loader.dataset)))
```

### 1.5.2 Question 10

What does the `.to(device)` method do and why is it needed - what would happen if that step was forgotten or removed?

```
[37]: n_epochs = 5
device = torch.device("cpu")
torch_model = Net().to(device)
optimizer = optim.Adam(torch_model.parameters())

for epoch in range(n_epochs):
    train(torch_model, device, train_loader, optimizer, epoch+1)
    test(torch_model, device, test_loader)
```

```
Train Epoch: 1 [0/60000 (0%)] Loss: 2.304742
Train Epoch: 1 [12800/60000 (21%)] Loss: 0.240950
Train Epoch: 1 [25600/60000 (43%)] Loss: 0.149276
Train Epoch: 1 [38400/60000 (64%)] Loss: 0.168552
Train Epoch: 1 [51200/60000 (85%)] Loss: 0.193709
```

```
Test set: Average loss: 0.0829, Accuracy: 9741/10000 (97%)
```

```
Train Epoch: 2 [0/60000 (0%)] Loss: 0.093725
Train Epoch: 2 [12800/60000 (21%)] Loss: 0.114255
Train Epoch: 2 [25600/60000 (43%)] Loss: 0.097460
Train Epoch: 2 [38400/60000 (64%)] Loss: 0.057916
Train Epoch: 2 [51200/60000 (85%)] Loss: 0.151071
```

Test set: Average loss: 0.0598, Accuracy: 9813/10000 (98%)

```
Train Epoch: 3 [0/60000 (0%)] Loss: 0.129273
Train Epoch: 3 [12800/60000 (21%)] Loss: 0.093605
Train Epoch: 3 [25600/60000 (43%)] Loss: 0.064539
Train Epoch: 3 [38400/60000 (64%)] Loss: 0.081091
Train Epoch: 3 [51200/60000 (85%)] Loss: 0.115277
```

Test set: Average loss: 0.0447, Accuracy: 9847/10000 (98%)

```
Train Epoch: 4 [0/60000 (0%)] Loss: 0.094472
Train Epoch: 4 [12800/60000 (21%)] Loss: 0.080404
Train Epoch: 4 [25600/60000 (43%)] Loss: 0.052637
Train Epoch: 4 [38400/60000 (64%)] Loss: 0.126223
Train Epoch: 4 [51200/60000 (85%)] Loss: 0.133436
```

Test set: Average loss: 0.0436, Accuracy: 9857/10000 (99%)

```
Train Epoch: 5 [0/60000 (0%)] Loss: 0.081542
Train Epoch: 5 [12800/60000 (21%)] Loss: 0.067959
Train Epoch: 5 [25600/60000 (43%)] Loss: 0.051706
Train Epoch: 5 [38400/60000 (64%)] Loss: 0.087659
Train Epoch: 5 [51200/60000 (85%)] Loss: 0.106181
```

Test set: Average loss: 0.0378, Accuracy: 9868/10000 (99%)

## 1.6 Compare/Contrast

### 1.6.1 Question 11

Write a paragraph to summarize the similarities and differences between Tensorflow and Pytorch.  
Cite or link to any sources you find that help with your answer.

References:

<https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b>  
<https://towardsdatascience.com/keras-vs-pytorch-for-deep-learning-a013cb63870d>  
<https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article>

### 1.6.2 Question 12

If you needed to build a neural network for image recognition right now, would you prefer to use Tensorflow or Pytorch? And why?

*Note: Either choice is fine, this question is about matching the reasoning “why” behind the one of the choices.*

[ ]:

# Notebook

August 2, 2023

## 1 Lab 2

```
[334]: FIRST_NAME = "Leng"  
LAST_NAME = "Her"  
STUDENT_ID = "5445877"
```

### 1.1 Introduction

Scikit Learn is one of the most prominent tools for building machine learning models in the data science industry. It contains a plethora of standardized tools for managing the entire machine learning development workflow.

In this lab, you will use the same simulated data set as last week to build, validate, and evaluate machine learning models with various regression algorithms. We will also save this model to a file so it can be utilized to make future predictions.

### 1.2 The Data Set

#### Data Description

This is a simulated data set of students performance in the INET 4062 class. *None of these are actual students.*

#### Data Dictionary

Column Name	Type	Description
studentId	int	Unique Id of student
gpa	float	Current cumulative GPA
labHours	float	Number of hours spent per week on labs
studyHours	float	Number of hours spent studying for each exam
took4061	int	Binary if student took INET 4061 (0=No, 1=Yes)
pythonExp	int	A High, Medium, or Low rating from student on previous python experience (0=Low, 1=Medium, 2=High)
statsRating	int	A 0-5 rating of ability on statistics
height	float	Height of student in inches
eyeColor	str	Eye color of student

Column Name	Type	Description
followers	int	Number of followers on all social media accounts
grade	float	Percentage grade in INET 4062 out of 100
letterGrade	str	Letter grade derived from the percentage

## Data Sample

studentId	gpa	labHours	studyHours	took4061	pythonExperience	statsRating	height	eyeColor	followers	grade	letterGrade
0	3.6206	3.0089	4.36066	1	2	3	69.4933	brown	632	92.31	A-
1	3.1939	12.524	4.88687	0	2	2	67.3275	blue	44	85.59	B
2	3.1945	0.903686	2.0478	1	1	5	69.3401	green	181	88.39	B+
3	3.2779	4.88015	0.822806	1	2	4	67.8951	blue	347	90.91	A-
4	2.5	1.47281	7.51036	0	2	4	67.708	brown	1070	84.14	B
5	2.5616	4.53166	5.50934	0	1	5	69.6897	hazel	18	84.37	B
6	3.1558	2.60646	1.56167	0	1	5	69.045	hazel	7007	84.2	B
7	3.7340	2.41052	2.99812	1	2	5	66.5794	brown	5599	93.6	A
8	2.9845	2.68131	1.73898	1	2	4	69.1432	hazel	1206	88.73	B+
9	3.8450	0.43147	4.9316	1	1	1	68.034	blue	40097	92.12	A-

```
[335]: import numpy as np
import pandas as pd

n = 1000 # number of records to simulate
np.random.seed(40) # set the seed of the random number generator

# Current GPA
gpa = 0.4 * np.random.randn(n) + 3.25
gpa = np.clip(gpa, 2.5, 4.0)

# Average hours per week on Labs
labHours = 5.5/np.exp(2*np.random.rand(n))

# Number of hours studying for exam
studyHours = np.power(2*np.random.rand(n) + 0.75, 2)

# Junior or Senior
isSenior = np.random.binomial(size=n, n=1, p=0.67)

# Took 4061
took4061 = np.random.binomial(size=n, n=1, p=0.75)

# Previous Python Experience
```

```

pythonExp = np.random.binomial(size=n, n=2, p=0.70)

# Ability in statistics
statsRating = np.random.binomial(size=n, n=5, p=0.75)

# Height
height = 4 * np.random.rand(n) + 66.5

# Eye Color
eyeColor = np.random.choice(["blue", "green", "brown", "hazel"], n)

# Social media followers
followers = (10 ** (1+5*np.random.beta(3, 7, size=n))).round()

# simulate grades
grade = 72 + (((gpa**2)/3 + np.sqrt(statsRating+1)) *
               np.sqrt(labHours/3 + studyHours/6 + pythonExp + 3*took4061)) + \
        (1+3*np.random.rand())

# Compile columns into a DataFrame
students_df = pd.DataFrame({
    'gpa' : gpa,
    'labHours' : labHours,
    'studyHours' : studyHours,
    'took4061' : took4061,
    'pythonExp' : pythonExp,
    'statsRating' : statsRating,
    'height' : height,
    'eyeColor' : eyeColor,
    'followers' : followers,
    'grade' : grade.round(2)
})

# Define a function to calculate the letter grade based
# on the percentage in the class
def getLetterGrade(x):
    if x < 76.67:
        return("C")
    elif x < 80:
        return("C+")
    elif x < 83.33:
        return("B-")
    elif x < 86.67:
        return("B")
    elif x < 90:
        return("B+")
    elif x < 93.33:

```

```

        return("A-")
    else:
        return("A")

# Add the letter grade column to the DataFrame
students_df['letterGrade'] = students_df['grade'].apply(lambda row: ↴
    ↪getLetterGrade(row))

# Rename the index of the DataFrame to be `studentId`
# because that index uniquely identifies 1 student
students_df.index.rename("studentId", inplace=True)
students_df.reset_index(drop=False, inplace=True)

```

[336]: students\_df

```

[336]:      studentId      gpa  labHours  studyHours  took4061  pythonExp \
0            0  3.006981  3.013592   3.187031       1          1
1            1  3.199545  1.025472   2.750329       1          0
2            2  2.976157  3.032981   7.211016       1          2
3            3  3.621486  1.799475   5.882726       1          2
4            4  2.512240  3.950051   3.418261       1          2
..           ...
995          995  2.801776  2.742993   2.137272       1          2
996          996  3.618104  1.017808   4.031764       1          0
997          997  3.838771  1.576157   3.818294       0          2
998          998  3.688545  1.297447   0.953019       1          1
999          999  3.157349  2.109959   3.289108       1          2

      statsRating      height eyeColor  followers  grade letterGrade
0            2  66.605363     blue      154.0  85.57          B
1            3  68.511122    hazel      1091.0  84.96          B
2            4  68.770468    brown      749.0  88.34          B+
3            4  67.058603    brown      136.0  91.36          A-
4            3  69.320191    brown      802.0  85.17          B
..           ...
995          4  68.828923     blue      3712.0  86.56          B
996          4  69.533289     blue      5676.0  87.62          B+
997          3  69.373830    green      104.0  86.70          B+
998          3  68.642846     blue      262.0  88.41          B+
999          4  68.545741    hazel      52.0  88.30          B+

```

[1000 rows x 12 columns]

### 1.3 Question 1

The first step is to split data into the independent variables (features) and the dependent variable (label), and to split the data into a train and test set.

First, create a DataFrame named `X` with only the following columns “`gpa`”, “`labHours`”, “`studyHours`”, “`took4061`”, “`pythonExp`”, “`statsRating`”.

Next, create a Series (a 1 column DataFrame) name `y` that contains just the “`grade`” column of the DataFrame.

Then do a train/test split on the `X` and `y` datasets where 75% of the rows are in the train set, and 25% are in the test set. Set the random seed to 4062 for the train/test split.

### 1.3.1 Resources

**Scikit Learn Documentation** \* `sklearn.model_selection.train_test_split` ([link](#))

**Pandas Documentation** \* Indexing and Selecting data ([link](#))

**Examples** \* Selecting multiple columns `datagy` \* Selecting multiple columns `statology` \* Train Test Split (Real Python) \* Train Test Split ([Machine Learning Mastery](#))

### 1.3.2 Answer

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

[338]: # select the independent variables and store in X
X = students_df[["gpa", "labHours", "studyHours", "took4061", "pythonExp", ↴
    "statsRating"]]

# select the dependent variable (label) and store in y
y = students_df["grade"]

# split the data into a train and test set with 75% of the data in the train ↴
# set and 25% in the test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ↴
    random_state=4062)
```

```
[338]:
```

## 1.4 Question 2

One hot encode the categorical features for the model “`took4061`”, “`pythonExp`”, “`statsRating`”. Fit and transform the OneHotEncoder on the `X_train` DataFrame. Save the results of the one hot encoding transformation into a variable named `categoricalVars`.

Next, convert the `categoricalVars` array into a Pandas DataFrame named `categoricalVars_df` with the column names from the `.get_feature_names_out()` method of your OneHotEncoder object.

### 1.4.1 Resources

**Scikit Learn Documentation** \* `sklearn.preprocessing.OneHotEncoder` ([link](#))

**Examples** \* Datagy ([link](#)) \* Geeks4Geeks ([link](#))

### 1.4.2 Answer

```
[339]: from sklearn.preprocessing import OneHotEncoder  
  
[340]: encoder = OneHotEncoder()  
categoricalVars = encoder.fit_transform(X_train[["took4061", "pythonExp",  
    ↴"statsRating"]])  
categoricalVars_df = pd.DataFrame(categoricalVars.toarray(), columns=encoder.  
    ↴get_feature_names_out())
```

## 1.5 Question 3

Normalize the numeric features for the machine learning models “*gpa*”, “*labHours*”, “*studyHours*”. Fit and transform the StandardScaler on the *X\_train* DataFrame. Save the results of the transformation into a variable named *normalizedVars*.

Next, convert the *normalizedVars* array into a Pandas DataFrame named *normalizedVars\_df* with the column names from the *.get\_feature\_names\_out()* method of your StandardScaler object.

### 1.5.1 Resources

**Scikit Learn Documentation** \* `sklearn.preprocessing.StandardScaler` ([link](#))

**Examples** \* Machine Learning Mastery ([link](#)) \* BenAlexKeen ([link](#))

### 1.5.2 Answer

```
[341]: from sklearn.preprocessing import StandardScaler  
  
[342]: # Select the numeric features  
numeric_features = ["gpa", "labHours", "studyHours"]  
X_train_num = X_train[numeric_features]  
  
# Fit and transform the StandardScaler on the numeric features in X_train  
scaler = StandardScaler()  
normalizedVars = scaler.fit_transform(X_train_num)  
  
# Convert normalizedVars array into a Pandas DataFrame  
normalizedVars_df = pd.DataFrame(normalizedVars, columns=numeric_features)
```

## 1.6 Question 4

Combine the columns of the two DataFrames *categoricalVars\_df* and *normalizedVars\_df*. Use the `pd.concat()` function with `axis='columns'`. Save the resulting DataFrame into a variable named *X\_train\_cleaned*.

### 1.6.1 Resources

**Pandas Documentation** \* `pd.concat` ([link](#))

Examples \* Statology ([link](#)) \* W3Resource ([link](#))

### 1.6.2 Answer

```
[343]: X_train_cleaned = pd.concat([categoricalVars_df, normalizedVars_df],  
    ↪axis='columns')
```

```
[344]: X_train_cleaned
```

```
[344]:      took4061_0  took4061_1  pythonExp_0  pythonExp_1  pythonExp_2  \  
0          0.0        1.0        0.0        0.0        1.0  
1          0.0        1.0        0.0        1.0        0.0  
2          1.0        0.0        1.0        0.0        0.0  
3          0.0        1.0        0.0        0.0        1.0  
4          0.0        1.0        0.0        0.0        1.0  
..        ...        ...        ...        ...        ...  
745         0.0        1.0        0.0        0.0        1.0  
746         0.0        1.0        0.0        1.0        0.0  
747         0.0        1.0        0.0        1.0        0.0  
748         1.0        0.0        0.0        0.0        1.0  
749         0.0        1.0        0.0        1.0        0.0  
  
      statsRating_0  statsRating_1  statsRating_2  statsRating_3  \  
0          0.0        0.0        0.0        0.0  
1          0.0        1.0        0.0        0.0  
2          0.0        0.0        0.0        0.0  
3          0.0        0.0        0.0        1.0  
4          0.0        0.0        0.0        1.0  
..        ...        ...        ...        ...  
745         0.0        0.0        0.0        0.0        0.0  
746         0.0        0.0        0.0        1.0  
747         0.0        0.0        0.0        0.0  
748         0.0        0.0        0.0        0.0  
749         0.0        0.0        0.0        0.0  
  
      statsRating_4  statsRating_5       gpa  labHours  studyHours  
0          1.0        0.0  0.455838 -0.704009 -0.676952  
1          0.0        0.0  0.812140 -0.939033 -0.352502  
2          0.0        1.0  0.155625  1.593085  1.425394  
3          0.0        0.0  1.294941 -0.740124 -0.039968  
4          0.0        0.0 -1.462300 -1.150700  0.052982  
..        ...        ...        ...        ...  
745         1.0        0.0 -1.010872  0.527532 -0.844072  
746         0.0        0.0  1.936559  1.803265  1.516103  
747         0.0        1.0  0.694991 -1.129082  1.051331  
748         1.0        0.0  0.197053 -1.055185  0.639203  
749         1.0        0.0  1.082019 -0.961691  0.247238
```

```
[750 rows x 14 columns]
```

## 1.7 Question 5

Test the performance of 3 different algorithms to predict the grades of each student: Support Vector Machines, K-Nearest Neighbors, and Random Forest. Try various different hyperparameters for each algorithm (*except for this case, do not use weight=distance for KNN*).

Track the Mean Absolute Error (MAE) on the training data set for each set of hyperparameters for each algorithm that you try.

### 1.7.1 Resources

**Scikit Learn Documentation** \* LinearSVR ([link](#)) \* KNeighborsRegressor ([link](#)) \* RandomForestRegressor ([link](#)) \* mean\_absolute\_error ([link](#))

### 1.7.2 Answer

```
[345]: from sklearn.svm import LinearSVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_absolute_error
```

```
[346]: svm = LinearSVR()
knn = KNeighborsRegressor()
rf = RandomForestRegressor()
```

```
[347]: # Fitting the models
svm.fit(X_train_cleaned, y_train)
knn.fit(X_train_cleaned, y_train)
rf.fit(X_train_cleaned, y_train)
```

```
[347]: RandomForestRegressor()
```

```
[348]: # Predicting with the models
y_train_pred_svm = svm.predict(X_train_cleaned)
y_train_pred_knn = knn.predict(X_train_cleaned)
y_train_pred_rf = rf.predict(X_train_cleaned)
```

```
[349]: # Calculating MAE
mae_svm = mean_absolute_error(y_train, y_train_pred_svm)
mae_knn = mean_absolute_error(y_train, y_train_pred_knn)
mae_rf = mean_absolute_error(y_train, y_train_pred_rf)
```

```
[350]: print("Mean Absolute Error - SVM:", mae_svm)
print("Mean Absolute Error - KNN:", mae_knn)
```

```
print("Mean Absolute Error - RF:", mae_rf)
```

```
Mean Absolute Error - SVM: 0.2676725559973789  
Mean Absolute Error - KNN: 0.48866133333333284  
Mean Absolute Error - RF: 0.16256840000000358
```

## 1.8 Question 6

Evaluate the results of the models on the training data set. Select the model with the lowest Mean Absolute Error (MAE). Then re-train that algorithm with the same hyperparameters and calculate the MAE on the test data set.

```
[351]: encoder = OneHotEncoder()  
categoricalVars = encoder.fit_transform(X_train[["took4061", "pythonExp",  
"statsRating"]])  
categoricalVars_df = pd.DataFrame(categoricalVars.toarray(), columns=encoder.  
get_feature_names_out())
```

```
[352]: #Apply encoding on the test set  
  
# Categorical feature preprocessing  
categoricalVars = encoder.transform(X_test[["took4061", "pythonExp",  
"statsRating"]])  
categoricalVars_df = pd.DataFrame(categoricalVars.toarray(), columns=encoder.  
get_feature_names_out())  
  
# Numerical feature preprocessing  
normalizedVars = scaler.transform(X_test[["gpa", "labHours", "studyHours"]])  
normalizedVars_df = pd.DataFrame(normalizedVars, columns=scaler.  
get_feature_names_out())  
  
# Combine all features  
X_test_cleaned = pd.concat([categoricalVars_df, normalizedVars_df],  
axis='columns')
```

```
[353]: # Re-train the selected model with the same hyperparameters  
rf.fit(X_train_cleaned, y_train)  
y_pred = rf.predict(X_test_cleaned)  
best_model_mae = mean_absolute_error(y_test, y_pred)  
print('MAE on the test data set:', best_model_mae)
```

```
MAE on the test data set: 0.4021979999999989
```

## 1.9 Question 7

Save the model with the lowest mean absolute error (MAE) to a Pickle file named `model.pkl`. \* Retrain the algorithm with the same hyper parameters as in the previous step. \* Get the mean

average error of the model on the test data set \* Use the Pickle Python package the model object to a file named `model.pkl`

### 1.9.1 Resources

**Python Documentation** \* Pickle Examples ([link](#)) \* Pickle Docs ([link](#))

Examples \* Datacamp ([link](#)) \* Real Python ([link](#))

### 1.9.2 Answer

```
[354]: import pickle
```

```
[355]: rf.fit(X_train_cleaned, y_train)
```

```
[355]: RandomForestRegressor()
```

```
[356]: y_pred2 = rf.predict(X_test_cleaned)
```

```
[357]: best_model_mae2 = mean_absolute_error(y_test, y_pred)
print('MAE on the test data set:', best_model_mae2)
```

MAE on the test data set: 0.4021979999999989

```
[358]: with open("model.pkl", "wb") as file:
pickle.dump(rf, file)
```

## 1.10 Question 8

Write a Python function named `predict()` that takes in the data about new students in a dictionary, and returns the predictions from the `model.pkl` object that was recently saved.

On the example below, it should return something like:

```
{'predictions': [84.76219999999998]}
```

- First, read in the `model.pkl` file into a new python variable named “model”.
- Next, convert the input dictionary into a Pandas DataFrame using `pd.DataFrame()`.
- Then, run the `.predict()` function on the DataFrame to get an array with the prediction. Use the `.tolist()` method of the array to convert it into a list.
- Return the list in a dictionary with the key as ‘`predictions`’.

### 1.10.1 Answer

```
[359]: new_students = [{  
    'took4061_0' : 1,  
    'took4061_1' : 0,  
    'pythonExp_0' : 1,  
    'pythonExp_1' : 0,  
    'pythonExp_2' : 0,  
}]
```

```
'statsRating_0' : 0,  
'statsRating_1' : 0,  
'statsRating_2' : 0,  
'statsRating_3' : 1,  
'statsRating_4' : 0,  
'statsRating_5' : 0,  
'gpa' : 3.1,  
'labHours' : 2.2,  
'studyHours' : 4.7  
}]
```

```
[360]: def predict(new_students: dict) -> dict:  
    # Read in the model  
    with open('model.pkl', 'rb') as f:  
        model = pickle.load(f)  
  
    # Convert the input dictionary into a pandas DataFrame  
    X_new = pd.DataFrame(new_students)  
  
    # Ensure that the feature names are in the same order as they were in the  
    # training data  
    X_new = X_new[['took4061_0', 'took4061_1', 'pythonExp_0', 'pythonExp_1',  
    'pythonExp_2', 'statsRating_0',  
    'statsRating_1', 'statsRating_2', 'statsRating_3',  
    'statsRating_4', 'statsRating_5',  
    'gpa', 'labHours', 'studyHours']]  
  
    # Get predictions from the model  
    predictions = model.predict(X_new).tolist()  
  
    # Return the predictions in a dictionary with the key 'predictions'  
    return {'predictions': predictions}
```

```
[361]: print(predict(new_students))
```

```
{'predictions': [84.77680000000004]}
```

## 2 End

```
[361]:
```

# Notebook

August 2, 2023

## 1 Lab 11

```
[ ]: FIRST_NAME = "Leng"  
      LAST_NAME = "Her"  
      STUDENT_ID = "5445877"
```

### 1.1 Introduction

In this lab we will solve the [diet](#) problem, or otherwise known as “Mix and Blend”. Traditionally this problem is trying to decide which foods to include in a diet to try to meet various specifications. In this lab, we will mix and blend various different types of wines to make a blend that meets our quality and cost specifications.

### 1.2 The Data Set

#### Data Description

This lab uses an open source data set about wine quality from the UCI Machine Learning [repository](#). It was originally used in the paper [Cortez et al., 2009](#). The 2 datasets are separated into red and white wines and includes columns related to the physicochemical inputs and average subjective quality ratings. Cost information was excluded from the open source data so we will simulate that parameter below.

#### Acknowledgements

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

#### Data Dictionary

More information about the data set is also available on [Kaggle](#).

Column Name	Type	Description
fixed acidity	float	physicochemical input
volatile acidity	float	physicochemical input
citric acid	float	physicochemical input
residual sugar'	float	physicochemical input
chlorides	float	physicochemical input
free sulfur dioxide	float	physicochemical input

Column Name	Type	Description
total sulfur dioxide	float	physicochemical input
density	float	physicochemical input
pH	float	physicochemical input
sulphates	float	physicochemical input
alcohol	float	physicochemical input
quality	int	0-10 quality output

## Data Sample

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	pH	sulphates	alcohol	quality
0	7.4	0.7	0	1.9	0.076	11	34	0.99783.51	0.56	9.4	5
1	7.8	0.88	0	2.6	0.098	25	67	0.99683.2	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15	54	0.997 3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17	60	0.998 3.16	0.58	9.8	6
4	7.4	0.7	0	1.9	0.076	11	34	0.99783.51	0.56	9.4	5
5	7.4	0.66	0	1.8	0.075	13	40	0.99783.51	0.56	9.4	5
6	7.9	0.6	0.06	1.6	0.069	15	59	0.99643.3	0.46	9.4	5
7	7.3	0.65	0	1.2	0.065	15	21	0.99463.39	0.47	10	7
8	7.8	0.58	0.02	2	0.073	9	18	0.99683.36	0.57	9.5	7
9	7.5	0.5	0.36	6.1	0.071	17	102	0.99783.35	0.8	10.5	5

```
[1]: ! wget https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
      ↵ winequality-red.csv
! wget https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
      ↵ winequality-white.csv
```

```
--2023-04-16 18:18:01-- https://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 84199 (82K) [application/x-httdp-php]
Saving to: 'winequality-red.csv'
```

```
winequality-red.csv 100%[=====] 82.23K 315KB/s in 0.3s
```

```
2023-04-16 18:18:02 (315 KB/s) - 'winequality-red.csv' saved [84199/84199]
```

```
--2023-04-16 18:18:02-- https://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-white.csv
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
```

```
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443...
connected.
```

```
HTTP request sent, awaiting response... 200 OK
Length: 264426 (258K) [application/x-httdp-php]
Saving to: 'winequality-white.csv'
```

```
winequality-white.c 100%[=====] 258.23K 488KB/s in 0.5s
```

```
2023-04-16 18:18:03 (488 KB/s) - 'winequality-white.csv' saved [264426/264426]
```

### 1.3 Setup

```
[17]: import os
import pandas as pd
import numpy as np
```

#### Download data

```
[18]: # Read files into data frames
red_df = pd.read_csv("winequality-red.csv", sep=";")
white_df = pd.read_csv("winequality-white.csv", sep=";")
```

```
[19]: red_df
```

```
[19]:      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides \
0            7.4          0.700        0.00           1.9       0.076
1            7.8          0.880        0.00           2.6       0.098
2            7.8          0.760        0.04           2.3       0.092
3           11.2          0.280        0.56           1.9       0.075
4            7.4          0.700        0.00           1.9       0.076
...
1594          6.2          0.600        0.08           2.0       0.090
1595          5.9          0.550        0.10           2.2       0.062
1596          6.3          0.510        0.13           2.3       0.076
1597          5.9          0.645        0.12           2.0       0.075
1598          6.0          0.310        0.47           3.6       0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates \
0                11.0              34.0  0.99780  3.51       0.56
1                25.0              67.0  0.99680  3.20       0.68
2                15.0              54.0  0.99700  3.26       0.65
3                17.0              60.0  0.99800  3.16       0.58
4                11.0              34.0  0.99780  3.51       0.56
...
1594              32.0              44.0  0.99490  3.45       0.58
1595              39.0              51.0  0.99512  3.52       0.76
1596              29.0              40.0  0.99574  3.42       0.75
```

```

1597           32.0        44.0  0.99547  3.57      0.71
1598           18.0        42.0  0.99549  3.39      0.66

    alcohol  quality
0         9.4        5
1        9.8        5
2        9.8        5
3        9.8        6
4        9.4        5
...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]

```

```
[20]: # Simulate wine costs
np.random.seed(4062)
red_df["cost"] = 2 * np.sqrt(np.exp(red_df["quality"]) + np.random.
                             ↪randn(len(red_df)))
white_df["cost"] = 2 * np.sqrt(np.exp(white_df["quality"]) + np.random.
                               ↪randn(len(white_df)))
```

**Configure Gurobi Solver** Follow the instructions on Canvas to download an academic Gurobi license. Upload the `gurobi.lic` file to this Colab Noteboook. Then run the following piece of code to install and authenticate the solver package.

```
[21]: ! pip install gurobipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: gurobipy in /usr/local/lib/python3.9/dist-
packages (10.0.1)
```

```
[23]: os.environ['GRB_LICENSE_FILE'] = f'{os.getcwd()}/gurobi.lic'
```

## 1.4 Question 1

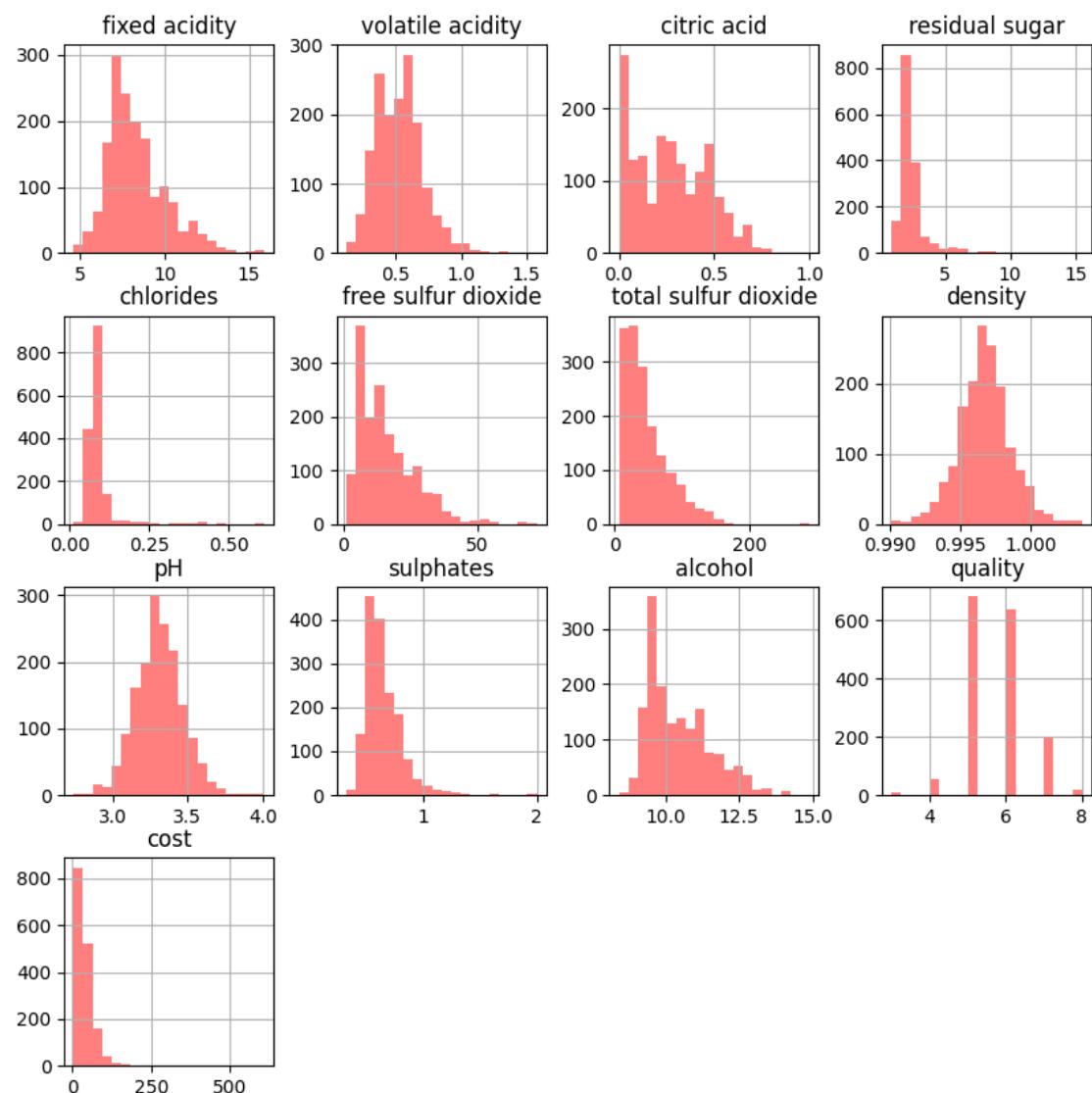
### 1.4.1 EDA

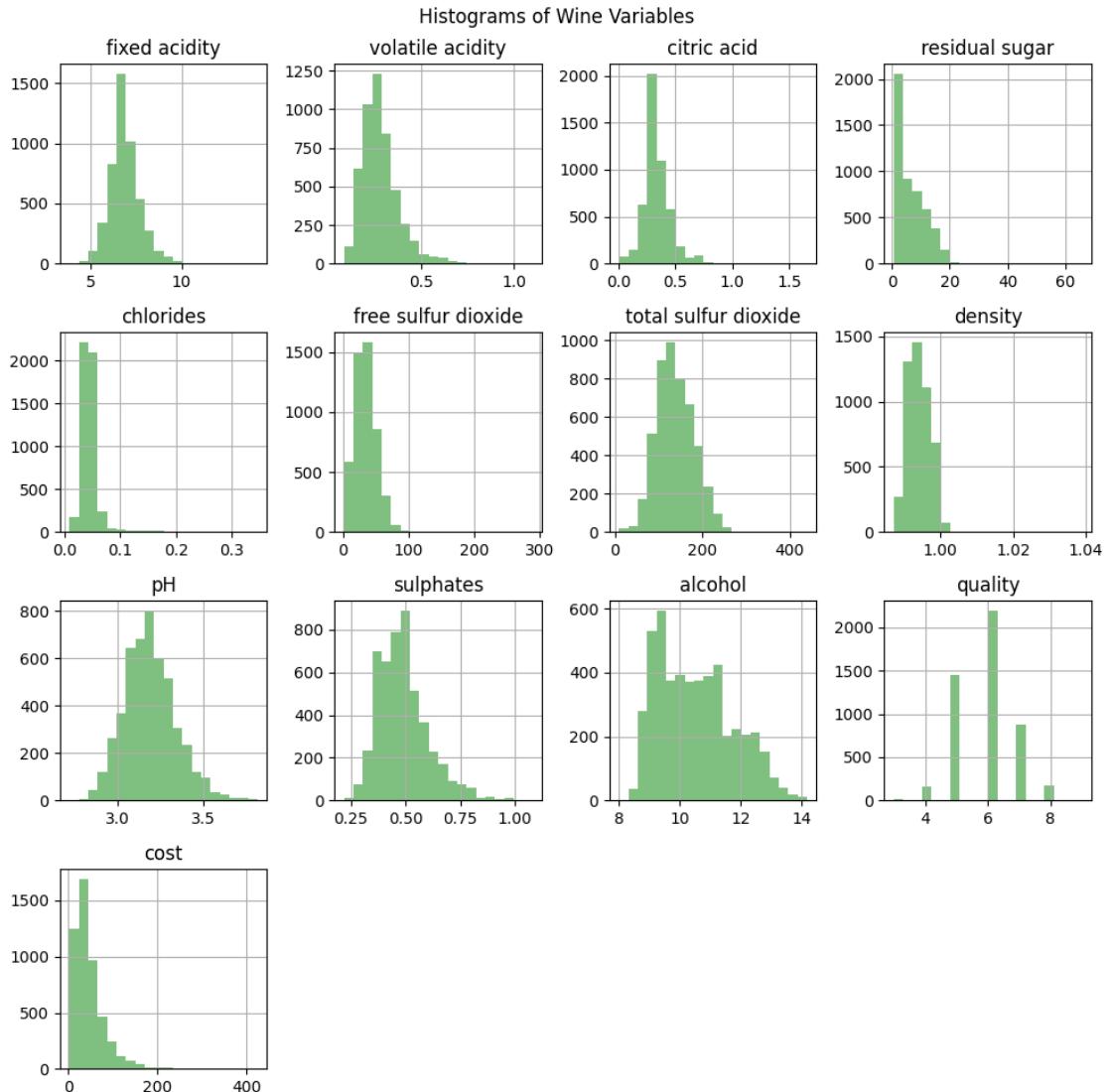
Create histogram plots for each variable in the data set to visualize their distributions. Overlap the red and white datasets on the same plots to see the difference between the two types of wines.

```
[24]: import seaborn as sns
import matplotlib.pyplot as plt
```

```
[25]: # Create histogram plots for each variable
red_df.hist(alpha=0.5, color='red', bins=20, figsize=(10,10))
white_df.hist(alpha=0.5, color='green', bins=20, figsize=(10,10))

# Add titles and labels
plt.suptitle("Histograms of Wine Variables")
plt.tight_layout()
plt.show()
```





## 1.5 Mathematical Formulations

## Sets

- Set of all wines -  $w \in W$
  - Set of all traits -  $t \in T$

## Parameters

- Cost of wine w -  $C_w$
  - Amount of train t in wine w - \$ A\_{tw} \$
  - Minimum amount of trait t in the final blend -  $L_t$
  - Maximum amount of trait t in the final blend -  $U_t$

## Variables

- $X_w$  - percent of blend that comes from wine w

**Constraints** The percentage totals for the wines must add up to 100%

$$\sum_w X_w = 1$$

Each variable must be within the upper and lower bounds

$$L_t \leq \sum_w A_{wt} \cdot X_w \leq U_t \quad \forall t \in T$$

## Objective Function

$$\sum_w C_w \cdot X_w$$

### 1.6 Question 2

#### 1.6.1 Solving

Find the optimal solution to the linear program above using the data from the red wine data set and the parameters below.

```
[27]: from gurobipy import GRB, Model, quicksum
```

Define Model

```
[28]: # Define Model
model = Model('wine_blend')
```

Set parameter WLSAccessID  
Set parameter WLSSecret  
Set parameter LicenseID to value 2366536  
Academic license - for non-commercial use only - registered to her00024@umn.edu

Sets

```
[31]: wines = red_df.index.tolist()
traits = red_df.drop(columns=["quality", "cost"]).columns.tolist()
```

Parameters

```
[51]: ## Parameters

# Minimum
min_values = {
    'fixed acidity': 5,
    'volatile acidity': 0.1,
    'citric acid': 0,
```

```

'residual sugar': 1.2,
'chlorides': 0.012,
'free sulfur dioxide': 1,
'total sulfur dioxide': 6,
'density': 0.99,
'pH': 2.74,
'sulphates': 0.33,
'alcohol': 8.4
}

# Maximum
max_values = {
    'fixed acidity': 15.9,
    'volatile acidity': 1.58,
    'citric acid': 1,
    'residual sugar': 15.5,
    'chlorides': 0.611,
    'free sulfur dioxide': 72,
    'total sulfur dioxide': 289,
    'density': 1.00369,
    'pH': 4.01,
    'sulphates': 2,
    'alcohol': 14.9
}

```

[ ]:

Variables

[45]: `x = model.addVars(wines, name="wine")`

Constraints

[46]: `# Constraints`  
`model.addConstr(quicksum(x[wine] for wine in wines) == 1,`  
`↳name="total_percentage")`

[46]: <gurobi.Constr \*Awaiting Model Update\*>

[52]: `for trait in traits:`  
 `model.addConstr(`  
 `quicksum(red_df.loc[wine, trait] * x[wine] for wine in wines) >=`  
`↳min_values[trait],`  
 `name=f"min_{trait}"`  
 `)`  
 `model.addConstr(`  
 `quicksum(red_df.loc[wine, trait] * x[wine] for wine in wines) <=`  
`↳max_values[trait],`

```
    name=f"max_{trait}"  
)
```

[ ]:

Objective

```
[55]: # Objective Function  
Cw = red_df['cost'].to_dict()  
model.setObjective(quicksum(Cw[i] * x[i] for i in wines), GRB.MINIMIZE)
```

Optimize

```
[56]: # Optimize  
model.optimize()
```

Gurobi Optimizer version 10.0.1 build v10.0.1rc0 (linux64)

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]  
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Academic license - for non-commercial use only - registered to her00024@umn.edu

Optimize a model with 68 rows, 3198 columns and 107808 nonzeros

Model fingerprint: 0xb627aa1c

Coefficient statistics:

```
Matrix range      [1e-02, 3e+02]  
Objective range   [4e+00, 6e+02]  
Bounds range     [0e+00, 0e+00]  
RHS range        [1e-02, 3e+02]
```

Presolve removed 46 rows and 484 columns

Presolve time: 0.11s

Presolved: 22 rows, 2728 columns, 29636 nonzeros

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	4.4388337e+00	4.415444e+01	0.000000e+00	0s
18	4.4388337e+00	0.000000e+00	0.000000e+00	0s

Solved in 18 iterations and 0.17 seconds (0.04 work units)

Optimal objective 4.438833742e+00

```
[58]: # Print Solution  
#print("Optimal Blend:")  
#for wine in wines:  
#    print(f"{wine}: {x[wine].x*100:.2f}%")  
#print(f"\nTotal Cost: ${model.objVal:.2f}")
```

# Notebook

August 2, 2023

**Title:** INET 4061 Final Project

**Authors:** Brian Bianchi, Maaz Mohammad, Leng Her

**Date:** December 10, 2022

#Install Requirements

```
[ ]: !pip install category_encoders  
!pip install lazypredict
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/  
Collecting category_encoders  
  Downloading category_encoders-2.5.1.post0-py2.py3-none-any.whl (72 kB)  
 | 72 kB 342 kB/s  
Requirement already satisfied: patsy>=0.5.1 in  
/usr/local/lib/python3.8/dist-packages (from category_encoders) (0.5.3)  
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from category_encoders) (1.7.3)  
Requirement already satisfied: scikit-learn>=0.20.0 in  
/usr/local/lib/python3.8/dist-packages (from category_encoders) (1.0.2)  
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.8/dist-packages (from category_encoders) (1.21.6)  
Requirement already satisfied: statsmodels>=0.9.0 in  
/usr/local/lib/python3.8/dist-packages (from category_encoders) (0.12.2)  
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.8/dist-packages (from category_encoders) (1.3.5)  
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.0.5->category_encoders) (2022.6)  
Requirement already satisfied: python-dateutil>=2.7.3 in  
/usr/local/lib/python3.8/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)  
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from patsy>=0.5.1->category_encoders) (1.15.0)  
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.2.0)  
Requirement already satisfied: threadpoolctl>=2.0.0 in  
/usr/local/lib/python3.8/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.1.0)
```

```

Installing collected packages: category-encoders
Successfully installed category-encoders-2.5.1.post0

[ ]: ['Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/',
      'Collecting lazypredict',
      '  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)',
      'Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.2.0)',
      'Requirement already satisfied: lightgbm in /usr/local/lib/python3.8/dist-packages (from lazypredict) (2.2.3)',
      'Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.3.5)',
      'Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from lazypredict) (7.1.2)',
      'Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.0.2)',
      'Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from lazypredict) (4.64.1)',
      'Requirement already satisfied: xgboost in /usr/local/lib/python3.8/dist-packages (from lazypredict) (0.90)',
      'Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from lightgbm->lazypredict) (1.21.6)',
      'Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from lightgbm->lazypredict) (1.7.3)',
      'Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas->lazypredict) (2.8.2)',
      'Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->lazypredict) (2022.6)',
      'Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7.3->pandas->lazypredict) (1.15.0)',
      'Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->lazypredict) (3.1.0)',
      'Installing collected packages: lazypredict',
      'Successfully installed lazypredict-0.2.12']

```

#Overview

### Our Objective:

“We are seeking to determine which data science processes and models yield the most accurate predictions of housing prices from some descriptive data.”

