Student: Duncan Ferguson
Student Id: 871641260
Class: Comp 4431-1
Assignment: Assignment 3
Date: 11/20/2021

You are to apply the following techniques to mine this data set:

- Association Rules
- Decision tree Classification
- Naive Bayes Classification
- kmeans and dbscan clustering

Also, you are to use Principle Component Analysis to reduce the number of attributes (aka features)

1. What can you deduce from the data set? (in otherwords, what attribute values are indicative of "success")

poutcome and campaign are very indicative of success. This makes sense as poutcome is people that subscribed before and campaign is the type of campaign that they are targeted with. Association rules did not find anything any rules that were actually interesting or strong with relation to predicting success. The Decision tree gave the best results followed by the Gaussian Model. K-means and DBscan seemed to be a bit lacking which was probably due to the fact that there were a lot of features.

2. Which mining techniques yielded the best results for what? How do you define "best results"?

Best result is defined by the highest accuracy

The technique that yielded the best results was running a backward selection with a decision tree.
Accuracy: 0.8911865531350216 Features: ['poutcome', 'campaign']

This makes the most sense on the campaign being successful. They had previously subscribed. And the type of campaign that is run predicted if they would subscribe.

This was followed up by decision tree using the forward model selection which added a few features. Accuracy: 0.8916288842198385 Add Feature: contact
Model Features ['contact', 'poutcome', 'campaign', 'loan']
Contact and Loan seem to have added to the forward selection. This is due to me just dropping values in the backward selection model that made the model drop the first value that

The backward model selection with the gaussian found the most accurate model to just be:
Accuracy: 0.8824505142098861
Features: ['campaign']

Forward Selection with the gaussian model
accuracy = 0.885546831803605
selection = ['housing', 'duration', "age"]

3. How useful was Principle Component Analysis? For what number of components did you get the "best results."

PCA was alright. There was a variety of different answers. There was an even spread and it did not contain a noticable dip that made it easy to just cut off the data. It was useful for DBSCAN and K-means Clustering to aid in the visualization. The PCA only explained about 33 percent of the variation with only three features. If there was a more noticable dip instead of a even spread on the features determining variation I would have ben able to utilize it a bit more.

Please include tables and/or graphs to justify your statements about which are best.
All of the graphs and tables are below following with the analysis.

Start of The Code

Main Library Imports Below

```
import graphviz
import pandas as pd
import numpy as np
from csv import reader

# Sklearn Libraries
from sklearn import tree
from sklearn.model_selection import train_test_split, ParameterGrid
from sklearn.metrics import *
from sklearn.naive_bayes import *
from sklearn.cluster import KMeans, DBSCAN
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.neighbors import NearestNeighbors
from kneed import KneeLocator

# Importing mlxtend Libraries
```

```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Graphing Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing the main document

Setting the random seed and setting view options

3
```
np.random.seed(0)
df =  pd.read_csv("bank-full.csv")

pd.set_option('display.max_columns', None)
pd.set_option('max_columns', None)
pd.set_option('max_rows', None)
pd.options.display.width = 0   # automatically ajust to window length

df_a = df.copy()
df_b = df.copy()
```

4
```
df_a.head()
```

4

	age	job	marital	education	default	balance	housing	loan	contact
0	58	management	married	tertiary	no	2143	yes	no	unknown
1	44	technician	single	secondary	no	29	yes	no	unknown
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown
4	33	unknown	single	unknown	no	1	no	no	unknown

Association Rule Analysis Below

Feature Engineering

First Cleaning up the data and creating categories

5
```
for index, row in df_a.iterrows():
        if row['balance'] < 0:
                row['balance'] = 'inDebt'
        else:
                row['balance'] = 'notInDebt'

# Creating Categories for balance
df_a.loc[(df_a['balance'] >= 10000), 'balanceSummary'] = 'veryPositive'
df_a.loc[((df_a['balance'] < 0) & (df_a['balance'] >= -10000)), 'balanceSummary'] = 'negative'
df_a.loc[((df_a['balance'] < 10000) & (df_a['balance'] >= 0)), 'balanceSummary'] = 'positive'
df_a.loc[df_a['balance'] < -500, 'balanceSummary'] = 'veryNegative'
```