Presentation link with more info: <https://docs.google.com/presentation/d/1qPi01IXcnE6CukFmFAVa38ihbEJmuj>

### #Exploratory Data Analysis

Data Source:

<https://www.kaggle.com/datasets/gyanshashwat1611/housing-prices-data>

```
[ ]: import pandas as pd
import numpy as np
import scipy
import statsmodels
from category_encoders import *
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
import math
original = pd.read_csv("train.csv")
# 1460 rows

train = original[:730]

test = original[730:]

# enc = TargetEncoder().fit(df)

# df_new = enc.transform(df)
train.shape
```

[ ]: (730, 81)

[ ]: original.head

0	1	60	RL		65.0	8450	Pave	NaN	Reg	
1	2	20	RL		80.0	9600	Pave	NaN	Reg	
2	3	60	RL		68.0	11250	Pave	NaN	IR1	
3	4	70	RL		60.0	9550	Pave	NaN	IR1	
4	5	60	RL		84.0	14260	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL		62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL		85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL		66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL		68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL		75.0	9937	Pave	NaN	Reg	
...	...	...	...	...	...	...	...	...	...	
0	Lvl	AllPub	...	0	NaN	NaN		NaN	0	
1	Lvl	AllPub	...	0	NaN	NaN		NaN	0	
2	Lvl	AllPub	...	0	NaN	NaN		NaN	0	
3	Lvl	AllPub	...	0	NaN	NaN		NaN	0	
4	Lvl	AllPub	...	0	NaN	NaN		NaN	0	
...	...	...	...	...	...	...	...	...	...	

```

1455      Lvl    AllPub ...      0    NaN    NaN      NaN      0
1456      Lvl    AllPub ...      0    NaN    MnPrv      NaN      0
1457      Lvl    AllPub ...      0    NaN    GdPrv      Shed    2500
1458      Lvl    AllPub ...      0    NaN    NaN      NaN      0
1459      Lvl    AllPub ...      0    NaN    NaN      NaN      0

```

	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	2	2008	WD	Normal	208500
1	5	2007	WD	Normal	181500
2	9	2008	WD	Normal	223500
3	2	2006	WD	Abnorml	140000
4	12	2008	WD	Normal	250000
...	...	...	...	...	...
1455	8	2007	WD	Normal	175000
1456	2	2010	WD	Normal	210000
1457	5	2010	WD	Normal	266500
1458	4	2010	WD	Normal	142125
1459	6	2008	WD	Normal	147500

[1460 rows x 81 columns]>

[ ]: original.info()

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Id               1460 non-null   int64  
 1   MSSubClass        1460 non-null   int64  
 2   MSZoning          1460 non-null   object  
 3   LotFrontage       1201 non-null   float64 
 4   LotArea           1460 non-null   int64  
 5   Street            1460 non-null   object  
 6   Alley              91 non-null    object  
 7   LotShape           1460 non-null   object  
 8   LandContour        1460 non-null   object  
 9   Utilities          1460 non-null   object  
 10  LotConfig          1460 non-null   object  
 11  LandSlope          1460 non-null   object  
 12  Neighborhood       1460 non-null   object  
 13  Condition1         1460 non-null   object  
 14  Condition2         1460 non-null   object  
 15  BldgType           1460 non-null   object  
 16  HouseStyle         1460 non-null   object  
 17  OverallQual        1460 non-null   int64  
 18  OverallCond        1460 non-null   int64  

```

19	YearBuilt	1460	non-null	int64
20	YearRemodAdd	1460	non-null	int64
21	RoofStyle	1460	non-null	object
22	RoofMatl	1460	non-null	object
23	Exterior1st	1460	non-null	object
24	Exterior2nd	1460	non-null	object
25	MasVnrType	1452	non-null	object
26	MasVnrArea	1452	non-null	float64
27	ExterQual	1460	non-null	object
28	ExterCond	1460	non-null	object
29	Foundation	1460	non-null	object
30	BsmtQual	1423	non-null	object
31	BsmtCond	1423	non-null	object
32	BsmtExposure	1422	non-null	object
33	BsmtFinType1	1423	non-null	object
34	BsmtFinSF1	1460	non-null	int64
35	BsmtFinType2	1422	non-null	object
36	BsmtFinSF2	1460	non-null	int64
37	BsmtUnfSF	1460	non-null	int64
38	TotalBsmtSF	1460	non-null	int64
39	Heating	1460	non-null	object
40	HeatingQC	1460	non-null	object
41	CentralAir	1460	non-null	object
42	Electrical	1459	non-null	object
43	1stFlrSF	1460	non-null	int64
44	2ndFlrSF	1460	non-null	int64
45	LowQualFinSF	1460	non-null	int64
46	GrLivArea	1460	non-null	int64
47	BsmtFullBath	1460	non-null	int64
48	BsmtHalfBath	1460	non-null	int64
49	FullBath	1460	non-null	int64
50	HalfBath	1460	non-null	int64
51	BedroomAbvGr	1460	non-null	int64
52	KitchenAbvGr	1460	non-null	int64
53	KitchenQual	1460	non-null	object
54	TotRmsAbvGrd	1460	non-null	int64
55	Functional	1460	non-null	object
56	Fireplaces	1460	non-null	int64
57	FireplaceQu	770	non-null	object
58	GarageType	1379	non-null	object
59	GarageYrBlt	1379	non-null	float64
60	GarageFinish	1379	non-null	object
61	GarageCars	1460	non-null	int64
62	GarageArea	1460	non-null	int64
63	GarageQual	1379	non-null	object
64	GarageCond	1379	non-null	object
65	PavedDrive	1460	non-null	object
66	WoodDeckSF	1460	non-null	int64

```

67 OpenPorchSF    1460 non-null    int64
68 EnclosedPorch  1460 non-null    int64
69 3SsnPorch      1460 non-null    int64
70 ScreenPorch    1460 non-null    int64
71 PoolArea       1460 non-null    int64
72 PoolQC         7 non-null      object
73 Fence          281 non-null    object
74 MiscFeature    54 non-null     object
75 MiscVal        1460 non-null    int64
76 MoSold         1460 non-null    int64
77 YrSold         1460 non-null    int64
78 SaleType       1460 non-null    object
79 SaleCondition   1460 non-null    object
80 SalePrice      1460 non-null    int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB

```

[ ]: original.describe()

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	\
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	
std	421.610009	42.300571	24.284752	9981.264932	1.382997	
min	1.000000	20.000000	21.000000	1300.000000	1.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	
	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	\
count	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...
mean	5.575342	1971.267808	1984.865753	103.685262	443.639726	...
std	1.112799	30.202904	20.645407	181.066207	456.098091	...
min	1.000000	1872.000000	1950.000000	0.000000	0.000000	...
25%	5.000000	1954.000000	1967.000000	0.000000	0.000000	...
50%	5.000000	1973.000000	1994.000000	0.000000	383.500000	...
75%	6.000000	2000.000000	2004.000000	166.000000	712.250000	...
max	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...
	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	\
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	
mean	94.244521	46.660274	21.954110	3.409589	15.060959	
std	125.338794	66.256028	61.119149	29.317331	55.757415	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	25.000000	0.000000	0.000000	0.000000	
75%	168.000000	68.000000	0.000000	0.000000	0.000000	

```
max      857.000000  547.000000  552.000000  508.000000  480.000000  
PoolArea      MiscVal      MoSold      YrSold      SalePrice  
count   1460.000000  1460.000000  1460.000000  1460.000000  1460.000000  
mean     2.758904    43.489041   6.321918   2007.815753  180921.195890  
std      40.177307   496.123024   2.703626   1.328095   79442.502883  
min      0.000000    0.000000   1.000000   2006.000000  34900.000000  
25%     0.000000    0.000000   5.000000   2007.000000  129975.000000  
50%     0.000000    0.000000   6.000000   2008.000000  163000.000000  
75%     0.000000    0.000000   8.000000   2009.000000  214000.000000  
max     738.000000  15500.000000  12.000000  2010.000000  755000.000000
```

[8 rows x 38 columns]

```
[ ]: original.duplicated().sum()
```

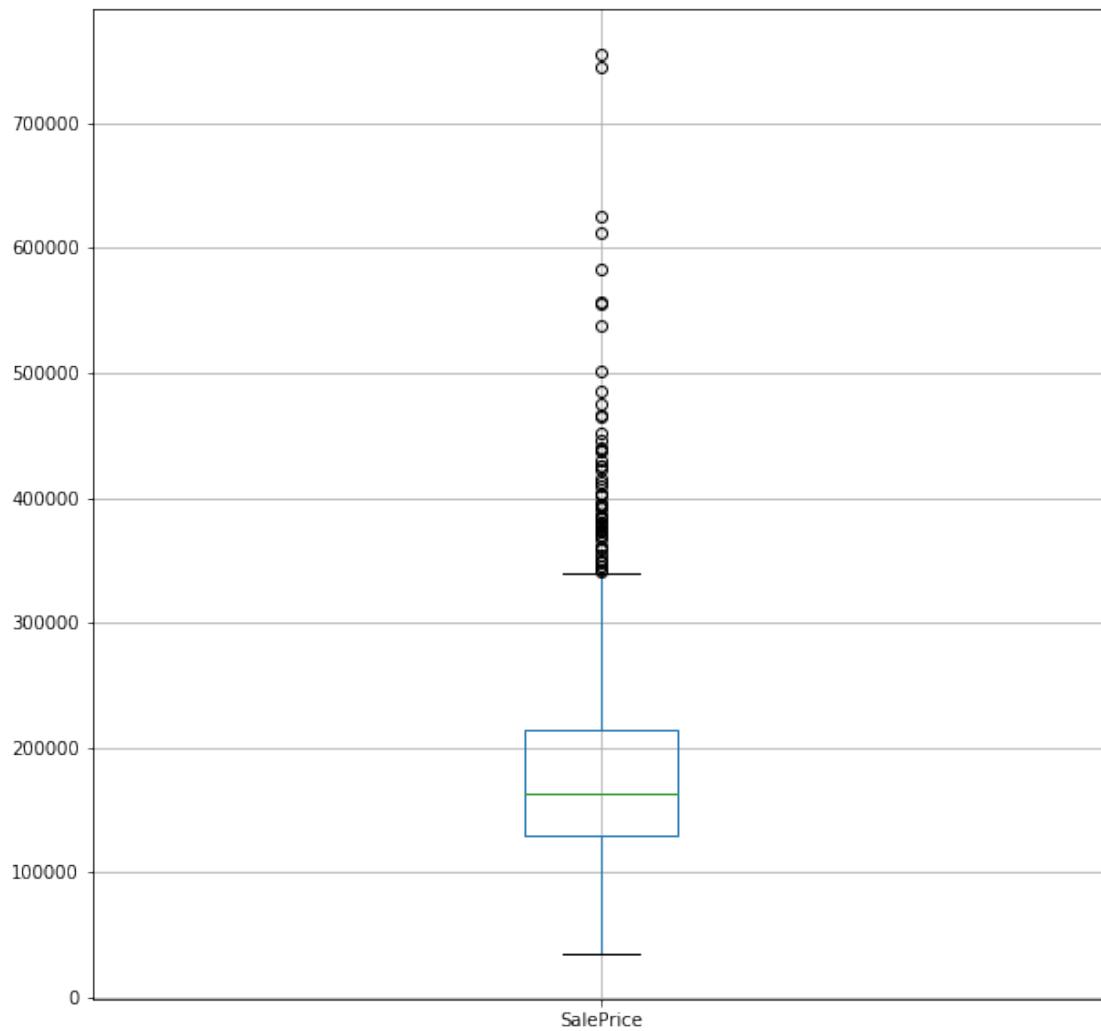
```
[ ]: 0
```

```
[ ]: original.isnull().sum()
```

```
[ ]: Id                  0  
MSSubClass            0  
MSZoning              0  
LotFrontage           259  
LotArea                0  
...  
MoSold                 0  
YrSold                 0  
SaleType                0  
SaleCondition            0  
SalePrice                0  
Length: 81, dtype: int64
```

```
[ ]: original[['SalePrice']].boxplot(figsize=(10,10))
```

```
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d3344ac40>
```



```
[ ]: original.corr()
```

```
[ ]: 
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	\
Id	1.000000	0.011156	-0.010601	-0.033226	-0.028365	
MSSubClass	0.011156	1.000000	-0.386347	-0.139781	0.032628	
LotFrontage	-0.010601	-0.386347	1.000000	0.426095	0.251646	
LotArea	-0.033226	-0.139781	0.426095	1.000000	0.105806	
OverallQual	-0.028365	0.032628	0.251646	0.105806	1.000000	
OverallCond	0.012609	-0.059316	-0.059213	-0.005636	-0.091932	
YearBuilt	-0.012713	0.027850	0.123349	0.014228	0.572323	
YearRemodAdd	-0.021998	0.040581	0.088866	0.013788	0.550684	
MasVnrArea	-0.050298	0.022936	0.193458	0.104160	0.411876	
BsmtFinSF1	-0.005024	-0.069836	0.233633	0.214103	0.239666	
BsmtFinSF2	-0.005968	-0.065649	0.049900	0.111170	-0.059119	
BsmtUnfSF	-0.007940	-0.140759	0.132644	-0.002618	0.308159	

	TotalBsmtSF	-0.015415	-0.238518	0.392075	0.260833	0.537808	\
1stFlrSF	0.010496	-0.251758	0.457181	0.299475	0.476224		
2ndFlrSF	0.005590	0.307886	0.080177	0.050986	0.295493		
LowQualFinSF	-0.044230	0.046474	0.038469	0.004779	-0.030429		
GrLivArea	0.008273	0.074853	0.402797	0.263116	0.593007		
BsmtFullBath	0.002289	0.003491	0.100949	0.158155	0.111098		
BsmtHalfBath	-0.020155	-0.002333	-0.007234	0.048046	-0.040150		
FullBath	0.005587	0.131608	0.198769	0.126031	0.550600		
HalfBath	0.006784	0.177354	0.053532	0.014259	0.273458		
BedroomAbvGr	0.037719	-0.023438	0.263170	0.119690	0.101676		
KitchenAbvGr	0.002951	0.281721	-0.006069	-0.017784	-0.183882		
TotRmsAbvGrd	0.027239	0.040380	0.352096	0.190015	0.427452		
Fireplaces	-0.019772	-0.045569	0.266639	0.271364	0.396765		
GarageYrBlt	0.000072	0.085072	0.070250	-0.024947	0.547766		
GarageCars	0.016570	-0.040110	0.285691	0.154871	0.600671		
GarageArea	0.017634	-0.098672	0.344997	0.180403	0.562022		
WoodDeckSF	-0.029643	-0.012579	0.088521	0.171698	0.238923		
OpenPorchSF	-0.000477	-0.006100	0.151972	0.084774	0.308819		
EnclosedPorch	0.002889	-0.012037	0.010700	-0.018340	-0.113937		
3SsnPorch	-0.046635	-0.043825	0.070029	0.020423	0.030371		
ScreenPorch	0.001330	-0.026030	0.041383	0.043160	0.064886		
PoolArea	0.057044	0.008283	0.206167	0.077672	0.065166		
MiscVal	-0.006242	-0.007683	0.003368	0.038068	-0.031406		
MoSold	0.021172	-0.013585	0.011200	0.001205	0.070815		
YrSold	0.000712	-0.021407	0.007450	-0.014261	-0.027347		
SalePrice	-0.021917	-0.084284	0.351799	0.263843	0.790982		
	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	\
Id	0.012609	-0.012713	-0.021998	-0.050298	-0.005024		
MSSubClass	-0.059316	0.027850	0.040581	0.022936	-0.069836		
LotFrontage	-0.059213	0.123349	0.088866	0.193458	0.233633		
LotArea	-0.005636	0.014228	0.013788	0.104160	0.214103		
OverallQual	-0.091932	0.572323	0.550684	0.411876	0.239666		
OverallCond	1.000000	-0.375983	0.073741	-0.128101	-0.046231		
YearBuilt	-0.375983	1.000000	0.592855	0.315707	0.249503		
YearRemodAdd	0.073741	0.592855	1.000000	0.179618	0.128451		
MasVnrArea	-0.128101	0.315707	0.179618	1.000000	0.264736		
BsmtFinSF1	-0.046231	0.249503	0.128451	0.264736	1.000000		
BsmtFinSF2	0.040229	-0.049107	-0.067759	-0.072319	-0.050117		
BsmtUnfSF	-0.136841	0.149040	0.181133	0.114442	-0.495251		
TotalBsmtSF	-0.171098	0.391452	0.291066	0.363936	0.522396		
1stFlrSF	-0.144203	0.281986	0.240379	0.344501	0.445863		
2ndFlrSF	0.028942	0.010308	0.140024	0.174561	-0.137079		
LowQualFinSF	0.025494	-0.183784	-0.062419	-0.069071	-0.064503		
GrLivArea	-0.079686	0.199010	0.287389	0.390857	0.208171		
BsmtFullBath	-0.054942	0.187599	0.119470	0.085310	0.649212		
BsmtHalfBath	0.117821	-0.038162	-0.012337	0.026673	0.067418		

FullBath	-0.194149	0.468271	0.439046	0.276833	0.058543
HalfBath	-0.060769	0.242656	0.183331	0.201444	0.004262
BedroomAbvGr	0.012980	-0.070651	-0.040581	0.102821	-0.107355
KitchenAbvGr	-0.087001	-0.174800	-0.149598	-0.037610	-0.081007
TotRmsAbvGrd	-0.057583	0.095589	0.191740	0.280682	0.044316
Fireplaces	-0.023820	0.147716	0.112581	0.249070	0.260011
GarageYrBlt	-0.324297	0.825667	0.642277	0.252691	0.153484
GarageCars	-0.185758	0.537850	0.420622	0.364204	0.224054
GarageArea	-0.151521	0.478954	0.371600	0.373066	0.296970
WoodDeckSF	-0.003334	0.224880	0.205726	0.159718	0.204306
OpenPorchSF	-0.032589	0.188686	0.226298	0.125703	0.111761
EnclosedPorch	0.070356	-0.387268	-0.193919	-0.110204	-0.102303
3SsnPorch	0.025504	0.031355	0.045286	0.018796	0.026451
ScreenPorch	0.054811	-0.050364	-0.038740	0.061466	0.062021
PoolArea	-0.001985	0.004950	0.005829	0.011723	0.140491
MiscVal	0.068777	-0.034383	-0.010286	-0.029815	0.003571
MoSold	-0.003511	0.012398	0.021490	-0.005965	-0.015727
YrSold	0.043950	-0.013618	0.035743	-0.008201	0.014359
SalePrice	-0.077856	0.522897	0.507101	0.477493	0.386420

...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	\
Id	...	-0.029643	-0.000477	0.002889	-0.046635
MSSubClass	...	-0.012579	-0.006100	-0.012037	-0.043825
LotFrontage	...	0.088521	0.151972	0.010700	0.070029
LotArea	...	0.171698	0.084774	-0.018340	0.020423
OverallQual	...	0.238923	0.308819	-0.113937	0.030371
OverallCond	...	-0.003334	-0.032589	0.070356	0.025504
YearBuilt	...	0.224880	0.188686	-0.387268	0.031355
YearRemodAdd	...	0.205726	0.226298	-0.193919	0.045286
MasVnrArea	...	0.159718	0.125703	-0.110204	0.018796
BsmtFinSF1	...	0.204306	0.111761	-0.102303	0.026451
BsmtFinSF2	...	0.067898	0.003093	0.036543	-0.029993
BsmtUnfSF	...	-0.005316	0.129005	-0.002538	0.020764
TotalBsmtSF	...	0.232019	0.247264	-0.095478	0.037384
1stFlrSF	...	0.235459	0.211671	-0.065292	0.056104
2ndFlrSF	...	0.092165	0.208026	0.061989	-0.024358
LowQualFinSF	...	-0.025444	0.018251	0.061081	-0.004296
GrLivArea	...	0.247433	0.330224	0.009113	0.020643
BsmtFullBath	...	0.175315	0.067341	-0.049911	-0.000106
BsmtHalfBath	...	0.040161	-0.025324	-0.008555	0.035114
FullBath	...	0.187703	0.259977	-0.115093	0.035353
HalfBath	...	0.108080	0.199740	-0.095317	-0.004972
BedroomAbvGr	...	0.046854	0.093810	0.041570	-0.024478
KitchenAbvGr	...	-0.090130	-0.070091	0.037312	-0.024600
TotRmsAbvGrd	...	0.165984	0.234192	0.004151	-0.006683
Fireplaces	...	0.200019	0.169405	-0.024822	0.011257
GarageYrBlt	...	0.224577	0.228425	-0.297003	0.023544

GarageCars	...	0.226342	0.213569	-0.151434	0.035765		
GarageArea	...	0.224666	0.241435	-0.121777	0.035087		
WoodDeckSF	...	1.000000	0.058661	-0.125989	-0.032771		
OpenPorchSF	...	0.058661	1.000000	-0.093079	-0.005842		
EnclosedPorch	...	-0.125989	-0.093079	1.000000	-0.037305		
3SsnPorch	...	-0.032771	-0.005842	-0.037305	1.000000		
ScreenPorch	...	-0.074181	0.074304	-0.082864	-0.031436		
PoolArea	...	0.073378	0.060762	0.054203	-0.007992		
MiscVal	...	-0.009551	-0.018584	0.018361	0.000354		
MoSold	...	0.021011	0.071255	-0.028887	0.029474		
YrSold	...	0.022270	-0.057619	-0.009916	0.018645		
SalePrice	...	0.324413	0.315856	-0.128578	0.044584		
		ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SalePrice
Id		0.001330	0.057044	-0.006242	0.021172	0.000712	-0.021917
MSSubClass		-0.026030	0.008283	-0.007683	-0.013585	-0.021407	-0.084284
LotFrontage		0.041383	0.206167	0.003368	0.011200	0.007450	0.351799
LotArea		0.043160	0.077672	0.038068	0.001205	-0.014261	0.263843
OverallQual		0.064886	0.065166	-0.031406	0.070815	-0.027347	0.790982
OverallCond		0.054811	-0.001985	0.068777	-0.003511	0.043950	-0.077856
YearBuilt		-0.050364	0.004950	-0.034383	0.012398	-0.013618	0.522897
YearRemodAdd		-0.038740	0.005829	-0.010286	0.021490	0.035743	0.507101
MasVnrArea		0.061466	0.011723	-0.029815	-0.005965	-0.008201	0.477493
BsmtFinSF1		0.062021	0.140491	0.003571	-0.015727	0.014359	0.386420
BsmtFinSF2		0.088871	0.041709	0.004940	-0.015211	0.031706	-0.011378
BsmtUnfSF		-0.012579	-0.035092	-0.023837	0.034888	-0.041258	0.214479
TotalBsmtSF		0.084489	0.126053	-0.018479	0.013196	-0.014969	0.613581
1stFlrSF		0.088758	0.131525	-0.021096	0.031372	-0.013604	0.605852
2ndFlrSF		0.040606	0.081487	0.016197	0.035164	-0.028700	0.319334
LowQualFinSF		0.026799	0.062157	-0.003793	-0.022174	-0.028921	-0.025606
GrLivArea		0.101510	0.170205	-0.002416	0.050240	-0.036526	0.708624
BsmtFullBath		0.023148	0.067616	-0.023047	-0.025361	0.067049	0.227122
BsmtHalfBath		0.032121	0.020025	-0.007367	0.032873	-0.046524	-0.016844
FullBath		-0.008106	0.049604	-0.014290	0.055872	-0.019669	0.560664
HalfBath		0.072426	0.022381	0.001290	-0.009050	-0.010269	0.284108
BedroomAbvGr		0.044300	0.070703	0.007767	0.046544	-0.036014	0.168213
KitchenAbvGr		-0.051613	-0.014525	0.062341	0.026589	0.031687	-0.135907
TotRmsAbvGrd		0.059383	0.083757	0.024763	0.036907	-0.034516	0.533723
Fireplaces		0.184530	0.095074	0.001409	0.046357	-0.024096	0.466929
GarageYrBlt		-0.075418	-0.014501	-0.032417	0.005337	-0.001014	0.486362
GarageCars		0.050494	0.020934	-0.043080	0.040522	-0.039117	0.640409
GarageArea		0.051412	0.061047	-0.027400	0.027974	-0.027378	0.623431
WoodDeckSF		-0.074181	0.073378	-0.009551	0.021011	0.022270	0.324413
OpenPorchSF		0.074304	0.060762	-0.018584	0.071255	-0.057619	0.315856
EnclosedPorch		-0.082864	0.054203	0.018361	-0.028887	-0.009916	-0.128578
3SsnPorch		-0.031436	-0.007992	0.000354	0.029474	0.018645	0.044584
ScreenPorch		1.000000	0.051307	0.031946	0.023217	0.010694	0.111447

```

PoolArea          0.051307  1.000000  0.029669 -0.033737 -0.059689  0.092404
MiscVal          0.031946  0.029669  1.000000 -0.006495  0.004906 -0.021190
MoSold           0.023217 -0.033737 -0.006495  1.000000 -0.145721  0.046432
YrSold           0.010694 -0.059689  0.004906 -0.145721  1.000000 -0.028923
SalePrice         0.111447  0.092404 -0.021190  0.046432 -0.028923  1.000000

```

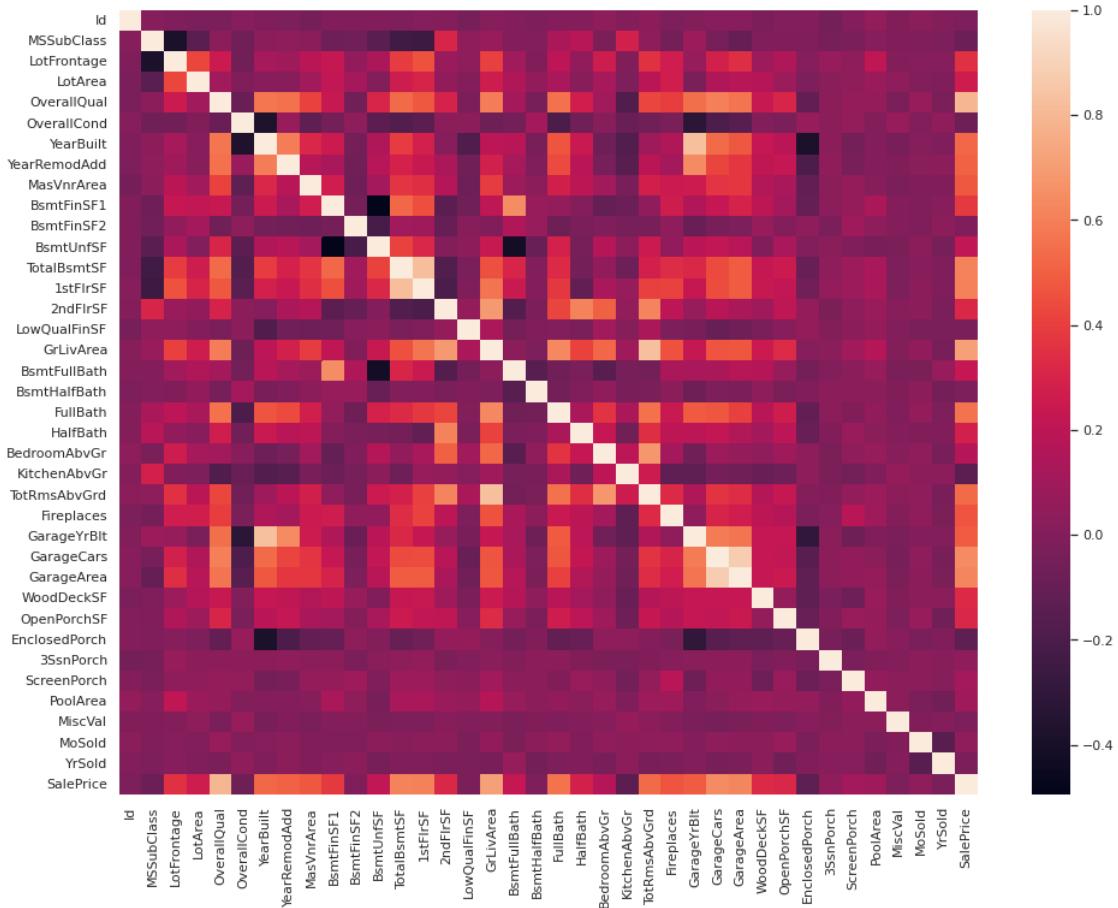
[38 rows x 38 columns]

```

[ ]: import seaborn as sns
sns.set(rc={"figure.figsize":(16, 12)}) # https://www.statology.org/
    ↪seaborn-figure-size/
sns.heatmap(original.corr(), linewidths=0)

```

[ ]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7f1d3311dd60>



[ ]: test.shape

[ ]: (730, 81)

```
[ ]: x_train = train.drop(["SalePrice" , "Id"], axis=1)
y_train = train["SalePrice"]

[ ]: x_test = test.drop(["SalePrice" , "Id"], axis=1)
y_test = test["SalePrice"]

[ ]: enc = TargetEncoder()

/usr/local/lib/python3.8/dist-packages/category_encoders/target_encoder.py:122:
FutureWarning: Default parameter min_samples_leaf will change in version 2.6. See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
    warnings.warn("Default parameter min_samples_leaf will change in version 2.6.")
/usr/local/lib/python3.8/dist-packages/category_encoders/target_encoder.py:127:
FutureWarning: Default parameter smoothing will change in version 2.6. See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
    warnings.warn("Default parameter smoothing will change in version 2.6.")

[ ]: x_train_numeric = enc.fit_transform(x_train, y_train)
x_test_numeric = enc.transform(x_test)

[ ]: x_train_numeric.shape

[ ]: (730, 79)

[ ]: train_temp = pd.concat([x_train_numeric, y_train], axis=1)
test_temp = pd.concat([x_test_numeric,y_test], axis=1)

[ ]: train_temp = train_temp.dropna()
test_temp = test_temp.dropna()
test_temp.isna().sum()

[ ]: MSSubClass      0
MSZoning          0
LotFrontage        0
LotArea            0
Street             0
...
MoSold             0
YrSold             0
SaleType            0
SaleCondition       0
SalePrice           0
Length: 80, dtype: int64

[ ]: x_train_numeric = train_temp.drop(["SalePrice"], axis=1)
y_train = train_temp["SalePrice"]
x_test_numeric = test_temp.drop(["SalePrice"], axis=1)
```

```

y_test = test_temp["SalePrice"]
pd.set_option('display.max_rows', 124)
# x_train_numeric.dtypes
x_train_numeric.isna().sum()

```

```

[ ]: MSSubClass      0
MSZoning          0
LotFrontage        0
LotArea            0
Street             0
Alley              0
LotShape            0
LandContour        0
Utilities           0
LotConfig           0
LandSlope           0
Neighborhood        0
Condition1         0
Condition2         0
BldgType            0
HouseStyle          0
OverallQual         0
OverallCond         0
YearBuilt           0
YearRemodAdd        0
RoofStyle           0
RoofMatl            0
Exterior1st         0
Exterior2nd         0
MasVnrType          0
MasVnrArea          0
ExterQual           0
ExterCond           0
Foundation          0
BsmtQual            0
BsmtCond            0
BsmtExposure        0
BsmtFinType1        0
BsmtFinSF1          0
BsmtFinType2        0
BsmtFinSF2          0
BsmtUnfSF           0
TotalBsmtSF         0
Heating              0
HeatingQC            0
CentralAir           0
Electrical           0

```

```
1stFlrSF      0
2ndFlrSF      0
LowQualFinSF  0
GrLivArea     0
BsmtFullBath  0
BsmtHalfBath  0
FullBath       0
HalfBath       0
BedroomAbvGr  0
KitchenAbvGr  0
KitchenQual    0
TotRmsAbvGrd  0
Functional     0
Fireplaces     0
FireplaceQu   0
GarageType     0
GarageYrBlt   0
GarageFinish   0
GarageCars     0
GarageArea     0
GarageQual     0
GarageCond     0
PavedDrive     0
WoodDeckSF    0
OpenPorchSF   0
EnclosedPorch  0
3SsnPorch     0
ScreenPorch   0
PoolArea       0
PoolQC         0
Fence          0
MiscFeature    0
MiscVal        0
MoSold         0
YrSold         0
SaleType        0
SaleCondition  0
dtype: int64
```

```
[ ]: def mse_rmse(test,pred):
    mse = mean_squared_error(test, pred)
    rmse = math.sqrt(mse)
    return mse, rmse
```

```
#Models
```

```
Random Forest
```

```
[ ]: from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error

rand_forest = RandomForestRegressor()
rand_forest.fit(x_train_numeric, y_train)
```

```
[ ]: RandomForestRegressor()
```

```
[ ]: score = rand_forest.score(x_train_numeric, y_train)
print(score)
```

0.9746150041282055

```
[ ]: y_pred = rand_forest.predict(x_test_numeric)
```

```
[ ]: base_random_mse, base_random_rmse = mse_rmse(y_test, y_pred)
print(base_random_mse)
print(base_random_rmse)
```

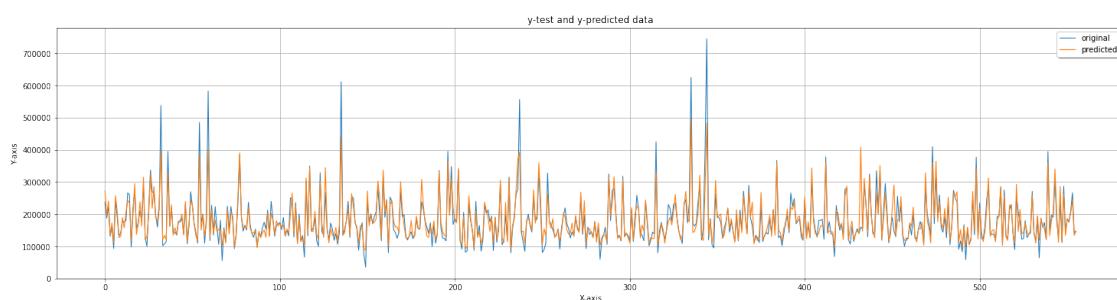
998656023.8840587

31601.519328729413

```
[ ]: score = rand_forest.score(x_test_numeric, y_test)
print(score)
```

0.8428300340322101

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, linewidth=1, label="original")
plt.plot(x_ax, y_pred, linewidth=1.1, label="predicted")
plt.title("y-test and y-predicted data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



## AdaBoost

```
[ ]: from sklearn.ensemble import AdaBoostRegressor  
adamodel = AdaBoostRegressor()  
adamodel.fit(x_train_numeric, y_train)  
adamodel.score(x_test_numeric,y_test)
```

```
[ ]: 0.7894228912926889
```

```
[ ]: y_pred_ada = adamodel.predict(x_test_numeric)
```

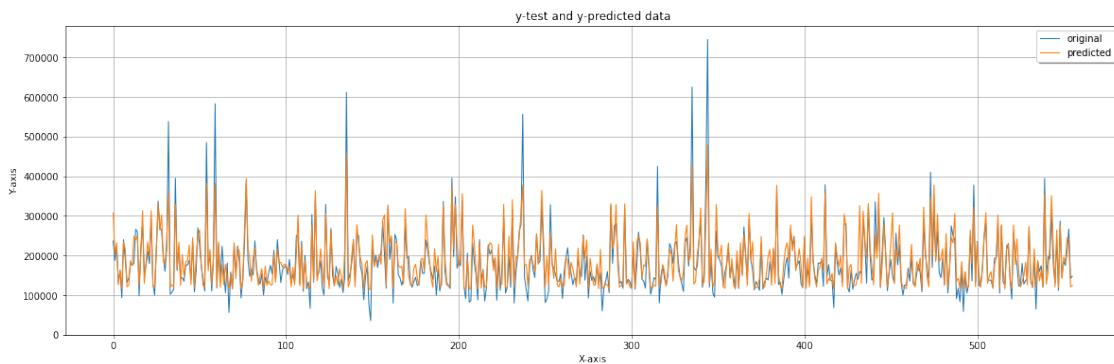
```
[ ]: base_ada_mse, base_ada_rmse = mse_rmse(y_test,y_pred_ada)
```

```
[ ]: print(base_ada_mse)  
print(base_ada_rmse)
```

1338004349.6716273

36578.741772669375

```
[ ]: plt.figure(figsize=(20,6))  
x_ax = range(len(y_test))  
plt.plot(x_ax, y_test, linewidth=1, label="original")  
plt.plot(x_ax, y_pred_ada, linewidth=1.1, label="predicted")  
plt.title("y-test and y-predicted data")  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')  
plt.legend(loc='best', fancybox=True, shadow=True)  
plt.grid(True)  
plt.show()
```



## Bagging Meta-Estimator

```
[ ]: from sklearn.ensemble import BaggingRegressor
from sklearn import tree
bagmeta = BaggingRegressor(tree.DecisionTreeRegressor(random_state=1))
bagmeta.fit(x_train_numeric, y_train)
bagmeta.score(x_test_numeric,y_test)
```

[ ]: 0.8342795129880863

```
[ ]: y_pred_bagmeta = bagmeta.predict(x_test_numeric)
```

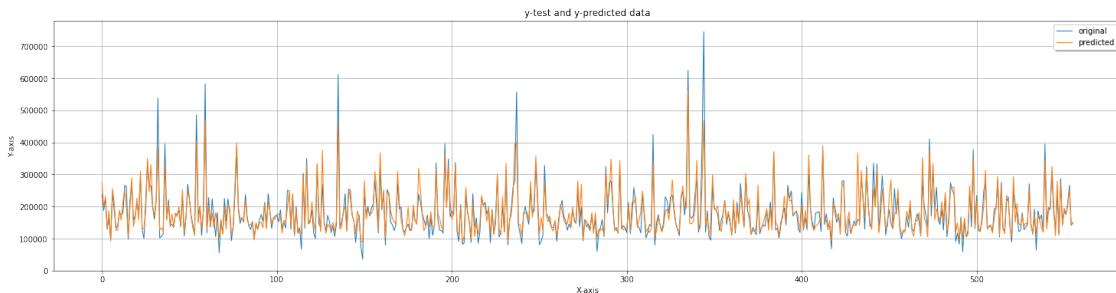
```
[ ]: base_bag_mse, base_bag_rmse = mse_rmse(y_test,y_pred_bagmeta )
```

```
[ ]: print(base_bag_mse)
print(base_bag_rmse)
```

1052985929.0633453

32449.744668692623

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, linewidth=1, label="original")
plt.plot(x_ax, y_pred_bagmeta, linewidth=1.1, label="predicted")
plt.title("y-test and y-predicted data")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



## XGBoost

```
[ ]: import xgboost as xgb
xgbmodel=xgb.XGBRegressor()
xgbmodel.fit(x_train_numeric, y_train)
xgbmodel.score(x_test_numeric,y_test)
```

```
[21:05:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear  
is now deprecated in favor of reg:squarederror.
```

```
[ ]: 0.867411990928935
```

```
[ ]: y_pred_xgb = xgbmodel.predict(x_test_numeric)
```

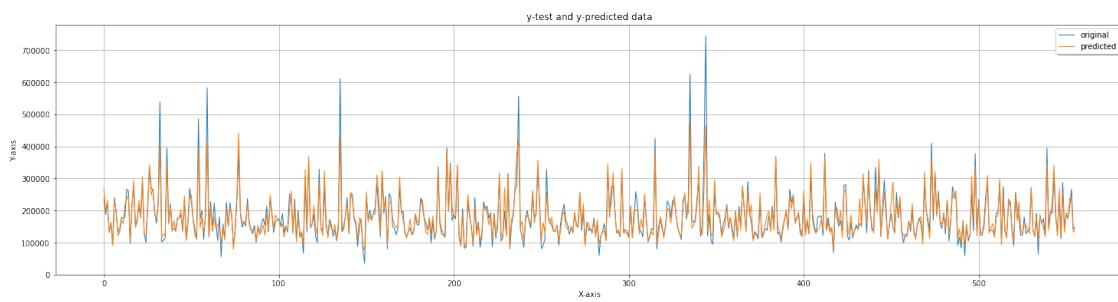
```
[ ]: base_xgb_mse, base_xgb_rmse = mse_rmse(y_test,y_pred_xgb)
```

```
[ ]: print(base_xgb_mse)  
print(base_xgb_rmse)
```

```
842462573.1658505
```

```
29025.205824694
```

```
[ ]: plt.figure(figsize=(25,6))  
x_ax = range(len(y_test))  
plt.plot(x_ax, y_test, linewidth=1, label="original")  
plt.plot(x_ax, y_pred_xgb, linewidth=1.1, label="predicted")  
plt.title("y-test and y-predicted data")  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')  
plt.legend(loc='best', fancybox=True, shadow=True)  
plt.grid(True)  
plt.show()
```



## 1 PCA Models

```
[ ]: pca_enc = TargetEncoder() #Create an encoder for pca  
  
pca_x_vars = original.drop(["SalePrice"], axis = 1).copy()  
pca_y = original["SalePrice"].copy()  
pca_x_train_numeric = pca_enc.fit_transform(pca_x_vars, pca_y) #fit new  
→dimensions for pca
```

```

pca_x_vars = pca_enc.transform(pca_x_vars)

/usr/local/lib/python3.8/dist-packages/category_encoders/target_encoder.py:122:
FutureWarning: Default parameter min_samples_leaf will change in version 2.6. See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
    warnings.warn("Default parameter min_samples_leaf will change in version 2.6.")
/usr/local/lib/python3.8/dist-packages/category_encoders/target_encoder.py:127:
FutureWarning: Default parameter smoothing will change in version 2.6. See
https://github.com/scikit-learn-contrib/category_encoders/issues/327
    warnings.warn("Default parameter smoothing will change in version 2.6.")

[ ]: tempConcat = pd.concat([pca_x_vars,pca_y],axis = 1)#concat both variables for  

↳dropping NaN values  

tempConcat = tempConcat.dropna()#drop NaN values

[ ]: final_pca_x = tempConcat.drop(["Id","SalePrice"], axis =1)#drop columns we dont  

↳need for x vars  

final_pca_y = tempConcat["SalePrice"]  

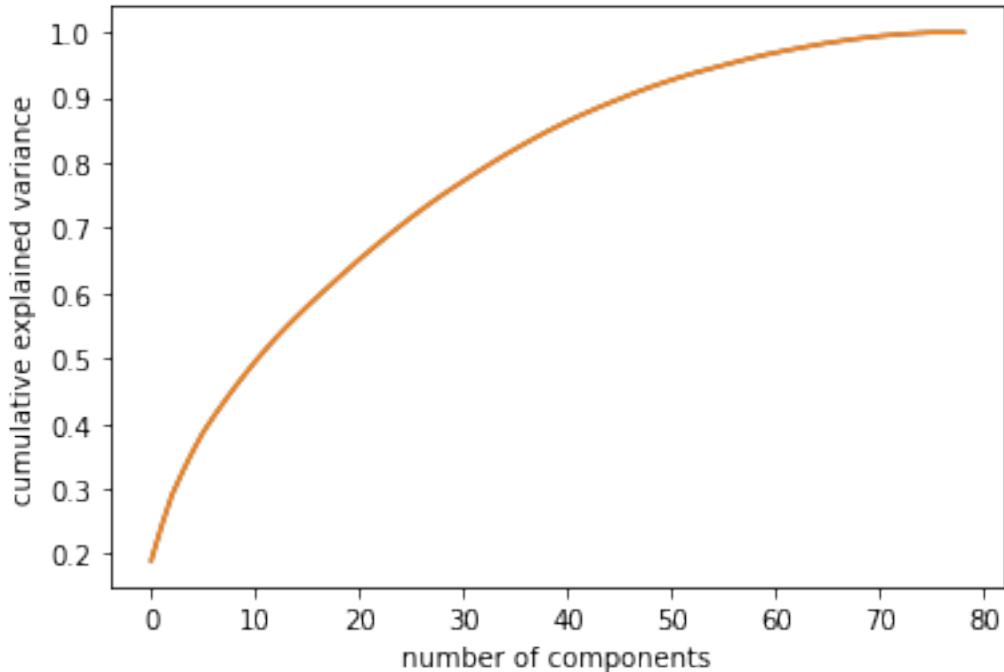
test_pca_x = tempConcat.drop(["Id","SalePrice"], axis =1)

[ ]: X_std = StandardScaler().fit_transform(test_pca_x)#standardize data  
  

pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

[ ]: Text(0, 0.5, 'cumulative explained variance')

```



```
[ ]: pca_explained = PCA(0.80).fit(X_std)
pca_explained.n_components_
```

[ ]: 34

The first 34 features explain about 80% of our variance.

```
[ ]: final_pca_x = final_pca_x.iloc[:, :34] #get the first 34 columns from the
    ↴datafram and its rows because PCA
```

```
[ ]: pca_features = final_pca_x.columns.tolist()
print(pca_features)
```

```
['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'Alley',
'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope',
'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle',
'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle',
'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'MasVnrArea',
'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
'BsmtFinType1', 'BsmtFinSF1']
```

Features included in PCA

```
[ ]: #SPLIT THE DATASET BEFORE TESTING ON RANDOMFOREST
from sklearn.model_selection import train_test_split
```

```
pca_x_train, pca_x_test, pca_y_train, pca_y_test = train_test_split(final_pca_x, final_pca_y, test_size=0.30, random_state=42)
#70:30 split for train and test -> train:test
```

## RandomForest Regression With PCA

```
[ ]: #Use randomforest regression on pca
```

```
rand_forest_pca = RandomForestRegressor() #create random forest test
rand_forest_pca.fit(pca_x_train, pca_y_train)
```

```
[ ]: RandomForestRegressor()
```

```
[ ]: pca_y_pred = rand_forest_pca.predict(pca_x_test) #randomforest test
```

```
[ ]: pca_score = rand_forest_pca.score(pca_x_test, pca_y_test)
print(pca_score)
```

0.8092555332736899

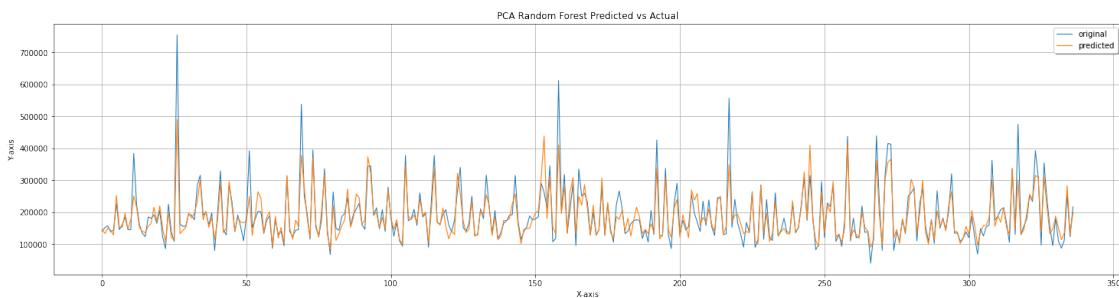
```
[ ]: random_forest_mse, random_forest_rmse = mse_rmse(pca_y_test, pca_y_pred)
```

```
[ ]: print(random_forest_mse)
print(random_forest_rmse)
```

1524328130.4066467

39042.645023187746

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(pca_y_test))
plt.plot(x_ax, pca_y_test, linewidth=1, label="original")
plt.plot(x_ax, pca_y_pred, linewidth=1.1, label="predicted")
plt.title("PCA Random Forest Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



## AdaBoost with PCA

```
[ ]: PCA_adamodel = AdaBoostRegressor()  
PCA_adamodel.fit(pca_x_train, pca_y_train)  
PCA_adamodel.score(pca_x_test,pca_y_test)
```

```
[ ]: 0.7816007831843239
```

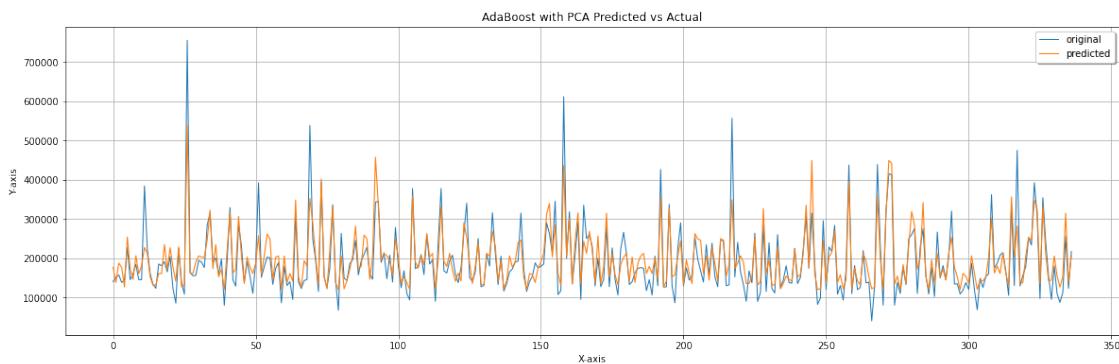
```
[ ]: PCA_y_pred_ada = PCA_adamodel.predict(pca_x_test)
```

```
[ ]: ada_mse, ada_rmse = mse_rmse(pca_y_test, PCA_y_pred_ada)
```

```
[ ]: print(ada_mse)  
print(ada_rmse)
```

```
1745330155.9126995  
41777.148728852946
```

```
[ ]: plt.figure(figsize=(20,6))  
x_ax = range(len(pca_y_test))  
plt.plot(x_ax,pca_y_test, linewidth=1, label="original")  
plt.plot(x_ax, PCA_y_pred_ada, linewidth=1.1, label="predicted")  
plt.title("AdaBoost with PCA Predicted vs Actual")  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')  
plt.legend(loc='best', fancybox=True, shadow=True)  
plt.grid(True)  
plt.show()
```



## Bagging Meta-Estimator with PCA

```
[ ]: pca_bagmeta = BaggingRegressor(tree.DecisionTreeRegressor(random_state=1))
pca_bagmeta.fit(pca_x_train, pca_y_train)
pca_bagmeta.score(pca_x_test,pca_y_test)
```

```
[ ]: 0.7921747283566334
```

```
[ ]: pca_y_pred_bagmeta = pca_bagmeta.predict(pca_x_test)
```

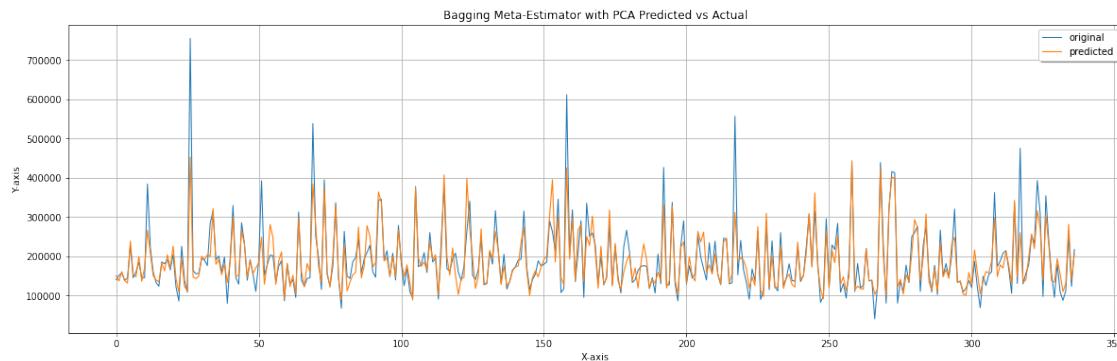
```
[ ]: baggingMeta_mse, baggingMeta_rmse = mse_rmse(pca_y_test, pca_y_pred_bagmeta)
```

```
[ ]: print(baggingMeta_mse)
print(baggingMeta_rmse)
```

```
1660828820.9478633
```

```
40753.26761068201
```

```
[ ]: plt.figure(figsize=(20,6))
x_ax = range(len(pca_y_test))
plt.plot(x_ax,pca_y_test, linewidth=1, label="original")
plt.plot(x_ax, pca_y_pred_bagmeta, linewidth=1.1, label="predicted")
plt.title("Bagging Meta-Estimator with PCA Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



## XGBoost With PCA

```
[ ]: pca_xgbmodel=xgb.XGBRegressor()
pca_xgbmodel.fit(pca_x_train, pca_y_train)
pca_xgbmodel.score(pca_x_test,pca_y_test)
```

```
[21:05:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

```
[ ]: 0.8325806551744855
```

```
[ ]: pca_y_pred_xgb = pca_xgbmodel.predict(pca_x_test)
```

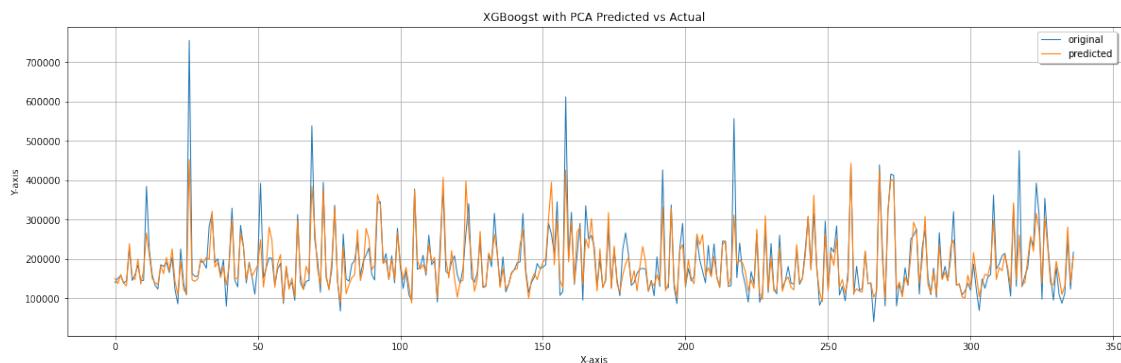
```
[ ]: xgb_mse,xgb_rmse = mse_rmse(pca_y_test, pca_y_pred_xgb)
```

```
[ ]: print(xgb_mse)
print(xgb_rmse)
```

1337926186.1260653

36577.6733284946

```
[ ]: plt.figure(figsize=(20,6))
x_ax = range(len(pca_y_test))
plt.plot(x_ax,pca_y_test, linewidth=1, label="original")
plt.plot(x_ax, pca_y_pred_bagmeta, linewidth=1.1, label="predicted")
plt.title("XGBoogst with PCA Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



#Stepwise feature selection

```
[ ]: stepwise_x = tempConcat.drop(["SalePrice", "Utilities"], axis = 1)#The Utilities
↳ column was causing ERRORS so it was removed. The P-Value obtained from it
↳ was 0.0 so regardless it would not have been included
#Therefore it shouold have no impact on the stepwise selection
stepwise_y = tempConcat["SalePrice"]
```

```
[ ]: import warnings #for ignoring warnings https://stackoverflow.com/questions/
↳ 14463277/how-to-disable-python-warnings
def stepwise_selection(X, y):
```

```

initial_list=[]
threshold_in=0.01,
threshold_out = 0.05,
verbose=True
""" Perform a forward-backward feature selection
based on p-value from statsmodels.api.OLS
Arguments:
    X - pandas.DataFrame with candidate features
    y - list-like with the target
    initial_list - list of features to start with (column names of X)
    threshold_in - include a feature if its p-value < threshold_in
    threshold_out - exclude a feature if its p-value > threshold_out
    verbose - whether to print the sequence of inclusions and exclusions
Returns: list of selected features
Always set threshold_in < threshold_out to avoid infinite looping.
See https://en.wikipedia.org/wiki/Stepwise_regression for the details
"""

included = list(initial_list)
while True:
    changed=False
    # forward step
    excluded = list(set(X.columns)-set(included))
    new_pval = pd.Series(index=excluded)
    for new_column in excluded:
        warnings.filterwarnings("ignore")#used to ignore warnings
        model = sm.OLS(y, sm.add_constant(pd.
>DataFrame(X[included+[new_column]]))).fit()
        new_pval[new_column] = model.pvalues[new_column]
    best_pval = new_pval.min()
    if best_pval < threshold_in:
        # best_feature = new_pval.argmin()
        best_feature = new_pval.idxmin()
        included.append(best_feature)
        changed=True
        if verbose:
            print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

    # backward step
    model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
    # use all coefs except intercept
    pvalues = model.pvalues.iloc[1:]
    worst_pval = pvalues.max() # null if pvalues is empty
    if worst_pval > threshold_out:
        changed=True
        worst_feature = pvalues.argmax()
        try:

```

```

        included.remove(worst_feature)

    except ValueError:
        print(f" {worst_feature} Not in list")
    if verbose:
        print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
    changed = False

    if not changed:
        break

return included

```

[ ]: result = stepwise\_selection(stepwise\_x, stepwise\_y)

<ipython-input-62-53029cf363d1>:25: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.