```
# Creating Categories for ages
df_a.loc[((df_a['age'] < 25) & (df_a['age'] >= 0)),  'ageBand'] = 'ageBand1'
df_a.loc[((df_a['age'] < 30) & (df_a['age'] >= 25)), 'ageBand'] = 'ageBand2'
df_a.loc[((df_a['age'] < 40) & (df_a['age'] >= 30)), 'ageBand'] = 'ageBand3'
df_a.loc[((df_a['age'] < 50) & (df_a['age'] >= 40)), 'ageBand'] = 'ageBand4'
df_a.loc[((df_a['age'] < 120) & (df_a['age'] >= 50)),'ageBand'] = 'ageBand5'

# Creating Categories for default
df_a.loc[(df_a['default'] == 'no'), 'defaultValue'] = 'defaultNo'
df_a.loc[(df_a['default'] == 'yes'), 'defaultValue'] = 'defaultYes'

# Creating Categories for Housing
df_a.loc[(df_a['housing'] == 'no'), 'housingVal'] = 'houseNo'
df_a.loc[(df_a['housing'] == 'yes'), 'housingVal'] = 'houseYes'

# Creating Categories for Loans
df_a.loc[(df_a['loan'] == 'no'), 'loanVal'] = 'loanNo'
df_a.loc[(df_a['loan'] == 'yes'), 'loanVal'] = 'loanYes'

# Dropping all values that are not strings
df_a = df_a.drop(['age', 'day', 'duration', 'balance', 'campaign', 'previous', 'default', 'housing', 'loa
                  axis=1)

# Displaying Head
df_a.head()
```

5

	job	marital	education	contact	month	poutcome	y	balanceSummary
0	management	married	tertiary	unknown	may	unknown	no	positive
1	technician	single	secondary	unknown	may	unknown	no	positive
2	entrepreneur	married	secondary	unknown	may	unknown	no	positive
3	blue-collar	married	unknown	unknown	may	unknown	no	positive
4	unknown	single	unknown	unknown	may	unknown	no	positive

Displaying all the Unique Values

6

```
for col in df_a:
    print(col, ":", df_a[col].unique())
```

```
job : ['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']
marital : ['married' 'single' 'divorced']
education : ['tertiary' 'secondary' 'unknown' 'primary']
contact : ['unknown' 'cellular' 'telephone']
month : ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
poutcome : ['unknown' 'failure' 'other' 'success']
y : ['no' 'yes']
balanceSummary : ['positive' 'negative' 'veryPositive' 'veryNegative']
ageBand : ['ageBand5' 'ageBand4' 'ageBand3' 'ageBand2' 'ageBand1']
defaultValue : ['defaultNo' 'defaultYes']
housingVal : ['houseYes' 'houseNo']
loanVal : ['loanNo' 'loanYes']
```

Exporting the dataframe as a CSV and reimporting it to speed up run times.

7
```
df_a.to_csv("outfile.csv", index=False)
file = []
with open("outfile.csv", 'r') as read_obj:
    csv_reader = reader(read_obj)
    for row in csv_reader:
        file.append(row)
```

Creating transact 1 hot boolean encoded numpy array and displaying dataframe head

8
```
te = TransactionEncoder()
te_ary = te.fit(file).transform(file)
df_a_encoded = pd.DataFrame(te_ary, columns=te.columns_)
df_a_encoded.head()
```

8

	admin.	ageBand	ageBand1	ageBand2	ageBand3	ageBand4	ageBand5	apr	aug
0	False	True	False	False	False	False	False	False	False
1	False	False	False	False	False	False	True	False	False
2	False	False	False	False	False	True	False	False	False
3	False	False	False	False	True	False	False	False	False
4	False	False	False	False	False	True	False	False	False

Creating Filter Function for frequent Item Sets

9
```
def filter_freq_items(freq_items,items=1):
    """This function filters the frequent items so that we can look a certain n-itemsets"""
    filtered_freq_items = freq_items[freq_items["length"] == items].copy()
    return filtered_freq_items
```

Creating Frequent Items List and Association Rules

10
```
# Creating Frequent Items List
freq_items = apriori(df_a_encoded, min_support=.5, use_colnames=True)
freq_items['length'] = freq_items['itemsets'].apply(lambda x: len(x))
freq_items.sort_values(["support", "length"], inplace=True, ascending=False)
freq_items.head(10)
```

10

	support	itemsets	length
1	0.981952	(defaultNo)	1
6	0.898346	(positive)	1
17	0.890073	(positive, defaultNo)	2

	support	itemsets	length
5	0.882996	(no)	1
16	0.866120	(no, defaultNo)	2
3	0.839755	(loanNo)	1
14	0.828386	(loanNo, defaultNo)	2
8	0.826528	(unknown)	1
19	0.809741	(defaultNo, unknown)	2
27	0.788994	(positive, no)	2

11
```
# Creating Association Rules
rules = association_rules(freq_items, metric="confidence", min_threshold=0.3)
rules.sort_values(["lift"], inplace=True, ascending=False)
rules.head(10)
```

11

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
232	(houseYes)	(no, defaultNo)	0.555826	0.866120	0.504048	0.906844	1.047020	0.
229	(no, defaultNo)	(houseYes)	0.866120	0.555826	0.504048	0.581961	1.047020	0.
227	(houseYes)	(no)	0.555826	0.882996	0.513028	0.923000	1.045306	0.
226	(no)	(houseYes)	0.882996	0.555826	0.513028	0.581008	1.045306	0.
231	(no)	(houseYes, defaultNo)	0.882996	0.546205	0.504048	0.570838	1.045100	0.
230	(houseYes, defaultNo)	(no)	0.546205	0.882996	0.504048	0.922818	1.045100	0.
177	(positive, unknown)	(no, loanNo, defaultNo)	0.738830	0.722928	0.557463	0.754520	1.043701	0.
176	(no, loanNo, defaultNo)	(positive, unknown)	0.722928	0.738830	0.557463	0.771118	1.043701	0.
169	(positive, defaultNo, unknown)	(no, loanNo)	0.731045	0.733478	0.557463	0.762556	1.039644	0.
184	(no, loanNo)	(positive, defaultNo, unknown)	0.733478	0.731045	0.557463	0.760027	1.039644	0.

Association rules analysis

Below are different tables that show the break down of the frequent item sets

Frequent Items