```

new_pval = pd.Series(index=excluded)

Add OverallQual           with p-value 3.17361e-248
Add GrLivArea             with p-value 7.51658e-60
Add Neighborhood          with p-value 4.64119e-59
Add BsmtExposure          with p-value 1.06688e-24
Add KitchenQual           with p-value 2.27198e-21
Add MSSubClass             with p-value 5.90547e-19
Add BsmtQual               with p-value 1.28034e-12
Add BsmtFinSF1             with p-value 1.74117e-08
Add RoofMatl               with p-value 2.92471e-08
Add OverallCond            with p-value 2.55581e-06
Add GarageCars              with p-value 2.70328e-08
Add PoolQC                 with p-value 6.19418e-05
Add PoolArea                with p-value 3.32161e-07
Add SaleCondition           with p-value 6.94228e-05
Add LotConfig               with p-value 0.000128068
Add ScreenPorch             with p-value 0.000239651
Add MasVnrArea              with p-value 0.000312057
Add Condition2              with p-value 0.000876595
Add Condition1              with p-value 0.000916061
Add Fireplaces               with p-value 0.00168037
Add TotRmsAbvGrd            with p-value 0.0063221
Add ExterQual                with p-value 0.00623091
Add BsmtFinSF2              with p-value 0.00666727
Add LotFrontage              with p-value 0.00836915
Add LandContour              with p-value 0.00770106

```

```
[ ]: print(result)

['OverallQual', 'GrLivArea', 'Neighborhood', 'BsmtExposure', 'KitchenQual',
'MSSubClass', 'BsmtQual', 'BsmtFinSF1', 'RoofMatl', 'OverallCond', 'GarageCars',
'PoolQC', 'PoolArea', 'SaleCondition', 'LotConfig', 'ScreenPorch', 'MasVnrArea',
'Condition2', 'Condition1', 'Fireplaces', 'TotRmsAbvGrd', 'ExterQual',
'BsmtFinSF2', 'LotFrontage', 'LandContour']
```

```
[ ]: len(result)
```

```
[ ]: 25
```

```
[ ]: final_step_x = tempConcat[result]
final_step_y = tempConcat["SalePrice"]
```

```
[ ]: #Split data for testing
```

```
step_x_train, step_x_test, step_y_train, step_y_test =
    train_test_split(final_step_x, final_step_y, test_size=0.30, random_state=42)
```

RandomForest Regression with Stepwise Feature selection

```
[ ]: rand_forest_stepwise = RandomForestRegressor()#create random forest test
rand_forest_stepwise.fit(step_x_train, step_y_train)
```

```
[ ]: RandomForestRegressor()
```

```
[ ]: stepwise_y_pred = rand_forest_stepwise.predict(step_x_test)
```

```
[ ]: stepwise_score = rand_forest_stepwise.score(step_x_test, step_y_test)
print(stepwise_score)
```

0.8807565133889832

```
[ ]: random_step_mse,random_step_rmse = mse_rmse(step_y_test, stepwise_y_pred)
```

```
[ ]: print(random_step_mse)
print(random_step_rmse)
```

952930400.1764237

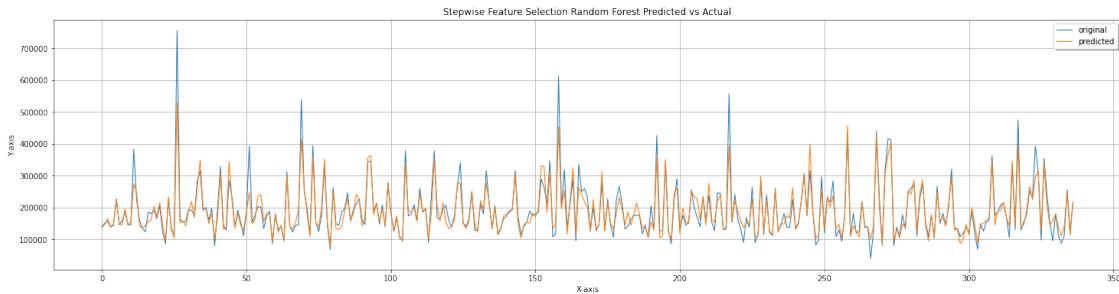
30869.570780566803

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(step_y_test))
plt.plot(x_ax, step_y_test, linewidth=1, label="original")
plt.plot(x_ax, stepwise_y_pred, linewidth=1.1, label="predicted")
plt.title("Stepwise Feature Selection Random Forest Predicted vs Actual")
plt.xlabel('X-axis')
```

```

plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()

```



### AdaBoost with Stepwise Feature Selection

```

[ ]: stepwise_adamodel = AdaBoostRegressor()
stepwise_adamodel.fit(step_x_train, step_y_train)
stepwise_adamodel.score(step_x_test, step_y_test)

```

```
[ ]: 0.8483483044431357
```

```
[ ]: stepwise_y_pred_ada = stepwise_adamodel.predict(step_x_test)
```

```
[ ]: ada_mse_step, ada_rmse_step = mse_rmse(step_y_test, stepwise_y_pred_ada)
```

```
[ ]: print(ada_mse_step)
print(ada_rmse_step)
```

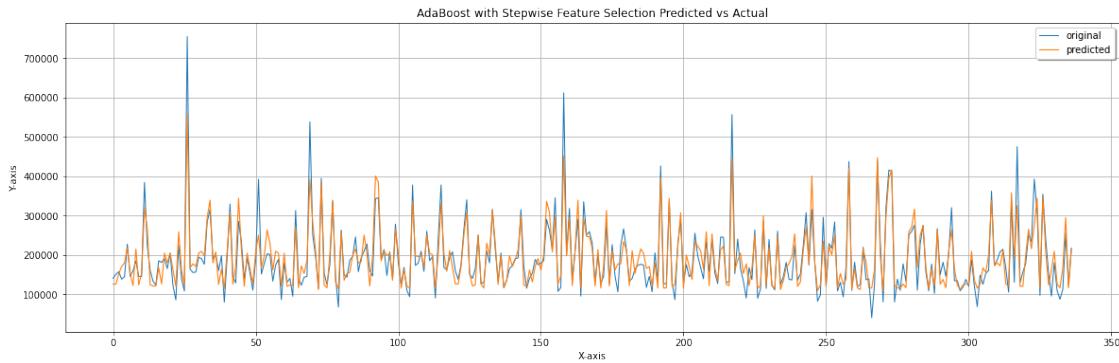
1211919535.7466547

34812.63471423349

```

[ ]: plt.figure(figsize=(20,6))
x_ax = range(len(step_y_test))
plt.plot(x_ax, step_y_test, linewidth=1, label="original")
plt.plot(x_ax, stepwise_y_pred_ada, linewidth=1.1, label="predicted")
plt.title("AdaBoost with Stepwise Feature Selection Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()

```



### Bagging Meta-Estimator with Stepwise Feature Selection

```
[ ]: stepwise_bagmeta = BaggingRegressor(tree.DecisionTreeRegressor(random_state=1))
stepwise_bagmeta.fit(step_x_train, step_y_train)
stepwise_bagmeta.score(step_x_test, step_y_test)
```

```
[ ]: 0.8702336031597314
```

```
[ ]: stepwise_y_pred_bagmeta = stepwise_bagmeta.predict(step_x_test)
```

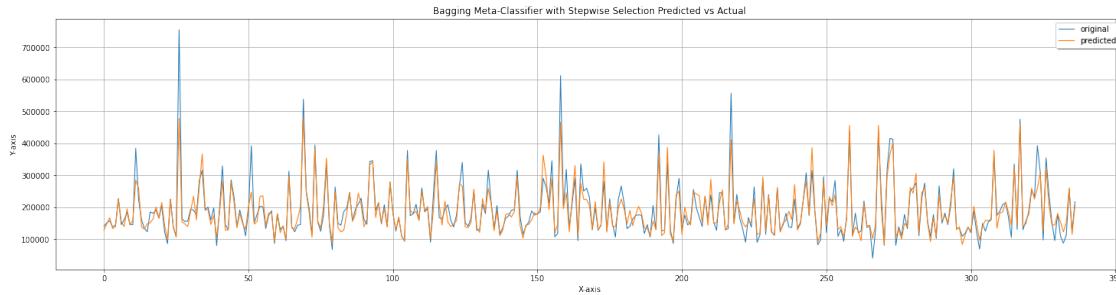
```
[ ]: bagging_mse_step,bagging_rmse_step = mse_rmse(step_y_test,stepwise_y_pred_bagmeta)
```

```
[ ]: print(bagging_mse_step)
print(bagging_rmse_step)
```

```
1037023891.0728488
```

```
32202.855324844237
```

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(step_y_test))
plt.plot(x_ax, step_y_test, linewidth=1, label="original")
plt.plot(x_ax, stepwise_y_pred_bagmeta, linewidth=1.1, label="predicted")
plt.title("Bagging Meta-Classifier with Stepwise Selection Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



### XGBoost with Stepwise Feature Selection

```
[ ]: stepwise_xgbmodel=xgb.XGBRegressor()
stepwise_xgbmodel.fit(step_x_train, step_y_train)
stepwise_xgbmodel.score(step_x_test,step_y_test)
```

[21:06:02] WARNING: /workspace/src/objective/regression\_obj.cu:152: reg:linear  
is now deprecated in favor of reg:squarederror.

[ ]: 0.899743937268905

```
[ ]: stepwise_y_pred_xgb = stepwise_xgbmodel.predict(step_x_test)
```

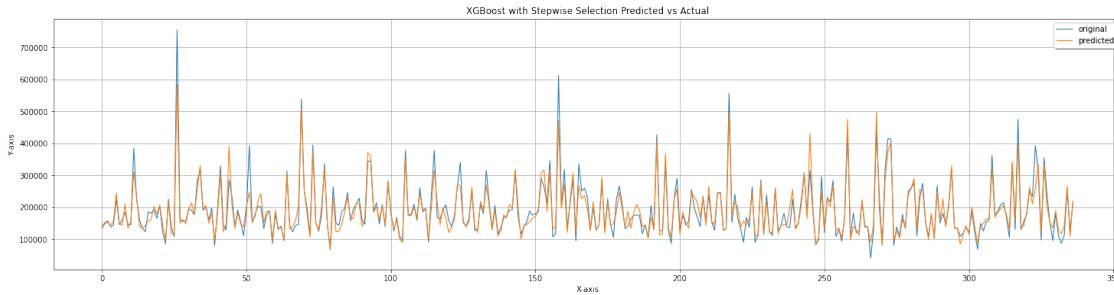
```
[ ]: xgb_step_mse, xgb_step_rmse = mse_rmse(step_y_test,stepwise_y_pred_xgb )
```

```
[ ]: print(xgb_step_mse)
print(xgb_step_rmse)
```

801193026.9206706

28305.353326193803

```
[ ]: plt.figure(figsize=(25,6))
x_ax = range(len(step_y_test))
plt.plot(x_ax, step_y_test, linewidth=1, label="original")
plt.plot(x_ax, stepwise_y_pred_xgb, linewidth=1.1, label="predicted")
plt.title("XGBoost with Stepwise Selection Predicted vs Actual")
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



#Final Scores

```
[ ]: print("*****Base Model Accuracy Scores*****")
print(f"Random Forest Regressor Model Score: {score*100:,.3f}%")
print(f"AdaBoost Model Score: {adamodel.score(x_test_numeric,y_test)*100:,.3f}%")
print(f"Bagging Meta-Estimator Model Score: {bagmeta.
    .score(x_test_numeric,y_test)*100:,.3f}%")
print(f"XGBoost Model Score: {xgbmodel.score(x_test_numeric,y_test)*100:,.3f}%
    \n")
print("*****Principle Component Analysis Accuracy Scores*****")
print(f"PCA Random Forest Regressor Model Score: {pca_score*100:,.3f}%")
print(f"PCA AdaBoost Model Score: {PCA_adamodel.
    .score(pca_x_test,pca_y_test)*100:,.3f}%")
print(f"PCA Bagging Meta-Estimator Model Score: {pca_bagmeta.
    .score(pca_x_test,pca_y_test)*100:,.3f}%")
print(f"PCA XGBoost Model Score: {pca_xgbmodel.score(pca_x_test,pca_y_test)*100:,
    ,.3f}% \n")
print("*****Stepwise Feature Selection Accuracy Scores*****")
print(f"Stepwise Selection Random Forest Regressor Model Score: {stepwise_
    .score*100:,.3f}%")
print(f"Stepwise Selection AdaBoost Model Score: {stepwise_adamodel.
    .score(step_x_test,step_y_test)*100:,.3f}%")
print(f"Stepwise Selection Bagging Meta-Estimator Model Score: {stepwise_
    .bagmeta.score(step_x_test,step_y_test)*100:,.3f}%")
print(f"Stepwise Selection XGBoost Model Score: {stepwise_xgbmodel.
    .score(step_x_test,step_y_test)*100:,.3f}%")
```

\*\*\*\*\*Base Model Accuracy Scores\*\*\*\*\*

Random Forest Regressor Model Score: 84.283%  
AdaBoost Model Score: 78.942%  
Bagging Meta-Estimator Model Score: 83.428%  
XGBoost Model Score: 86.741%

\*\*\*\*\*Principle Component Analysis Accuracy Scores\*\*\*\*\*  
PCA Random Forest Regressor Model Score: 80.926%

```
PCA AdaBoost Model Score: 78.160%
PCA Bagging Meta-Estimator Model Score: 79.217%
PCA XGBoost Model Score: 83.258%
```

```
*****Stepwise Feature Selection Accuracy Scores*****
Stepwise Selection Random Forest Regressor Model Score: 88.076%
Stepwise Selection AdaBoost Model Score: 84.835%
Stepwise Selection Bagging Meta-Estimator Model Score: 87.023%
Stepwise Selection XGBoost Model Score: 89.974%
```

```
#Lazypredict
```

```
Stepwise Feature Selection
```

```
[ ]: from lazypredict.Supervised import LazyRegressor#https://pypi.org/project/
      ↵lazypredict/
```

```
reg = LazyRegressor(verbose=0,ignore_warnings=False, custom_metric=None )
models,predictions = reg.fit(step_x_train, step_x_test, step_y_train, ↵
                             ↵step_y_test)
```

```
100%|     | 42/42 [00:20<00:00,  2.01it/s]
```

```
[21:06:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
```

```
[ ]: models
```

Model	Adjusted R-Squared	R-Squared	RMSE	\
XGBRegressor	0.89	0.90	28305.62	
GradientBoostingRegressor	0.89	0.90	28862.67	
PoissonRegressor	0.88	0.89	29262.83	
BaggingRegressor	0.87	0.88	30529.80	
GammaRegressor	0.87	0.88	30967.12	
RandomForestRegressor	0.87	0.88	30983.56	
HuberRegressor	0.87	0.88	31155.30	
PassiveAggressiveRegressor	0.87	0.88	31335.30	
LGBMRegressor	0.86	0.87	31867.27	
HistGradientBoostingRegressor	0.86	0.87	32295.22	
RANSACRegressor	0.85	0.87	32793.81	
ExtraTreesRegressor	0.85	0.86	32918.82	
ElasticNet	0.84	0.85	34203.14	
AdaBoostRegressor	0.84	0.85	34845.60	
BayesianRidge	0.82	0.83	36626.44	
RidgeCV	0.82	0.83	36650.94	
LassoLarsCV	0.81	0.83	37044.08	
LarsCV	0.81	0.83	37044.08	

TweedieRegressor	0.81	0.83	37091.19
Ridge	0.81	0.82	37411.04
LassoLars	0.81	0.82	37416.10
Lasso	0.81	0.82	37502.29
LinearRegression	0.81	0.82	37505.53
TransformedTargetRegressor	0.81	0.82	37505.53
Lars	0.81	0.82	37505.53
LassoLarsIC	0.81	0.82	37505.53
LassoCV	0.81	0.82	37770.53
SGDRegressor	0.78	0.80	40020.71
OrthogonalMatchingPursuitCV	0.77	0.79	41301.55
DecisionTreeRegressor	0.75	0.77	42660.42
KNeighborsRegressor	0.75	0.77	43311.93
OrthogonalMatchingPursuit	0.72	0.74	45376.79
ExtraTreeRegressor	0.69	0.71	47809.16
ElasticNetCV	0.05	0.12	83933.90
DummyRegressor	-0.09	-0.01	89857.23
NuSVR	-0.13	-0.05	91445.35
SVR	-0.19	-0.10	93908.07
QuantileRegressor	-0.19	-0.10	93941.85
GaussianProcessRegressor	-2.94	-2.65	170736.53
KernelRidge	-3.92	-3.56	190810.83
LinearSVR	-5.02	-4.57	210964.86
MLPRegressor	-5.02	-4.57	211001.11

#### Time Taken

Model	Time Taken
XGBRegressor	0.11
GradientBoostingRegressor	0.25
PoissonRegressor	0.02
BaggingRegressor	0.08
GammaRegressor	0.03
RandomForestRegressor	0.42
HuberRegressor	0.02
PassiveAggressiveRegressor	0.11
LGBMRegressor	0.08
HistGradientBoostingRegressor	4.48
RANSACRegressor	0.13
ExtraTreesRegressor	0.51
ElasticNet	0.01
AdaBoostRegressor	0.22
BayesianRidge	0.04
RidgeCV	0.02
LassoLarsCV	0.07
LarsCV	0.07
TweedieRegressor	0.01
Ridge	0.01

LassoLars	0.02
Lasso	0.02
LinearRegression	0.01
TransformedTargetRegressor	0.01
Lars	0.03
LassoLarsIC	0.03
LassoCV	0.13
SGDRegressor	0.03
OrthogonalMatchingPursuitCV	0.01
DecisionTreeRegressor	0.09
KNeighborsRegressor	0.01
OrthogonalMatchingPursuit	0.01
ExtraTreeRegressor	0.03
ElasticNetCV	0.14
DummyRegressor	0.03
NuSVR	0.06
SVR	0.12
QuantileRegressor	12.61
GaussianProcessRegressor	0.14
KernelRidge	0.04
LinearSVR	0.01
MLPRegressor	0.61

## PCA

```
[ ]: reg2 = LazyRegressor(verbose=0, ignore_warnings=False, custom_metric=None )
models2, predictions2 = reg.fit(pca_x_train, pca_x_test, pca_y_train, pca_y_test)
```

```
'tuple' object has no attribute '__name__'
Invalid Regressor(s)

98%|   | 41/42 [00:15<00:00,  2.44it/s]

[21:06:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.

100%|   | 42/42 [00:15<00:00,  2.71it/s]
```

```
[ ]: models2
```

Model	Adjusted R-Squared	R-Squared	RMSE	\
XGBRegressor	0.81	0.83	36577.51	
BaggingRegressor	0.81	0.83	36617.36	
ExtraTreesRegressor	0.81	0.83	36761.78	
GradientBoostingRegressor	0.81	0.83	36838.98	
PoissonRegressor	0.81	0.83	37028.51	
LGBMRegressor	0.80	0.82	38304.05	

HistGradientBoostingRegressor	0.79	0.82	38426.48
RandomForestRegressor	0.79	0.81	38884.27
HuberRegressor	0.78	0.80	40195.76
RidgeCV	0.77	0.80	40352.79
BayesianRidge	0.77	0.80	40364.09
LassoLars	0.77	0.80	40373.68
Ridge	0.77	0.80	40388.93
Lasso	0.77	0.80	40394.91
Lars	0.77	0.80	40395.75
TransformedTargetRegressor	0.77	0.80	40395.75
LinearRegression	0.77	0.80	40395.75
LassoLarsIC	0.77	0.79	40604.52
SGDRegressor	0.77	0.79	40812.30
AdaBoostRegressor	0.76	0.79	41168.57
LassoLarsCV	0.76	0.79	41398.86
LarsCV	0.76	0.79	41398.86
PassiveAggressiveRegressor	0.76	0.78	41705.43
LassoCV	0.74	0.77	43091.83
ElasticNet	0.73	0.76	43771.09
GammaRegressor	0.73	0.76	43911.39
ExtraTreeRegressor	0.71	0.74	45806.16
OrthogonalMatchingPursuitCV	0.70	0.73	46296.16
TweedieRegressor	0.69	0.72	46889.53
KNeighborsRegressor	0.67	0.70	48650.42
OrthogonalMatchingPursuit	0.67	0.70	48849.61
RANSACRegressor	0.65	0.69	49852.82
DecisionTreeRegressor	0.63	0.67	51364.66
ElasticNetCV	0.02	0.12	84067.50
DummyRegressor	-0.12	-0.01	89857.23
NuSVR	-0.16	-0.05	91450.34
SVR	-0.23	-0.10	93896.93
QuantileRegressor	-0.23	-0.10	93941.85
GaussianProcessRegressor	-3.64	-3.17	182616.19
KernelRidge	-4.10	-3.59	191443.86
MLPRegressor	-5.19	-4.56	210783.54
LinearSVR	-5.20	-4.57	210964.86

#### Time Taken

Model	Time Taken
XGBRegressor	0.15
BaggingRegressor	0.05
ExtraTreesRegressor	0.40
GradientBoostingRegressor	0.23
PoissonRegressor	0.02
LGBMRegressor	0.09
HistGradientBoostingRegressor	0.30
RandomForestRegressor	0.51

HuberRegressor	0.04
RidgeCV	0.03
BayesianRidge	0.02
LassoLars	0.04
Ridge	0.01
Lasso	0.01
Lars	0.02
TransformedTargetRegressor	0.01
LinearRegression	0.02
LassoLarsIC	0.03
SGDRegressor	0.03
AdaBoostRegressor	0.11
LassoLarsCV	0.08
LarsCV	0.05
PassiveAggressiveRegressor	0.09
LassoCV	0.13
ElasticNet	0.03
GammaRegressor	0.01
ExtraTreeRegressor	0.04
OrthogonalMatchingPursuitCV	0.02
TweedieRegressor	0.03
KNeighborsRegressor	0.02
OrthogonalMatchingPursuit	0.01
RANSACRegressor	0.15
DecisionTreeRegressor	0.03
ElasticNetCV	0.12
DummyRegressor	0.02
NuSVR	0.06
SVR	0.12
QuantileRegressor	11.54
GaussianProcessRegressor	0.13
KernelRidge	0.02
MLPRegressor	0.66
LinearSVR	0.02

## #Conclusion

- The model with the highest Adjusted R-Squared and lowest Root Mean Squared Error (RMSE) is usually the best, as it explains the most variance in the data (Adjusted R-Squared) while predicting with the lowest difference from the actual values.
- By those metrics, the best model we produced was the XGB Regression Model *with* Stepwise Feature Selection

<https://www.statology.org/rmse-vs-r-squared/>

## Why did that model perform the best?

An XGBoost model is ultimately a very thorough derivative of a gradient boosted trees algorithm. Gradient boosted trees already tend to be fairly accurate because they are thorough, and this

particular variant seems to be especially good at predictions in this case.

The Stepwise Feature Selection applied also is a high accuracy way of reducing to features which truly matter in predictions; this puts the XGBoost with Stepwise Feature Selection ahead of the XGBoost without, and helps make it the most accurate model we attempted.

In conclusion, we accomplished our objective - we determined what the best model for predicting housing prices from a dataset comprised of various details about homes was - XGB Regression with Stepwise Feature Selection. We also were able to yield a confidently high prediction from this model of that data, meaning we didn't *just* find the most accurate model relative to other models, we found a model that was in general good at predicting these prices. We also found several other models that, while not as good, would likely be sufficiently accurate alternatives.

All of this being said, it should be noted that this modeling method may not be the best for other locations; Iowa is not representative of the entire United States, let alone the world. The nature in which housing features effect the price of the home may vary between Iowa and other regions.

That being said, the same reasons XGB Regression excelled here likely would compel it to be successful elsewhere.

## 2 Additional References

<https://datascience.stackexchange.com/questions/24405/how-to-do-stepwise-regression-using-sklearn/24447#24447>

[https://planspace.org/20150423-forward\\_selection\\_with\\_statsmodels/](https://planspace.org/20150423-forward_selection_with_statsmodels/)

[https://www.statsmodels.org/dev/generated/statsmodels.regression.linear\\_model.OLS.html](https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html)

<https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/>

<https://www.simplilearn.com/tutorials/statistics-tutorial/mean-squared-error>

<https://corporatefinanceinstitute.com/resources/data-science/r-squared/>

<https://www.investopedia.com/terms/p/p-value.asp>

<https://pypi.org/project/lazypredict/>

# Notebook

August 2, 2023

## Lab 6

Leng Her

October 15, 2022

#Overview

In order to understand which features are important to help us predict who to offer the CARAVAN policy to, we are using logistic regression, PCA, and random forest technique. Logistic regression is a predictive analysis used to describe data and to explain relationships between variables (Statistics Solutions). PCA also known as principal component analysis and is linear dimensionality reduction on our data. Lastly random forest classifier from sklearn is defined as “a random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting” (scikit). By using these techniques we will attempt to create models that predict who else to offer the CARAVAN policy to.

References (Direct Links):

[https://chrisalbon.com/code/machine\\_learning/trees\\_and\\_forests/feature\\_selection\\_using\\_random\\_forest/](https://chrisalbon.com/code/machine_learning/trees_and_forests/feature_selection_using_random_forest/)  
<https://districtdatalabs.silvrback.com/principal-component-analysis-with-python>  
<https://machinelearningmastery.com/feature-selection-machine-learning-python/>  
<https://www.kaggle.com/datasets/uciml/caravan-insurance-challenge>  
<https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a>  
<https://seaborn.pydata.org/generated/seaborn.catplot.html>  
<https://www.python-graph-gallery.com/92-control-color-in-seaborn-heatmaps>  
<https://gust.dev/python/dimensionality-reduction>  
<https://thispointer.com/select-first-n-columns-of-pandas-dataframe/>  
<https://numpy.org/doc/stable/reference/generated/numpy.cov.html>  
<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>  
<https://machinelearningmastery.com/random-oversampling-andundersampling-for-imbalanced-classification/>

<https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-logistic-regression/>

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

#Data

The data contains information on customers of an insurance company. Dataset contains 86 various variables in relation to who has insurance for caravan and who does not. For more information on the dataset please check: <https://www.kaggle.com/uciml/caravan-insurance-challenge/home>.

[3]: #Load dependencies

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
%matplotlib inline
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
from imblearn.under_sampling import RandomUnderSampler
```

[4]: df = pd.read\_csv("/content/caravan-insurance-challenge.csv")
df.head()

[4]:

	ORIGIN	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	\
0	train	33	1	3	2	8	0	5	
1	train	37	1	2	2	8	1	4	
2	train	37	1	2	2	8	0	4	
3	train	9	1	3	3	3	2	3	
4	train	40	1	4	2	10	1	4	

	MGODOV	MGODGE	...	APERSONG	AGEZONG	AWAOREG	ABRAND	AZEILPL	APLEZIER	\
0	1	3	...	0	0	0	1	0	0	
1	1	4	...	0	0	0	1	0	0	
2	2	4	...	0	0	0	1	0	0	
3	2	4	...	0	0	0	1	0	0	
4	1	4	...	0	0	0	1	0	0	

	AFIETS	AINBOED	ABYSTAND	CARAVAN
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

```
3      0      0      0      0  
4      0      0      0      0
```

[5 rows x 87 columns]

#EDA (Exploratory Data Analysis)

[5]: *#check for null values*  
df.info()

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 9822 entries, 0 to 9821  
Data columns (total 87 columns):  
 #   Column    Non-Null Count  Dtype     
---  --    
 0   ORIGIN    9822 non-null   object    
 1   MOSTYPE   9822 non-null   int64     
 2   MAANTHUI  9822 non-null   int64     
 3   MGEMOMV   9822 non-null   int64     
 4   MGEMLEEF  9822 non-null   int64     
 5   MOSHOOFD  9822 non-null   int64     
 6   MGODRK    9822 non-null   int64     
 7   MGODPR    9822 non-null   int64     
 8   MGODOV    9822 non-null   int64     
 9   MGODGE    9822 non-null   int64     
 10  MRELGE    9822 non-null   int64     
 11  MRELSA    9822 non-null   int64     
 12  MRELOV    9822 non-null   int64     
 13  MFALLEEN  9822 non-null   int64     
 14  MFGEKIND  9822 non-null   int64     
 15  MFWEKIND  9822 non-null   int64     
 16  MOPLHOOG  9822 non-null   int64     
 17  MOPLMIDD  9822 non-null   int64     
 18  MOPLLAAG  9822 non-null   int64     
 19  MBERHOOG  9822 non-null   int64     
 20  MBERZELF  9822 non-null   int64     
 21  MBERBOER  9822 non-null   int64     
 22  MBERMIDD  9822 non-null   int64     
 23  MBERARBG  9822 non-null   int64     
 24  MBERARBO  9822 non-null   int64     
 25  MSKA       9822 non-null   int64     
 26  MSKB1     9822 non-null   int64     
 27  MSKB2     9822 non-null   int64     
 28  MSKC       9822 non-null   int64     
 29  MSKD       9822 non-null   int64     
 30  MHHUUR    9822 non-null   int64     
 31  MHKOOP    9822 non-null   int64     
 32  MAUT1     9822 non-null   int64
```

33	MAUT2	9822	non-null	int64
34	MAUTO	9822	non-null	int64
35	MZFONDS	9822	non-null	int64
36	MZPART	9822	non-null	int64
37	MINKM30	9822	non-null	int64
38	MINK3045	9822	non-null	int64
39	MINK4575	9822	non-null	int64
40	MINK7512	9822	non-null	int64
41	MINK123M	9822	non-null	int64
42	MINKGEM	9822	non-null	int64
43	MKOOPKLA	9822	non-null	int64
44	PWAPART	9822	non-null	int64
45	PWABEDR	9822	non-null	int64
46	PWALAND	9822	non-null	int64
47	PPERSAUT	9822	non-null	int64
48	PBESAUT	9822	non-null	int64
49	PMOTSCO	9822	non-null	int64
50	PVRAAUT	9822	non-null	int64
51	PAANHANG	9822	non-null	int64
52	PTRACTOR	9822	non-null	int64
53	PWERKT	9822	non-null	int64
54	PBROM	9822	non-null	int64
55	PLEVEN	9822	non-null	int64
56	PPERSONG	9822	non-null	int64
57	PGEZONG	9822	non-null	int64
58	PWAOREG	9822	non-null	int64
59	PBRAND	9822	non-null	int64
60	PZEILPL	9822	non-null	int64
61	PPLEZIER	9822	non-null	int64
62	PFIETS	9822	non-null	int64
63	PINBOED	9822	non-null	int64
64	PBYSTAND	9822	non-null	int64
65	AWAPART	9822	non-null	int64
66	AWABEDR	9822	non-null	int64
67	AWALAND	9822	non-null	int64
68	APERSAUT	9822	non-null	int64
69	ABESAUT	9822	non-null	int64
70	AMOTSCO	9822	non-null	int64
71	AVRAAUT	9822	non-null	int64
72	AAANHANG	9822	non-null	int64
73	ATRACTOR	9822	non-null	int64
74	AWERKT	9822	non-null	int64
75	ABROM	9822	non-null	int64
76	ALEVEN	9822	non-null	int64
77	APERSONG	9822	non-null	int64
78	AGEZONG	9822	non-null	int64
79	AWAOREG	9822	non-null	int64
80	ABRAND	9822	non-null	int64

```
81 AZEILPL    9822 non-null    int64
82 APLEZIER   9822 non-null    int64
83 AFIETS     9822 non-null    int64
84 AINBOED    9822 non-null    int64
85 ABYSTAND   9822 non-null    int64
86 CARAVAN    9822 non-null    int64
dtypes: int64(86), object(1)
memory usage: 6.5+ MB
```

```
[6]: df.shape
print(f"Number of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
```

```
Number of rows: 9822
Number of columns: 87
```

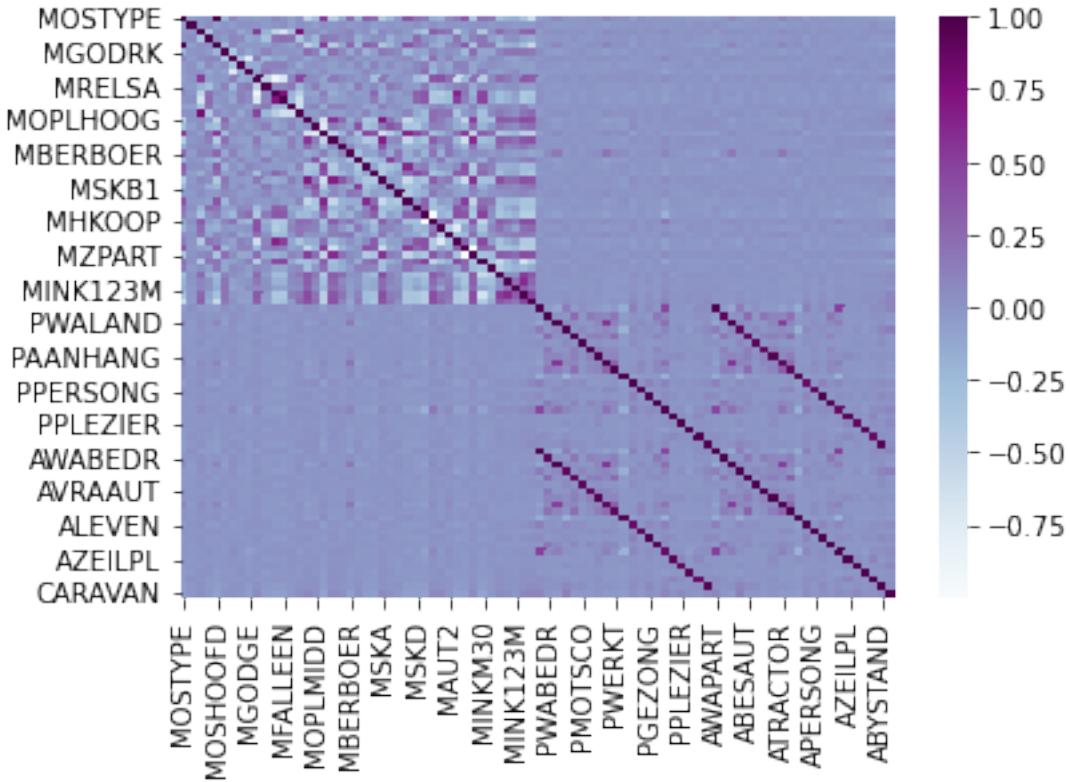
```
[7]: df.columns
```

```
[7]: Index(['ORIGIN', 'MOSTYPE', 'MAANTHUI', 'MGEMOMV', 'MGEMLEEF', 'MOSHOOFD',
       'MGODRK', 'MGODPR', 'MGODOV', 'MGODGE', 'MRELGE', 'MRELSA', 'MRELOV',
       'MFALLEEN', 'MFGEKIND', 'MFWEKIND', 'MOPLHOOG', 'MOPLMIDD', 'MOPLLAAG',
       'MBERHOOG', 'MBERZELF', 'MBERBOER', 'MBERMIDD', 'MBERARBG', 'MBERARBO',
       'MSKA', 'MSKB1', 'MSKB2', 'MSKC', 'MSKD', 'MHUUR', 'MHKOOP', 'MAUT1',
       'MAUT2', 'MAUTO', 'MZFONDS', 'MZPART', 'MINKM30', 'MINK3045',
       'MINK4575', 'MINK7512', 'MINK123M', 'MINKGEM', 'MKOOPKLA', 'PWAPART',
       'PWABEDR', 'PWALAND', 'PPERSAUT', 'PBESAUT', 'PMOTSCO', 'PVRAAUT',
       'PAANHANG', 'PTRACTOR', 'PWERKT', 'PBROM', 'PLEVEN', 'PPERSONG',
       'PGEZONG', 'PWAOREG', 'PBRAND', 'PZEILPL', 'PPLEZIER', 'PFIETS',
       'PINBOED', 'PBYSTAND', 'AWAPART', 'AWABEDR', 'AWALAND', 'APERSAUT',
       'ABESAUT', 'AMOTSCO', 'AVRAAUT', 'AAANHANG', 'ATRACTOR', 'AWERKT',
       'ABROM', 'ALEVEN', 'APERSONG', 'AGEZONG', 'AWAOREG', 'ABRAND',
       'AZEILPL', 'APLEZIER', 'AFIETS', 'AINBOED', 'ABYSTAND', 'CARAVAN'],
      dtype='object')
```

From the above code we can see that there are no null values and most values being of the int64 data type except for the ORIGIN column. There are 9822 rows of data and 82 columns.

```
[8]: import seaborn as sns
sns.heatmap(df.corr(), cmap="BuPu")
#Color changes: https://www.pythongraph-gallery.com/
˓→92-control-color-in-seaborn-heatmaps
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe652dd5f90>
```



The above code shows a correlation heatmap. From the heat map we can see that the first few features are more correlated with one another than anywhere else on the correlation heatmap.

```
[9]: not_insured_with_caraven = sum(df["CARAVAN"] == 0)
insured_with_caraven = sum(df["CARAVAN"] == 1)

plt.bar("Not Insured", not_insured_with_caraven, color = 'b', width = 0.5, ▾
        Label = "Not Insured")

plt.bar("Insured", insured_with_caraven, color = 'r', width = 0.5, Label = ▾
        'Insured')

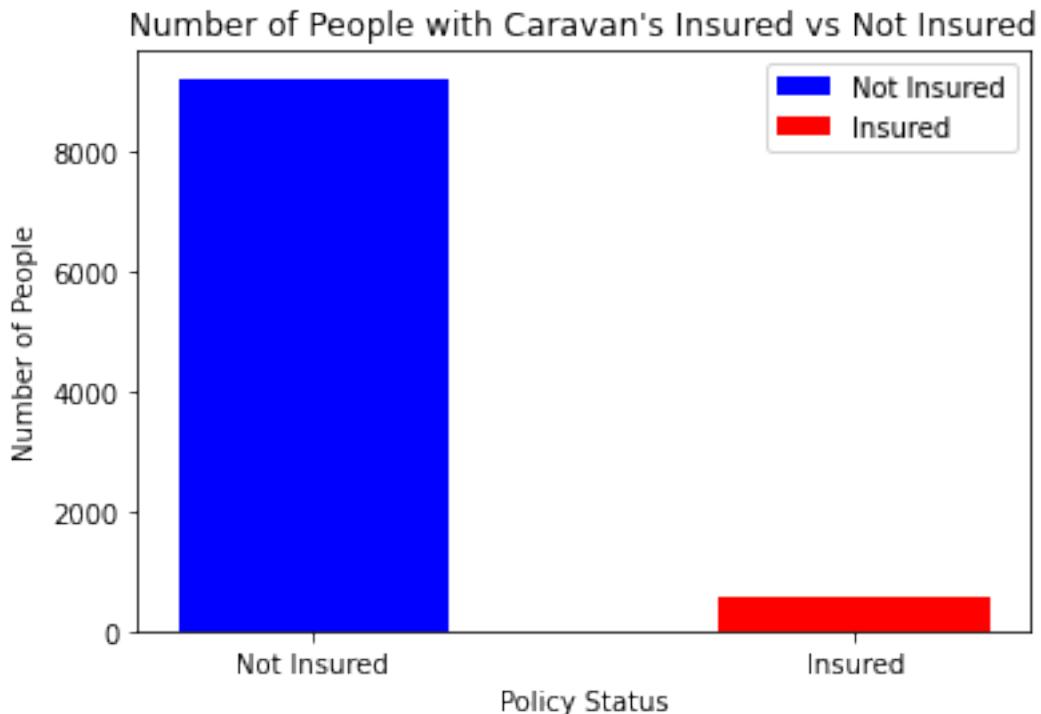
print(f'Not insured: {not_insured_with_caraven}')
print(f'Insured: {insured_with_caraven}')

plt.title("Number of People with Caravan's Insured vs Not Insured")
plt.xlabel("Policy Status")
plt.ylabel("Number of People")
plt.legend(loc='upper right')

per_insured = (insured_with_caraven/not_insured_with_caraven) * 100
```

```
print(f"Percentage that is insured with CARAVAN {per_insured}")
```

```
Not insured: 9236  
Insured: 586  
Percentage that is insured with CARAVAN 6.344737981810307
```



From the bar graph we can see that the data seems to be very unbalanced with only about 6% of customers being insured with CARAVAN our target variable. Because of the unbalanced data our models section will be using the data however, undersampled.

#Models

The models in this section will be shown in the order below:

Logistic regression with all features (base model)

Logistic regression with stepwise selection based on p-values

Random Forest based on feature importance feature selection

Random Forest based on Principal component analysis feature selection

Because of the unbalanced data from the EDA section we will be undersampling our data in order to achieve more accurate results.