```
12 filter_freq_items(freq_items,5)
```

12

	support	itemsets	length
51	0.557463	(unknown, positive, no, loanNo, defaultNo)	5

```
13 filter_freq_items(freq_items,4)
```

13

	support	itemsets	length
46	0.660444	(positive, no, loanNo, defaultNo)	4
49	0.659692	(positive, no, defaultNo, unknown)	4
48	0.622180	(positive, loanNo, defaultNo, unknown)	4
47	0.611298	(no, loanNo, defaultNo, unknown)	4
50	0.562107	(positive, no, loanNo, unknown)	4

```
14 filter_freq_items(freq_items,3).head(10)
```

14

	support	itemsets	length
39	0.781253	(positive, no, defaultNo)	3
35	0.760440	(positive, loanNo, defaultNo)	3
40	0.733323	(no, defaultNo, unknown)	3
41	0.731045	(positive, defaultNo, unknown)	3
34	0.722928	(no, loanNo, defaultNo)	3
36	0.679842	(loanNo, defaultNo, unknown)	3
45	0.666991	(positive, no, unknown)	3
42	0.665399	(positive, no, loanNo)	3
44	0.627157	(positive, loanNo, unknown)	3
43	0.621096	(no, loanNo, unknown)	3

```
15 filter_freq_items(freq_items,2).head(10)
```

15

	support	itemsets	length
17	0.890073	(positive, defaultNo)	2

	support	itemsets	length
16	0.866120	(no, defaultNo)	2
14	0.828386	(loanNo, defaultNo)	2
19	0.809741	(defaultNo, unknown)	2
27	0.788994	(positive, no)	2
22	0.765770	(positive, loanNo)	2
28	0.749049	(no, unknown)	2
29	0.738830	(positive, unknown)	2
21	0.733478	(no, loanNo)	2
23	0.690392	(loanNo, unknown)	2

After breaking down the frequent item sets it is interesting to learn that there is only 1 frequent item set that contains 5 words with a support over 5. This will make our cluster analysis a bit more interesting later. With that said. Positive balance, unknown job, no default and no loan is not is not likely to subscribe at least with a .557463 support.

It is also interesting to see that the majority of the frequent item sets contain "no" meaning that they are not likely to subscribe. The majority of the banks customers had defaultno with a positive balance summary, or were not likely to subscribe.

It is also interesting to see that yes is not included in the frequent item sets. meaning that the majority of people that say yes do not have a whole lot in common with others that said yes

Association rules

16
```
rules.sort_values('lift', ascending=False).head(5)
```

16

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
232	(houseYes)	(no, defaultNo)	0.555826	0.866120	0.504048	0.906844	1.047020	0.
229	(no, defaultNo)	(houseYes)	0.866120	0.555826	0.504048	0.581961	1.047020	0.
227	(houseYes)	(no)	0.555826	0.882996	0.513028	0.923000	1.045306	0.
226	(no)	(houseYes)	0.882996	0.555826	0.513028	0.581008	1.045306	0.
231	(no)	(houseYes, defaultNo)	0.882996	0.546205	0.504048	0.570838	1.045100	0.

The top 5 most interesting rules, defined by the highest lift values is that homeowners are not likely to default and subscribe. This not exactly shocking to find out. It is intresting to see that the lift values are not extremely high.

```
17 rules.sort_values('confidence', ascending=False).head(5)
```

17

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	le
18	(positive, loanNo)	(defaultNo)	0.765770	0.981952	0.760440	0.993039	1.011291	0.
68	(positive, no, loanNo)	(defaultNo)	0.665399	0.981952	0.660444	0.992554	1.010797	0.
105	(positive, loanNo, unknown)	(defaultNo)	0.627157	0.981952	0.622180	0.992065	1.010299	0.
162	(positive, no, loanNo, unknown)	(defaultNo)	0.562107	0.981952	0.557463	0.991737	1.009965	0.
209	(positive, married)	(defaultNo)	0.541206	0.981952	0.536694	0.991663	1.009890	0.

Looking at the top five strong rules reveals th default no has the strongest confidence if the antecedents are, loanNo, positive bank account and are married or have an unknown job.

Unfortunately these word associations are not leading us to find anything that is helping find rules that are indicative of a successful campaign. It is for that reason that I will continue data mining with the remain algorithms.

Feature Engineering for the following algorithms

- Decision tree Classification
- Naive Bayes Classification

First we must take all the data and numericalize it

```
18 # Creating Categories for ages then making categories
   df_b['job'] = df_b['job'].astype('category')
   df_b['job'] = df_b['job'].cat.codes
   df_b['marital'] = df_b['marital'].astype('category')
   df_b['marital'] = df_b['marital'].cat.codes
   df_b['education'] = df_b['education'].astype('category')
   df_b['education'] = df_b['education'].cat.codes
   df_b['default'] = df_b['default'].astype('category')
```

```
df_b['default'] = df_b['default'].cat.codes
df_b['contact'] = df_b['contact'].astype('category')
df_b['contact'] = df_b['contact'].cat.codes
df_b['month'] = df_b['month'].astype('category')
df_b['month'] = df_b['month'].cat.codes
df_b['poutcome'] = df_b['poutcome'].astype('category')
df_b['poutcome'] = df_b['poutcome'].cat.codes
df_b['housing'] = df_b['housing'].astype('category')
df_b['housing'] = df_b['housing'].cat.codes
df_b['loan'] = df_b['loan'].astype('category')
df_b['loan'] = df_b['loan'].cat.codes
df_b['y'] = df_b['y'].astype('category')
df_b['y'] = df_b['y'].cat.codes
df_b = df_b.drop('pdays',axis=1)
df_b['balance'] = (df_b['balance'] - df_b['balance'].min()) / ( df_b['balance'].max() - df_b['balance'].m
print("Post Conversion of Data into numeric, dropping pdays, and normalizing balance:")
df_b.head(10)
```