```
[10]: #prepare data for modeling  
from sklearn import preprocessing
```

```
all_vars = df.drop(['ORIGIN'], axis=1)
x_vars = all_vars.drop(['CARAVAN'], axis=1)
y = df['CARAVAN']
```

```
[11]: #split data and scale data
```

```
x_train, x_test, y_train, y_test = train_test_split(x_vars, y, test_size=0.3,
random_state=0)

rus = RandomUnderSampler(random_state=77)
x_train_under, y_train_under = rus.fit_resample(x_train, y_train)
```

```
[12]: from sklearn.linear_model import LogisticRegression
logisticRegr = LogisticRegression(max_iter=9823)
```

```
[13]: # APPLY LOGISTIC REGRESSION ON TRAINING SET
logisticRegr.fit(x_train_under, y_train_under)
predictions = logisticRegr.predict(x_test)

score = logisticRegr.score(x_test, y_test)

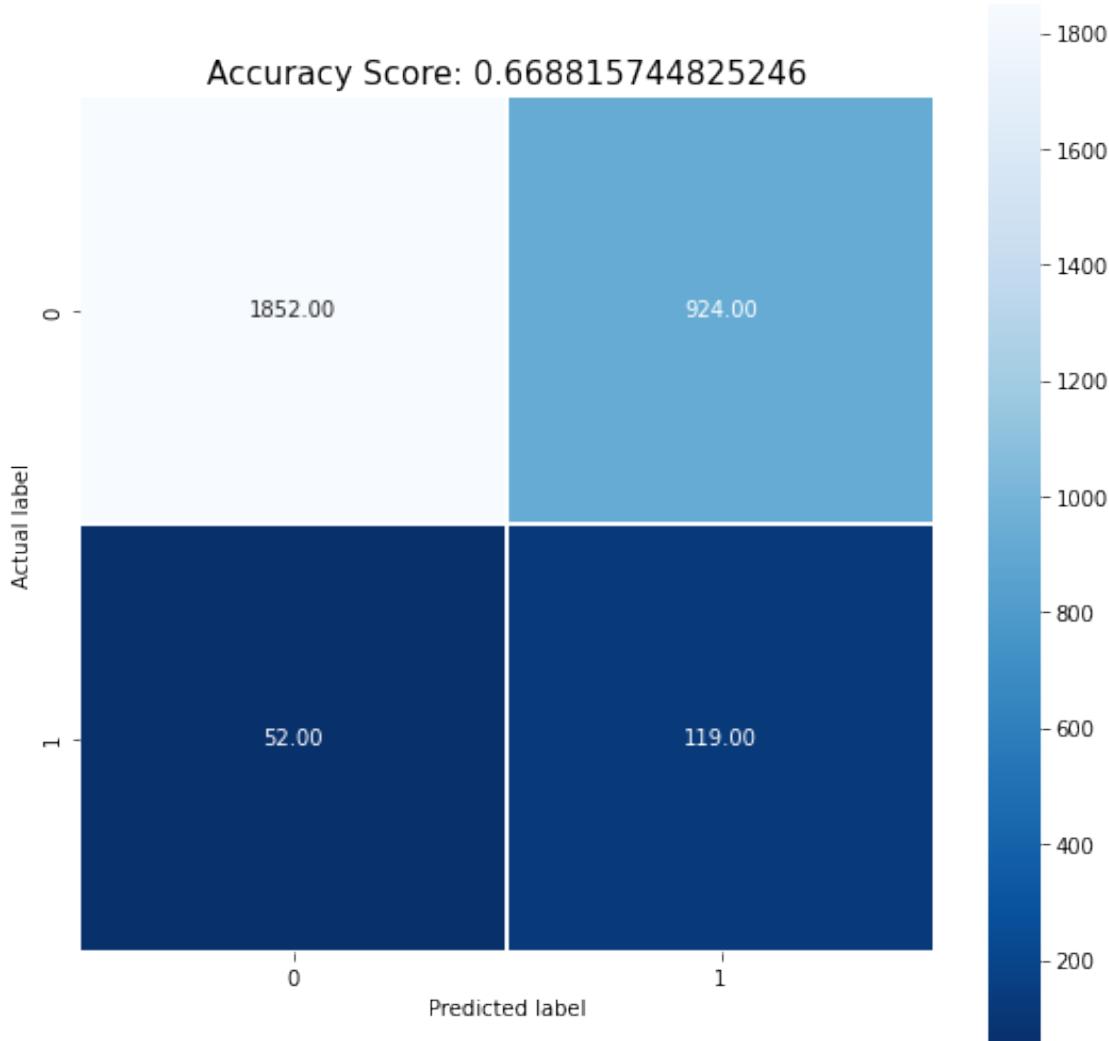
print(score)
```

0.668815744825246

```
[14]: confusion_matrix = metrics.confusion_matrix(y_test,predictions)
print(confusion_matrix)
```

```
[[1852  924]
 [ 52  119]]
```

```
[15]: #https://towardsdatascience.com/
      ↴logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b
plt.figure(figsize=(9,9))
sns.heatmap(confusion_matrix, annot=True, fmt=".2f", linewidths=.5, square =
    ↴True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



The code above is logistic regression of all feature variables and will act as our base model. From the base model we can see that it has an accuracy score of 0.668, meaning about 33% of our predicted guess from the base model are incorrect and about 67% of our prediction are correct.

The confusion matrix is can also be shown to express the details above,

```
[16]: #LOGISITIC REGRESSION WITH SPECIFIC FEATURE SET BASED ON P-VALUE
def stepwise_selection(X, y):
    initial_list=[]
    threshold_in=0.01,
    threshold_out = 0.05,
    verbose=True
    """
    Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
```

```

X - pandas.DataFrame with candidate features
y - list-like with the target
initial_list - list of features to start with (column names of X)
threshold_in - include a feature if its p-value < threshold_in
threshold_out - exclude a feature if its p-value > threshold_out
verbose - whether to print the sequence of inclusions and exclusions
>Returns: list of selected features
Always set threshold_in < threshold_out to avoid infinite looping.
See https://en.wikipedia.org/wiki/Stepwise\_regression for the details
"""

included = list(initial_list)
while True:
    changed=False
    # forward step
    excluded = list(set(X.columns)-set(included))
    new_pval = pd.Series(index=excluded)
    for new_column in excluded:
        model = sm.OLS(y, sm.add_constant(pd.
>DataFrame(X[included+[new_column]]))).fit()
        new_pval[new_column] = model.pvalues[new_column]
    best_pval = new_pval.min()
    if best_pval < threshold_in:
        # best_feature = new_pval.argmin()
        best_feature = new_pval.idxmin()
        included.append(best_feature)
        changed=True
        if verbose:
            print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

    # backward step
    model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
    # use all coefs except intercept
    pvalues = model.pvalues.iloc[1:]
    worst_pval = pvalues.max() # null if pvalues is empty
    if worst_pval > threshold_out:
        changed=True
        worst_feature = pvalues.argmax()
        included.remove(worst_feature)
        if verbose:
            print('Drop  {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
    if not changed:
        break
return included

result = stepwise_selection(x_vars, y)

```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
  
Add  PPERSAUT           with p-value 2.14684e-42  
  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```











```

the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
Add MKOOPKLA                         with p-value 1.36739e-21
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
Add PWAPART                      with p-value 3.66711e-15  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```









```

the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

Add APLEZIER                         with p-value 8.20766e-15

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```













```
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
```



```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
Add  MBERBOER                      with p-value 8.31838e-06
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for

```











```
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
```

```

of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
Add  MRELGE                      with p-value 1.41977e-05
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
Add PWALAND                         with p-value 0.000361295
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

```











```
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
Add ABRAND                                with p-value 0.000937601
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
Add AZEILPL                               with p-value 0.00153041
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
```













```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
  
Add PBYSTAND                               with p-value 0.00243579  
  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```











```
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
```

```
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

Add  PGEZONG           with p-value 0.00485648

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:  
DeprecationWarning: The default dtype for empty Series will be 'object' instead  
of 'float64' in a future version. Specify a dtype explicitly to silence this  
warning.  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
  
Add AGEZONG                      with p-value 0.00450709  
  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for
```











```
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:25:
DeprecationWarning: The default dtype for empty Series will be 'object' instead
of 'float64' in a future version. Specify a dtype explicitly to silence this
warning.
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
```

```
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)

Add  MHHUUR                               with p-value 0.00630075

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
  x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)  
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:  
FutureWarning: In a future version of pandas all arguments of concat except for  
the argument 'objs' will be keyword-only  
    x = pd.concat(x[::-order], 1)
```

Code above was obtained from Exercise 5. From the code above we use stepwise selection to differentiate the best from the worse features in the data set based on the p-value. We will later use the information obtained by this to do another logistic regression with the specified feature variables.

```
[17]: print('resulting features:')
```

```
resulting features:  
['PPERSAUT', 'MKOOPKLA', 'PWAPART', 'APLEZIER', 'MOPLHOOG', 'PBRAND',  
'MBERBOER', 'MRELGE', 'PWALAND', 'ABRAND', 'AZEILPL', 'MINK123M', 'PBYSTAND',  
'PGEZONG', 'AGEZONG', 'MHUUR']
```

The code above prints out the best features for our dataset.

```
[18]: feature_col = df[result].copy()  
  
feature_col.head()
```

```
[18]: PPERSAUT MKOOPKLA PWAPART APLEZIER MOPLHOOG PBRAND MBERBOER MRELGE \
0      6      3      0      0      1      5      1      7
1      0      4      2      0      0      2      0      6
2      6      4      2      0      0      2      0      3
3      6      4      0      0      3      2      0      5
4      0      3      0      0      5      6      4      7

PWALAND ABRAND AZEILPL MINK123M PBYSTAND PGEZONG AGEZONG MHUUR
0      0      1      0      0      0      0      0      1
1      0      1      0      0      0      0      0      2
2      0      1      0      0      0      0      0      7
3      0      1      0      0      0      0      0      5
4      0      1      0      0      0      0      0      4
```

```
[19]: X_train, X_test, Y_train, Y_test = train_test_split(feature_col, y, test_size=0.
    ↪3, random_state=0)

rus2 = RandomUnderSampler(random_state=77)
x_train_under2, y_train_under2 = rus2.fit_resample(X_train, Y_train)

logisticRegr = LogisticRegression(max_iter=9823)
logisticRegr.fit(x_train_under2, y_train_under2)
predictions = logisticRegr.predict(X_test)

score2 = logisticRegr.score(X_test, Y_test)

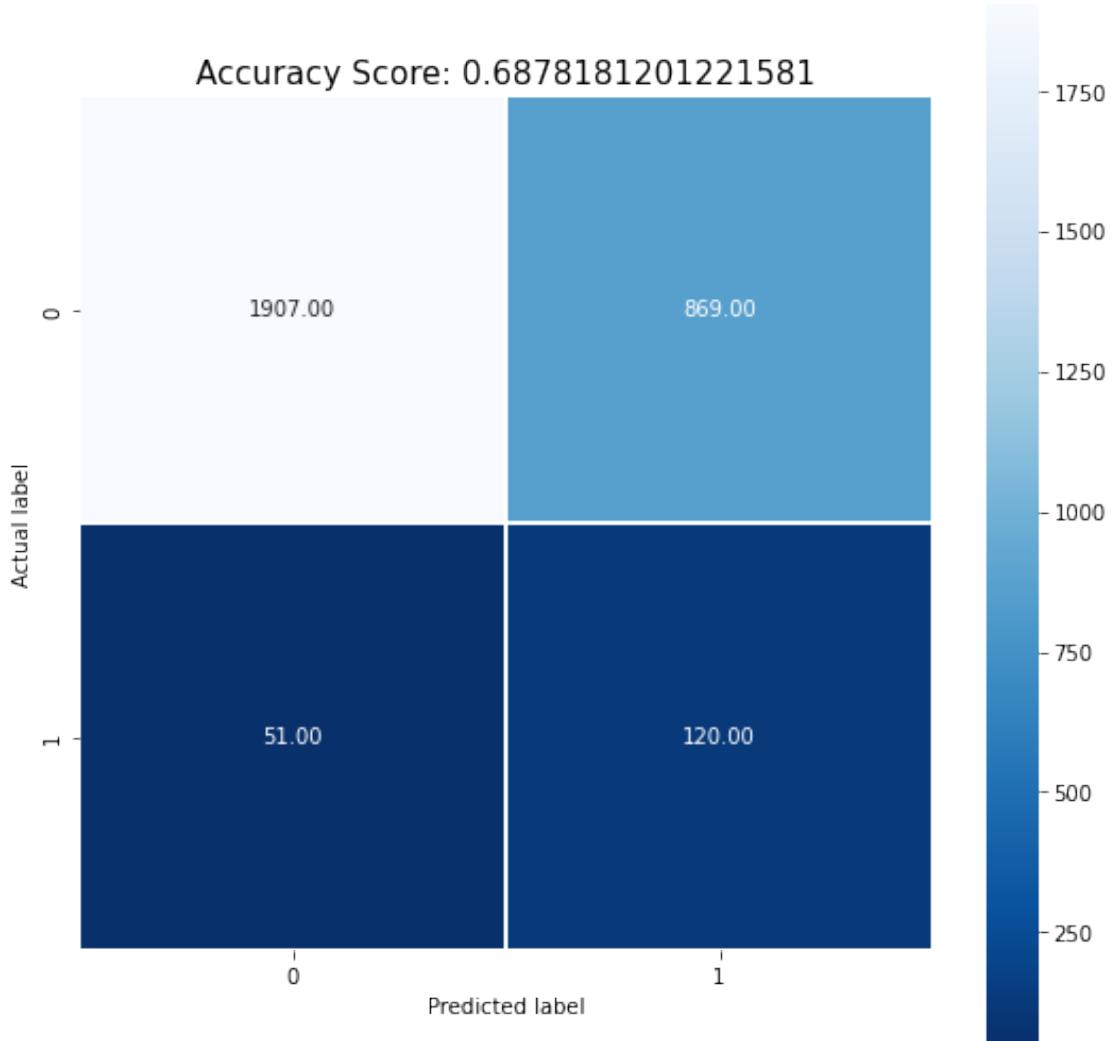
print(score2)
```

0.6878181201221581

```
[20]: confusion_matrix2 = metrics.confusion_matrix(Y_test,predictions)
print(confusion_matrix2)
```

`[[1907 869]
 [ 51 120]]`

```
[21]: plt.figure(figsize=(9,9))
sns.heatmap(confusion_matrix2, annot=True, fmt=".2f", linewidths=.5, square = True,
    ↪True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {}'.format(score2)
plt.title(all_sample_title, size = 15);
```



The above code shows logistic regression applied to the model with specific features. The confusion matrix shows us that the accuracy has actually increased in comparison to our base model. However, the difference between our base model and feature selection model (via stepwise) is only about a 2% increase.

```
[22]: #Code from https://chrisalbon.com/code/machine_learning/trees_and_forests/
    ↪feature_selection_using_random_forest/

randomX_train, randomX_test, randomY_train, randomY_test = □
    ↪train_test_split(x_vars, y, test_size=0.3, random_state=0)

rus3 = RandomUnderSampler(random_state=77)
x_train_under3, y_train_under3 = rus3.fit_resample(randomX_train, randomY_train)
```

```

clf = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)
clf.fit(x_train_under3, y_train_under3)
cols = df.columns.tolist()
cols.remove('ORIGIN')
cols.remove('CARAVAN')

ls = []
# Print the name and gini importance of each feature
for feature in zip(cols, clf.feature_importances_):
    print(feature)

```

('MOSTYPE', 0.03474891115363239)  
 ('MAANTHUI', 0.004504134266575562)  
 ('MGEMOMV', 0.011528704229786893)  
 ('MGEMLEEF', 0.010986631734914095)  
 ('MOSHOOFD', 0.020904331562674162)  
 ('MGODRK', 0.013107503234025934)  
 ('MGODPR', 0.020866642176881375)  
 ('MGODOV', 0.014499417707442607)  
 ('MGODGE', 0.0174914546312731)  
 ('MRELGE', 0.018346752390950266)  
 ('MRELSA', 0.011064958258116181)  
 ('MRELOV', 0.014916178050507338)  
 ('MFALLEEN', 0.017305716475165406)  
 ('MFGEKIND', 0.019420754281766005)  
 ('MFWEKIND', 0.020007177903020732)  
 ('MOPLHOOG', 0.01911058992259323)  
 ('MOPLMIDD', 0.02121244257666205)  
 ('MOPLLAAG', 0.02606945544158058)  
 ('MBERHOOG', 0.01740821504346696)  
 ('MBERZELF', 0.008888265046783956)  
 ('MBERBOER', 0.01208259524785035)  
 ('MBERMIDD', 0.02022320723032383)  
 ('MBERARBG', 0.01892251999203062)  
 ('MBERARBO', 0.017816748680177248)  
 ('MSKA', 0.01495200821543731)  
 ('MSKB1', 0.016706128632715794)  
 ('MSKB2', 0.0175105014601339)  
 ('MSKC', 0.019035738937072298)  
 ('MSKD', 0.013890251636793827)  
 ('MHUUR', 0.021549841554439488)  
 ('MHKOOP', 0.02120600893636681)  
 ('MAUT1', 0.019416510507939556)  
 ('MAUT2', 0.014609897095301258)  
 ('MAUTO', 0.015553011308342312)  
 ('MZFONDS', 0.01604253421564284)  
 ('MZPART', 0.01546138416770067)

('MINKM30', 0.022892317392993817)  
('MINK3045', 0.019759888926853847)  
('MINK4575', 0.025926478648800152)  
('MINK7512', 0.014496810264726444)  
('MINK123M', 0.008577278726249919)  
('MINKGEM', 0.019227055081187834)  
('MKOOPKLA', 0.025469720858062254)  
('PWAPART', 0.020807439164580954)  
('PWABEDR', 0.0005895036537455104)  
('PWALAND', 0.0006612578489321803)  
('PPERSAUT', 0.05516781098974348)  
('PBESAUT', 0.0010939235408145508)  
('PMOTSCO', 0.0028401805118886663)  
('PVRAAUT', 2.649150083159275e-05)  
('PAANHANG', 0.0011605396469442709)  
('PTRACTOR', 0.0013050309332071877)  
('PWERKT', 0.0)  
('PBROM', 0.0035664178190207623)  
('PLEVEN', 0.006521232349126935)  
('PPERSONG', 0.0008130296010780511)  
('PGEZONG', 0.0011082770980095202)  
('PWAOREG', 0.000560122206453803)  
('PBRAND', 0.04575330928465032)  
('PZEILPL', 0.00030670762945486055)  
('PPLEZIER', 0.002337260133772068)  
('PFIETS', 0.0027850354602321015)  
('PINBOED', 0.00027725683212840587)  
('PBYSTAND', 0.003278575345977462)  
('AWAPART', 0.014282986718919128)  
('AWABEDR', 0.0005383894155893424)  
('AWALAND', 0.0005770141356042792)  
('APERSAUT', 0.035739977739156124)  
('ABESAUT', 0.0010533402297640546)  
('AMOTSCO', 0.0023239306714063386)  
('AVRAAUT', 3.1226736848909785e-05)  
('AAANHANG', 0.0010813240448958046)  
('ATRACTOR', 0.0011931600183535463)  
('AWERKT', 0.0)  
('ABROM', 0.0033230546491931088)  
('ALEVEN', 0.005354704141849184)  
('APERSONG', 0.0007276771886486227)  
('AGEZONG', 0.001081564220935082)  
('AWAOREG', 0.0005836646016891815)  
('ABRAND', 0.018535020823020762)  
('AZEILPL', 0.0003173570548851277)  
('APLEZIER', 0.00231638153771194)  
('AFIETS', 0.003209981166127026)  
('AINBOED', 0.00026737238714706407)

```
('ABYSTAND', 0.0027857971627013893)
```

```
[23]: # Create a selector object that will use the random forest classifier to
      # identify
      # features that have an importance of more than 0.02
sfm = SelectFromModel(clf, threshold=0.02)

sfm.fit(x_train_under3, y_train_under3)

# Print the names of the most important features
for feature_list_index in sfm.get_support(indices=True):
    print(cols[feature_list_index])
ls.append(cols[feature_list_index])
```

```
MOSTYPE
MOSHOOFD
MGODPR
MFWEKIND
MOPLMIDD
MOPLLAAG
MBERMIDD
MHUUR
MHKOOP
MINKM30
MINK4575
MKOOPKLA
PWAPART
PPERSAUT
PBRAND
APERSAUT
```

The printed list above are the most important features obtained by random forest based on feature importance feature selection

```
[24]: fi = pd.DataFrame({'feature':cols, 'importance':clf.feature_importances_})
print(fi)
```

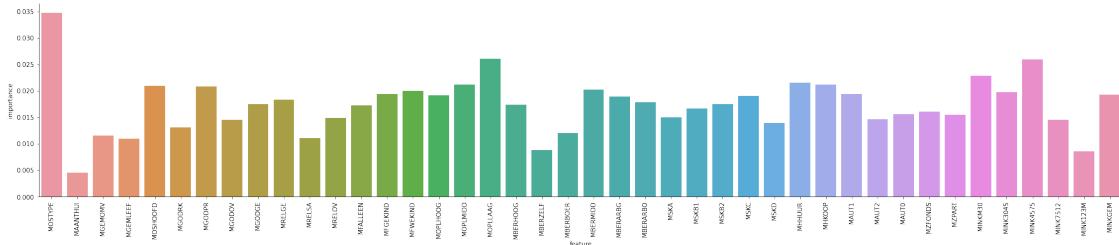
	feature	importance
0	MOSTYPE	0.034749
1	MAANTHUI	0.004504
2	MGEMOMV	0.011529
3	MGEMLEEF	0.010987
4	MOSHOOFD	0.020904
..	...	...
80	AZEILPL	0.000317
81	APLEZIER	0.002316
82	AFIETS	0.003210
83	AINBOED	0.000267

84 ABYSTAND 0.002786

[85 rows x 2 columns]

```
[25]: sns.catplot(x = "feature", y = "importance", data = fi[:42], kind = 'bar', aspect = 5).set_xticklabels(rotation = 90) #https://gust.dev/python/  
    ↪dimensionality-reduction  
#More information on catplot: https://seaborn.pydata.org/generated/seaborn.  
    ↪catplot.html
```

[25]: <seaborn.axisgrid.FacetGrid at 0x7fe64c833d10>



From the code above we got the most important features and graphed the importance value of most features in our dataset.

```
[26]: #predict using only important features.
```

```
X_important_train = sfm.transform(x_train_under3)
X_important_test = sfm.transform(randomX_test)

clf_important = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)

clf_important.fit(X_important_train, y_train_under3)
```

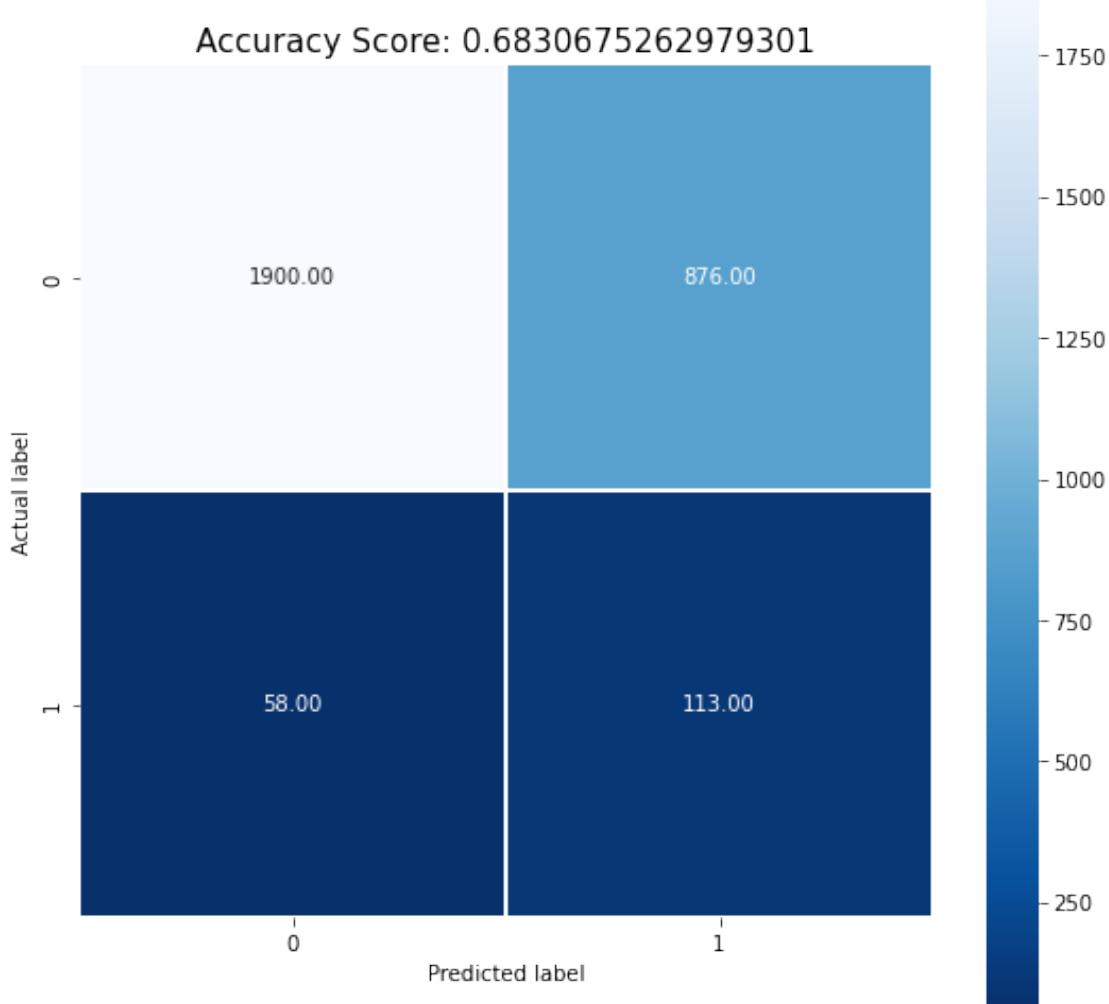
[26]: RandomForestClassifier(n\_estimators=10000, n\_jobs=-1, random\_state=0)

```
[27]: y_important_pred = clf_important.predict(X_important_test)
acc_important = accuracy_score(randomY_test, y_important_pred)

print(acc_important)
```

0.6830675262979301

```
[28]: randomForestcm = metrics.confusion_matrix(randomY_test,y_important_pred)
plt.figure(figsize=(9,9))
sns.heatmap(randomForestcm, annot=True, fmt=".2f", linewidths=.5, square =True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(acc_important)
plt.title(all_sample_title, size = 15);
```



From the confusion matrix above we can see that our accuracy score for the random forest important feature model is lower than our logistic regression models in prediction.

```
[29]: #Random Forest based on Principal component analysis feature selection
```

```
pca_df = df.copy()
```

```
pca_df.head()

vars = pca_df.drop(['ORIGIN'], axis = 1)
pca_x_vars = vars.drop(['CARAVAN'], axis=1)

pca_y = pca_df['CARAVAN']
```

[30]: # standardize the data

```
X_std = StandardScaler().fit_transform(pca_x_vars)
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
```

Covariance matrix

```
[[ 1.00010182 -0.04033167 -0.0065945 ... -0.0255354 -0.01633792
-0.04375385]
[-0.04033167  1.00010182 -0.00431565 ... -0.01252764  0.0319394
-0.005852 ]
[-0.0065945 -0.00431565  1.00010182 ...  0.01848839  0.01092878
 0.03085877]
...
[-0.0255354 -0.01252764  0.01848839 ...  1.00010182  0.00204416
 0.00712214]
[-0.01633792  0.0319394   0.01092878 ...  0.00204416  1.00010182
 0.01727922]
[-0.04375385 -0.005852     0.03085877 ...  0.00712214  0.01727922
 1.00010182]]
```

`X_std = StandardScaler().fit_transform(x_vars)` - This code fits our `x_vars` data and transforms it at the same time. `standardscaler()` normalizes our data for fitting and transforming so that each column will have a mean of 0 and a standard deviation of 1.

`mean_vec = np.mean(X_std, axis=0)` - We get the mean from the dataset that has been normalized, fitted and transformed.

`cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)` - After getting the normalized, fitted, transformed data and mean we create a covariance matrix. A covariance matrix tells us the distribution magnitude and direction of the data. Covariance matrix also estimates the level to which two variables vary together. For more information on covariance matrix: <https://numpy.org/doc/stable/reference/generated/numpy.cov.html>

[31]: #Perform eigendecomposition on covariance matrix

```
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

Eigenvectors

```
[[ -1.80741073e-01  1.90012238e-01 -1.08200120e-02 ... -1.19823729e-02
```

```

1.86481597e-03 -3.99575753e-04]
[ 6.86446049e-05 -9.97766755e-03  1.42353206e-02 ... -5.62264287e-03
 1.64017734e-03 -1.88420999e-03]
[ 1.21471618e-01  2.78081107e-01  9.89312897e-02 ...  1.32823262e-03
 -4.33591381e-03 -1.66503718e-03]
...
[ 1.26127757e-02 -1.12937278e-02  2.14048183e-02 ...  9.27489842e-03
 -2.15410748e-02 -4.32546502e-03]
[ 1.32624344e-02 -5.62562226e-03 -1.97657957e-02 ...  1.93069813e-03
 -1.94621753e-03 -5.83645630e-03]
[ 2.36962778e-02  1.36883444e-02 -4.79283762e-02 ...  4.76677274e-02
 1.50691498e-02 -3.00046503e-02]]

```

### Eigenvalues

```

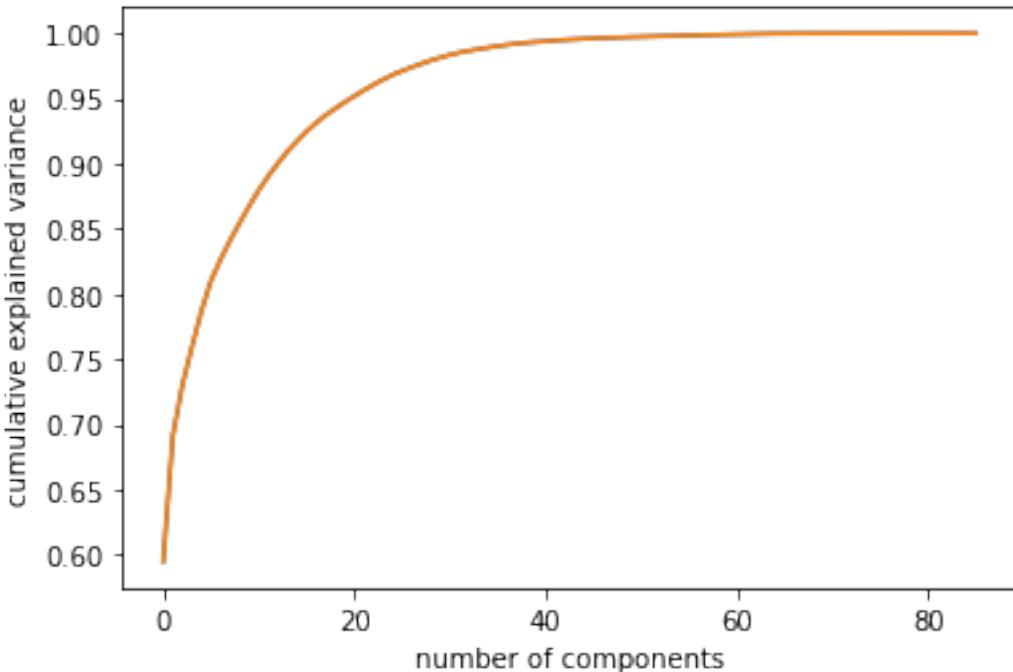
[9.38114298e+00 4.90789884e+00 3.98336652e+00 3.34322942e+00
 2.92824840e+00 2.64384083e+00 2.26219960e+00 2.23659410e+00
 2.14010672e+00 2.10857567e+00 2.00996208e+00 1.95746407e+00
 1.91389055e+00 1.90310544e+00 1.86888982e+00 1.84630241e+00
 1.79586996e+00 1.81596659e+00 1.72663381e+00 1.68813177e+00
 1.55440514e+00 1.62031830e+00 1.64806302e+00 1.38457956e+00
 1.45373536e+00 1.42746322e+00 1.31847563e+00 1.27461790e+00
 1.19260062e+00 1.20852203e+00 1.13484625e+00 1.05794702e+00
 9.73410107e-01 9.48684989e-01 8.76109506e-01 8.52179201e-01
 7.13488960e-01 7.73204970e-01 7.95741040e-01 8.14269572e-01
 7.53824663e-01 6.19100476e-01 6.79104244e-01 5.62045509e-01
 5.95793872e-01 4.63305819e-01 4.69505046e-01 3.81630781e-01
 3.21680502e-01 3.38308150e-01 1.97275996e-01 1.70937409e-01
 1.46517388e-01 1.44544847e-01 1.34015386e-01 1.26700732e-01
 1.15349363e-01 1.07555511e-01 1.03837949e-01 9.80237167e-02
 5.45284011e-03 2.62882378e-04 6.41300056e-04 9.12843707e-02
 8.99486814e-02 8.16324392e-02 7.75176397e-02 5.44035528e-02
 6.23525567e-02 1.52485450e-02 6.65622373e-02 1.70374630e-02
 4.40246373e-02 2.63267265e-02 1.84601060e-02 1.94393481e-02
 3.45140033e-02 3.94323005e-02 3.13134055e-02 2.86393954e-02
 2.79075675e-02 7.48788307e-02 3.22847283e-02 1.97996292e-02
 3.61504284e-02]

```

Above we perform an eigendecomposition on covariance matrix where the eigenvalues represents all of the variance within the entire dataset (Labs, D. D).

```
[32]: pca = PCA().fit(vars)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

```
[32]: Text(0, 0.5, 'cumulative explained variance')
```



```
[33]: pca = PCA(0.75).fit(vars)
pca.n_components_
#about 75% of our variance can be explained by the first 4 features of our
#dataset
print(f"About 75% of our variance in the dataset can be explained by the first
{pca.n_components_} features")
```

About 75% of our variance in the dataset can be explained by the first 4 features

From the principal component analysis and variance ratio graph we can observe that about 95% of our variance can be explained by the first 20 features of our dataset. And from the code directly above we can see that the first 4 features from our dataset can explain about 75% of the variance in our dataset. Therefore we will continue and create a predictive model based on the first 4 features in our dataset.

```
[34]: four_features = vars.iloc[:, :4].copy()
pca_y = df['CARAVAN'].copy()

#https://thispointer.com/select-first-n-columns-of-pandas-dataframe/
```

```
[39]: pcaX_train, pcaX_test, pcaY_train, pcaY_test = train_test_split(four_features,
#pca_y, test_size=0.3, random_state=0)
```

```
rus4 = RandomUnderSampler(random_state=77)
x_train_under4, y_train_under4 = rus4.fit_resample(pcaX_train, pcaY_train)

pca_random = RandomForestClassifier(n_estimators=10000, random_state=0, n_jobs=-1)

pca_random.fit(x_train_under4, y_train_under4)

pca_y_pred = pca_random.predict(pcaX_test)
pca_acc = accuracy_score(pcaY_test, pca_y_pred)
```

```
[36]: print(pca_acc)
```

0.5846623685103495

```
[37]: pca_cm = metrics.confusion_matrix(pcaY_test,pca_y_pred)
plt.figure(figsize=(9,9))
sns.heatmap(pca_cm, annot=True, fmt=".2f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {}'.format(pca_acc)
plt.title(all_sample_title, size = 15);
```



```
[38]: print(f"The percent accuracy score of the base model for logistic regression is:  
      ↪ {score *100:,.2f}%)  
print(f"The percent accuracy score of the stepwise model for logistic  
      ↪ regression is: {score2*100:,.2f}%)  
print(f"The percent accuracy score of the important features with random forest  
      ↪ model is: {acc_important*100:,.2f}%)  
print(f"The percent accuracy score of the model using PCA is: {pca_acc*100:,.  
      ↪ 2f}%)")
```

The percent accuracy score of the base model for logistic regression is: 66.88%  
The percent accuracy score of the stepwise model for logistic regression is:  
68.78%

The percent accuracy score of the important features with random forest model  
is: 68.31%

The percent accuracy score of the model using PCA is: 58.47%

All of our models used undersampling technique due to the extreme unbalance in the data. From our models we can observe that the best model for predicting which features show which customers are more likely to be insured with CARAVAN is the stepwise logistic regression model, with the important features random forest model not too far behind. The lowest model for prediction is the PCA predictive model. Although the PCA model is the lowest for prediction it used only four features that explained 75% of variation our dataset.

## #Conclusion

Overall, in this lab we used logistic regression, PCA, and random forest techniques to determine which features are important in predicting which customers who also bought caravan insurance. By doing so we can try and predict who to offer the CARAVAN policy to. During the EDA section we noticed that the data was extremely unbalanced, therefore we decided to use an undersampling technique to make our predictive models more accurate. From our models above we noticed that the best model for prediction was the stepwise model for logistic regression having an 68.78% success rate in predicting correctly. The stepwise model used only important features obtained through the stepwise process. The worst performing model was the PCA model with a 58.47% success rate in prediction. The PCA model showed us that about 75% of variation in our dataset was determined by the first four features in the dataset, therefore when creating the dataset we only used the first four features. The other two models also did relatively well in relation to the best model with differences in predictability ranging from 0.4% to 1.9%. In conclusion, the best model for predict who to offer the CARAVAN policy to is the stepwise logistic regression model.

## #References

1. Labs, D. D. (2016, August 31). Principal component analysis with python. District Data Labs. Retrieved October 16, 2022, from <https://districtdatalabs.silvrback.com/principal-component-analysis-with-python>
2. What is logistic regression? Statistics Solutions. (2022, June 14). Retrieved October 16, 2022, from <https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-logistic-regression/>
3. Sklearn.ensemble.randomforestclassifier. scikit. (n.d.). Retrieved October 16, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

# Notebook

August 2, 2023

**Leng Her**

**Lab 7**

**October 23 2022**

#Overview

In order to understand where wines originated from which of three regions, we will be utilizing different ensemble methods and classifiers. We will be using bagging, boosting, random forest, KNeighbors, and stacking classifier methods for differentiating the three regions of where the wines originated. Bagging, boosting and random forest are ensemble methods, a machine learning technique that consists of several base model that produces one predictive model (Lutins, E). Stacking is also an ensemble technique where predictions from multiple classifiers are used for new features to train a meta classifier (Ceballos, F). By using these ensemble models and methods we will attempt to predict which of the three regions the wines were originated.

References:

<https://medium.com/@saugata.paul1010/ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9c66cb271674>

<https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f>

<https://towardsdatascience.com/stacking-classifiers-for-higher-predictive-performance-566f963e4840>

[https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_wine.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html)

#Data

The dataset below was obtained from one of sciklars datasets. The origins of the dataset is from UCI ML. For more information on the data set please refer to the link: [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_wine.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html)

```
[ ]: import six
```

```
[ ]: from sklearn.datasets import load_wine
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
```

```

from sklearn.ensemble import GradientBoostingClassifier
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from mlxtend.classifier import StackingClassifier
from mlxtend.classifier import StackingCVClassifier
from mlxtend.classifier import EnsembleVoteClassifier

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

```

```

[ ]: wine = load_wine()
# Convert to Pandas Dataframe
wine_dataframe = pd.DataFrame(data = np.c_[wine['data'], wine['target']],
                               columns = wine['feature_names'] + ['target'])
print(wine.DESCR)

```

... \_wine\_dataset:

Wine recognition dataset

\*\*Data Set Characteristics:\*\*

- :Number of Instances: 178 (50 in each of three classes)
- :Number of Attributes: 13 numeric, predictive attributes and the class
- :Attribute Information:
  - Alcohol
  - Malic acid
  - Ash
  - Alcalinity of ash
  - Magnesium
  - Total phenols
  - Flavanoids
  - Nonflavanoid phenols
  - Proanthocyanins
  - Color intensity
  - Hue
  - OD280/OD315 of diluted wines
  - Proline

```
- class:  
  - class_0  
  - class_1  
  - class_2
```

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None

:Class Distribution: class\_0 (59), class\_1 (71), class\_2 (48)

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.

<https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -

An Extendible Package for Data Exploration, Classification and Correlation.  
Institute of Pharmaceutical and Food Analysis and Technologies,  
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository  
[<https://archive.ics.uci.edu/ml>]. Irvine, CA: University of California,  
School of Information and Computer Science.

.. topic:: References

(1) S. Aeberhard, D. Coomans and O. de Vel,  
Comparison of Classifiers in High Dimensional Settings,  
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Technometrics).

The data was used with many others for comparing various  
classifiers. The classes are separable, though only RDA  
has achieved 100% correct classification.

(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))  
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,  
"THE CLASSIFICATION PERFORMANCE OF RDA"  
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of  
Mathematics and Statistics, James Cook University of North Queensland.  
(Also submitted to Journal of Chemometrics).

```
[ ]: wine_dataframe.head()
```

```
[ ]:   alcohol  malic_acid    ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43                  15.6       127.0          2.80
1      13.20        1.78  2.14                  11.2       100.0          2.65
2      13.16        2.36  2.67                  18.6       101.0          2.80
3      14.37        1.95  2.50                  16.8       113.0          3.85
4      13.24        2.59  2.87                  21.0       118.0          2.80

  flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity  hue  \
0         3.06                  0.28              2.29            5.64  1.04
1         2.76                  0.26              1.28            4.38  1.05
2         3.24                  0.30              2.81            5.68  1.03
3         3.49                  0.24              2.18            7.80  0.86
4         2.69                  0.39              1.82            4.32  1.04

od280/od315_of_diluted_wines  proline  target
0                      3.92  1065.0     0.0
1                      3.40  1050.0     0.0
2                      3.17  1185.0     0.0
3                      3.45  1480.0     0.0
4                      2.93    735.0     0.0
```

```
#EDA (Exploratory Data Analysis)
```

```
[ ]: wine_dataframe.round(4).describe()
```

```
[ ]:      alcohol  malic_acid      ash  alcalinity_of_ash  magnesium  \
count    178.000000  178.000000  178.000000  178.000000  178.000000
mean     13.000618   2.336348   2.366517   19.494944  99.741573
std      0.811827   1.117146   0.274344   3.339564  14.282484
min     11.030000   0.740000   1.360000  10.600000  70.000000
25%    12.362500   1.602500   2.210000  17.200000  88.000000
50%    13.050000   1.865000   2.360000  19.500000  98.000000
75%    13.677500   3.082500   2.557500  21.500000 107.000000
max    14.830000   5.800000   3.230000  30.000000 162.000000

      total_phenols  flavanoids  nonflavanoid_phenols  proanthocyanins  \
count    178.000000  178.000000  178.000000  178.000000
mean     2.295112   2.029270   0.361854   1.590899
std      0.625851   0.998859   0.124453   0.572359
min     0.980000   0.340000   0.130000   0.410000
25%    1.742500   1.205000   0.270000   1.250000
50%    2.355000   2.135000   0.340000   1.555000
75%    2.800000   2.875000   0.437500   1.950000
max    3.880000   5.080000   0.660000   3.580000

      color_intensity      hue  od280/od315_of_diluted_wines      proline  \
count    178.000000  178.000000  178.000000  178.000000
mean     5.058090   0.957449   2.611685  746.893258
std      2.318286   0.228572   0.709990  314.907474
min     1.280000   0.480000   1.270000  278.000000
25%    3.220000   0.782500   1.937500  500.500000
50%    4.690000   0.965000   2.780000  673.500000
75%    6.200000   1.120000   3.170000  985.000000
max    13.000000   1.710000   4.000000 1680.000000

      target
count  178.000000
mean    0.938202
std     0.775035
min     0.000000
25%    0.000000
50%    1.000000
75%    2.000000
max    2.000000
```

```
[ ]: wine_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
```

```
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
 ---  --  
 0   alcohol           178 non-null    float64 
 1   malic_acid        178 non-null    float64 
 2   ash               178 non-null    float64 
 3   alcalinity_of_ash 178 non-null    float64 
 4   magnesium         178 non-null    float64 
 5   total_phenols     178 non-null    float64 
 6   flavanoids        178 non-null    float64 
 7   nonflavanoid_phenols 178 non-null    float64 
 8   proanthocyanins  178 non-null    float64 
 9   color_intensity   178 non-null    float64 
 10  hue               178 non-null    float64 
 11  od280/od315_of_diluted_wines 178 non-null    float64 
 12  proline           178 non-null    float64 
 13  target             178 non-null    float64 

dtypes: float64(14)
memory usage: 19.6 KB
```

Check for null values. No Null values

```
[ ]: print(f"The number of columns in the dataset is: {wine_dataframe.shape[1]}")
print(f"The number of rows in the dataset is: {wine_dataframe.shape[0]}")
```

```
The number of columns in the dataset is: 14
The number of rows in the dataset is: 178
```

```
[ ]: wine_dataframe['target']
```

```
[ ]: 0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
...
173    2.0
174    2.0
175    2.0
176    2.0
177    2.0
Name: target, Length: 178, dtype: float64
```

```
[ ]: class1 = len(wine_dataframe[wine_dataframe['target']== 0.0])
class2 = len(wine_dataframe[wine_dataframe['target']== 1.0])
class3 = len(wine_dataframe[wine_dataframe['target']== 2.0])

print("Class 1 Occurrences: " + str(class1))
```

```

print("Class 2 Occurrences: " + str(class2))
print("Class 3 Occurrences: " + str(class3))

```

Class 1 Occurrences: 59  
 Class 2 Occurrences: 71  
 Class 3 Occurrences: 48

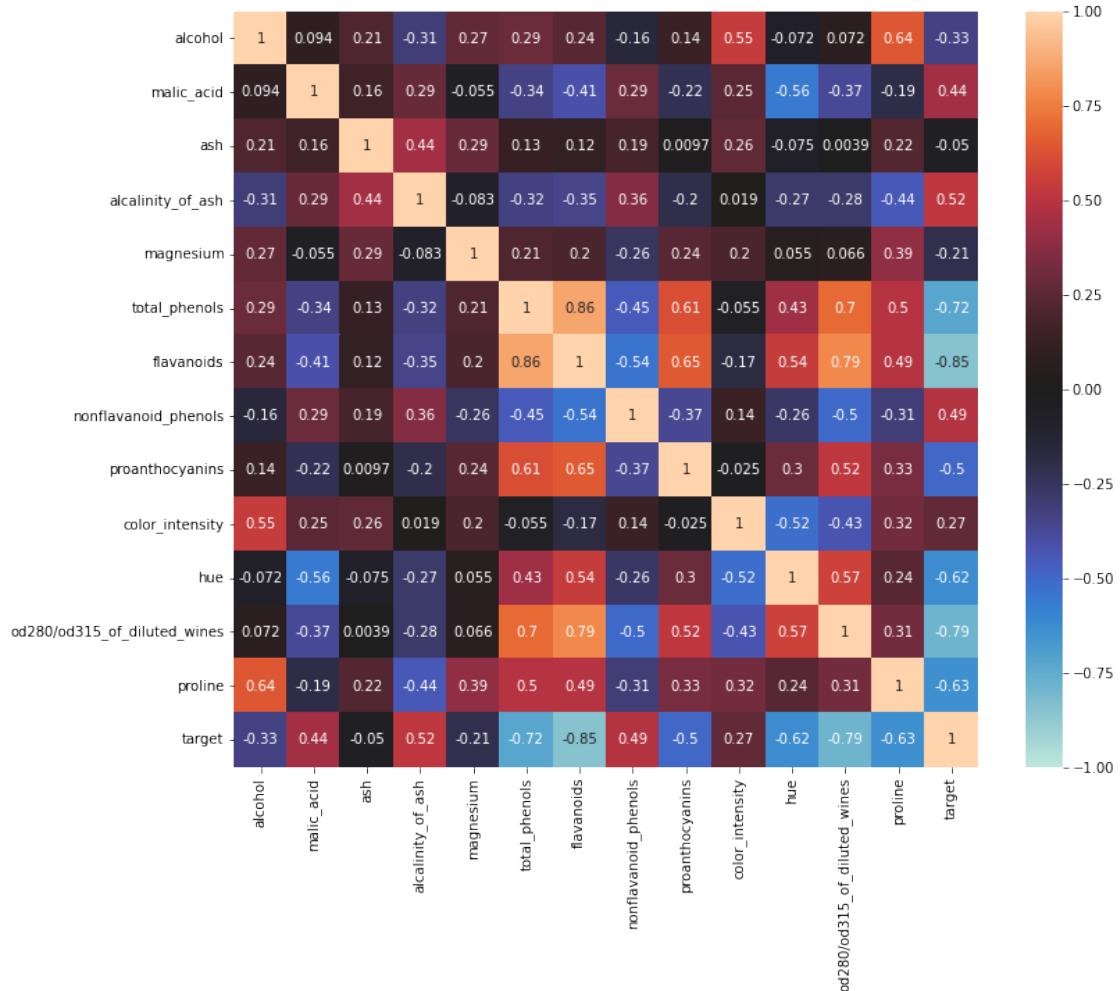
```

[ ]: fig, ax = plt.subplots(figsize = (12,10))

corrmat = wine_dataframe.corr()
sns.heatmap(corrmat,-1,1,ax=ax, center = 0, annot = True)

```

[ ]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7fdf671a7b10>



From the correlation matrix, we can observe that proline and alcohol have a moderately strong positive correlation along with proline and total\_phenols; diluted wines has a moderately strong positive correlation with total\_phenols, flavnoids, proanthocyanins and hue.

The strongest positive correlation between features is flavnoids and total\_phenols with a correlation of 0.86.