```
Post Conversion of Data into numeric, dropping pdays, and normalizing balance:
```

18

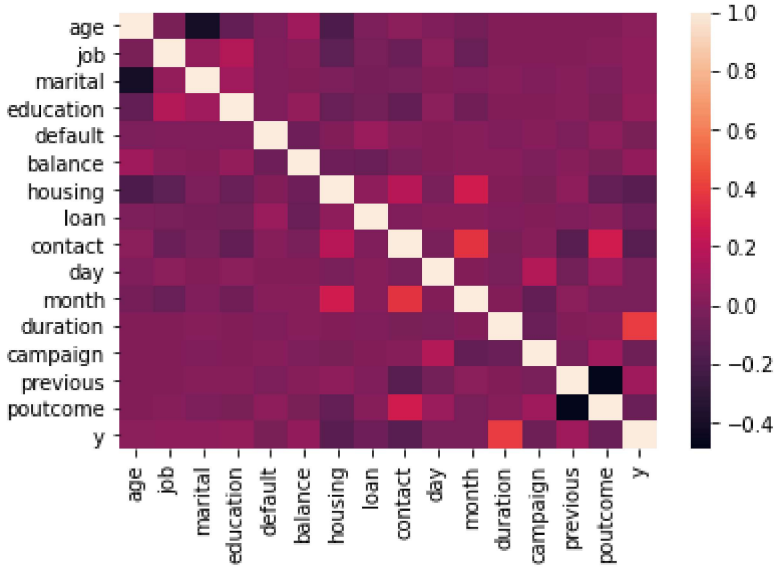
	age	job	marital	education	default	balance	housing	loan	contact	day	n
0	58	4	1	2	0	0.092259	1	0	2	5	8
1	44	9	2	1	0	0.073067	1	0	2	5	8
2	33	2	1	1	0	0.072822	1	1	2	5	8
3	47	1	1	3	0	0.086476	1	0	2	5	8
4	33	11	2	3	0	0.072812	0	0	2	5	8
5	35	4	1	2	0	0.074901	1	0	2	5	8
6	28	4	2	2	0	0.076862	1	1	2	5	8
7	42	2	0	2	1	0.072822	1	0	2	5	8
8	58	5	1	0	0	0.073902	1	0	2	5	8
9	43	9	2	1	0	0.078187	1	0	2	5	8

Exploratory Data Analysis

Now that we have numericalized the data it is time to do some preliminary exploration

19
```
sns.heatmap(df_b.corr())
```

19
```
<AxesSubplot:>
```



20
```
df_b.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   age        45211 non-null  int64  
 1   job        45211 non-null  int8   
 2   marital    45211 non-null  int8   
 3   education  45211 non-null  int8   
 4   default    45211 non-null  int8   
 5   balance    45211 non-null  float64
 6   housing    45211 non-null  int8   
 7   loan       45211 non-null  int8   
 8   contact    45211 non-null  int8   
 9   day        45211 non-null  int64  
 10  month      45211 non-null  int8   
 11  duration   45211 non-null  int64  
 12  campaign   45211 non-null  int64  
 13  previous   45211 non-null  int64  
 14  poutcome   45211 non-null  int8   
 15  y          45211 non-null  int8   
dtypes: float64(1), int64(5), int8(10)
memory usage: 2.5 MB
```

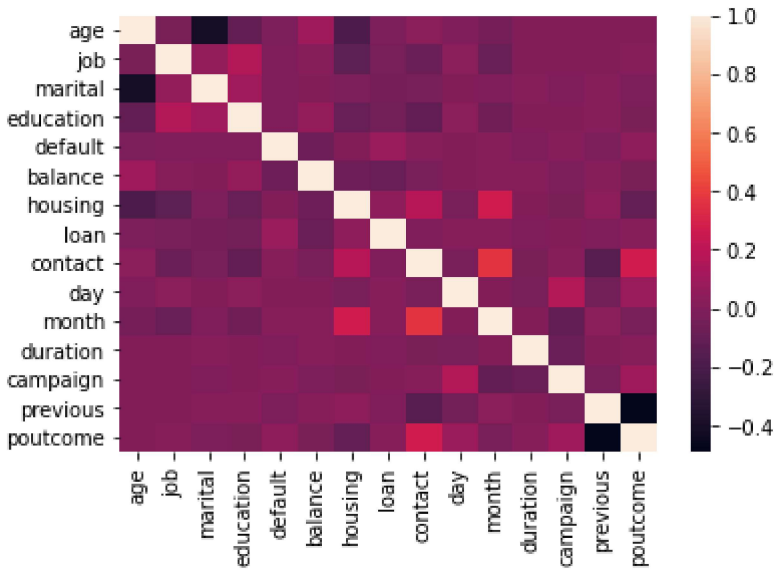
From the Correlation Map we can see that Y has a high correlation with marital status followed by education and housing.

Now to split the data into training sets and do some more exploration. But first we will split the data into feature{x} and outcome{y}. Followed by a quick heatmap, just go get a vizualization of the features correlated to themselves

21
```
y = df_b["y"]
X = df_b.drop("y", axis=1)

# Standardizing the feature data
X[X.columns] = StandardScaler().fit_transform(X)
sns.heatmap(X.corr())
```

21 <AxesSubplot:>



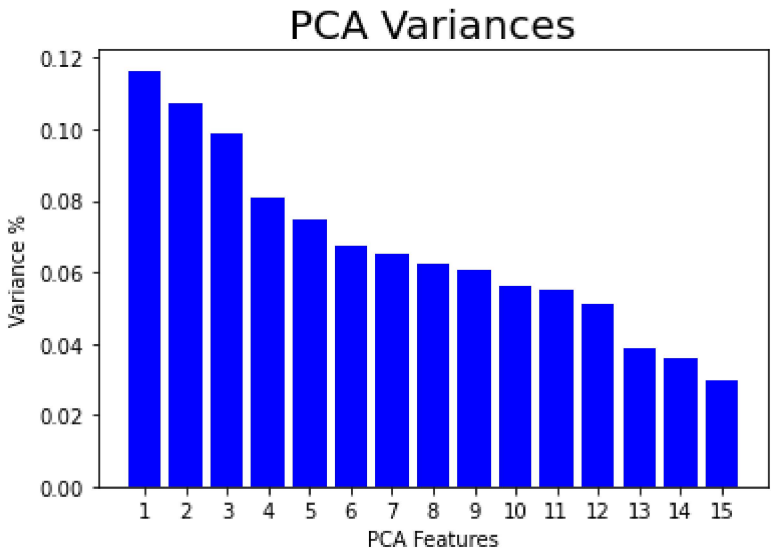
22
```
# Splitting the dataset into Train Set and Test Set
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=0)
```

PCA Analysis

The Principal Component Analysis Will help use look to see how many variables affect our algorithm to see if we can include less features to help speed up run times.

23
```
# Standardizing the Data
pca = PCA(n_components=15)
principalComponents = pca.fit_transform(X)

#Plotting the explained variances
features = range(1,pca.n_components_+1)
plt.bar(features, pca.explained_variance_ratio_, color='b')
plt.xlabel('PCA Features')
plt.ylabel('Variance %')
plt.xticks(features)
plt.title("PCA Variances", fontsize=20)
plt.show()
```



24
```
# First Three Feature variance Explanation
sum(pca.explained_variance_ratio_[:3])
```

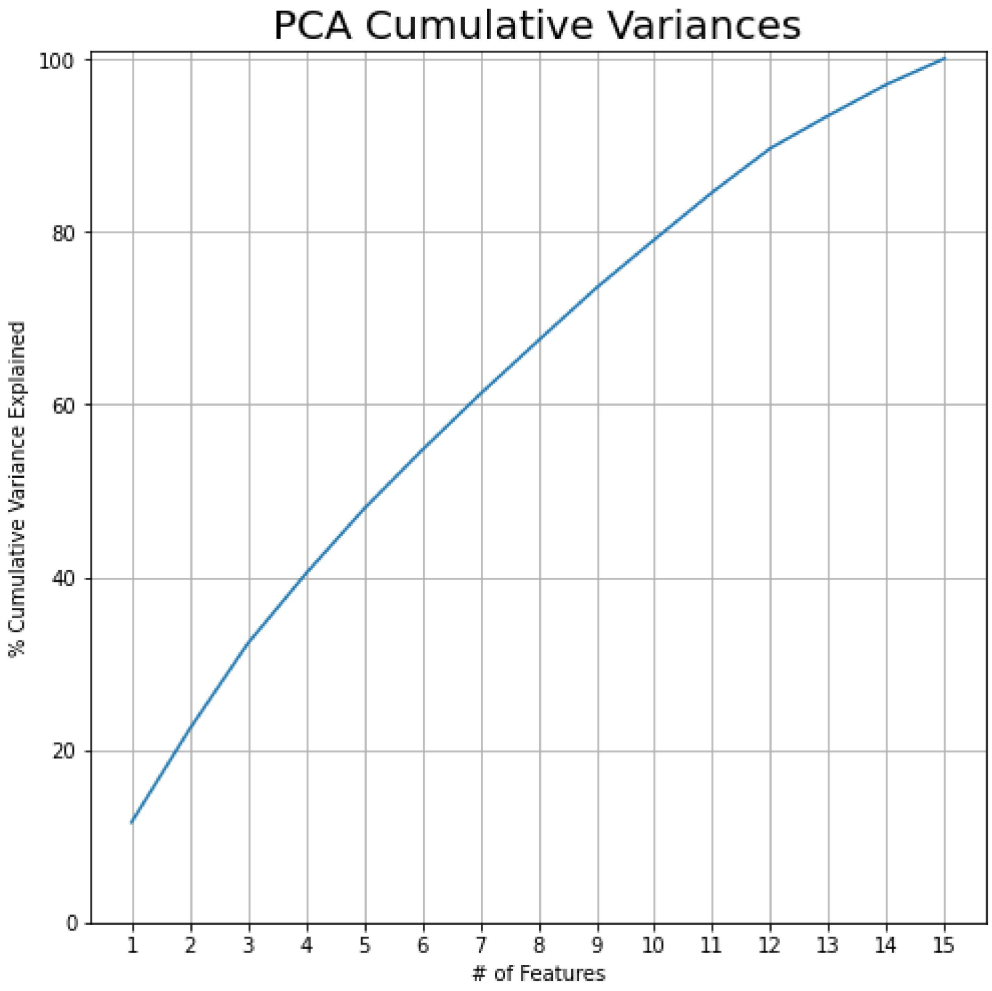
24
```
0.322735471272194
```

This first graph of the PCA's shows us a constant decline in variance. There is a bit of a dip around the first three give us an explanation of about 32%.

Next Graph will show us the cumulative PCA

25
```
var = np.cumsum(np.round(pca.explained_variance_ratio_, 3)*100)
plt.figure(figsize=(8,8))