```
[ ]: plt.close(); #code from: https://medium.com/@saugata.paul1010/
    ↪ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9
sns.set_style("whitegrid");
sns.pairplot(wine_dataframe, hue="target", size=3);
plt.show()
```

Output hidden; open in <https://colab.research.google.com> to view.

From the above plots we can see that proline is the best separator of class 1 compared to the other features. A close second to proline is the dilluted wines feature for class 1. In addiotion, both hue and dilluted wines feature show promise in seperating class 3 from the other two classes. All other features based on the plots do not seperate the three classes as accurately.

#Models

```
[ ]: # Code Taken From
# https://medium.com/@saugata.paul1010/
    ↪ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9

#baseline models

X = wine_dataframe.drop("target", 1)
y = wine_dataframe.target

RANDOM_SEED = 550

#Base Learners
rf_clf = RandomForestClassifier(n_estimators=10, random_state=RANDOM_SEED)
knn_clf = KNeighborsClassifier(n_neighbors=2)
svc_clf = SVC(C=10000.0, kernel='rbf', random_state=RANDOM_SEED)
lr_clf = LogisticRegression(C=20000, penalty='l2', random_state=RANDOM_SEED)

classifier_array = [rf_clf, knn_clf, svc_clf, lr_clf]
labels = [clf.__class__.__name__ for clf in classifier_array]

normal_accuracy = []
normal_std = []
bagging_accuracy = []
bagging_std = []
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the
argument 'labels' will be keyword-only
```

```
[ ]: for clf in classifier_array:
    cv_scores = cross_val_score(clf, X, y, cv=3, n_jobs=-1)

    normal_accuracy.append(np.round(cv_scores.mean(),4))
    normal_std.append(np.round(cv_scores.std(),4))

    print("Accuracy: %0.4f (+/- %0.4f) [Normal %s]" % (cv_scores.mean(),cv_scores.std(), clf.__class__.__name__))

```

Accuracy: 0.9161 (+/- 0.0543) [Normal RandomForestClassifier]  
 Accuracy: 0.6465 (+/- 0.0662) [Normal KNeighborsClassifier]  
 Accuracy: 0.9273 (+/- 0.0476) [Normal SVC]  
 Accuracy: 0.9108 (+/- 0.0925) [Normal LogisticRegression]

From our baseline models we can see that the most accurate model is the SVC model and following it is, the normal logistic regression model. The worse performing model would be the KNN model with only a ~65% accuracy, which could be due to only observing 2 nearest neighbors.

```
[ ]: for clf in classifier_array:
    cv_scores = cross_val_score(clf, X, y, cv=3, n_jobs=-1)
    bagging_clf = BaggingClassifier(clf, max_samples=0.4, max_features=3,random_state=RANDOM_SEED)
    bagging_scores = cross_val_score(bagging_clf, X, y, cv=3, n_jobs=-1)

    bagging_accuracy.append(np.round(bagging_scores.mean(),4))
    bagging_std.append(np.round(bagging_scores.std(),4))
    print("Accuracy: %0.4f (+/- %0.4f) [Bagging %s]\n" % (bagging_scores.mean(), bagging_scores.std(), clf.__class__.__name__))

```

Accuracy: 0.9158 (+/- 0.0236) [Bagging RandomForestClassifier]  
 Accuracy: 0.8990 (+/- 0.0363) [Bagging KNeighborsClassifier]  
 Accuracy: 0.8933 (+/- 0.0421) [Bagging SVC]  
 Accuracy: 0.9160 (+/- 0.0356) [Bagging LogisticRegression]

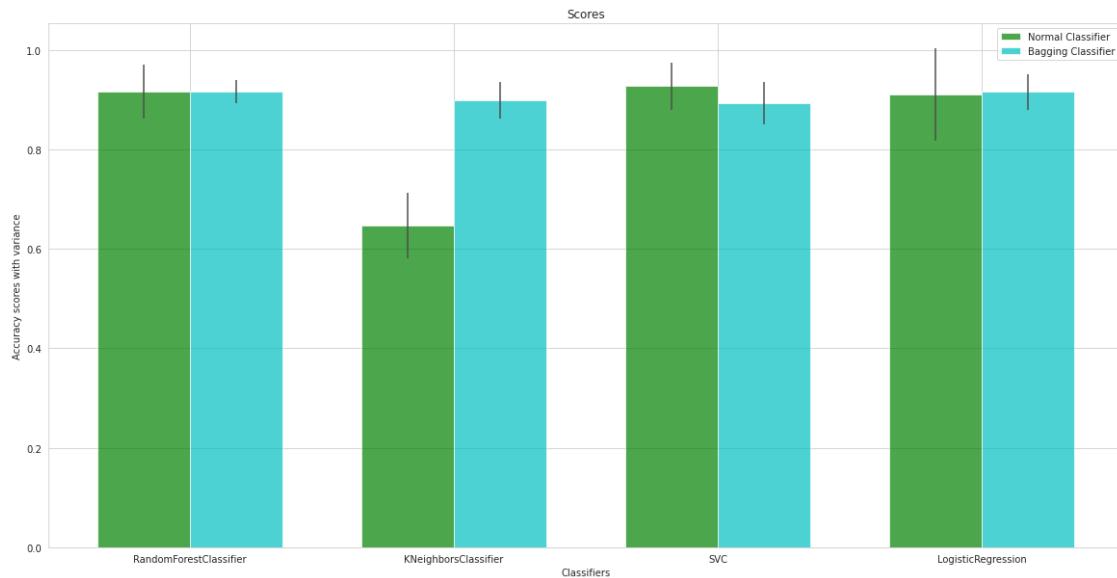
```
[ ]: fig, ax = plt.subplots(figsize=(20,10))
n_groups = 4
index = np.arange(n_groups)
bar_width = 0.35
opacity = .7
error_config = {'ecolor': '0.3'}
normal_clf = ax.bar(index, normal_accuracy, bar_width, alpha=opacity,color='g', yerr=normal_std, error_kw=error_config, label='Normal Classifier')
```

```

bagging_clf = ax.bar(index + bar_width, bagging_accuracy, bar_width,
                     alpha=opacity, color='c', yerr=bagging_std, error_kw=error_config,
                     label='Bagging Classifier')
ax.set_xlabel('Classifiers')
ax.set_ylabel('Accuracy scores with variance')
ax.set_title('Scores')
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels((labels))
ax.legend()

```

[ ]: <matplotlib.legend.Legend at 0x7ff6c1a4bf10>



From the bar graph we can observe that bagging had little affect on our normal accuracy scores except for the KNeighbors Classifier with a dramatic increase in accuracy. Each classifier did increase by a little except for one, the SVC score actually decreased from the bagging method.

```

[ ]: boosting_acc = []
boosting_std = []
ada_boost = AdaBoostClassifier(n_estimators=5)
grad_boost = GradientBoostingClassifier(n_estimators=10)
xgb_boost = XGBClassifier(max_depth=5, learning_rate=0.001)
ensemble_clf = EnsembleVoteClassifier(clfs=[ada_boost, grad_boost, xgb_boost],
                                      voting='hard')
boosting_labels = ['Ada Boost', 'Gradient Boost', 'XG Boost', 'Ensemble']
for clf, label in zip([ada_boost, grad_boost, xgb_boost, ensemble_clf],
                      boosting_labels):
    scores = cross_val_score(clf, X, y, cv=3, scoring='accuracy')
    boosting_acc.append(np.round(scores.mean(), 4))

```

```

boosting_std.append(np.round(scores.std(),4))
print("Accuracy: {:.3f}, Variance: (+/-) {:.3f} [{2}]".format(scores.
˓→mean(), scores.std(), label))

```

```

Accuracy: 0.843, Variance: (+/-) 0.035 [Ada Boost]
Accuracy: 0.853, Variance: (+/-) 0.070 [Gradient Boost]
Accuracy: 0.871, Variance: (+/-) 0.070 [XG Boost]
Accuracy: 0.876, Variance: (+/-) 0.056 [Ensemble]

```

From the accuracy scores we can see that boosting showed a decrease in accuracy compared to our normal tests scores. The best accuracy score out of the boosting methods was the Ensemble method with the worst accuracy score from the tested methods being ADA boost.

```
[ ]: #Decision Regions for all the boosting algorithms.
proline = wine_dataframe[['proline','alcohol']]
X = np.array(proline)
y = np.array(y)
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools
gs = gridspec.GridSpec(2, 2)
fig = plt.figure(figsize=(20,16))
for clf, label, grd in zip([ada_boost, grad_boost, xgb_boost, ensemble_clf], u
˓→boosting_labels, itertools.product([0, 1], repeat=2)):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y.astype(np.integer), clf=clf, legend=2)
    plt.title(label)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
```

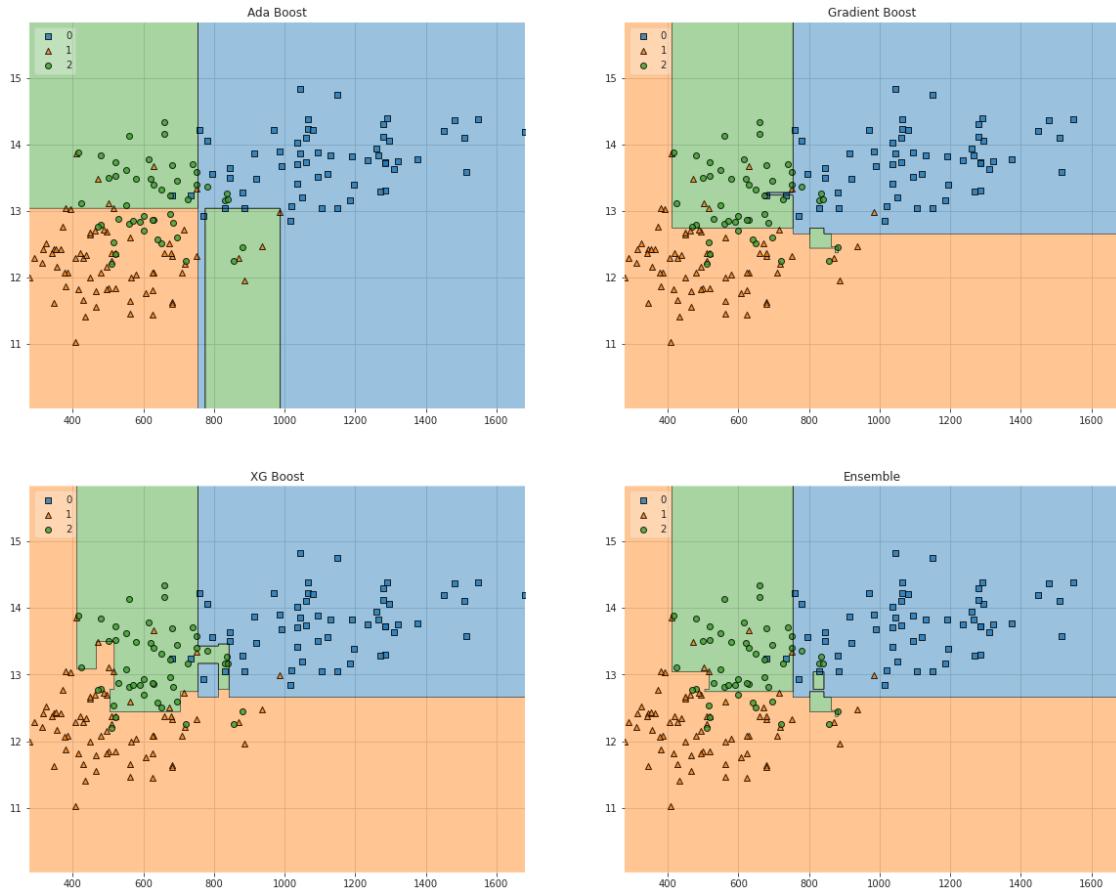
```

MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.

/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())

```



From the plotted decision regions we used proline and alcohol features as the more important features based on the plotted pair plots above. From the decison region plot above we can observe that each boosting method does a decent job out classifying each region based on the target variable. The best decision region plot would be between Ensemble, XG boost and Gradient Boost.

```
[46]: RANDOM_SEED = 0
X = wine_dataframe.drop("target", 1)
y = wine_dataframe.target

#Base Learners
rf_clf = RandomForestClassifier(n_estimators=10, random_state=RANDOM_SEED)
knn_clf = KNeighborsClassifier(n_neighbors=2)
svc_clf = SVC(C=10000.0, kernel='rbf', random_state=RANDOM_SEED)
lr_clf = LogisticRegression(C=20000, penalty='l2', random_state=RANDOM_SEED)
lr = LogisticRegression(random_state=RANDOM_SEED) # meta classifier

sclf = StackingClassifier(classifiers=[rf_clf, knn_clf, svc_clf, lr_clf], meta_classifier=lr)

classifier_array = [rf_clf, knn_clf, svc_clf, lr_clf, sclf]
labels = [clf.__class__.__name__ for clf in classifier_array]
acc_list = []
var_list = []
for clf, label in zip(classifier_array, labels):
    cv_scores = model_selection.cross_val_score(clf, X, y, cv=3, scoring='accuracy')
    print("Accuracy: %0.4f (+/- %0.4f) [%s]" % (cv_scores.mean(), cv_scores.std(), label))
    acc_list.append(np.round(cv_scores.mean(), 4))
    var_list.append(np.round(cv_scores.std(), 4))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning:
In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfsgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfsgs failed to converge (status=1):
```

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,  
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:  
ConvergenceWarning: lbfsgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,  
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:  
ConvergenceWarning: lbfsgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,  
Accuracy: 0.9046 (+/- 0.0442) [RandomForestClassifier]  
Accuracy: 0.6465 (+/- 0.0662) [KNeighborsClassifier]  
Accuracy: 0.9273 (+/- 0.0476) [SVC]  
Accuracy: 0.9108 (+/- 0.0925) [LogisticRegression]  
Accuracy: 0.9161 (+/- 0.0543) [StackingClassifier]
```

/usr/local/lib/python3.7/dist-packages/sklearn/linear\_model/\_logistic.py:818:  
ConvergenceWarning: lbfsgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,  
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:  
ConvergenceWarning: lbfsgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (`max_iter`) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

`extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,`

NOTE: Results printed again for readability

```
[49]: print("Accuracy: 0.9046 (+/- 0.0442) [RandomForestClassifier]")
print("Accuracy: 0.6465 (+/- 0.0662) [KNeighborsClassifier]")
print("Accuracy: 0.9273 (+/- 0.0476) [SVC]")
print("Accuracy: 0.9108 (+/- 0.0925) [LogisticRegression]")
print("Accuracy: 0.9161 (+/- 0.0543) [StackingClassifier]")
```

```
Accuracy: 0.9046 (+/- 0.0442) [RandomForestClassifier]
Accuracy: 0.6465 (+/- 0.0662) [KNeighborsClassifier]
Accuracy: 0.9273 (+/- 0.0476) [SVC]
Accuracy: 0.9108 (+/- 0.0925) [LogisticRegression]
Accuracy: 0.9161 (+/- 0.0543) [StackingClassifier]
```

```
[51]: #Decision Regions for 3 algorithms.
proline = wine_dataframe[['proline', 'alcohol']]
X = np.array(proline)
y = np.array(y)
gs = gridspec.GridSpec(2, 2)
fig = plt.figure(figsize=(20,16))
for clf, label, grd in zip([rf_clf, svc_clf, sclf], ["Random Forest Classifier", "Support Vector Classifier", "Stacking Classifier"], itertools.product([0, 1], repeat=2)):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y.astype(np.integer), clf=clf, legend=2)
    plt.title(label)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
```

```
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.
```

```
# Remove the CWD from sys.path while we load stuff.
```

```
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
```

```
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
```

```
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
```

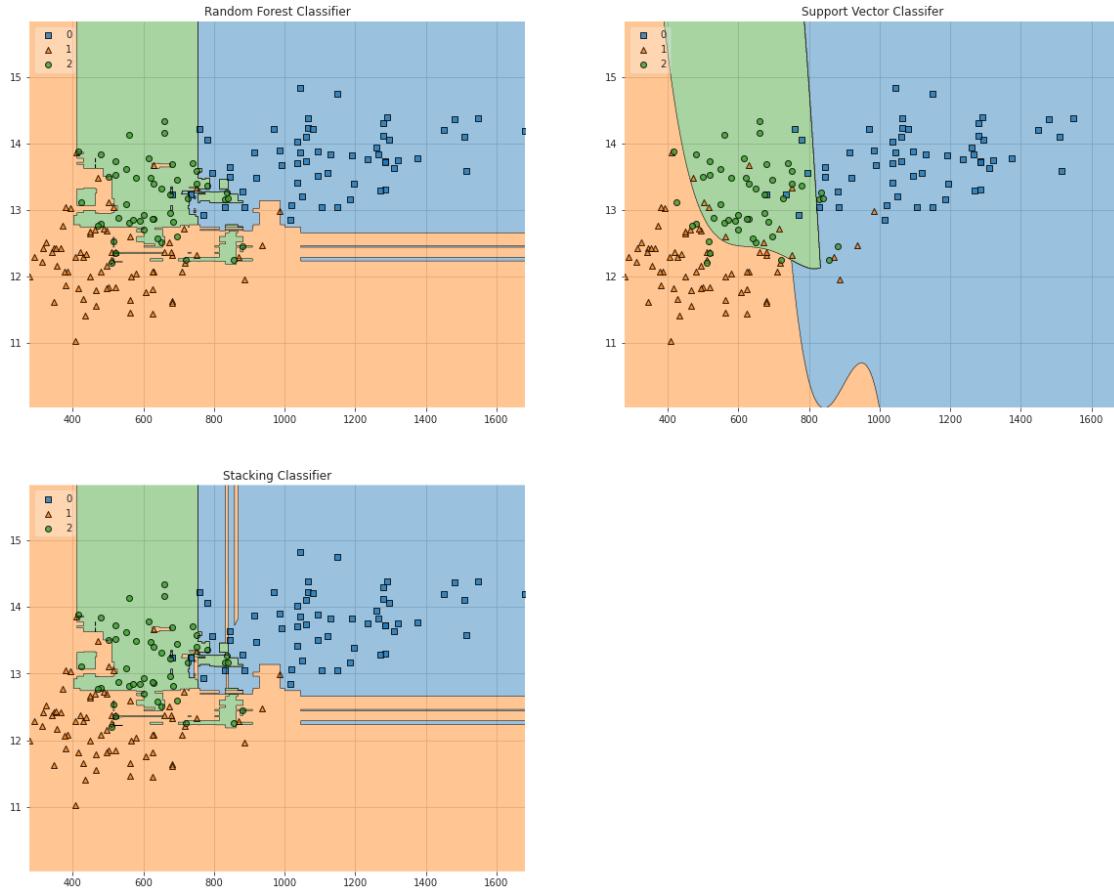
deprecated. The current result is `np.dtype(np.int\_)` which is not strictly correct. Note that the result depends on the system. To ensure stable results use may want to use `np.int64` or `np.int32`.

```
# Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.

    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.

# Remove the CWD from sys.path while we load stuff.
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.

    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
```



From the above decision region plot we can observe that SVC excels at differentiating regions while

the stacking and random forest classifier excel at accuracy of regions. The stacking plot seems to focus on specificity while the SVC plot focuses on a generalization of the majority.

```
#stuff
```

```
[52]: #Stacking classifiers using Grid Search cross-validation
```

```
RANDOM_SEED = 789
X = wine_dataframe.drop("target", 1)
y = wine_dataframe.target

rf_clf = RandomForestClassifier(random_state=RANDOM_SEED, n_jobs=-1)

knn_clf = KNeighborsClassifier(p=2, metric='minkowski', n_jobs=-1)

lr = LogisticRegression(random_state=RANDOM_SEED) # meta classifier

sclf = StackingClassifier(classifiers=[rf_clf, knn_clf], meta_classifier=lr)

print("\nAccuracies of all classifiers using grid search cross validation.")
params = {'randomforestclassifier__n_estimators': np.arange(10,20),  
          'randomforestclassifier__max_depth': np.arange(1,5),  
          'kneighborsclassifier__n_neighbors': np.arange(1,20,2),  
          'meta-logisticregression__C': [0.001,0.01,0.1,1,10,100,1000]}
```

Accuracies of all classifiers using grid search cross validation.

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning:  
In a future version of pandas all arguments of DataFrame.drop except for the  
argument 'labels' will be keyword-only  
    after removing the cwd from sys.path.
```

```
[53]: gsearch_cv = GridSearchCV(estimator=sclf, param_grid=params, cv=5, refit=True)
gsearch_cv.fit(X, y)
cv_keys = ('mean_test_score', 'std_test_score', 'params')
print('Best parameters: %s' % gsearch_cv.best_params_)
print('Accuracy: %.2f' % gsearch_cv.best_score_)
```

```
Best parameters: {'kneighborsclassifier__n_neighbors': 3, 'meta-  
logisticregression__C': 10, 'randomforestclassifier__max_depth': 4,  
'randomforestclassifier__n_estimators': 10}  
Accuracy: 0.98
```

After performing stacking classifiers using grid Search cross-validation, we can observe that the accuracy has gone up drastically being of 0.98. Compared to our other methods this is the most accurate model we have.

```
[55]: proline = wine_dataframe[['proline', 'alcohol']]
X = np.array(proline)
y = np.array(y)

rf_clf = RandomForestClassifier(max_depth=3, n_estimators=15, random_state=RANDOM_SEED, n_jobs=-1)
knn_clf = KNeighborsClassifier(n_neighbors=7, p=2, metric='minkowski', n_jobs=-1)
lr = LogisticRegression(C=0.1, random_state=RANDOM_SEED) # meta classifier
sclf = StackingClassifier(classifiers=[rf_clf, knn_clf], meta_classifier=lr)

gs = gridspec.GridSpec(2, 2)
fig = plt.figure(figsize=(20,16))

for clf, label, grd in zip([rf_clf, knn_clf, sclf], ["RandomForestClassifier", "KNeighborsClassifier", "StackingClassifier"], itertools.product([0, 1], repeat=2)):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=y.astype(np.integer), clf=clf, legend=2)
    plt.title(label)

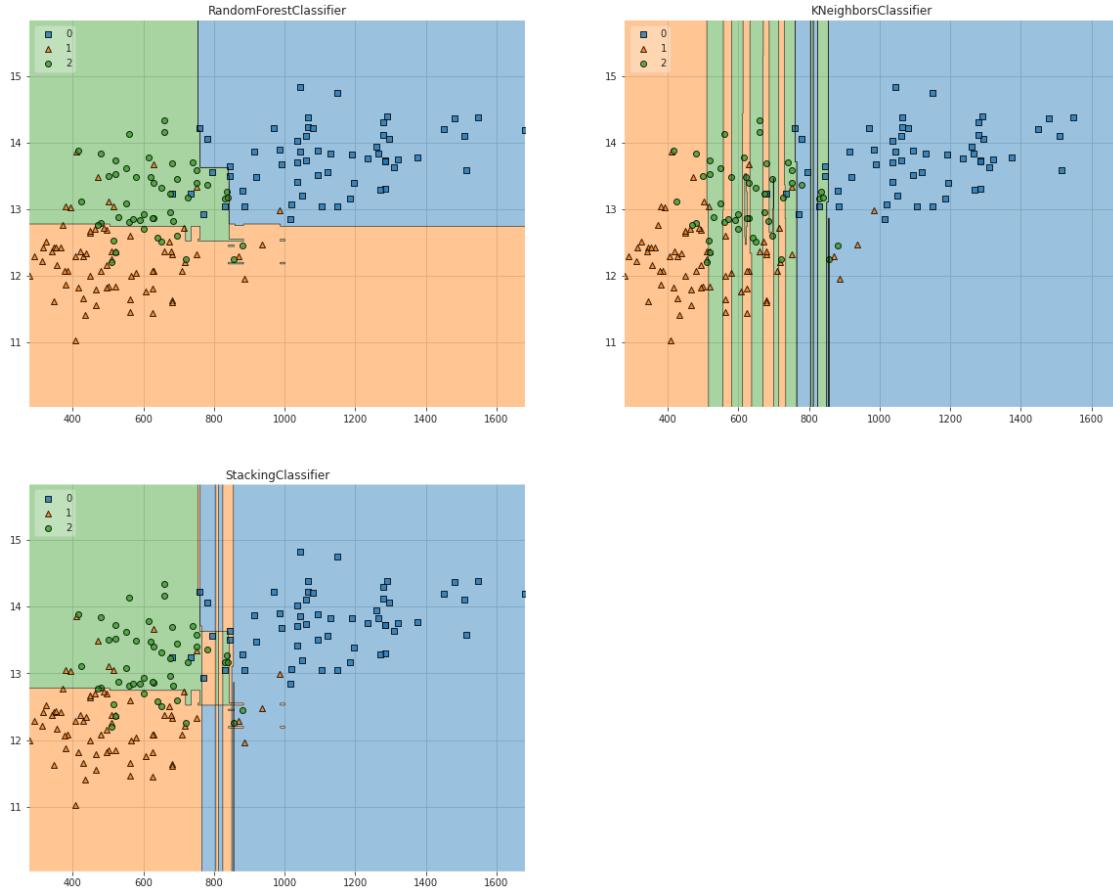
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.
    app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
use may want to use `np.int64` or `np.int32`.
    app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
    ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16:
DeprecationWarning: Converting `np.integer` or `np.signedinteger` to a dtype is
deprecated. The current result is `np.dtype(np.int_)` which is not strictly
correct. Note that the result depends on the system. To ensure stable results
```

```

use may want to use `np.int64` or `np.int32`.
app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/mlxtend/plotting/decision_regions.py:244:
MatplotlibDeprecationWarning: Passing unsupported keyword arguments to axis()
will raise a TypeError in 3.3.
ax.axis(xmin=xx.min(), xmax=xx.max(), y_min=yy.min(), y_max=yy.max())

```



From the plots we can observe how accurate our models are with randomforest and stacking classifier being quite accurate. From the plot we can obeserve that the KNeighbors classifier seems to differ from the other two when differentiating regions properly. Although the stacking classifier make be more accurate than the random forest classifier, the random forest classifier has a cleaner look in differentiating regions.

## #Conclusion

Overall, in this lab we used ensemble methods to determine where of the three regions did the wines given originate from. From our data we can observe that there is a strong relationship between two features in predicting which region each wine is from. These features being proline and alcohol. Continuing on, by using ensemble methods we were able to predict where wine was from within ~80-92% accuracy. However, after utilizing stacking and cross-grid validation we were able to create a model with 98% accuracy in determining where each wine was from. The least

accurate overall was the KNeighbors model, however this could be due to only observing two N nearest neighbor which lead to the inaccuracy. Overall, because we were able to use stacking and cross-grid validation along with ensemble methods, we could predict where wines originated from based on the model that was created with a 98% accuracy score.

References:

1. Lutins, E. (2017, August 2). Ensemble Methods in Machine Learning: What are they and why use them? Medium. Retrieved October 24, 2022, from <https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f>
2. Ceballos, F. (2019, September 14). Stacking classifiers for higher predictive performance. Medium. Retrieved October 24, 2022, from <https://towardsdatascience.com/stacking-classifiers-for-higher-predictive-performance-566f963e4840>

# Notebook

August 2, 2023

**Leng Her**

**Lab 10**

**November 15, 2022**

#Overview

In order to determine if there will be an overall decrease in area thickness of the Artic and Antractic ice (ice melting), we will be using two univariate time series models to forecast the area thickness of the Artic and Antarctic for the next few months/years. The two univariate time series models that will be used are the ARIMA and SARIMA models. The ARIMA model is a machine learning technique that utilizes statistical analysis, to understand the time series dataset or predict future trends (Hayes, A.). The SARIMA models is very similar to the ARIMA model, the only difference is that the SARIMA model also takes into consideration any seasonality patterns, whereas the ARIMA model does not (Real statistics using Excel). By utilizing these techniques we will try to forecast the area thickness of the Artic and Antractic ice.

**References:**

1. <https://machinelearningmastery.com/sarima-for-time-series-forecasting-in-python/>
2. <https://www.datasciencesmachinelearning.com/2019/01/arimasarima-in-python.html>
3. <https://www.wisdomgeek.com/development/machine-learningsarima-forecast-seasonal-data-using-python/>
4. <https://medium.com/mlearning-ai/how-to-build-sarima-model-in-python-7ae83b14c884>
5. [https://www.statsmodels.org/dev/generated/statsmodels.tsa.arima.model.ARIMAResults.get\\_forecast.html](https://www.statsmodels.org/dev/generated/statsmodels.tsa.arima.model.ARIMAResults.get_forecast.html)
6. [https://www.statsmodels.org/dev/examples/notebooks/generated/statespace\\_forecasting.html](https://www.statsmodels.org/dev/examples/notebooks/generated/statespace_forecasting.html)
7. <https://medium.datadriveninvestor.com/time-series-prediction-using-sarimax-a6604f258c56>
8. <https://analyticsindiamag.com/quick-way-to-find-p-d-and-q-values-for-arima/>
9. Hayes, A. (2022, June 24). Autoregressive Integrated moving average (ARIMA). Investopedia. Retrieved November 15, 2022, from <https://www.investopedia.com/terms/a/autoregressive-integrated-moving-average-arima.asp>
10. Real statistics using Excel. Real Statistics Using Excel. (n.d.). Retrieved November 15, 2022, from <https://www.real-statistics.com/time-series-analysis/seasonal-arima-sarima/>

#Data

The dataset was obtained from National Snow and Ice Data Center (<https://nsidc.org/data/g02135>). The data we will be using is a combined dataset which contains the area of thickness and the month and year as the index. For more information on the dataset please refer to the link: <https://nsidc.org/data/g02135>.

```
[5]: import pandas as pd
import numpy as np
%matplotlib inline
from matplotlib import pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima.model import ARIMAResults
from statsmodels.tsa.seasonal import seasonal_decompose

from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.statespace.sarimax import SARIMAXResults
```

```
[6]: #Loading data into dataframe to be used, combine all 12 csv files into one
      ↪dataframe so we have proper dataset with all 12 months
df = pd.concat([pd.read_csv('N_01_extent_v3.0.csv'), 'N_02_extent_v3.0.
      ↪csv','N_03_extent_v3.0.csv','N_04_extent_v3.0.csv','N_05_extent_v3.0.
      ↪csv','N_06_extent_v3.0.csv','N_07_extent_v3.0.csv',
      'N_08_extent_v3.0.csv','N_09_extent_v3.0.csv','N_10_extent_v3.0.
      ↪csv','N_11_extent_v3.0.csv' , 'N_12_extent_v3.0.csv']])
#remove the spaces that appear in column headings
df.columns = df.columns.str.replace(' ', '')
#dropping bad rows from dataset
df = df[df['area'] != -9999.00]
#dropping columns we don't need
df=df.drop(['data-type','region','extent'], axis=1)
#create a unique time column to sort on
df["time"] = df["year"].apply(str) + "-" + df["mo"].apply(str).str.zfill(2)
#drop the old columns we don't need now
df=df.drop(['year', 'mo'], axis=1)
#make sure our new unique column is the index which is important for time
      ↪series data (needs to be in order!)
df = df.set_index('time')
#sort it!
df = df.sort_values(by=['time'])
#make sure it's in order
df.head(50)
```

```
[6]:      area
      time
1978-11    9.04
1978-12   10.90
```

1979-01	12.41
1979-02	13.18
1979-03	13.21
1979-04	12.53
1979-05	11.11
1979-06	9.34
1979-07	6.69
1979-08	5.06
1979-09	4.58
1979-10	6.19
1979-11	8.37
1979-12	10.63
1980-01	11.94
1980-02	12.90
1980-03	12.99
1980-04	12.52
1980-05	10.96
1980-06	9.00
1980-07	6.53
1980-08	4.94
1980-09	4.87
1980-10	6.50
1980-11	8.73
1980-12	10.78
1981-01	11.91
1981-02	12.53
1981-03	12.71
1981-04	12.24
1981-05	10.99
1981-06	9.03
1981-07	6.31
1981-08	4.48
1981-09	4.44
1981-10	6.27
1981-11	8.31
1981-12	10.54
1982-01	12.19
1982-02	12.87
1982-03	13.07
1982-04	12.63
1982-05	11.17
1982-06	9.46
1982-07	6.87
1982-08	5.00
1982-09	4.43
1982-10	6.67
1982-11	9.06

```
1982-12 10.88
```

```
#EDA
```

```
[7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 513 entries, 1978-11 to 2021-10
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  --     --           --    
 0   area    513 non-null    float64 
dtypes: float64(1)
memory usage: 8.0+ KB
```

No null values

```
[8]: df.columns
```

```
[8]: Index(['area'], dtype='object')
```

```
[9]: print(f"The number of columns in the dataset is: {df.shape[1]}")
print(f"The number of rows in the dataset is: {df.shape[0]}")
```

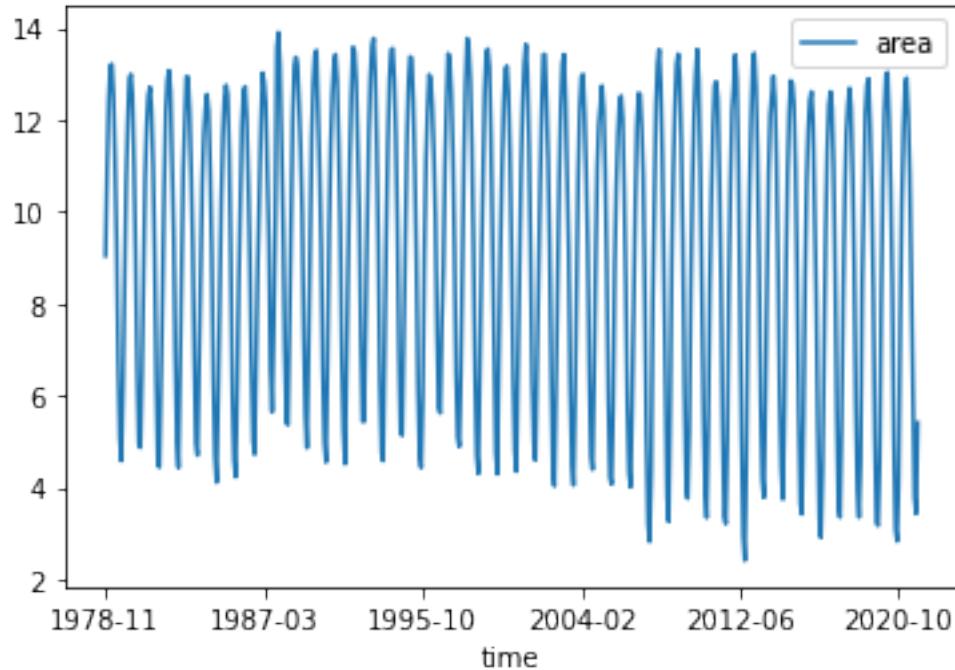
```
The number of columns in the dataset is: 1
The number of rows in the dataset is: 513
```

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

```
[10]:      area
count    513.000000
mean     9.272300
std      3.246864
min      2.410000
25%     6.270000
50%     9.950000
75%    12.290000
max     13.900000
```

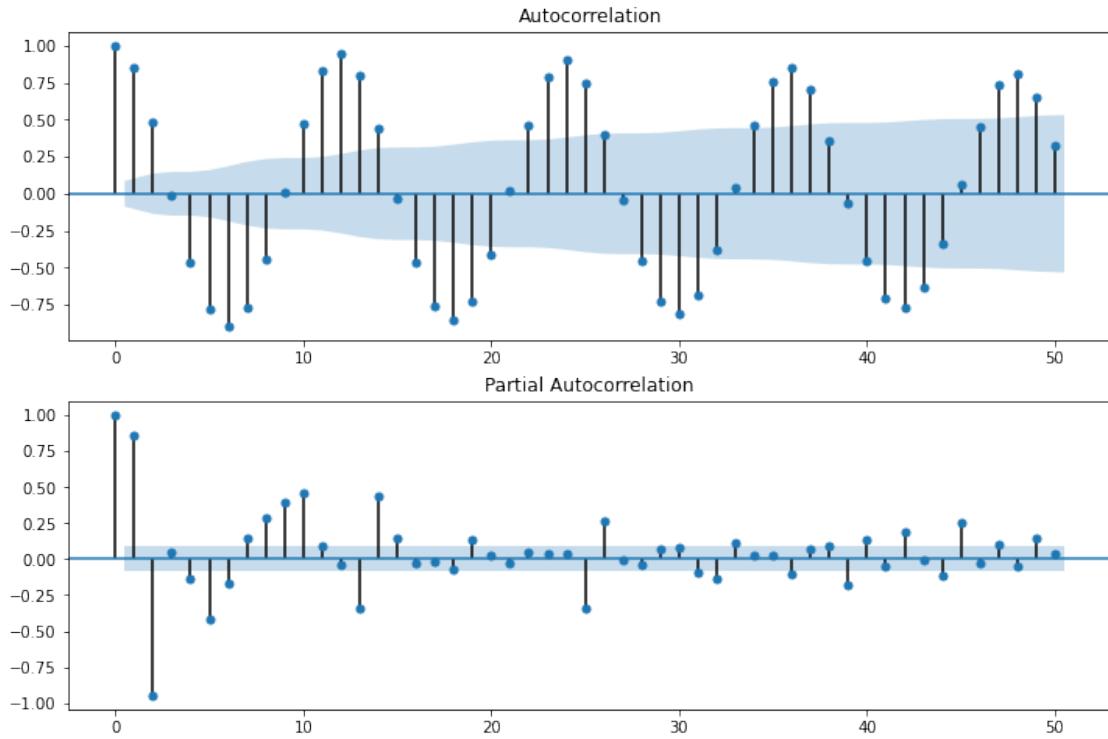
```
[11]: df.plot()
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7efea584e990>
```



From this plot we can see that the data is stationary and we can use the original series for our model. We can double-check via ACF and PACF plots.

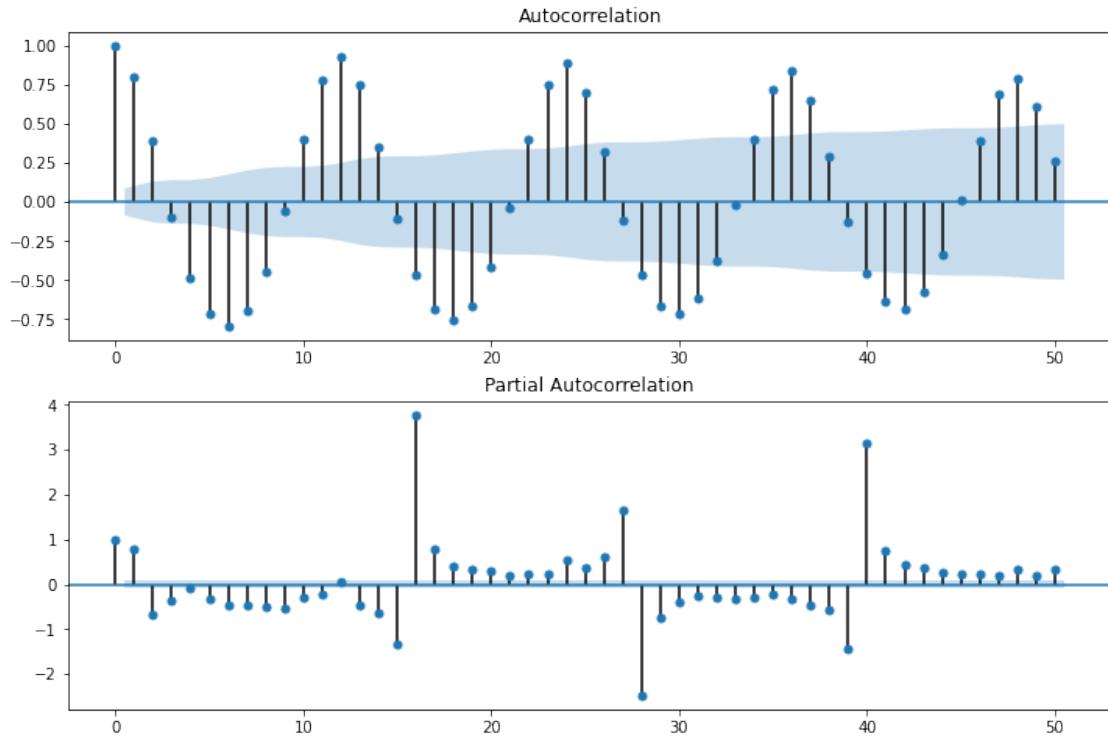
```
[12]: fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['area'], lags=50, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['area'], lags=50, ax=ax2)
plt.show()
```



```
[13]: #First order differencing
```

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['area'].diff().dropna(), lags=50, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['area'].diff().dropna(), lags=50, ax=ax2)
plt.show()
```

```
/usr/local/lib/python3.7/dist-
packages/statsmodels/regression/linear_model.py:1434: RuntimeWarning: invalid
value encountered in sqrt
    return rho, np.sqrt(sigmasq)
```



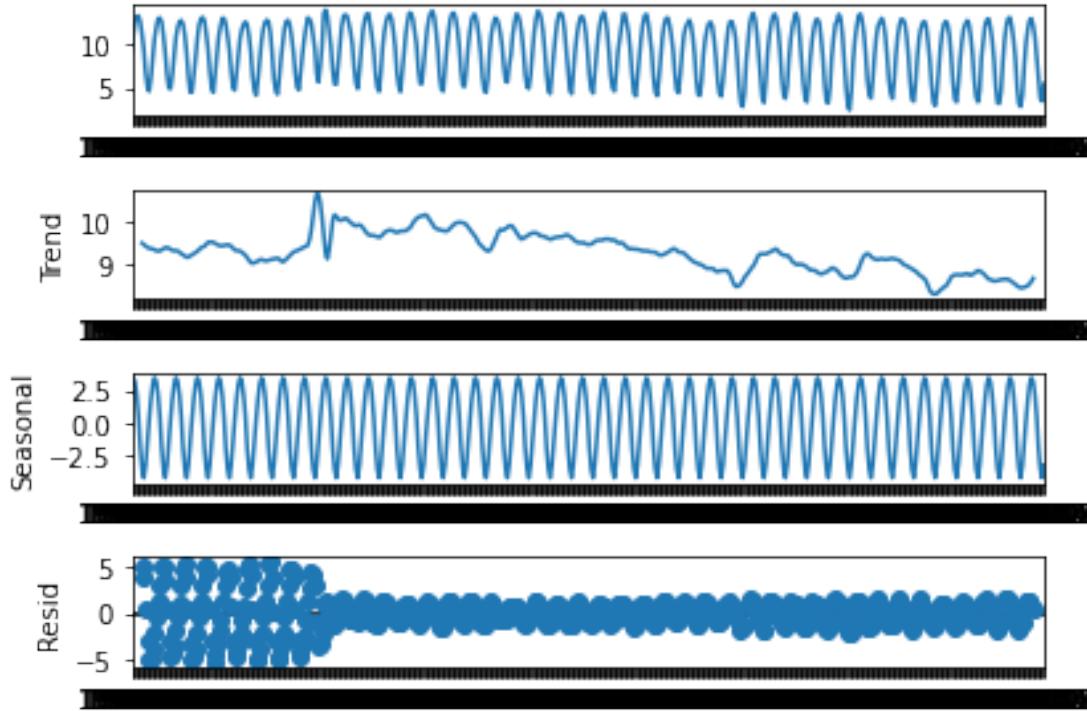
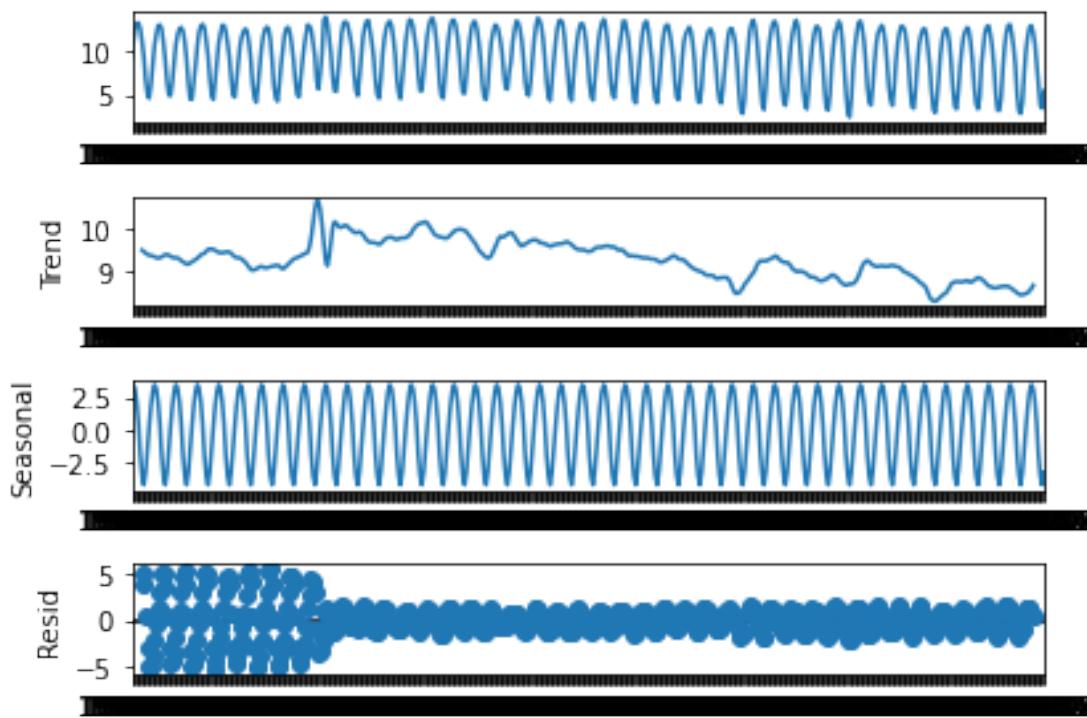
$$p = 2, q = 3$$

From the PACF graph if the original series contains 3 lags that are greater than the critical value setting our  $q$  to 3. From the ACF graph we can observe that 2 lags that are greater than the critical boundary with a third lag, however the third lag is not too far from the critical boundary so we will set  $p$  to 2.