plt.title('PCA Cumulative Analysis')
plt.ylim(0,101)
plt.xticks(features)
plt.ylabel('% Cumulative Variance Explained')
plt.xlabel('# of Features')
plt.title("PCA Cumulative Variances", fontsize=20)
plt.plot(features, var)
plt.grid()
plt.show()
```



Looking at the cumulative PCA variance graph offers us the same insight. there is no major elbow that makes limiting the amount of features drastically change. instead it gets closer and closer to explaining the variance the more features that we add.

The next few graphs will go through looking at graphing the different amount of accuracy for the both the GaussianNB model and decision trees.

26
```
def gather_model_accuracies(model_type, name):
    """This function takes in Different Classifiers to gather accuracies"""
    accuracy = []
    for i in range(1,16):
        pipe = Pipeline([('preprocessor', PCA(n_components=i, random_state=0)),
                         ('clf', model_type)])
        pipe.fit(X_train, y_train)
        y_prediction = pipe.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_prediction))
    accuracy = pd.DataFrame(accuracy, index=[i for i in range(1,16)], columns=[name])
    return accuracy
```

The Function Below Graphs the Accuracies Gathered from above

27
```
def graph_accuracies(model_type, model_name):
    """This Function Graphs The Accuracies"""
```

```
    x = model_type.index.tolist()
    y = model_type.iloc[:,0].tolist()
    fig, ax = plt.subplots(1, figsize=(5,5))
    sns.lineplot(x=x, y=y, linewidth=4, color='b', ax=ax)
    ax.set_xlabel("Number of Features")
    ax.set_ylabel("Model Accuracy")
    plt.xticks(np.arange(min(x),max(x)+1))
    plt.title(str("Accuracy for " + model_name), fontsize=20)
    plt.grid()
    plt.show()
```

Gathering the Decision Tree and GaussianNB Model Accuracy

28
```
dtree_accuracy = gather_model_accuracies(tree.DecisionTreeClassifier(random_state=0), "Decision Tree")
dtree_accuracy
```

28

	Decision Tree
1	0.801946
2	0.806480
3	0.818865
4	0.823952
5	0.838107
6	0.839323
7	0.848944
8	0.846069
9	0.845848
10	0.846732
11	0.849607
12	0.851598
13	0.848833
14	0.850050
15	0.854915

29
```
gnb_accuracy = gather_model_accuracies(GaussianNB(), "Gaussian")
gnb_accuracy
```

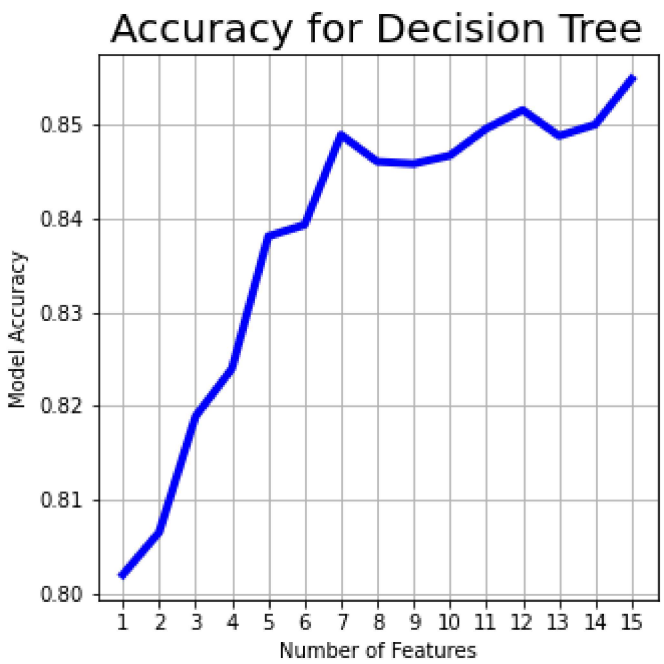
29

	Gaussian
1	0.882451
2	0.882451

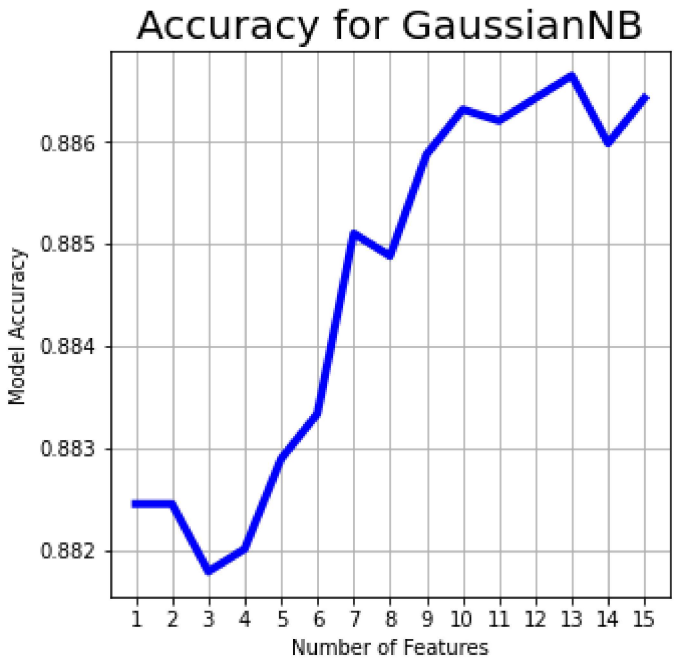
	Gaussian
3	0.881787
4	0.882008
5	0.882893
6	0.883335
7	0.885105
8	0.884883
9	0.885879
10	0.886321
11	0.886210
12	0.886431
13	0.886653
14	0.885989
15	0.886431

Graphing the best features for the GaussianND and the Decision Tree Model