From the second plot we used first order differencing and nothing changed for the ACF plot in the original and the first order. However there seems to be a difference in the PACF plot with the first order plot showing a trend similar to the ACF plot.

```
[14]: seasonal_decompose(df, model='additive', period= 12).plot() #choose 12 as period
      ↴because data is based on a monthly basis.
```

[14] :



Because our plot shows that our data is stationary in the seasonal section, this tells us that the d value for our ARIMA model can be 0.

```
#Models
```

```
[15]: train, test = df.iloc[:371,0], df.iloc[371:,0]
print(train.tail())
test.head()
```

```
time
2009-08    4.16
2009-09    3.76
2009-10    5.24
2009-11    8.36
2009-12   10.59
Name: area, dtype: float64
```

```
[15]: time
2010-01    12.04
2010-02    12.96
2010-03    13.53
2010-04    12.85
2010-05    10.88
Name: area, dtype: float64
```

```
[16]: model = ARIMA(train, order = (2,0,3))
model_fit = model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:472:
FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.
```

```
statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework and
is both well tested and maintained.
```

To silence this warning and continue using ARMA and ARIMA until they are removed, use:

```
import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                      FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                      FutureWarning)

warnings.warn(ARIMA_DEPRECATED_WARN, FutureWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:583:
```

```
ValueWarning: A date index has been provided, but it has no associated frequency  
information and so will be ignored when e.g. forecasting.  
' ignored when e.g. forecasting.', ValueWarning)
```

```
[17]: print(model_fit.summary())
```

```
ARMA Model Results  
=====
```

Dep. Variable:	area	No. Observations:	371
Model:	ARMA(2, 3)	Log Likelihood	-295.072
Method:	css-mle	S.D. of innovations	0.532
Date:	Wed, 16 Nov 2022	AIC	604.145
Time:	02:50:56	BIC	631.558
Sample:	0	HQIC	615.033

```
=====
```

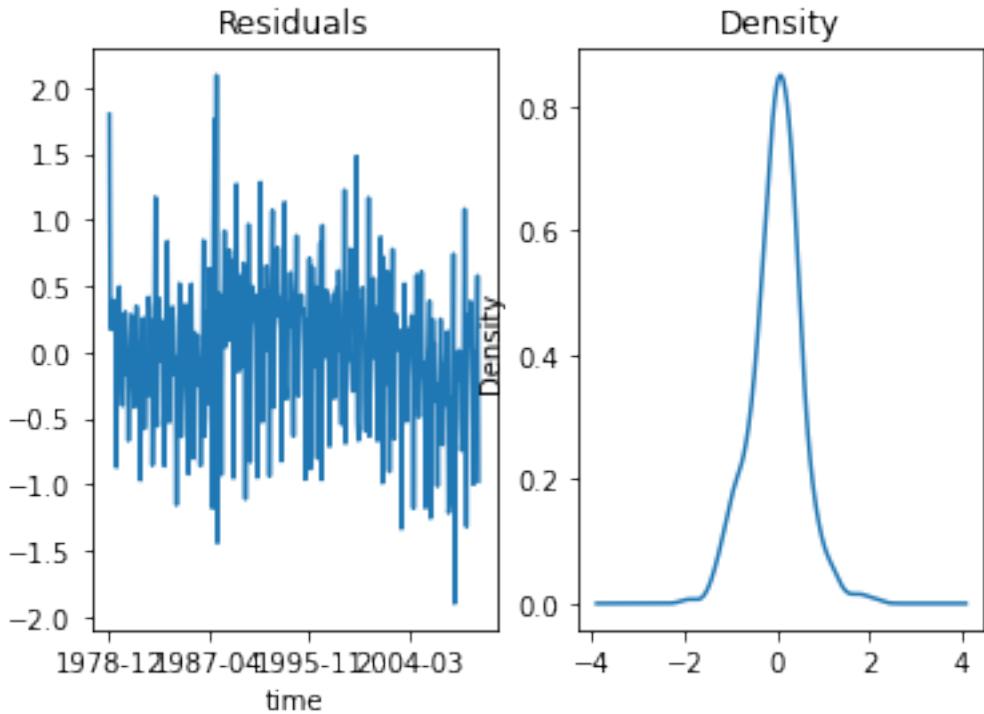
	coef	std err	z	P> z	[0.025	0.975]
const	9.4675	0.056	169.660	0.000	9.358	9.577
ar.L1.area	1.6789	0.017	100.072	0.000	1.646	1.712
ar.L2.area	-0.9479	0.016	-57.826	0.000	-0.980	-0.916
ma.L1.area	-0.1323	0.050	-2.626	0.009	-0.231	-0.034
ma.L2.area	-0.0462	0.050	-0.924	0.355	-0.144	0.052
ma.L3.area	-0.2817	0.049	-5.752	0.000	-0.378	-0.186

```
Roots  
=====
```

	Real	Imaginary	Modulus	Frequency
AR.1	0.8856	-0.5203j	1.0271	-0.0845
AR.2	0.8856	+0.5203j	1.0271	0.0845
MA.1	1.3741	-0.0000j	1.3741	-0.0000
MA.2	-0.7691	-1.4112j	1.6072	-0.3294
MA.3	-0.7691	+1.4112j	1.6072	0.3294

```
=====
```

```
[18]: #plot residues  
residuals = model_fit.resid[1:]  
fig, ax = plt.subplots(1,2)  
residuals.plot(title = "Residuals", ax = ax[0])  
residuals.plot(title = "Density", kind = "kde", ax = ax[1])  
  
plt.show()
```



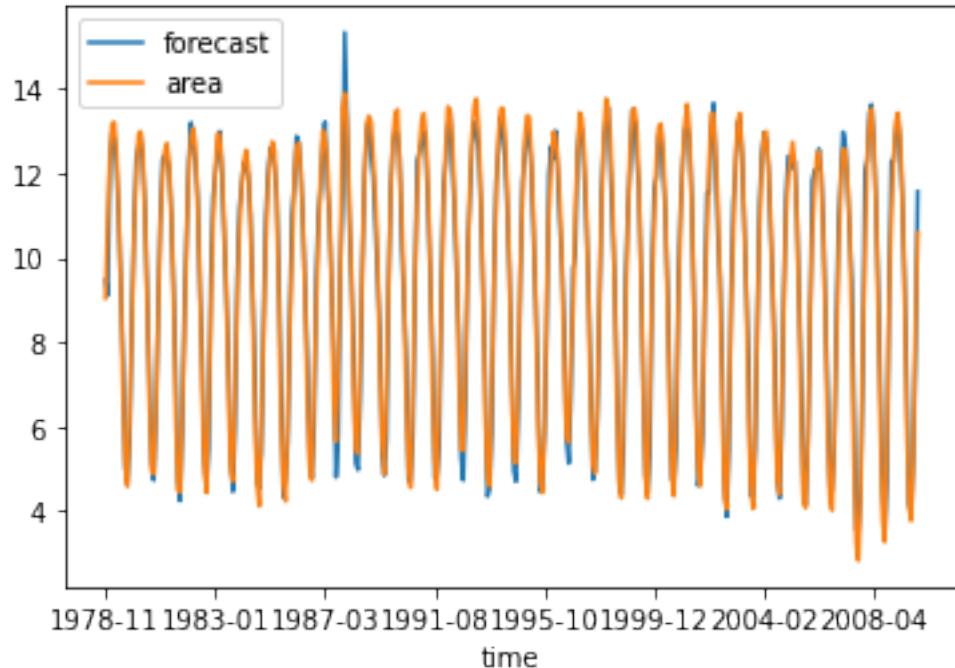
Density plot shows that our values are normally distributed.

```
[19]: residuals.describe()
```

```
[19]: count    370.000000
mean      0.005808
std       0.540403
min     -1.898491
25%     -0.281239
50%      0.030063
75%      0.321142
max      2.092207
dtype: float64
```

Our mean is extremely close to 0 so this means our ARIMA model has very little deficiency.

```
[20]: model_fit.plot_predict(start = train.index[0], end = train.index[-1] )
plt.show()
```



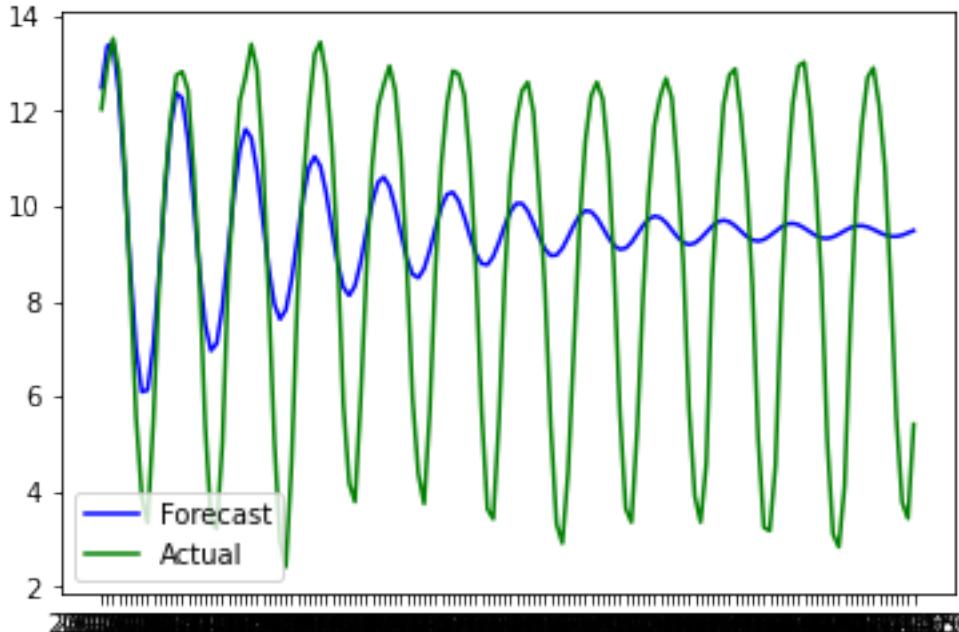
From this graph we can see that our ARIMA model is accurate following along the same trend as the actual values for our training set.

```
[21]: fore_values = model_fit.forecast(steps = 142) #get forecast values to compare with actual
```

```
[32]: len(fore_values[0])
```

```
[32]: 142
```

```
[29]: plt.plot(fore_values[0],c = "b", label = "Forecast")
plt.plot(test,c="g", label = "Actual")
plt.legend()
plt.show()
```



Here we can see that the ARIMA model fails to accurately predict the values after the first few values. Although the ARIMA value does follow the general trend of the actual values it fails in accuracy of values as seasonality increases.

#### SARIMA model

```
[23]: Smodel = SARIMAX(train, order=(2, 0, 3), seasonal_order=(2, 0, 3, 12)) #fourth
      ↪parameter for seasonal order is the frequency and from our data we choose 12
      ↪for 12 month basis

Smodel_fit = Smodel.fit()

print(Smodel_fit.summary())
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:583:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:583:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.7/dist-
packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.')
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:568:
```

```

ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
    ConvergenceWarning)

```

### SARIMAX Results

Dep. Variable:	area	No. Observations:				
371						
Model:	SARIMAX(2, 0, 3)x(2, 0, 3, 12)	Log Likelihood				
-229.591						
Date:	Wed, 16 Nov 2022	AIC				
481.182						
Time:	02:51:13	BIC				
524.260						
Sample:	0	HQIC				
498.291						
	- 371					
Covariance Type:	opg					
	coef	std err	z	P> z	[0.025	0.975]
ar.L1	1.5560	0.046	34.147	0.000	1.467	1.645
ar.L2	-0.5574	0.039	-14.410	0.000	-0.633	-0.482
ma.L1	-0.3455	0.061	-5.672	0.000	-0.465	-0.226
ma.L2	-0.2145	0.035	-6.210	0.000	-0.282	-0.147
ma.L3	-0.4266	0.038	-11.220	0.000	-0.501	-0.352
ar.S.L12	0.6685	0.364	1.837	0.066	-0.045	1.382
ar.S.L24	0.3087	0.356	0.867	0.386	-0.389	1.007
ma.S.L12	-0.0435	0.371	-0.117	0.907	-0.771	0.684
ma.S.L24	0.0101	0.146	0.069	0.945	-0.275	0.295
ma.S.L36	-0.1040	0.082	-1.273	0.203	-0.264	0.056
sigma2	0.1848	0.007	25.110	0.000	0.170	0.199
====						
Ljung-Box (L1) (Q):		3.68	Jarque-Bera (JB):			
2394.20						
Prob(Q):		0.05	Prob(JB):			
0.00						
Heteroskedasticity (H):		0.25	Skew:			
1.24						
Prob(H) (two-sided):		0.00	Kurtosis:			
15.19						
====						
====						

### Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-

```
step).
```

Our SARIMA model is considerably better than our ARIMA model as both the AIC and BIC values are lower than the ARIMA models.

The SARIMA model scores are: AIC = 481.182, BIC = 524.260.

The ARIMA model scores are: AIC = 604.145, BIC = 631.558.

```
[24]: forecast_values = Smodel_fit.forecast(steps = 160) #142 is for the data for the  
        ↪test split, however we want a to predict more in the future so we will add  
        ↪more steps
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:379:  
ValueWarning: No supported index is available. Prediction results will be given  
with an integer index beginning at `start`.  
ValueWarning)
```

```
[25]: print(forecast_values)
```

```
371    12.388331  
372    13.303074  
373    13.428960  
374    12.786594  
375    11.263847  
      ..  
526    9.149512  
527    10.457228  
528    11.154268  
529    11.262319  
530    10.775443  
Name: predicted_mean, Length: 160, dtype: float64
```

These are our forecasted values from the SARIMA model. The index values on the left can be replaced with dates later on following the original format of the dataset.

```
[35]: forecast = forecast_values.to_frame().reset_index()  
forecast.tail(30)
```

```
[35]:   index  predicted_mean  
130    501      7.638931  
131    502      9.373467  
132    503     10.728105  
133    504     11.450305  
134    505     11.562525  
135    506     11.058615  
136    507      9.855303  
137    508      8.035794  
138    509      5.615617  
139    510      4.012918
```

```
140    511      3.674176
141    512      5.077465
142    513      7.557180
143    514      9.261453
144    515     10.592423
145    516     11.301931
146    517     11.412044
147    518     10.916723
148    519      9.734138
149    520      7.946052
150    521      5.567725
151    522      3.992686
152    523      3.659669
153    524      5.038449
154    525      7.474972
155    526      9.149512
156    527     10.457228
157    528     11.154268
158    529     11.262319
159    530     10.775443
```

This is the forecast values of our dataset for the next months.

```
[36]: test2 = test.to_frame().reset_index()
test2.tail(20)
```

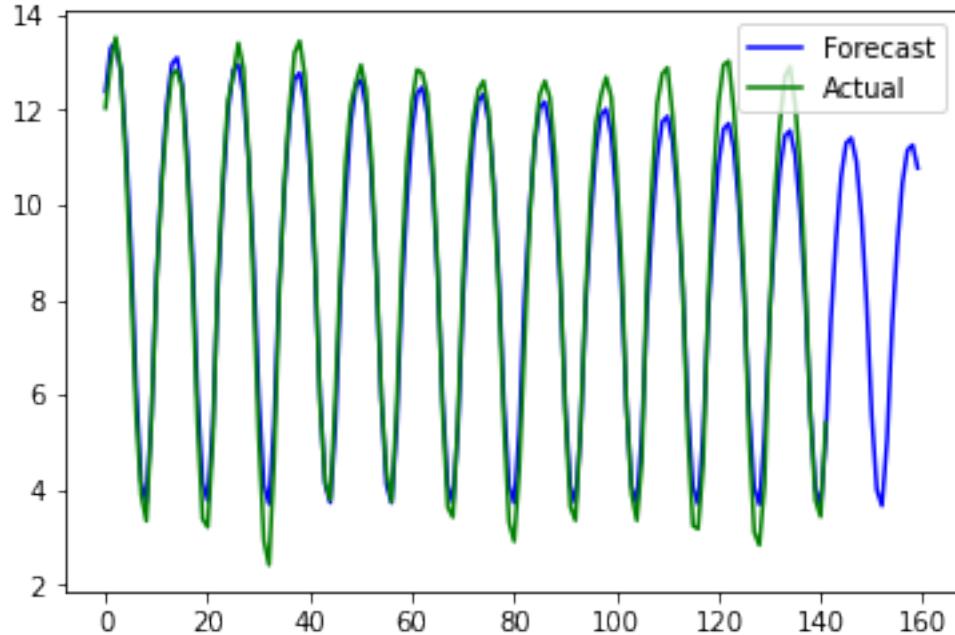
```
[36]:      time   area
122  2020-03  13.03
123  2020-04  12.02
124  2020-05  10.51
125  2020-06   8.41
126  2020-07   5.07
127  2020-08   3.10
128  2020-09   2.83
129  2020-10   4.11
130  2020-11   7.53
131  2020-12  10.16
132  2021-01  11.71
133  2021-02  12.71
134  2021-03  12.91
135  2021-04  12.13
136  2021-05  10.82
137  2021-06   8.38
138  2021-07   5.55
139  2021-08   3.77
140  2021-09   3.43
141  2021-10   5.41
```

Reset the index of the test set as we only want the values to determine if our forecast values match. We also turned the forecast values into a dataframe in order to plot without the date values

```
[28]: plt.plot(forecast["predicted_mean"], c = "b", label = "Forecast")
plt.plot(test2["area"], c = "g", label = "Actual")

plt.legend()
```

```
[28]: <matplotlib.legend.Legend at 0x7efea0f66f50>
```



From the plot we can see that our forecast and actual values follow the same trend, meaning our SARIMA model is very accurate. Our plot also shows a forecast of values.

## #Conclusion

To determine the area thickness of the Artic and Antarctic ice, we utilized two time series models; ARIMA and SARIMA. To use these models to forecast the area thickness of the ice, we split our dataset into two groups one for training and the other for testing. After doing so we ran the ARIMA model which did not consider seasonality. Although the ARIMA model had a relatively low AIC and BIC score of: AIC = 604.145, BIC = 631.558, the model failed to predict the test values accurately. Because the ARIMA model does not consider seasonality, we ran the data using the SARIMA model which does consider seasonality and patterns associated. After running the SARIMA model we received an even lower score for our AIC and BIC: AIC = 481.182, BIC = 524.260, indicating a more accurate model. The SARIMA model was then shown to be able to predict the test values correctly and even forecasted future values which has been shown above. From these forecast values it indicates that there will be an increase in area thickness of ice in the Artic and Antarctic. The forecast values also suggest a decrease after, following a periodic

like wave; increasing then decreasing then increasing, etc. It is also important to note that the horizontal asymptotes for both decreasing to increasing, and increasing to decreasing seem to be decreasing in value overall. This could indicate a decrease in fluctuation and an overall decrease in area ice thickness. Overall, we used ARIMA and SARIMA time series model to forecast the thickness of ice in the Arctic and Antarctic with forecast values for the next months/years.

# Notebook

August 2, 2023

**Leng Her**

**Lab 9**

**November 8 2022**

**#OVERVIEW**

In order to understand customer segments, we will be utilizing unsupervised clustering techniques, such as: K-means, and hierarchical clustering. K-means clustering is a clustering technique that groups similar data points together in order to observe patterns in the dataset (Understanding K-means clustering in machine learning). Hierarchical clustering is essentially the same thing, grouping together similar objects, however in this lab we will be using a dendrogram to show the relationships between data points and clusters. By using these unsupervised clustering techniques, we will attempt to understand more about customer segments or profiles of similar customers.

References:

<https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff>

<https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/>

<https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/>

[https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_silhouette\\_analysis.html](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html)

<https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>

<https://archive.ics.uci.edu/ml/datasets/wholesale+customers>

1. Understanding K-means clustering in machine learning. (n.d.). Retrieved November 9, 2022, from <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>

**#DATA**

Data obtained from UCI machine learning repository (LINK: <https://archive.ics.uci.edu/ml/datasets/wholesale+customers>). There is no target variable for our dataset. The dataset contains two categorical columns called: Channel, and Region; with Channel having the values of 1 or 2, with 1 representing horeaca (food catering) and 2 representing retail. In the region column it can have values of 1,2, and 3, with 1 representing Lisbon, 2 representing Oporto, and 3 representing other. For more information on the dataset please refer to the link above.

```
[140]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
import seaborn as sns
from sklearn.cluster import MeanShift, estimate_bandwidth
import matplotlib.cm as cm
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
[141]: df = pd.read_csv("/content/Wholesale customers data.csv")

df.head()
```

```
[141]:   Channel  Region  Fresh  Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0          2        3    12669    9656      7561       214                2674         1338
1          2        3     7057    9810      9568      1762                3293         1776
2          2        3     6353    8808      7684      2405                3516         7844
3          1        3    13265    1196      4221      6404                  507         1788
4          2        3    22615    5410      7198      3915                1777         5185
```

Based solely on the head of the data, this data may need to be scaled.

#EDA

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

```
[142]:   Channel  Region  Fresh  Milk  Grocery  \
count  440.000000  440.000000  440.000000  440.000000  440.000000
mean   1.322727  2.543182  12000.297727  5796.265909  7951.277273
std    0.468052  0.774272  12647.328865  7380.377175  9503.162829
min    1.000000  1.000000  3.000000   55.000000   3.000000
25%   1.000000  2.000000  3127.750000  1533.000000  2153.000000
50%   1.000000  3.000000  8504.000000  3627.000000  4755.500000
75%   2.000000  3.000000  16933.750000  7190.250000  10655.750000
max   2.000000  3.000000  112151.000000 73498.000000  92780.000000

                           Frozen  Detergents_Paper  Delicassen
count  440.000000      440.000000      440.000000
mean   3071.931818      2881.493182      1524.870455
std    4854.673333      4767.854448      2820.105937
min    25.000000       3.000000       3.000000
25%   742.250000      256.750000      408.250000
50%  1526.000000      816.500000      965.500000
75%  3554.250000      3922.000000      1820.250000
max  60869.000000     40827.000000     47943.000000
```

From the summary statistic table of our dataframe we can see that values can range from 1-3, 3-113000, 55-74000, 3-93000, 25-61000, and 3-480000. Because of the wide ranges of data we will be scaling and standardizing our data. It is also important to note that not just our ranges differ, but also that values differ in ranges over the widths. With these wide ranges it can affect our results, as each feature can impact our models in different magnitudes, in such a case we will be scaling our data to prevent this.

[143]: `#scaling data`

```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns = [
    "Channel", "Region", "Fresh", "Milk", "Grocery", "Frozen", "Detergents_Paper", "Delicassen"])
scaled_df.head()
```

[143]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	\
0	1.448652	0.590668	0.052933	0.523568	-0.041115	-0.589367	
1	1.448652	0.590668	-0.391302	0.544458	0.170318	-0.270136	
2	1.448652	0.590668	-0.447029	0.408538	-0.028157	-0.137536	
3	-0.690297	0.590668	0.100111	-0.624020	-0.392977	0.687144	
4	1.448652	0.590668	0.840239	-0.052396	-0.079356	0.173859	

	Detergents_Paper	Delicassen
0	-0.043569	-0.066339
1	0.086407	0.089151
2	0.133232	2.243293
3	-0.498588	0.093411
4	-0.231918	1.299347

[144]: `scaled_df.describe()`

[144]:

	Channel	Region	Fresh	Milk	Grocery	\
count	4.400000e+02	4.400000e+02	4.400000e+02	440.000000	4.400000e+02	
mean	1.614870e-17	3.552714e-16	-3.431598e-17	0.000000	-4.037175e-17	
std	1.001138e+00	1.001138e+00	1.001138e+00	1.001138	1.001138e+00	
min	-6.902971e-01	-1.995342e+00	-9.496831e-01	-0.778795	-8.373344e-01	
25%	-6.902971e-01	-7.023369e-01	-7.023339e-01	-0.578306	-6.108364e-01	
50%	-6.902971e-01	5.906683e-01	-2.767602e-01	-0.294258	-3.366684e-01	
75%	1.448652e+00	5.906683e-01	3.905226e-01	0.189092	2.849105e-01	
max	1.448652e+00	5.906683e-01	7.927738e+00	9.183650	8.936528e+00	

	Frozen	Detergents_Paper	Delicassen
count	4.400000e+02	4.400000e+02	4.400000e+02
mean	3.633457e-17	2.422305e-17	-8.074349e-18
std	1.001138e+00	1.001138e+00	1.001138e+00
min	-6.283430e-01	-6.044165e-01	-5.402644e-01
25%	-4.804306e-01	-5.511349e-01	-3.964005e-01

```
50% -3.188045e-01 -4.336004e-01 -1.985766e-01  
75% 9.946441e-02 2.184822e-01 1.048598e-01  
max 1.191900e+01 7.967672e+00 1.647845e+01
```

```
[145]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 440 entries, 0 to 439  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype  
---  
 0 Channel    440 non-null   int64  
 1 Region     440 non-null   int64  
 2 Fresh      440 non-null   int64  
 3 Milk       440 non-null   int64  
 4 Grocery    440 non-null   int64  
 5 Frozen     440 non-null   int64  
 6 Detergents_Paper 440 non-null   int64  
 7 Delicassen 440 non-null   int64  
dtypes: int64(8)  
memory usage: 27.6 KB
```

There are no null values with each value being of the int64 data type. There are 440 rows of data with 8 columns.

```
[146]: scaled_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 440 entries, 0 to 439  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype  
---  
 0 Channel    440 non-null   float64  
 1 Region     440 non-null   float64  
 2 Fresh      440 non-null   float64  
 3 Milk       440 non-null   float64  
 4 Grocery    440 non-null   float64  
 5 Frozen     440 non-null   float64  
 6 Detergents_Paper 440 non-null   float64  
 7 Delicassen 440 non-null   float64  
dtypes: float64(8)  
memory usage: 27.6 KB
```

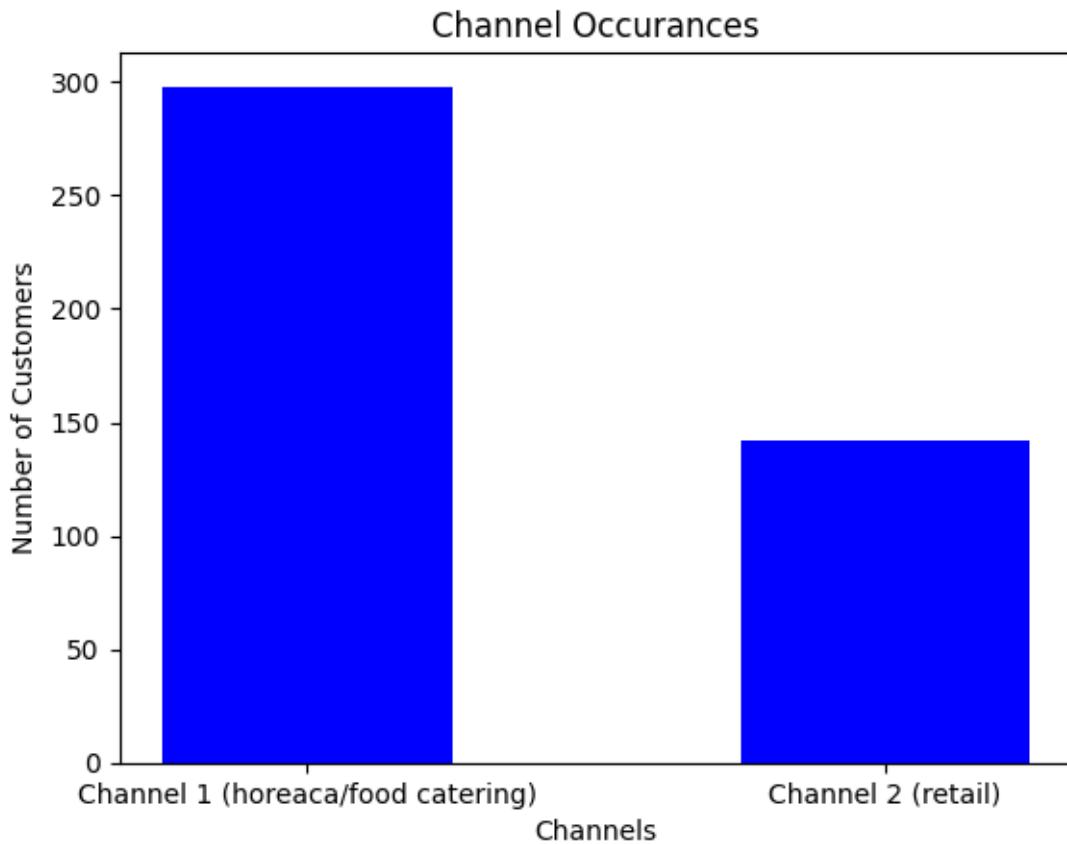
There are no null values with each value being of the float64 data type. There are 440 rows of data with 8 columns.

```
[147]: print(f"The number of columns in the dataset is: {df.shape[1]}")  
print(f"The number of rows in the dataset is: {df.shape[0]}")
```

```
The number of columns in the dataset is: 8  
The number of rows in the dataset is: 440
```

```
[148]: channel1 = len(df[df['Channel']== 1.0])  
channel2 = len(df[df['Channel']== 2.0])  
  
print("Channel 1 (horeaca/food catering) Occurrences: " + str(channel1))  
print("Channel 2 (retail) Occurrences: " + str(channel2))  
  
plt.bar(["Channel 1 (horeaca/food catering)","Channel 2 (retail)"],  
        [channel1,channel2], color ="blue",width = 0.5)  
  
plt.xlabel("Channels")  
plt.ylabel("Number of Customers")  
plt.title("Channel Occurrences")  
plt.show()
```

```
Channel 1 (horeaca/food catering) Occurrences: 298  
Channel 2 (retail) Occurrences: 142
```



From the bar graph we can observe most of our data comes from channel 1 (food catering/horeaca)

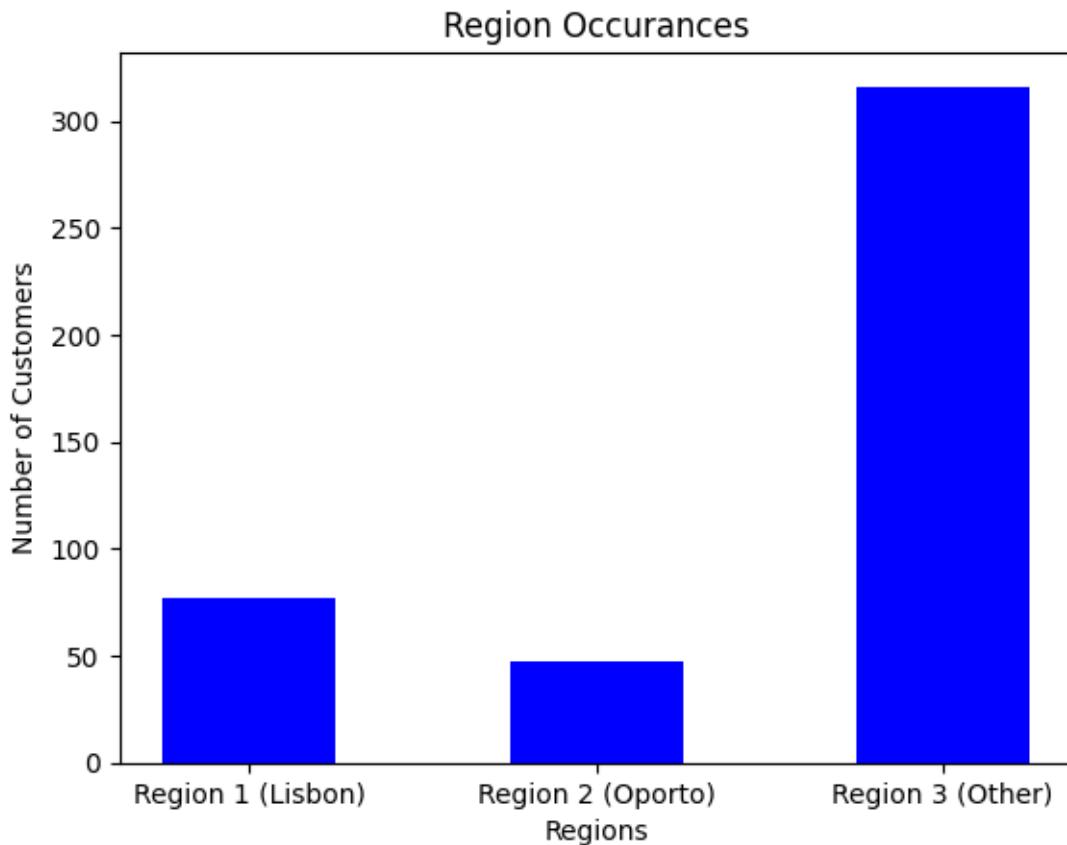
with more than double the occurrences of channel 2 (retail).

```
[149]: region1 = len(df[df['Region']== 1.0])
region2 = len(df[df['Region']== 2.0])
region3 = len(df[df['Region']== 3.0])

print("Region 1 (Lisbon) Occurrences: " + str(region1))
print("Region 2 (Oporto) Occurrences: " + str(region2))
print("Region 3 (Other) Occurrences: " + str(region3))

plt.bar(["Region 1 (Lisbon)","Region 2 (Oporto)","Region 3 (Other)"], [region1,region2,region3], color ="blue",width = 0.5)
plt.xlabel("Regions")
plt.ylabel("Number of Customers")
plt.title("Region Occurrences")
plt.show()
```

```
Region 1 (Lisbon) Occurrences: 77
Region 2 (Oporto) Occurrences: 47
Region 3 (Other) Occurrences: 316
```



From this bar graph we can observe that most of the observed values are from the region other with region 1 (Lisbon) as the second most occurrences in our dataset.

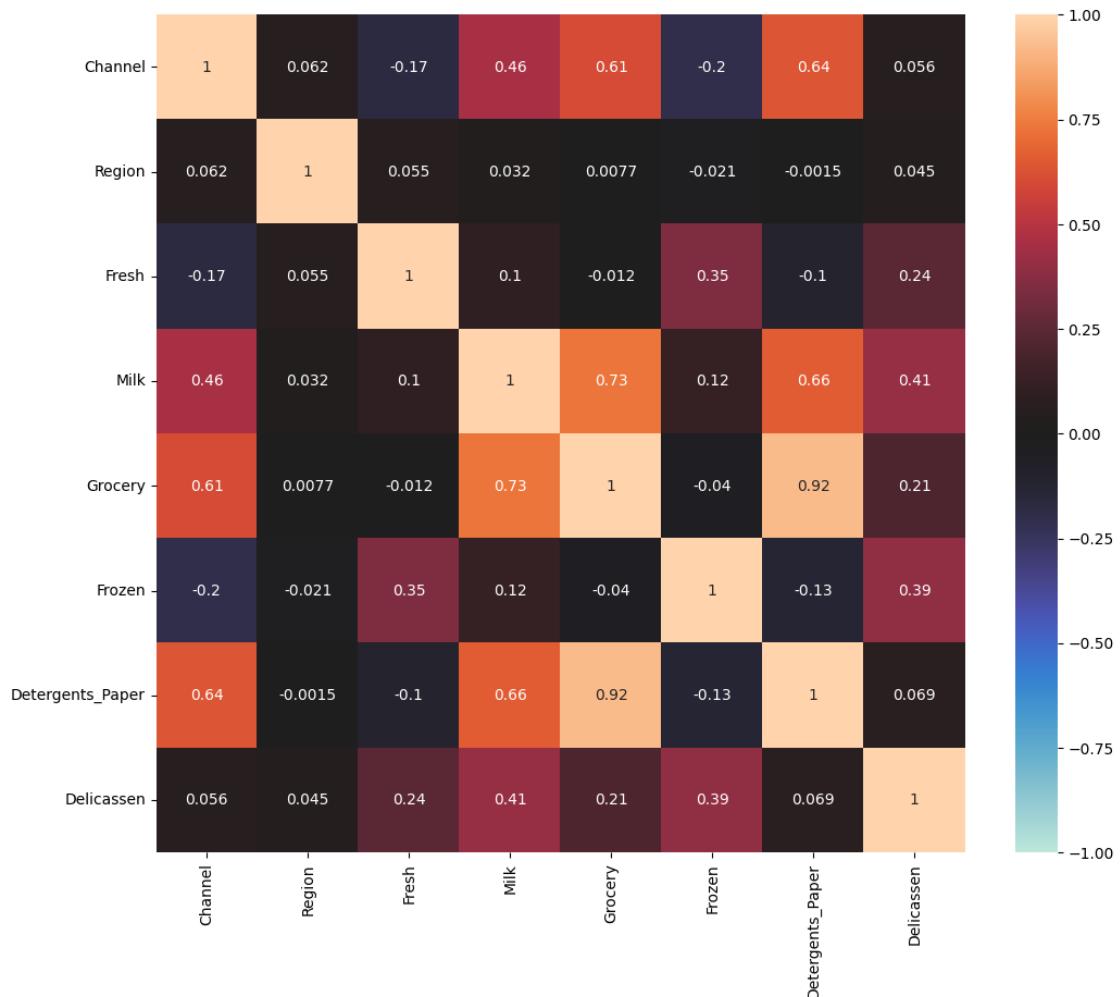
```
[150]: fig, ax = plt.subplots(figsize = (12,10))

corrmat = df.corr()
sns.heatmap(corrmat,-1,1,ax=ax, center = 0, annot = True)
```

/usr/local/lib/python3.7/dist-packages/seaborn/\_decorators.py:43: FutureWarning:  
 Pass the following variables as keyword args: vmin, vmax. From version 0.12, the  
 only valid positional argument will be `data`, and passing other arguments  
 without an explicit keyword will result in an error or misinterpretation.

FutureWarning

```
[150]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4afa063390>
```



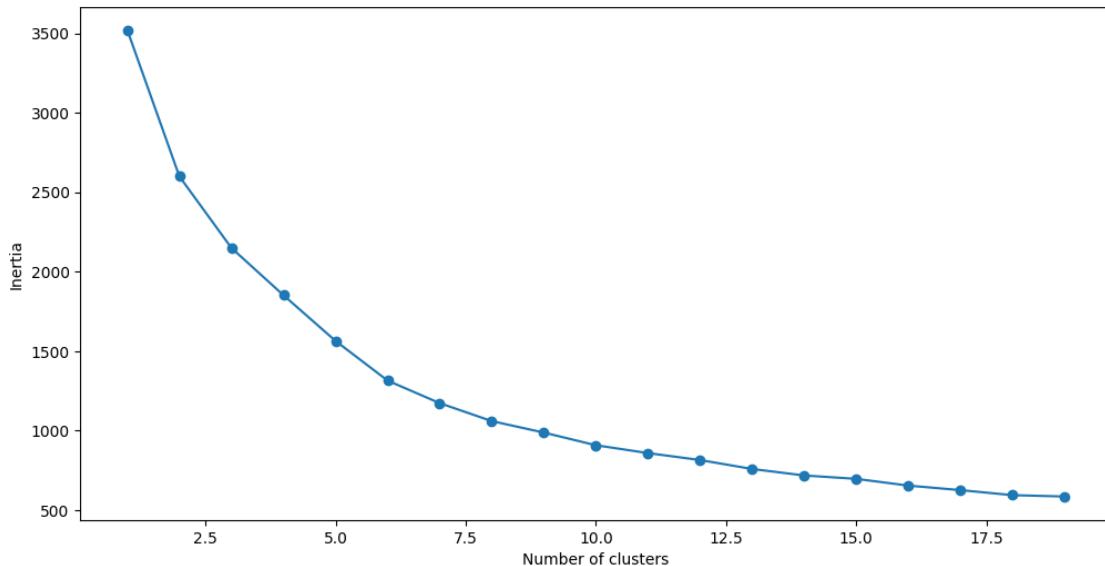
From the correlation matrix we can observe that dtergents\_paper and grocery have a strong positive correlation at 0.92, the highest correlation in the dataset. Other moderate-strong correlation attributes are milk to grocery, detergents\_paper to milk and channel to detergents\_paper and grocery.

#Models

```
[151]: SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_clusters = cluster, init='k-means++')
    kmeans.fit(scaled_df)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

```
[151]: Text(0, 0.5, 'Inertia')
```



The number of clusters we will choose will be 6, because it is better to have a lower value inertia than a higher value inertia. If the inertia value is high then it means the data points in the cluster are not similar to each other. The best way to find the “ideal” inertia is to use the elbow method, which is when the value of inertia begins to slow in decreasing value. From the inertia plot we can observe that the best values to choose from via the elbow method is between 5-8.

```
[152]: kmeans = KMeans(n_clusters = 6, init='k-means++')
kmeans.fit(scaled_df)
pred = kmeans.predict(scaled_df)
```

```
[153]: frame = pd.DataFrame(scaled_df)
frame['cluster'] = pred
cluster_result = frame['cluster'].value_counts()
cluster_result = cluster_result.to_frame().reset_index()
cluster_result = cluster_result.rename(columns = {"index":"Cluster_No.", "cluster": "count"})
#cluster_result2 = cluster_result.set_index("Cluster_No.")
frame
#Create new datafram with cluster column identifying which row is in which cluster
```

```
[153]:      Channel   Region    Fresh     Milk   Grocery   Frozen \
0    1.448652  0.590668  0.052933  0.523568 -0.041115 -0.589367
1    1.448652  0.590668 -0.391302  0.544458  0.170318 -0.270136
2    1.448652  0.590668 -0.447029  0.408538 -0.028157 -0.137536
3   -0.690297  0.590668  0.100111 -0.624020 -0.392977  0.687144
4    1.448652  0.590668  0.840239 -0.052396 -0.079356  0.173859
..      ...
435  -0.690297  0.590668  1.401312  0.848446  0.850760  2.075222
436  -0.690297  0.590668  2.155293 -0.592142 -0.757165  0.296561
437  1.448652  0.590668  0.200326  1.314671  2.348386 -0.543380
438  -0.690297  0.590668 -0.135384 -0.517536 -0.602514 -0.419441
439  -0.690297  0.590668 -0.729307 -0.555924 -0.573227 -0.620094
```

```
      Detergents_Paper  Delicassen  cluster
0        -0.043569   -0.066339      1
1         0.086407    0.089151      1
2         0.133232   2.243293      1
3        -0.498588   0.093411      0
4        -0.231918   1.299347      1
..          ...
435       -0.566831   0.241091      2
436       -0.585519   0.291501      2
437        2.511218   0.121456      1
438       -0.569770   0.213046      0
439       -0.504888  -0.522869      0
```

[440 rows x 9 columns]

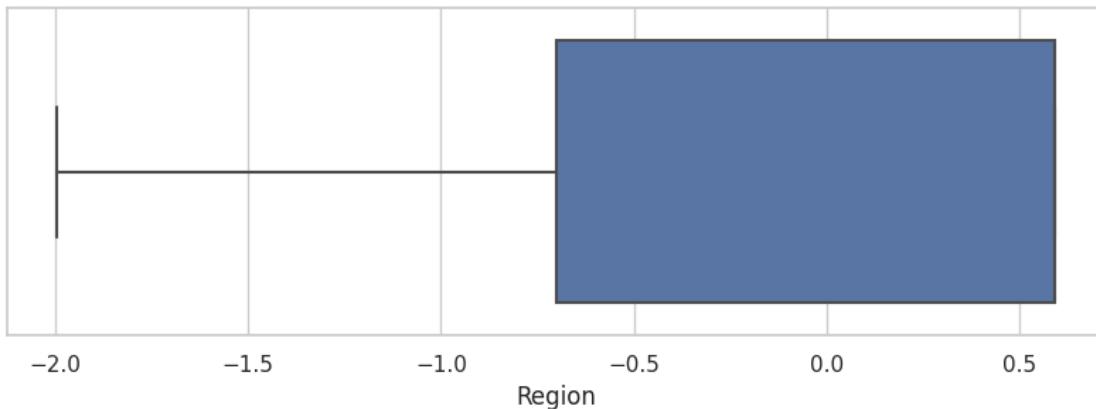
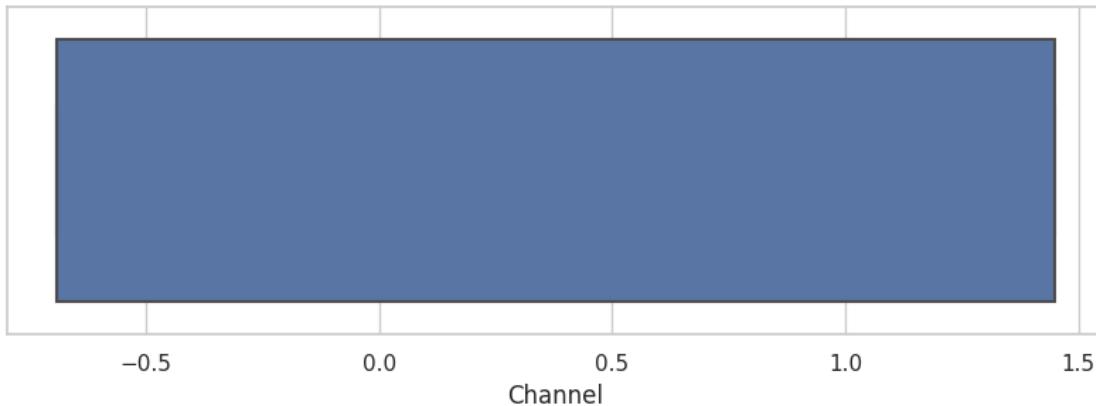
```
[154]: cluster_result
```

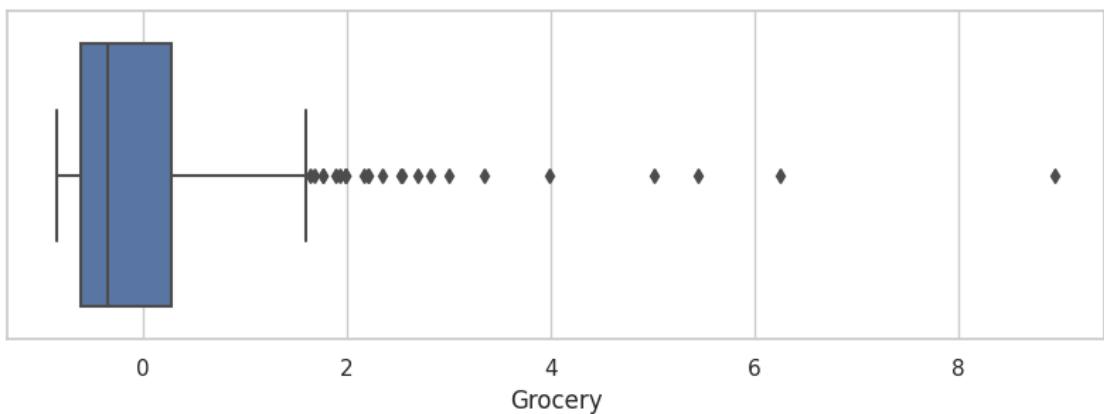
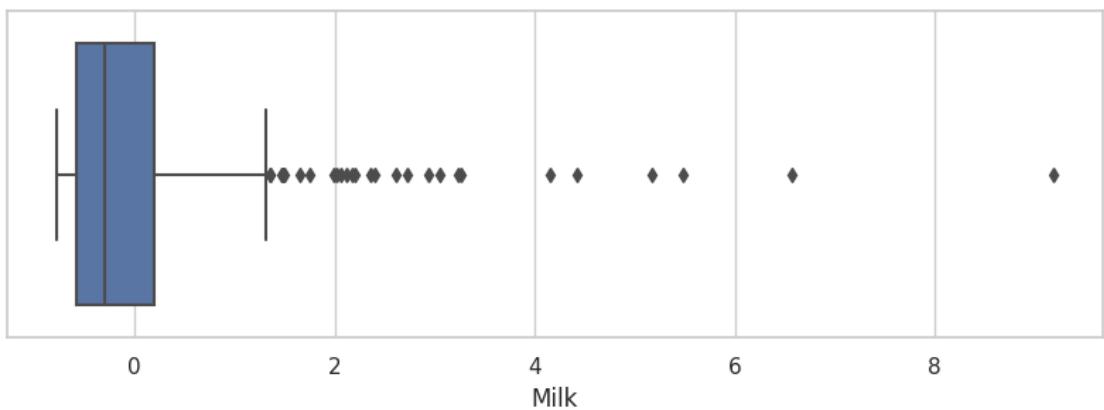
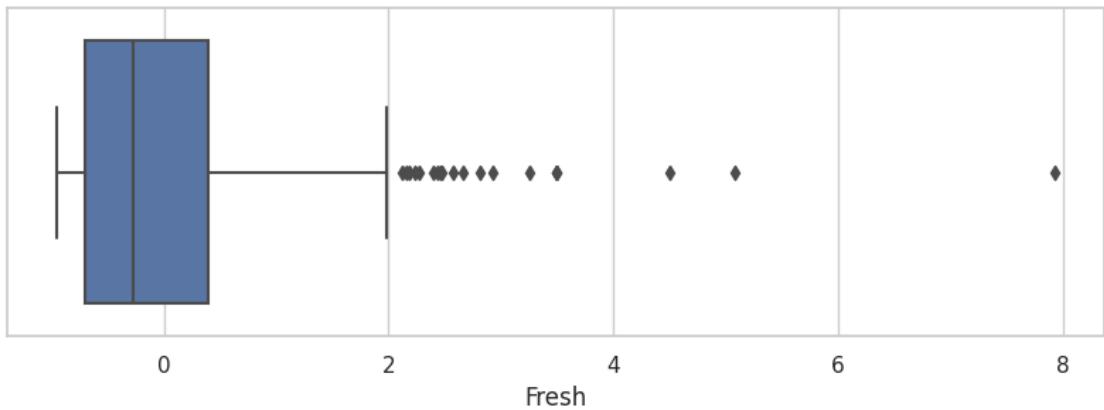
```
[154]:   Cluster_No.  count
0            0    172
```

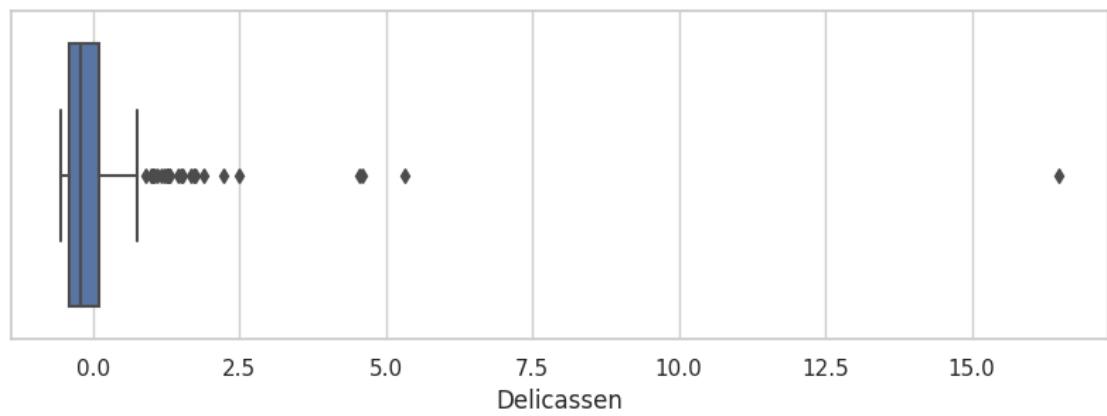
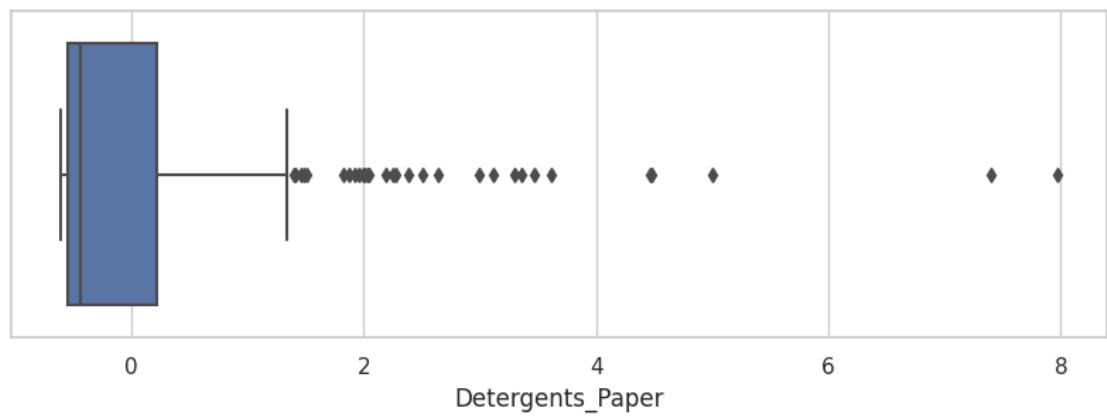
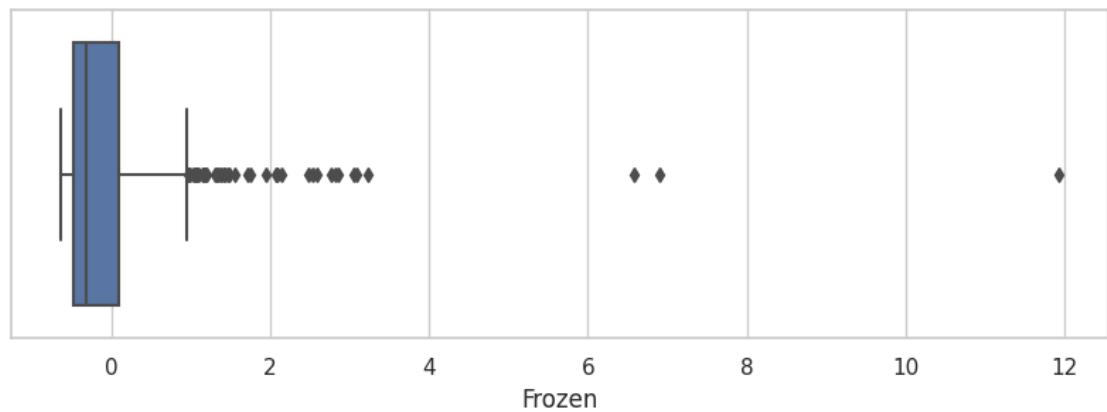
1	1	125
2	3	86
3	2	46
4	5	10
5	4	1

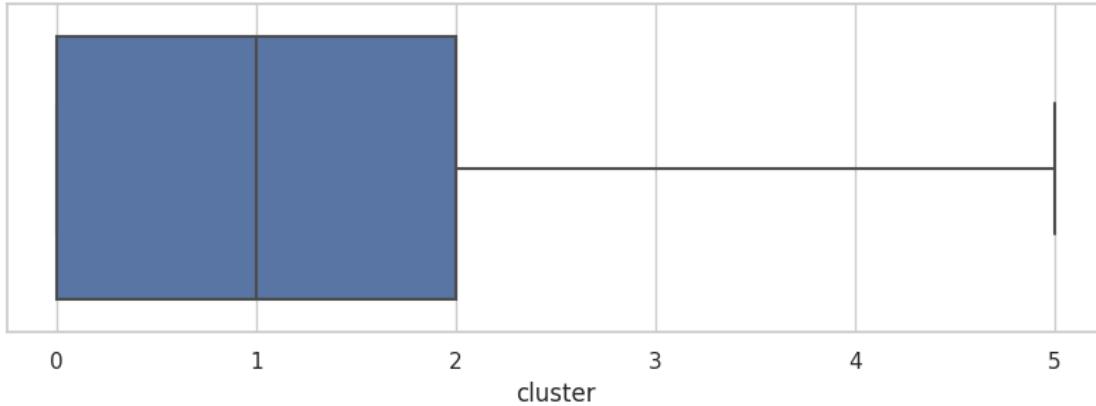
From the result above we can observe that 172 data values belong to cluster 1, and 125 data values for cluster 2. The most value belonging to cluster 1 and the least value belonging to cluster 5.

```
[155]: cols = frame.columns.to_list()
for i in cols:
    sns.set_theme(style="whitegrid")
    plt.figure(figsize=(10,3))
    ax = sns.boxplot(x=frame[i])
```









From the boxplot, we can observe that there seems to be a lot of data that are outliers in each attribute. However, focusing on the cluster boxplot we can observe that the median of the cluster column is 1, with the first quartile being 0 and third quartile being 2. The biggest box in the boxplot graph is the attribute Detergents\_Paper. From the boxplot graphs above we can observe the differences in size of box and the maximum of the data column, giving us insight to how much is usually spent on each product.

```
[156]: #CODE FROM: https://scikit-learn.org/stable/auto\_examples/cluster/plot\_kmeans\_silhouette\_analysis.html
range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(frame.to_numpy()) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(frame.to_numpy())

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
```

```

silhouette_avg = silhouette_score(frame.to_numpy(), cluster_labels)
print(
    "For n_clusters =",
    n_clusters,
    "The average silhouette_score is :",
    silhouette_avg,
)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(frame.to_numpy(), ▾
                                               cluster_labels)

y_lower = 10
for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels▴
                                                               == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(
        np.arange(y_lower, y_upper),
        0,
        ith_cluster_silhouette_values,
        facecolor=color,
        edgecolor=color,
        alpha=0.7,
    )

    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10 # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

```

```

ax1.set_yticks([]) # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(
    frame.to_numpy()[:, 0], frame.to_numpy()[:, 1], marker="o", s=30, lw=0, c=colors,
    alpha=0.7, c=colors, edgecolor="k"
)

# Labeling the clusters
centers = clusterer.cluster_centers_
# Draw white circles at cluster centers
ax2.scatter(
    centers[:, 0],
    centers[:, 1],
    marker="o",
    c="white",
    alpha=1,
    s=200,
    edgecolor="k",
)
for i, c in enumerate(centers):
    ax2.scatter(c[0], c[1], marker="$%d$" % i, alpha=1, s=50, edgecolor="k")

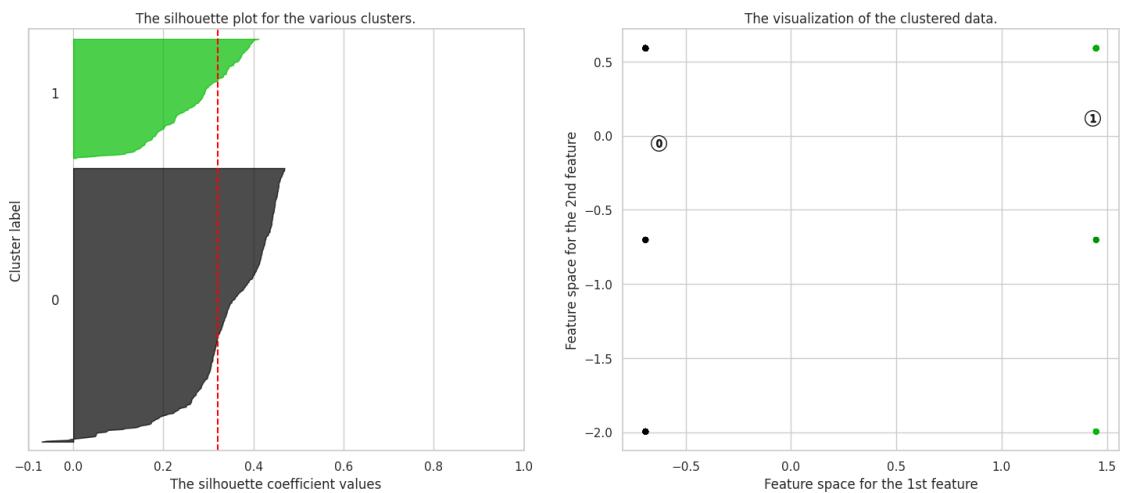
ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")

plt.suptitle(
    "Silhouette analysis for KMeans clustering on sample data with",
    n_clusters = "%d" % n_clusters,
    fontsize=14,
    fontweight="bold",
)
plt.show()

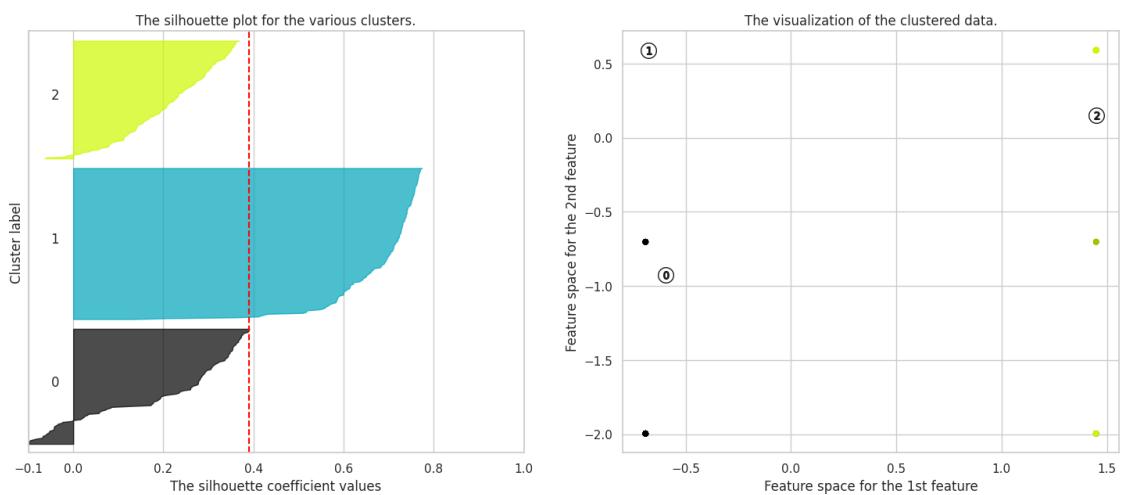
```

For n\_clusters = 2 The average silhouette\_score is : 0.31994052364013986  
 For n\_clusters = 3 The average silhouette\_score is : 0.3905860015885686  
 For n\_clusters = 4 The average silhouette\_score is : 0.422315001077336  
 For n\_clusters = 5 The average silhouette\_score is : 0.4662703414923966  
 For n\_clusters = 6 The average silhouette\_score is : 0.47300573904455817

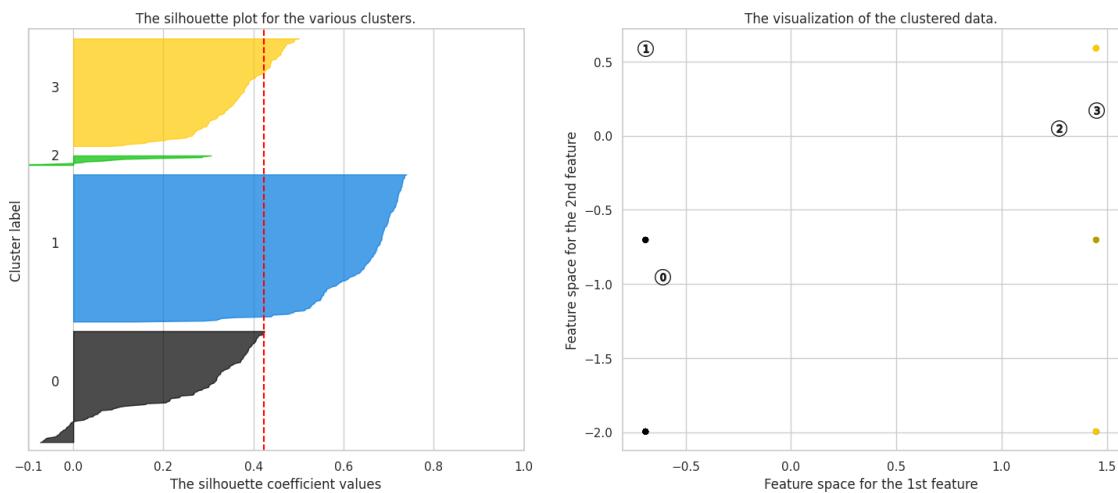
**Silhouette analysis for KMeans clustering on sample data with n\_clusters = 2**



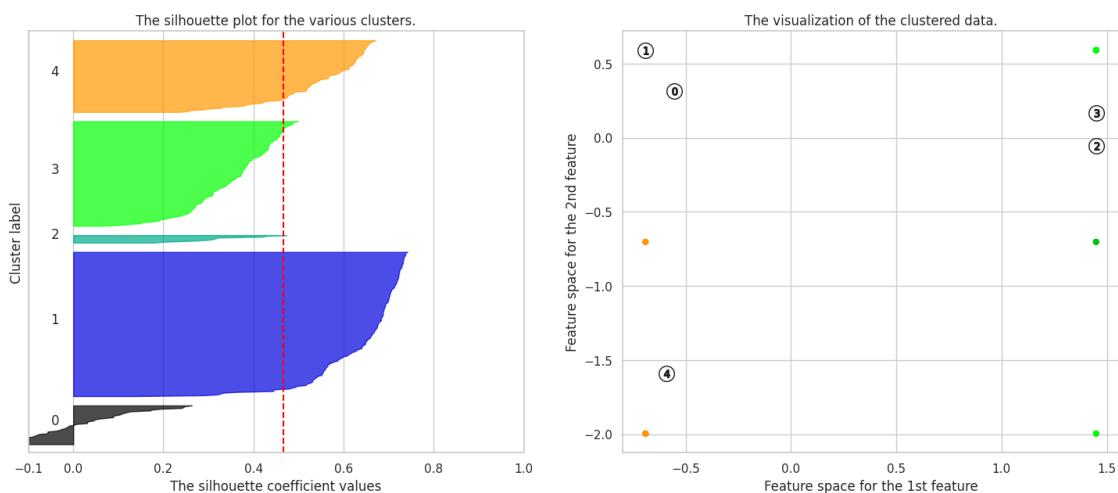
**Silhouette analysis for KMeans clustering on sample data with n\_clusters = 3**

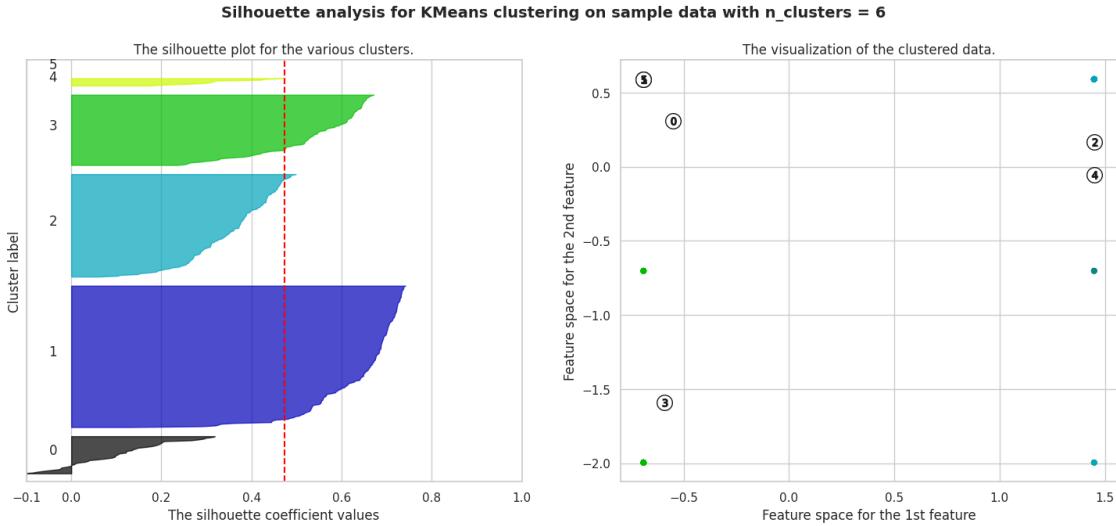


**Silhouette analysis for KMeans clustering on sample data with n\_clusters = 4**



**Silhouette analysis for KMeans clustering on sample data with n\_clusters = 5**



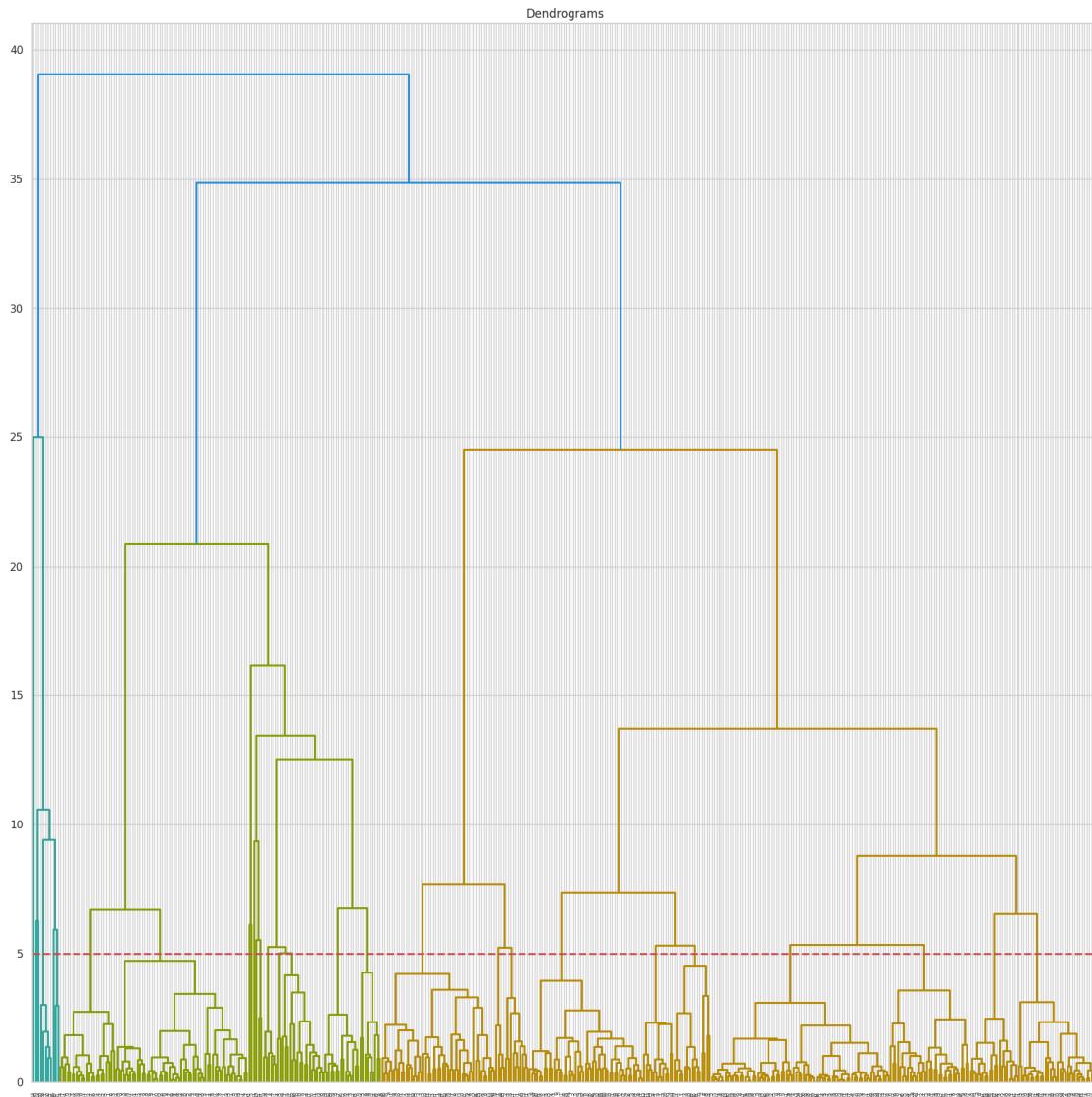


Our highest value silhouette score belongs to the cluster number 6 with a value of 0.473. Usually a silhouette score resides in the range [-1,1] with 1 indicating a compact cluster that is far away from other clusters and -1 being the worst possible value. Because our silhouette score for cluster number 6 is decent it shows that our model for clusters of 6 does well.

### Hierarchical Clustering

```
[157]: import scipy.cluster.hierarchy as shc
plt.figure(figsize=(20,20))
plt.title("Dendograms")
plt.style.use('Solarize_Light2')
test_df = scaled_df.drop(["Channel","Region"], axis = 1)
dend = shc.dendrogram(shc.linkage(test_df, method='ward'))
plt.axhline(y=5, color='r', linestyle='--')
```

[157]: <matplotlib.lines.Line2D at 0x7f4af835e850>



From the dendrogram we can observe that most of the verticles of shorter distance reside between 0-5. From the dendrogram we can infer that the best cluster value would be between 5-8 because of the distance of verticles are small; verticles show a degree of differences between clads.

## #Conclusion

Overall, in this lab we used unsupervised clustering techniques to understand customer segments. From our data we observed a strong suggestion that there be 5-8 different clusters of customer segments. These clusters show the similarities in each of the customer profiles. From our silhouette\_score, the best number of clusters to select would be 6 with a score of 0.473. Each cluster represents different patterns pertaining to each customer grouping. These groupings are dependent on the spending patterns of the customers, and our models suggest 6 different clusters of these spending patterns, with the most data points belonging to cluster 1 and the second most belonging in cluster 2.

# Notebook

August 2, 2023

## Lab 4

Leng Her

October 1, 2022

#Overview

In order to understand what factors have a possible influence on diabetes, we are using a measure of the progression of the disease and its observed relation to other variables. We will also be using a linear regression model to predict disease progression, and to measure whether or not an independent variable can be used to determine the progression of the disease. We will also be creating a multiple linear regression model “to estimate the relationship between two or more independent variables and one independent variable” (Bevans, R). With linear regression and multiple linear regression models we assume that error terms are independent of one another (Shweta).

Direct Links (References are at the bottom of Lab):

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt>

[https://github.com/Aerlinger/scikit-learn/blob/documentation\\_improvements/sklearn/datasets/descr/diabetes.rst](https://github.com/Aerlinger/scikit-learn/blob/documentation_improvements/sklearn/datasets/descr/diabetes.rst)

[https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_diabetes.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html)

<https://docs.google.com/document/d/1SRvCpvh5R6l4lQa0eGoSQdoYG2331APk84pFgiySfpE/edit>

<https://www.scribbr.com/statistics/multiple-linear-regression/>

<https://towardsdatascience.com/linear-regression-assumptions-why-is-it-important-af28438a44a1>

[https://dss.princeton.edu/online\\_help/analysis/interpreting\\_regression.htm](https://dss.princeton.edu/online_help/analysis/interpreting_regression.htm)

#Data

The data was obtained from a module sklearn containing the dataset. The dataset from sklearn is by default scaled whilst the dataset that is directly taken from the source is not scaled. The original source of the data can be found here: <https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.txt>. For more information about the source data refer to this link: [https://github.com/Aerlinger/scikit-learn/blob/documentation\\_improvements/sklearn/datasets/descr/diabetes.rst](https://github.com/Aerlinger/scikit-learn/blob/documentation_improvements/sklearn/datasets/descr/diabetes.rst).

And to understand more about the sklearn module with the dataset refer to this link: [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_diabetes.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html).

```
[11]: import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd

diabetes = datasets.load_diabetes(as_frame = True) #load dataset in scaled form
#refer to https://scikit-learn.org/stable/modules/generated/sklearn.datasets.
    ↪load_diabetes.html for more info

#print(type(diabetes.data)): <class 'pandas.core.frame.DataFrame'>

#import dataframe to pandas data, unscaled
df = pd.read_csv("https://www4.stat.ncsu.edu/~boos/var.select/diabetes.tab.
    ↪txt", sep="\t")

print(diabetes.DESCR)
```

.. \_diabetes\_dataset:

Diabetes dataset

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of n = 442 diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- age      age in years
- sex
- bmi      body mass index
- bp      average blood pressure
- s1      tc, total serum cholesterol
- s2      ldl, low-density lipoproteins
- s3      hdl, high-density lipoproteins
- s4      tch, total cholesterol / HDL

- s5 ltg, possibly log of serum triglycerides level
- s6 glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times `n\_samples` (i.e. the sum of squares of each column totals 1).

Source URL:

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>

For more information see:

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499.  
[https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle\\_2002.pdf](https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

[12]: df.head()# Y is representative of disease progression after a year  
 # s1-s6 are blood serum measurements  
 #unscaled data

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	151
1	48	1	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	75
2	72	2	30.5	93.0	156	93.6	41.0	4.0	4.6728	85	141
3	24	1	25.3	84.0	198	131.4	40.0	5.0	4.8903	89	206
4	50	1	23.0	101.0	192	125.4	52.0	4.0	4.2905	80	135

[13]: diabetes.data.head() #scaled data

	age	sex	bmi	bp	s1	s2	s3	\
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	
	s4	s5	s6					
0	-0.002592	0.019908	-0.017646					
1	-0.039493	-0.068330	-0.092204					
2	-0.002592	0.002864	-0.025930					
3	0.034309	0.022692	-0.009362					
4	-0.002592	-0.031991	-0.046641					

#EDA (Exploratory Data Analysis)

[14]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
```

```
Data columns (total 11 columns):
 #  Column  Non-Null Count  Dtype  
--- 
 0  AGE      442 non-null   int64  
 1  SEX      442 non-null   int64  
 2  BMI      442 non-null   float64 
 3  BP       442 non-null   float64 
 4  S1       442 non-null   int64  
 5  S2       442 non-null   float64 
 6  S3       442 non-null   float64 
 7  S4       442 non-null   float64 
 8  S5       442 non-null   float64 
 9  S6       442 non-null   int64  
 10 Y        442 non-null   int64  
dtypes: float64(6), int64(5)
memory usage: 38.1 KB
```

```
[15]: diabetes.data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #  Column  Non-Null Count  Dtype  
--- 
 0  age     442 non-null   float64 
 1  sex     442 non-null   float64 
 2  bmi     442 non-null   float64 
 3  bp      442 non-null   float64 
 4  s1      442 non-null   float64 
 5  s2      442 non-null   float64 
 6  s3      442 non-null   float64 
 7  s4      442 non-null   float64 
 8  s5      442 non-null   float64 
 9  s6      442 non-null   float64 
dtypes: float64(10)
memory usage: 34.7 KB
```

The above code gives us general information on the dataset and tells us if there are any missing values. From the result we can see that there are no missing entries with 442 entries and all columns containing that many entries. The difference between the scaled and unscaled seems to be that the unscaled are all floats meaning they are in decimal form while the scaled is not.

```
[16]: df.shape
```

```
[16]: (442, 11)
```

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

```
[17]: (442, 10)
```

```
[30]: diabetes['target']
```

```
[30]: 0      151.0
1      75.0
2     141.0
3    206.0
4    135.0
...
437   178.0
438   104.0
439   132.0
440   220.0
441   57.0
Name: target, Length: 442, dtype: float64
```

The above code shows the rows and columns of both the scaled and unscaled datasets. For the scaled dataset it has one less column than the unscaled as it is stored in a different array; the 'target' array which can be shown above aswell.

```
[29]: diabetes.feature_names
```

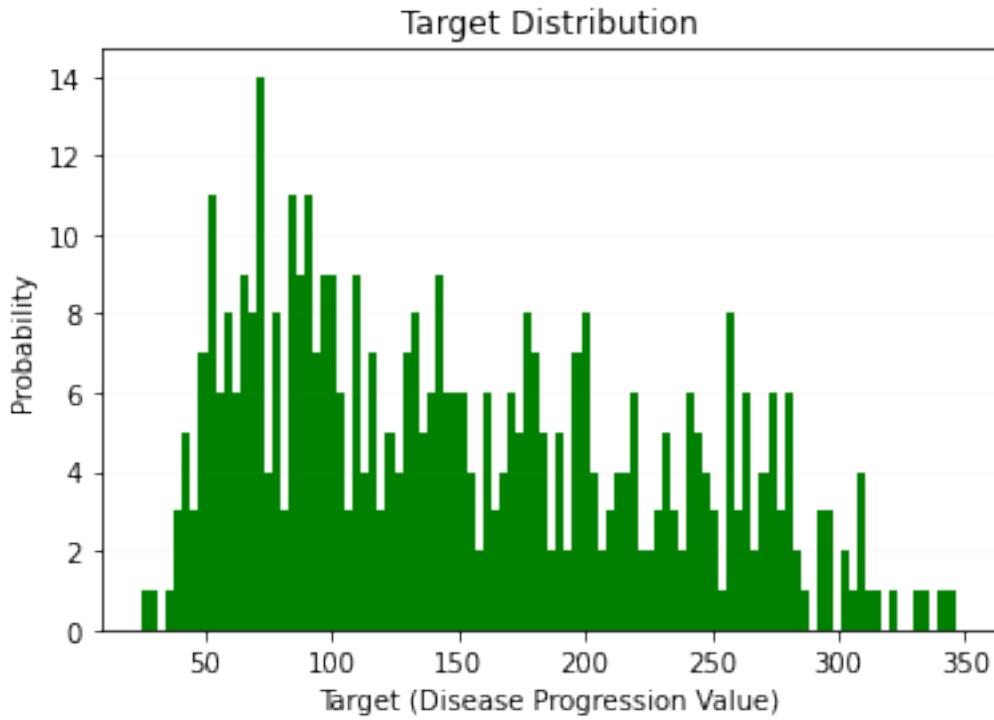
```
[29]: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
[27]: df.columns
```

```
[27]: Index(['AGE', 'SEX', 'BMI', 'BP', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'Y'],
dtype='object')
```

The above code shows the column names. The unscaled has an extra column which is the target variable in the scaled dataset. The Y is representative of the disease progression value.

```
[39]: n, bins, patches = plt.hist(x = diabetes.target, bins = 100, color = 'g')
plt.grid(axis='y', alpha=0.05)
plt.xlabel('Target (Disease Progression Value)')
plt.ylabel('Probability')
plt.title('Target Distribution')
plt.show()
```

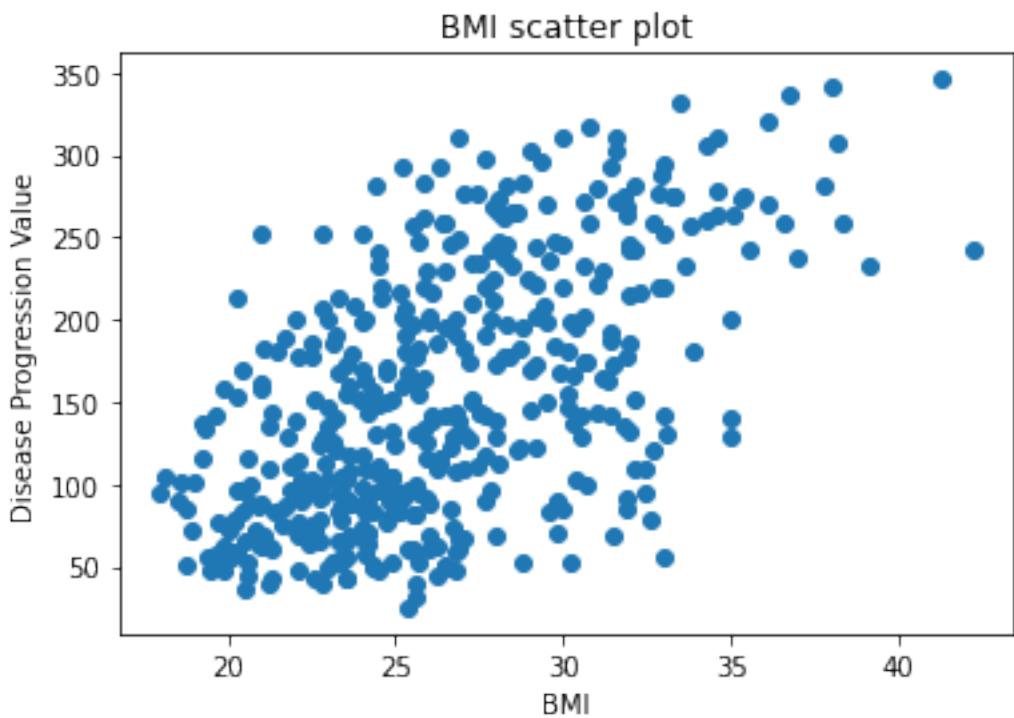


The code above is a histrogram of the ‘target’ value (Disease Progression Value). From the histogram we can see that the peak in our values is roughly around 75 which peaks at a probability of 14. Our target variable is skewed to the right.

Note: Code obatined for this can be shown here: <https://docs.google.com/document/d/1SRvCpvh5R6l4lQa0eGoS0>

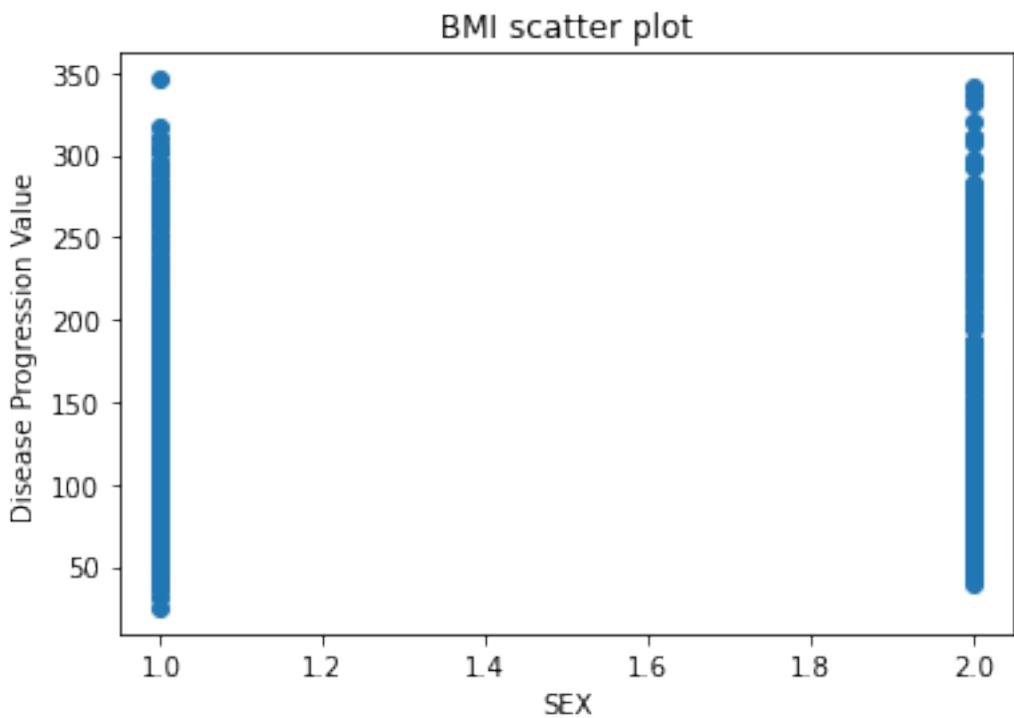
```
[41]: plt.scatter(df['BMI'], diabetes.target)

plt.xlabel('BMI')
plt.ylabel('Disease Progression Value')
plt.title('BMI scatter plot')
plt.show()
```



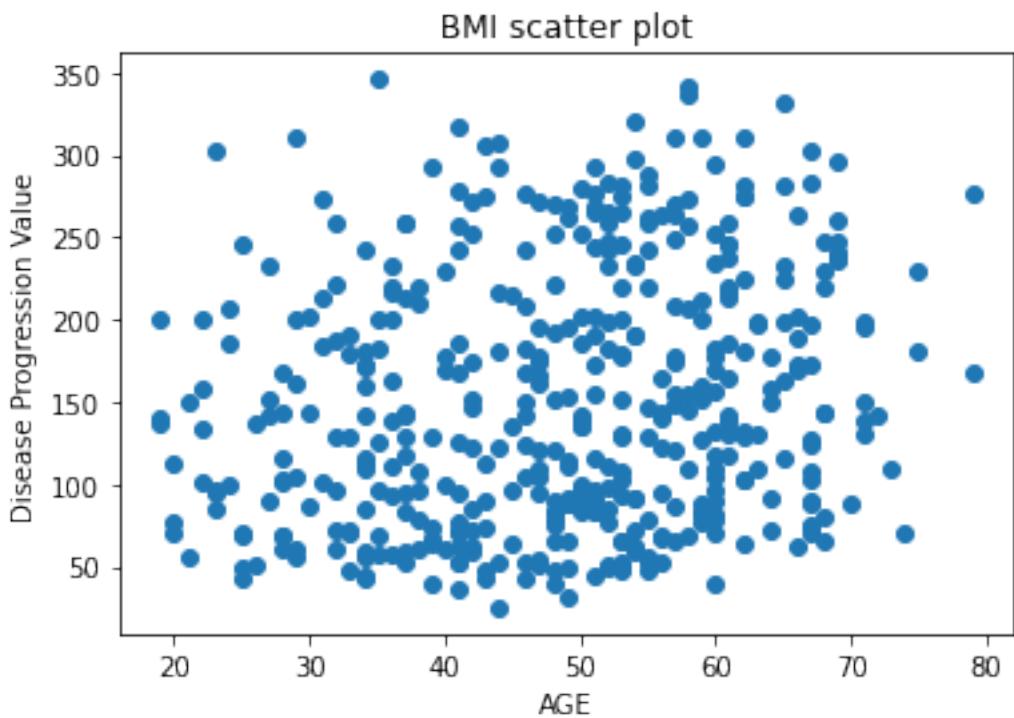
```
[44]: plt.scatter(df['SEX'], diabetes.target)

plt.xlabel('SEX')
plt.ylabel('Disease Progression Value')
plt.title('BMI scatter plot')
plt.show()
```



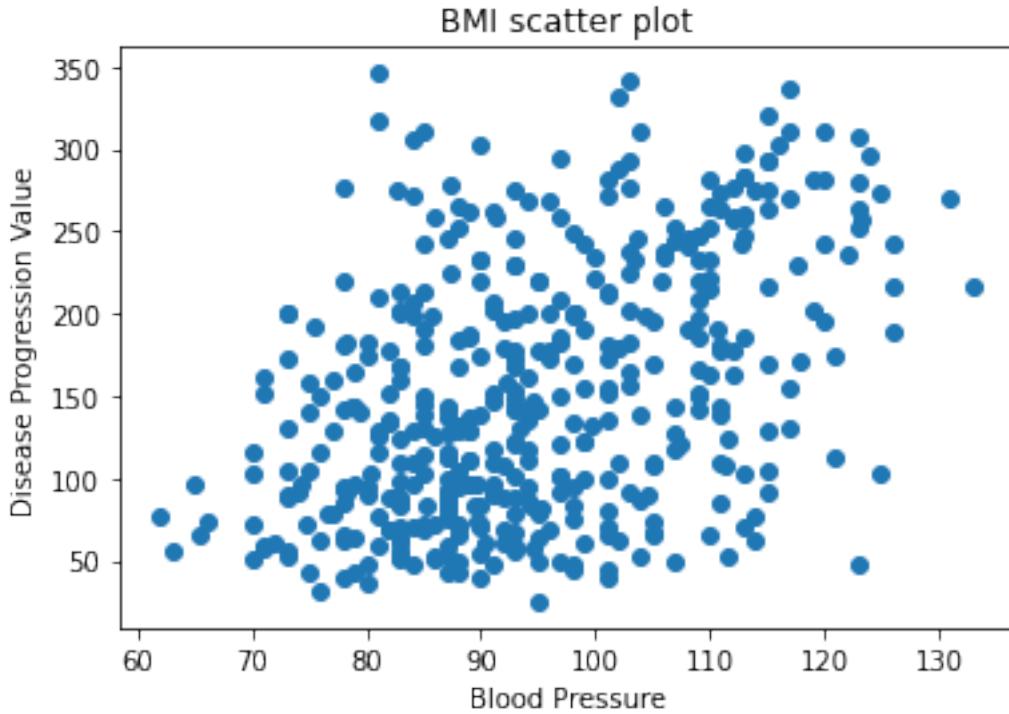
```
[45]: plt.scatter(df['AGE'], diabetes.target)
```

```
plt.xlabel('AGE')
plt.ylabel('Disease Progression Value')
plt.title('BMI scatter plot')
plt.show()
```



```
[46]: plt.scatter(df['BP'], diabetes.target)

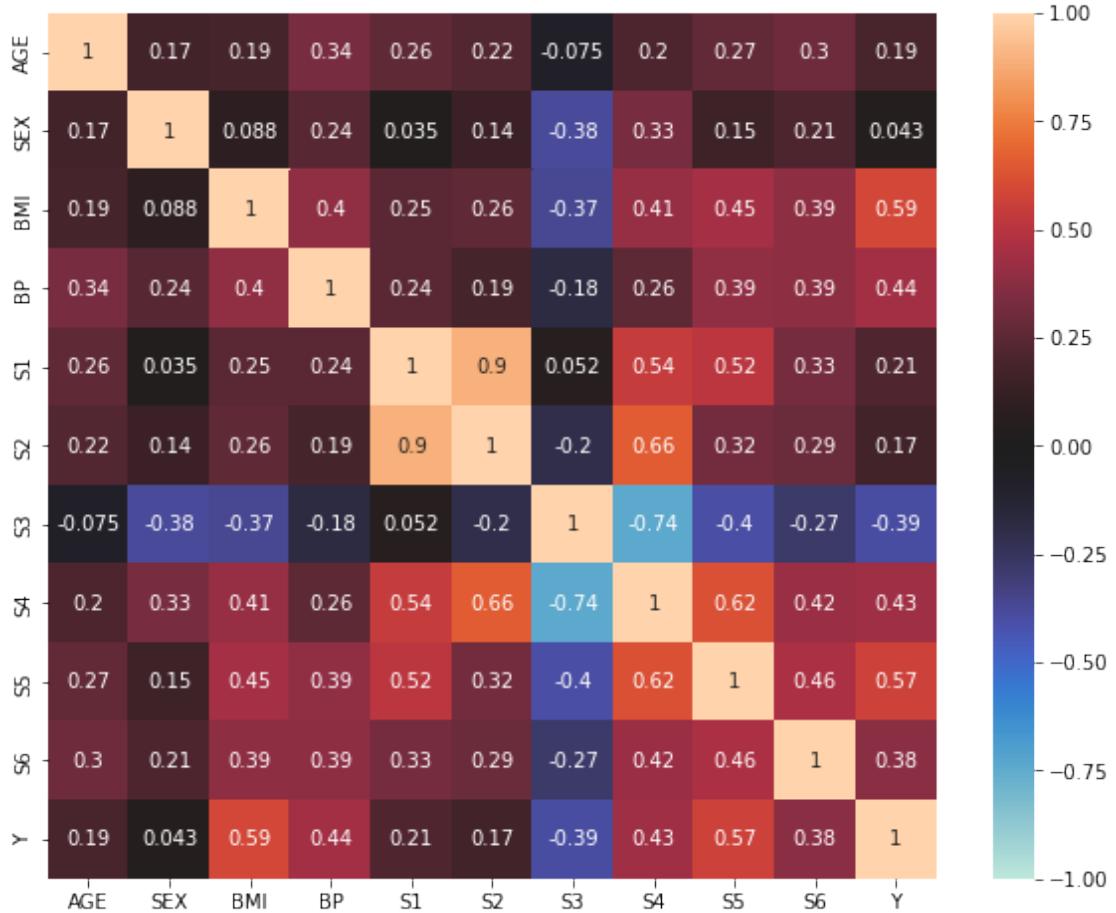
plt.xlabel('Blood Pressure')
plt.ylabel('Disease Progression Value')
plt.title('BMI scatter plot')
plt.show()
```



We use four scatter plots representing an independent variable and the dependent variable (Disease Progression Value) to observe if there exist a relationship. From the four scatter plots we can observe that BMI seems to have a positive relationship whereas the other independent values do not have a relationship with our dependent variable. In the case of BMI it would seem that as BMI increases the disease progression value also increases.

```
[50]: import seaborn as sns
fig, ax = plt.subplots(figsize = (10,8))
corrmat = df.corr()
sns.heatmap(corrmat,-1,1,ax=ax, center = 0, annot = True)
```

```
[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f73a8428910>
```



From the correlation matrix there does not seem to be a big correlation between progression of disease and independent variables other than BMI and S5. From the previous scatter plots we can see that BMI has a positive relationship with our dependent variable and with the correlation matrix we can see that it has a positive correlation with progression with a 0.59 value. Because the correlation value with BMI and other independent variables are not too high or moderate there does not seem to be a collinearity issue.

Note: S4 serum tends to be moderately or highly correlated to other serum tests.

#Models

We want to determine what kind of relationship BMI has with progression of disease. From the scatter plots and correlation matrix there seems to be a positive relationship so we will try a linear regression plot. Code for linear regression model can be found here with more information: [https://scikit-learn.org/stable/auto\\_examples/linear\\_model/plot\\_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py](https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py)

```
[112]: # Code source: Jaques Grobler
# License: BSD 3 clause
```

```

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature Feature is BMI
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()

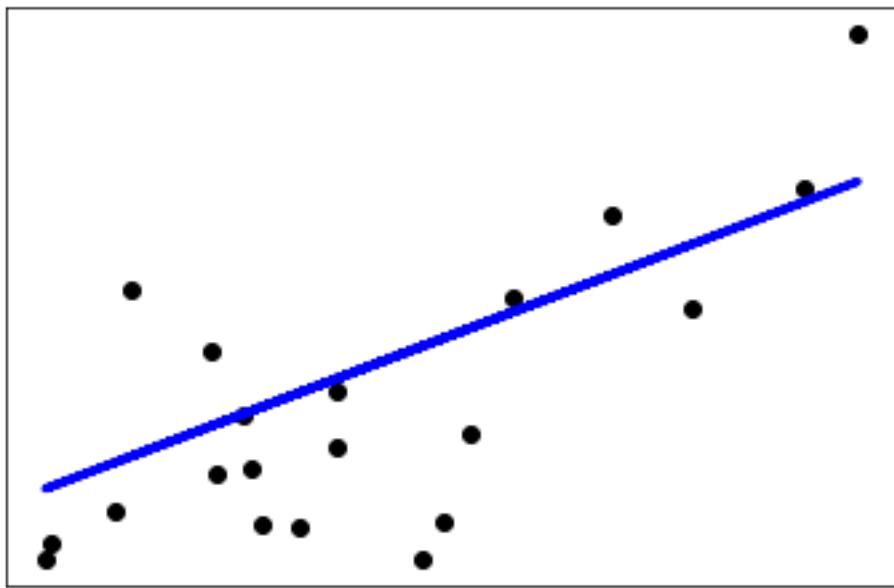
```

Coefficients:

```
[938.23786125]
```

```
Mean squared error: 2548.07
```

```
Coefficient of determination: 0.47
```



The model is not too accurate as the  $r^2$  value (coefficient of determination) is quite low. The coefficient of determination ranges from 0 to 100%; and in this case 0.47 can be interpreted as: 47% of our variance can be explained by BMI. Generally the higher the  $r^2$  the better the model is. It is also important to note that the mean squared error is quite high indicating that our model may have a large variance or bias. For this model the formula for the regression line with the coefficient values can be seen as:  $y = mx + b \rightarrow y = 0.47X + 938.23786125$

Because of the low  $r^2$  value and high MSE we should not accept this model.

More information on coefficient of determination : [https://dss.princeton.edu/online\\_help/analysis/interpreting\\_re](https://dss.princeton.edu/online_help/analysis/interpreting_re)

We will create a multiple linear regression model with all independent variables. Code can be found from:  
<https://docs.google.com/document/d/1SRvCpvh5R6l4lQa0eGoSQdoYG2331APk84pFgiySfpE/edit>

```
[114]: import statsmodels.api as sm
```

```
import statsmodels.api as sm

diabetes_X = diabetes.data

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

```

# Create linear regression object
lm = linear_model.LinearRegression()

# Train the model using the training sets
fit = lm.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = fit.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', fit.coef_)

# https://bigdata-madesimple.com/
# how-to-run-linear-regression-in-python-scikit-learn/
# get feature names
colnames = np.asarray(df.columns)

# https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
df_col = pd.DataFrame( data = np.asarray(df.columns) )
df_coef = pd.DataFrame( data = fit.coef_ )
df_tbl = pd.concat ( [df_col, df_coef] , axis = 1 )
df_tbl.columns = ['Columns', 'Coefficients']
df_tbl

```

```

# scipy.stats OLS linear regression model
X_train = sm.add_constant(diabetes_X_train)
lm2 = sm.OLS(diabetes_y_train, X_train).fit()
lm2.summary()

```

Coefficients:

```
[ 3.03499549e-01 -2.37639315e+02  5.10530605e+02  3.27736980e+02
-8.14131709e+02  4.92814588e+02  1.02848452e+02  1.84606489e+02
 7.43519617e+02  7.60951722e+01]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)
```

[114]: <class 'statsmodels.iolib.summary.Summary'>  
"""

#### OLS Regression Results

Dep. Variable:	y	R-squared:	0.512
Model:	OLS	Adj. R-squared:	0.500

Method:	Least Squares	F-statistic:	43.16			
Date:	Mon, 03 Oct 2022	Prob (F-statistic):	4.64e-58			
Time:	00:22:32	Log-Likelihood:	-2281.1			
No. Observations:	422	AIC:	4584.			
Df Residuals:	411	BIC:	4629.			
Df Model:	10					
Covariance Type:	nonrobust					
<hr/>						
	coef	std err	t	P> t	[0.025	0.975]
const	152.7643	2.658	57.469	0.000	147.539	157.990
age	0.3035	61.286	0.005	0.996	-120.169	120.776
sex	-237.6393	62.837	-3.782	0.000	-361.162	-114.117
bmi	510.5306	68.156	7.491	0.000	376.553	644.508
bp	327.7370	66.876	4.901	0.000	196.275	459.199
s1	-814.1317	424.044	-1.920	0.056	-1647.697	19.434
s2	492.8146	344.227	1.432	0.153	-183.850	1169.480
s3	102.8485	219.463	0.469	0.640	-328.561	534.258
s4	184.6065	167.336	1.103	0.271	-144.334	513.547
s5	743.5196	175.359	4.240	0.000	398.807	1088.232
s6	76.0952	68.293	1.114	0.266	-58.152	210.343
<hr/>						
Omnibus:	1.544	Durbin-Watson:	2.026			
Prob(Omnibus):	0.462	Jarque-Bera (JB):	1.421			
Skew:	0.004	Prob(JB):	0.491			
Kurtosis:	2.716	Cond. No.	224.			
<hr/>						

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

"""

Like our linear regression model with BMI, the multiple linear regression model does not do too well. Both models have similar  $r^2$  (coefficient of determination) values with the linear model regression being 0.47 and the multiple linear regression model being around ~0.5. With these  $r^2$  values we can infer that both models are not accurate. For the formula of this regression line it can be interpreted as:

$$y = mx + b \rightarrow y = \mathbf{0.5X} + \mathbf{152.7643}$$

We will create an different multiple linear regression model and compare it with the first multiple linear regression model. For our second multiple regression model we will remove the serum test independent variables ("s1", "s2", "s3", "s4", "s5", "s6"). We remove these independent variables as they are moderately or highly correlated with each other, which can be observed from the correlation matrix.

```
[115]: import statsmodels.api as sm

import statsmodels.api as sm

new_data = diabetes.data.drop(columns = ["s1", "s2", "s3", "s4", "s5", "s6"])
#remove correlated values

diabetes_X = new_data

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Create linear regression object
lm = linear_model.LinearRegression()

# Train the model using the training sets
fit = lm.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = fit.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', fit.coef_)

# https://bigdata-madesimple.com/
# ↵how-to-run-linear-regression-in-python-scikit-learn/
# get feature names
colnames = np.asarray(df.columns)

# https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html
df_col = pd.DataFrame( data = np.asarray(df.columns) )
df_coef = pd.DataFrame( data = fit.coef_ )
df_tbl = pd.concat ( [df_col, df_coef] , axis = 1 )
df_tbl.columns = ['Columns', 'Coefficients']
df_tbl

# scipy.stats OLS linear regression model
X_train = sm.add_constant(diabetes_X_train)
lm2 = sm.OLS(diabetes_y_train, X_train).fit()
lm2.summary()
```

Coefficients:  
[ 44.06337675 -101.00555176 776.95460996 419.97903542]

```

/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142:
FutureWarning: In a future version of pandas all arguments of concat except for
the argument 'objs' will be keyword-only
    x = pd.concat(x[::-order], 1)

[115]: <class 'statsmodels.iolib.summary.Summary'>
"""
            OLS Regression Results
=====
Dep. Variable:                      y      R-squared:                 0.394
Model:                            OLS      Adj. R-squared:             0.388
Method:                          Least Squares      F-statistic:                  67.74
Date:                Mon, 03 Oct 2022      Prob (F-statistic):        3.87e-44
Time:                    00:50:40      Log-Likelihood:           -2326.9
No. Observations:                  422      AIC:                     4664.
Df Residuals:                      417      BIC:                     4684.
Df Model:                           4
Covariance Type:            nonrobust
=====
              coef    std err          t      P>|t|      [0.025      0.975]
-----
const      152.8300     2.941      51.969      0.000     147.049     158.611
age         44.0634    65.523       0.672      0.502     -84.733     172.860
sex        -101.0056    63.749      -1.584      0.114     -226.316     24.305
bmi         776.9546    67.055      11.587      0.000     645.146     908.763
bp          419.9790    70.823       5.930      0.000     280.764     559.194
=====
Omnibus:                   10.370      Durbin-Watson:            1.928
Prob(Omnibus):               0.006      Jarque-Bera (JB):        6.439
Skew:                      0.135      Prob(JB):                  0.0400
Kurtosis:                   2.459      Cond. No.                  28.0
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

"""

Well...

It appears our second multiple linear regression model is much worse than our original multiple regression model. The  $r^2$  value is 0.394 meaning about ~40% of our variations can be explained by our independent variables (age, sex, bmi, blood pressure). With the coefficient of determination value being so low, it means that our model is accurate for prediction.

Our formula for this multiple regression line can be expressed as:

$$y = mx + b \rightarrow y = \mathbf{0.394X} + \mathbf{152.83}$$

#Conclusion

Overall, in this lab we used linear regression model and two multiple regression models to describe the relationship between our independent variables to our dependent variables. Our independent variables being: ‘age’, ‘sex’, ‘bmi’, ‘bp’, ‘s1’, ‘s2’, ‘s3’, ‘s4’, ‘s5’, ‘s6’; and our dependent variable being the progression of the disease (diabetes). From our data we can infer that there is a moderate positive relationship between the progression of the disease and BMI, however we fail to accurately predict values solely based on bmi. This could be due to multiple variables impacting our dependent variable. To verify this we conducted a multiple linear regression model to include all our independent variables and found a slightly better model, however the multiple regression model was observed to be not as significant either with a low coefficient of determination. Because there is an observed positive relationship between BMI and the progression of disease it can be recommended that lowering BMI can possibly lower the progression of the disease.

## 1 Refercenes

Bevans, R. (2022, June 1). Multiple linear regression: A quick guide (examples). Scribbr. Retrieved October 2, 2022, from <https://www.scribbr.com/statistics/multiple-linear-regression/>

Shweta. (2021, July 15). Linear regression assumptions-why is it important. Medium. Retrieved October 2, 2022, from <https://towardsdatascience.com/linear-regression-assumptions-why-is-it-important-af28438a44a1>

The Trustees of Princeton University. (n.d.). DSS - interpreting regression output. Princeton University. Retrieved October 2, 2022, from [https://dss.princeton.edu/online\\_help/analysis/interpreting\\_regression.htm](https://dss.princeton.edu/online_help/analysis/interpreting_regression.htm)

# Notebook

August 2, 2023

#LAB 3 Leng Her

INET 4061

September 25, 2022

## 1 Overview

In order to understand whether there is any statistical difference between age and race on voting, we will be conducting a statistical data analysis. The resultant data was analysed using a one-way ANOVA test. A one-way ANOVA test is a test that determines whether the means of at least two populations are different (Frost, J.). To conduct a one-way ANOVA test we need a continuous dependent variable as well as a categorical independent variable for a comparison of groups (Frost, J.). When conducting an ANOVA test we perform an F-test giving a(n) F -value which evaluates mean variances using an F-statistic ratio:  $F = \text{between-groups variance} / \text{within-group variance}$  (Frost, J.). With the ANOVA test we also assume a few things: the dependent variable is continuous, we have at least one categorical independent variable, the observations are independent, the groups have roughly equal variances, and the data in the groups follow a normal distribution (Frost, J.). Later, we also use the Bonferroni correction and Tukey's test for data analysis. The Bonferroni correction is a method to counteract multiple comparisons problem (Wikimedia Foundation). The Tukey's test is used to assess the significance of differences between pairs of group means (Post hoc tests – tukey HSD. bioSTTS).

Direct Links (References are at the bottom of Lab):

<https://statisticsbyjim.com/anova/>

[http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16\\_23.html](http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16_23.html)

[https://www.w3schools.com/python/pandas/ref\\_df\\_describe.asp](https://www.w3schools.com/python/pandas/ref_df_describe.asp)

<https://statisticsbyjim.com/anova/>

<https://statisticsbyjim.com/anova/f-tests-anova/>

[https://en.wikipedia.org/wiki/Bonferroni\\_correction](https://en.wikipedia.org/wiki/Bonferroni_correction)

<https://passel2.unl.edu/view/lesson/2e09f0055f13/14>

<https://biostats.w.uib.no/post-hoc-tests-tukey-hsd/>

#Data

The data method was taken from this site ([http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16\\_23.html](http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16_23.html)) containing two data sets. Both datasets values were randomly generated using numpy random. To get the data to show use the print function: print(variable) where variable is a dataframe or object. More information on the data can be found from the link above.

```
[ ]: #Load packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats

[ ]: #First dataset
np.random.seed(12)

races = ["asian", "black", "hispanic", "other", "white"]

# Generate random data
voter_race1 = np.random.choice(a= races,
                                p = [0.05, 0.15 ,0.25, 0.05, 0.5],
                                size=1000)

voter_age1 = stats.poisson.rvs(loc=18,
                               mu=30,
                               size=1000)

# Group age data by race
voter_frame1 = pd.DataFrame({"race":voter_race1, "age":voter_age1})
groups1 = voter_frame1.groupby("race").groups

# Extract individual groups
asian1 = voter_age1[groups1["asian"]]
black1 = voter_age1[groups1["black"]]
hispanic1 = voter_age1[groups1["hispanic"]]
other1 = voter_age1[groups1["other"]]
white1 = voter_age1[groups1["white"]]

print(voter_frame1)
#print(voter_age1)
#print(voter_race1)
```

	race	age
0	black	51
1	white	49
2	hispanic	51
3	white	48
4	asian	56
..	...	...

```

995    white   47
996    asian   40
997    white   50
998    white   51
999  hispanic  43

```

[1000 rows x 2 columns]

[ ]: *##Second Dataset ANOVA with group means differ*

```

np.random.seed(12)

# Generate random data
voter_race = np.random.choice(a= races,
                               p = [0.05, 0.15 ,0.25, 0.05, 0.5] ,
                               size=1000)

# Use a different distribution for white ages
white_ages = stats.poisson.rvs(loc=18,
                               mu=32,
                               size=1000)

voter_age = stats.poisson.rvs(loc=18,
                               mu=30,
                               size=1000)

voter_age = np.where(voter_race=="white", white_ages, voter_age)

# Group age data by race
voter_frame = pd.DataFrame({"race":voter_race, "age":voter_age})
groups = voter_frame.groupby("race").groups

# Extract individual groups
asian = voter_age[groups["asian"]]
black = voter_age[groups["black"]]
hispanic = voter_age[groups["hispanic"]]
other = voter_age[groups["other"]]
white = voter_age[groups["white"]]

print(voter_frame)
#print(voter_age)
#print(voter_race)

```

	race	age
0	black	54
1	white	51
2	hispanic	53
3	white	50

```

4      asian  51
..
995    white  51
996    asian  45
997    white  51
998    white  45
999  hispanic  56

[1000 rows x 2 columns]

##EDA (Exploratory data analysis)

[ ]: #For dataset 1 get average age based on races

asian_average = voter_frame1[voter_frame1.race == "asian"].age.mean() #gather
↳average age based on race for the asian sample population. Do same for rest
↳of races
black_average = voter_frame1[voter_frame1.race == "black"].age.mean()
hispanic_average = voter_frame1[voter_frame1.race == "hispanic"].age.mean()
other_average = voter_frame1[voter_frame1.race == "other"].age.mean()
white_average = voter_frame1[voter_frame1.race == "white"].age.mean()

average_dic = {"asian":asian_average,"black":black_average,"hispanic":
↳hispanic_average,"other":other_average,"white":white_average} #store data
↳into a dictionary which will then be turned into a dataframe using pandas

average_df = pd.DataFrame.from_dict(average_dic, orient= 'index') # turn
↳dictionary into a dataframe

sorted_average_df = average_df[0].sort_values(ascending = False) # sort the
↳values

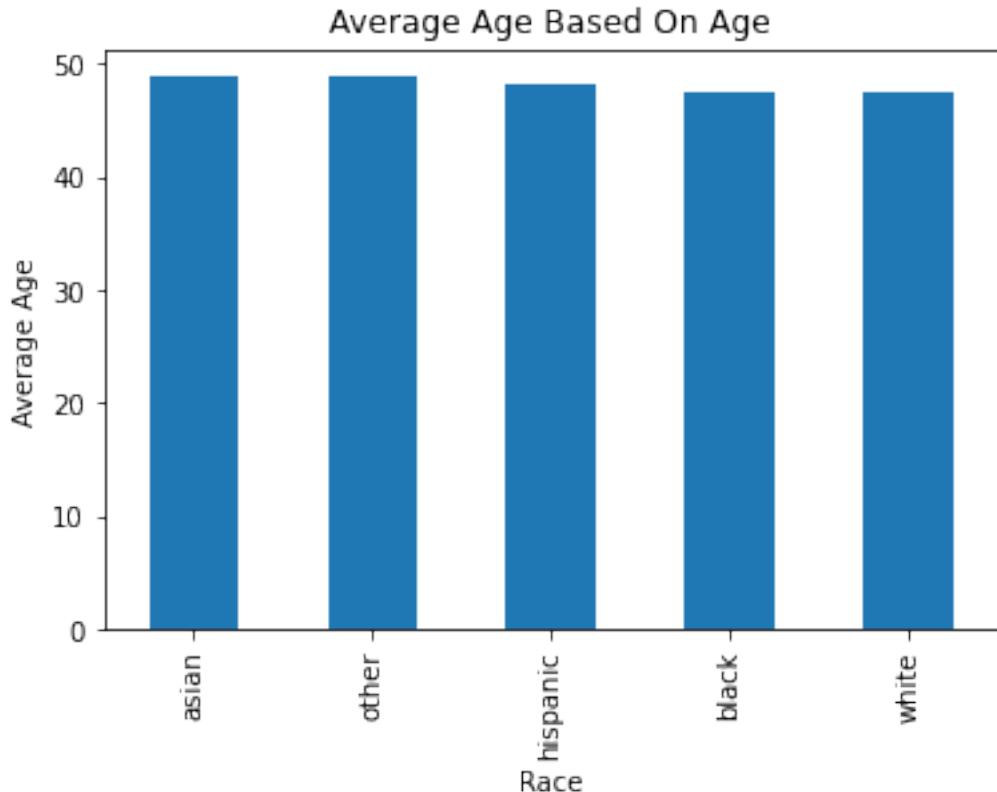
sorted_average_df.plot(kind = 'bar') # create a bar graph

#Labels for the graph
plt.xlabel("Race")
plt.ylabel("Average Age")
plt.title("Average Age Based On Age")

#Print out the findings for close to exact values
print("Race: Asian, Average age: %d" % asian_average)
print("Race: Black, Average age : %d" % black_average)
print("Race: Hispanic, Average age : %d" % hispanic_average)
print("Race: Other, Average age : %d" % other_average)
print("Race: White, Average age : %d\n" % white_average)

```

```
Race: Asian, Average age: 48  
Race: Black, Average age : 47  
Race: Hispanic, Average age : 48  
Race: Other, Average age : 48  
Race: White, Average age : 47
```



```
[ ]: #For dataset 2, we will apply the same methods as we did for dataset 1  
#Refer to comments on dataset 1 for more insight on code
```

```
asian_average2 = voter_frame[voter_frame.race == "asian"].age.mean()  
black_average2 = voter_frame[voter_frame.race == "black"].age.mean()  
hispanic_average2 = voter_frame[voter_frame.race == "hispanic"].age.mean()  
other_average2 = voter_frame[voter_frame.race == "other"].age.mean()  
white_average2 = voter_frame[voter_frame.race == "white"].age.mean()  
  
average_dic2 = {"asian":asian_average2,"black":black_average2,"hispanic":  
    ↪hispanic_average2,"other":other_average2,"white":white_average2}  
  
average_df2 = pd.DataFrame.from_dict(average_dic2, orient= 'index')
```

```

sorted_average_df2 = average_df2[0].sort_values(ascending = False)

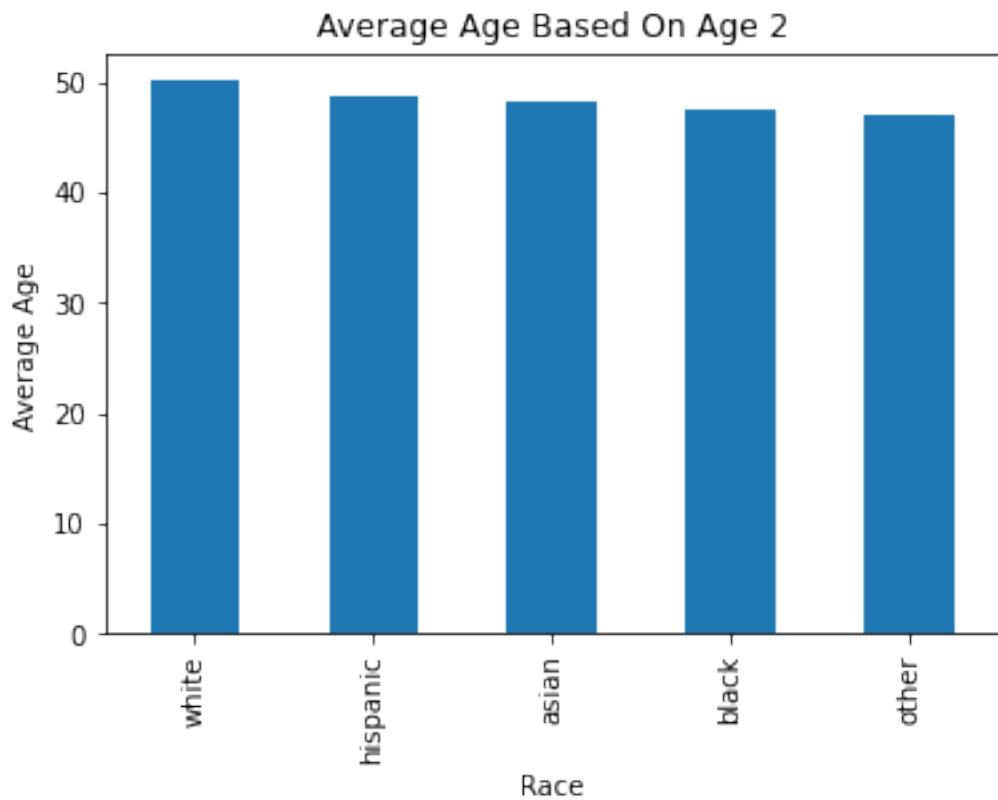
sorted_average_df2.plot(kind = 'bar')

plt.xlabel("Race")
plt.ylabel("Average Age")
plt.title("Average Age Based On Age 2")

print("Race: Asian, Average age: %d" % asian_average2)
print("Race: Black, Average age : %d" % black_average2)
print("Race: Hispanic, Average age : %d" % hispanic_average2)
print("Race: Other, Average age : %d" % other_average2)
print("Race: White, Average age : %d\n" % white_average2)

```

Race: Asian, Average age: 48  
 Race: Black, Average age : 47  
 Race: Hispanic, Average age : 48  
 Race: Other, Average age : 47  
 Race: White, Average age : 50



The above code gives us two bar graphs based on two separate datasets. The graphs show the average age based on race. From the graphs, in the first dataset it can be observed that the average age of voters regardless of race tend to be around 47-48 years old. The second dataset also expresses the same with a difference in the white race sample being a little older with an average age of 50.

(Note: The title of the graph representing the second dataset is denoted with a 2)

```
[ ]: #For data set 1 get mode based on race then across the whole dataset
asian_mode = voter_frame1[voter_frame1.race == "asian"].age.mode()
black_mode = voter_frame1[voter_frame1.race == "black"].age.mode()
hispanic_mode = voter_frame1[voter_frame1.race == "hispanic"].age.mode()
other_mode = voter_frame1[voter_frame1.race == "other"].age.mode()
white_mode = voter_frame1[voter_frame1.race == "white"].age.mode()

print("Race: Asian, this is the mode based on first dataset: %d" %
    ↪(asian_mode[0]))
print("Race: Black, this is the mode based on first dataset: %d" %
    ↪(black_mode[0]))
print("Race: Hispanic, this is the mode based first on dataset: %d" %
    ↪(hispanic_mode[0]))
print("Race: Other, this is the mode based on first dataset: %d" %
    ↪(other_mode[0]))
print("Race: White, this is the mode based on first dataset: %d" %
    ↪(white_mode[0]))
```

Race: Asian, this is the mode based on first dataset: 54  
 Race: Black, this is the mode based on first dataset: 45  
 Race: Hispanic, this is the mode based first on dataset: 45  
 Race: Other, this is the mode based on first dataset: 45  
 Race: White, this is the mode based on first dataset: 46

```
[ ]: #For dataset 2 do the same thing as we did for dataset 1
asian_mode2 = voter_frame[voter_frame.race == "asian"].age.mode()
black_mode2 = voter_frame[voter_frame.race == "black"].age.mode()
hispanic_mode2 = voter_frame1[voter_frame.race == "hispanic"].age.mode()
other_mode2 = voter_frame[voter_frame.race == "other"].age.mode()
white_mode2 = voter_frame[voter_frame.race == "white"].age.mode()

print("Race: Asian, this is the mode based on dataset 1: %d" % (asian_mode2[0]))
print("Race: Black, this is the mode based on dataset 1: %d" % (black_mode2[0]))
print("Race: Hispanic, this is the mode based on dataset 1: %d" %
    ↪(hispanic_mode2[0]))
print("Race: Other, this is the mode based on dataset 1: %d" % (other_mode2[0]))
print("Race: White, this is the mode based on dataset 1: %d" % (white_mode2[0]))
```

```
Race: Asian, this is the mode based on dataset 1: 43
Race: Black, this is the mode based on dataset 1: 50
Race: Hispanic, this is the mode based on dataset 1: 45
Race: Other, this is the mode based on dataset 1: 50
Race: White, this is the mode based on dataset 1: 47
```

Above we find the mode to denote, what the most occurring age is based on race. For the first dataset we can observe that the most occurring age for all the races other than Asian (54 years old) is around 45-46 years old. This tells us that most Asian voters tend to be a little older than other races. However for the second dataset we notice that the most occurring age tend to be a bit older for all races except for Hispanic. We also note that the second dataset shows that the most occurring race for Asian is 43, vastly different from our findings from the first dataset.

```
[ ]: #Number of people based on race for dataset 1, get percentage

showed_asian = len(voter_frame1[voter_frame1.race == "asian"]) #from the
    ↵dataframe obtain all occurrences of the asian race then count the length, do
    ↵this for all races
showed_black = len(voter_frame1[voter_frame1.race == "black"])
showed_hispanic = len(voter_frame1[voter_frame1.race == "hispanic"])
showed_other = len(voter_frame1[voter_frame1.race == "other"])
showed_white = len(voter_frame1[voter_frame1.race == "white"])

showed_total = showed_white + showed_other + showed_hispanic + showed_black +_
    ↵showed_asian #total the races: Should be 1000 as it was set in the Data
    ↵section

per_asian = (showed_asian/showed_total) * 100 #Get percentage for all races
per_black = (showed_black/showed_total) * 100
per_hispanic = (showed_hispanic/showed_total) * 100
per_other = (showed_other/showed_total) * 100
per_white = (showed_white/showed_total) * 100

sym = "%" #initialize symbol for printing

print("Below is the percentages of voters based on race for the first dataset:_"
    ↵") #print results
print("\nRace: Asian, the percentage is: %d%s" % (per_asian,sym))
print("Race: Black, the percentage is: %d%s" % (per_black,sym))
print("Race: Hispanic, the percentage is: %d%s" % (per_hispanic,sym))
print("Race: Other, the percentage is: %d%s" % (per_other,sym))
print("Race: White, the percentage is: %d%s" % (per_white,sym))
```

Below is the percentages of voters based on race for the first dataset:

```
Race: Asian, the percentage is: 4%
Race: Black, the percentage is: 14%
```

```
Race: Hispanic, the percentage is: 24%
Race: Other, the percentage is: 5%
Race: White, the percentage is: 51%
```

```
[ ]: #Number of people based on race for dataset 2, percentage
#Apply same method as we did for first dataset
#Refer to comments on first dataset for more insight

showed_asian2 = len(voter_frame[voter_frame.race == "asian"])
showed_black2 = len(voter_frame[voter_frame.race == "black"])
showed_hispanic2 = len(voter_frame[voter_frame.race == "hispanic"])
showed_other2 = len(voter_frame[voter_frame.race == "other"])
showed_white2 = len(voter_frame[voter_frame.race == "white"])

showed_total2 = showed_white2 + showed_other2 + showed_hispanic2 + showed_black2 + showed_asian2

per_asian2 = (showed_asian2/showed_total2) * 100
per_black2 = (showed_black2/showed_total2) * 100
per_hispanic2 = (showed_hispanic2/showed_total2) * 100
per_other2 = (showed_other2/showed_total2) * 100
per_white2 = (showed_white2/showed_total2) * 100

sym = "%"

print("Below is the percentages of voters based on race for the second dataset:")
print("\nRace: Asian, the percentage is: %d%s" % (per_asian2,sym))
print("Race: Black, the percentage is: %d%s" % (per_black2,sym))
print("Race: Hispanic, the percentage is: %d%s" % (per_hispanic2,sym))
print("Race: Other, the percentage is: %d%s" % (per_other2,sym))
print("Race: White, the percentage is: %d%s" % (per_white2,sym))
```

Below is the percentages of voters based on race for the second dataset:

```
Race: Asian, the percentage is: 4%
Race: Black, the percentage is: 14%
Race: Hispanic, the percentage is: 24%
Race: Other, the percentage is: 5%
Race: White, the percentage is: 51%
```

Above we obtain the percentage of voters based on race. From the results above it can be seen that the majority of voters are of the White race while the least amount being the Asian race. This is true for both datasets.

Note: Percentages can be seen set from the numpy.random() function above in the Data section.

```
[ ]: #Describe data
```

```
#Dataset 1
```

```
voter_frame1.describe()
```

```
[ ]: age
```

```
count    1000.000000
mean      47.773000
std       5.308074
min       34.000000
25%      44.000000
50%      48.000000
75%      51.000000
max       66.000000
```

```
[ ]: #Dataset 2
```

```
voter_frame.describe()
```

```
[ ]: age
```

```
count    1000.000000
mean      49.141000
std       5.722050
min       29.000000
25%      45.000000
50%      49.000000
75%      53.000000
max       72.000000
```

Above we use the `describe()` function from the pandas module, more insight on the function can be given here: [https://www.w3schools.com/python/pandas/ref\\_df\\_describe.asp](https://www.w3schools.com/python/pandas/ref_df_describe.asp). From the given tables we get statistical values pertaining to both datasets with key differences that can be observed at the minimum and maximum values. Everything else from the data tables are relatively similar.

```
#Models
```

```
[ ]: # Perform the ANOVA
```

```
set1 = stats.f_oneway(asian1, black1, hispanic1, other1, white1)
print(set1)
```

```
F_onewayResult(statistic=1.7744689357329695, pvalue=0.13173183201930463)
```

The above code is an ANOVA test performed on the first dataset with the null hypothesis being no significant difference between the means of each group, while the alternative hypothesis being a significant difference between the means of each group (More info: [http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16\\_23.html](http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16_23.html)). From the test we can observe that the p-value is greater than the significance level of 0.05 therefore we fail to reject the null hypothesis for the first dataset.

From dataset2

```
[ ]: set2 = stats.f_oneway(asian, black, hispanic, other, white)
      print(set2)
```

```
F_onewayResult(statistic=10.164699828386366, pvalue=4.5613242113994585e-08)
```

The above code is also an ANOVA test performed, however for the second data set with the same null hypothesis and alternative hypothesis. From the test we can observe that the p-value is extremely low with a 99% significance level, in which we can reject the null hypothesis.

post-hoc analysis with Bonferroni correction

```
[ ]: #run post-hoc analysis
      #perform separate t-test between all pairs
      #go in models
      # Get all race pairs
      race_pairs = []

      for race1 in range(4):
          for race2  in range(race1+1,5):
              race_pairs.append((races[race1], races[race2]))

      # Conduct t-test on each pair
      for race1, race2 in race_pairs:
          print(race1, race2)
          print(stats.ttest_ind(voter_age[groups[race1]],
                                voter_age[groups[race2]]))
```

```
asian black
Ttest_indResult(statistic=0.838644690974798, pvalue=0.4027281369339345)
asian hispanic
Ttest_indResult(statistic=-0.42594691924932293, pvalue=0.6704669004240726)
asian other
Ttest_indResult(statistic=0.9795284739636, pvalue=0.3298877500095151)
asian white
Ttest_indResult(statistic=-2.318108811252288, pvalue=0.020804701566400217)
black hispanic
Ttest_indResult(statistic=-1.9527839210712925, pvalue=0.05156197171952594)
black other
Ttest_indResult(statistic=0.28025754367057176, pvalue=0.7795770111117659)
black white
Ttest_indResult(statistic=-5.379303881281835, pvalue=1.039421216662395e-07)
hispanic other
Ttest_indResult(statistic=1.5853626170340225, pvalue=0.11396630528484335)
hispanic white
Ttest_indResult(statistic=-3.5160312714115376, pvalue=0.0004641298649066684)
other white
```

```
Ttest_indResult(statistic=-3.763809322077872, pvalue=0.00018490576317593065)
```

In the above code we run a post-hoc analysis to check which groups differ after receiving a positive ANOVA result (Python for Data Analysis Part 26). To conduct this test we do a pairwise t-test for each group (Python for Data Analysis Part 26). From the data above we can observe that some groups have a p-value greater than the significance level while others have lower p-values. This tells us that some groups may reject the null hypothesis while others may not.

Tukey test

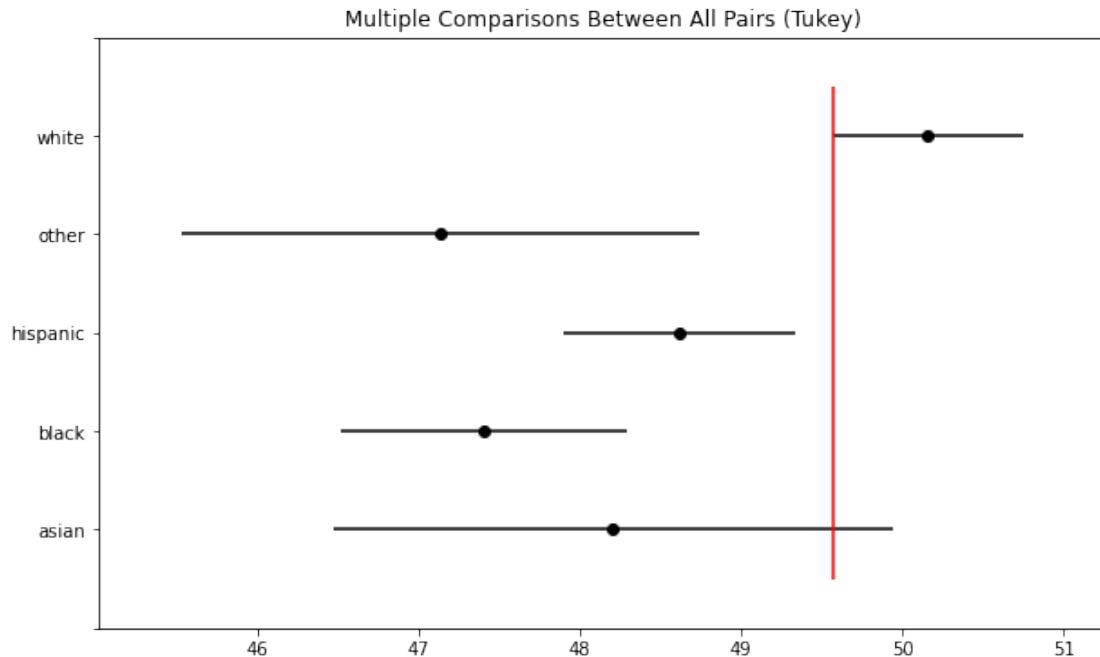
```
[ ]: from statsmodels.stats.multicomp import pairwise_tukeyhsd

tukey = pairwise_tukeyhsd(endog=voter_age,          # Data
                        groups=voter_race,      # Groups
                        alpha=0.05)             # Significance level

tukey.plot_simultaneous()    # Plot group confidence intervals
plt.vlines(x=49.57,ymin=-0.5,ymax=4.5, color="red")

tukey.summary()              # See test summary
```

```
[ ]: <class 'statsmodels.iolib.table.SimpleTable'>
```



Above we run another post-hoc test, however in this case we use the Bonferroni correction. The Bonferroni correction is used in the case that we may end up rejecting results that are actually significant (Python for Data Analysis Part 26: Analysis of

variance (ANOVA)). The Bonferroni correction has us adjust for a multiple comparison problem by dividing the significance level by the number of comparisons (Python for Data Analysis Part 26: Analysis of variance (ANOVA)). We also use another post-hoc test, the Tukey's test which access the significance of differences between pairs of group means (Post hoc tests – tukey HSD. bioSTTS).

Our resulting data after the Bonferroni correction and Tukey test show us whether we should reject the null hypothesis for each pair of groups (Python for Data Analysis Part 26). This is also represented in the graph.

## #Conclusions

Overall, in this lab we used ANOVA analysis to describe the average age of voters across a given group from our data. From our first resultant data, we can conclude that there is not a significance mean of variances thus failing to reject the null hypothesis. However, in the second dataset the resultant data shows us that we reject the null hypothesis for 3 pairs each including the “white” category (Python for Data Analysis Part 26). Because the “white” category is included all 3 instances of which we reject the null hypothesis, it can be inferred that the white group is likely to be different from the others. We also found that one other group overlaps with the white group’s confidence interval which can be observed from the graph (Python for Data Analysis Part 26).

## #Refernces

1. Frost, J. (n.d.). ANOVA Archives. Statistics By Jim. Retrieved September 24, 2022, from <https://statisticsbyjim.com/anova/>
2. Python for Data Analysis Part 26: Analysis of variance (ANOVA). Life Is Study. (n.d.). Retrieved September 25, 2022, from [http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16\\_23.html](http://hamelg.blogspot.com/2015/11/python-for-data-analysis-part-16_23.html)
3. Pandas DataFrame describe() Method. Pandas dataframe describe() method. (n.d.). Retrieved September 25, 2022, from [https://www.w3schools.com/python/pandas/ref\\_df\\_describe.asp](https://www.w3schools.com/python/pandas/ref_df_describe.asp)
4. Frost, J. (2022, May 19). How F-tests work in analysis of variance (ANOVA). Statistics By Jim. Retrieved September 25, 2022, from <https://statisticsbyjim.com/anova/f-tests-anova/>
5. Frost, J. (n.d.). ANOVA Archives. Statistics By Jim. Retrieved September 25, 2022, from <https://statisticsbyjim.com/anova/>
6. Wikimedia Foundation. (2022, August 31). Bonferroni correction. Wikipedia. Retrieved September 25, 2022, from [https://en.wikipedia.org/wiki/Bonferroni\\_correction](https://en.wikipedia.org/wiki/Bonferroni_correction)
7. Part 2: Statistical analysis and modeling. passel. (n.d.). Retrieved September 25, 2022, from <https://passel2.unl.edu/view/lesson/2e09f0055f13/14>
8. Post hoc tests – tukey HSD. bioSTTS. (n.d.). Retrieved September 25, 2022, from <https://biostats.w.uib.no/post-hoc-tests-tukey-hsd/>

```
[3]: !jupyter nbconvert --to HTML "Lab10.ipynb"
```

```
[NbConvertApp] Converting notebook Lab10.ipynb to HTML
[NbConvertApp] Writing 1066806 bytes to Lab10.html
```