```
30 graph_accuracies(dtree_accuracy, "Decision Tree")
```




```
31 graph_accuracies(gnb_accuracy, "GaussianNB")
```



Looking at the two graphs above. It looks like there is a slight peak for the Decision tree at 7 features then a valey, then another peak at 12 features with the total model accuracy being best at 15 features. The range for the accuracy is in between just above 80% with one features and 86% with 15 features.

For the GaussianNB Model the range of accuracies is in between 88.2% going all the way up to 88.65 % There isolated peaks at 2 features, 7 features 10 and the highest accuracy being at 13 features.

Now that we have looked at these graphs. It's time to actually build some models and discover if we are better at making some models.

The Next Few functions are set up so that I can run both the Decsion Tree and Gaussian Model with forward and backward selection. Adding and taking away features.

32
```
def train_data(df, drops=None):
    """This Function splits the data to train and test the data. It also an element for being able to
    drop certain rows."""
    # Y is the classification
    Y = df['y'].tolist()
    X = df.copy()
    X = X.drop(columns=["y"])
    X[X.columns] = StandardScaler().fit_transform(X)
    if drops != None:
        X = X.drop(columns=drops)
    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
    return x_train, x_test, y_train, y_test, X
```

33
```
def decisionTree_Model(x_train, x_test, y_train, y_test, X, yes_print=False):
    """This Function goes through building a decision tree and creating a confusion matrix
    computing accuracy, and listing the decision tree features and their importance"""
    # Building out a decision tree
```

```
    dtree = tree.DecisionTreeClassifier(criterion="gini")
    dtree = dtree.fit(x_train, y_train)
    y_predicted = dtree.predict(x_test)
    accuracy = accuracy_score(y_test, y_predicted)
    important = dtree.feature_importances_
    df_importance_list = []
    for i, v in enumerate(important):
        df_importance_list.append([X.columns[i], v])
    df_importance = pd.DataFrame(df_importance_list, columns=["FName", "Score"])
    df_importance.sort_values(by=['Score'], ascending=False, inplace=True)
    decisionTree_Model_Features = df_importance["FName"].tolist()
    if yes_print:
        print("DecisionTree Accuracy: ", accuracy)
        print("DecisionTree Confusion Matrix:")
        print(confusion_matrix(y_test, y_predicted))
        print("decision tree dtree feature importance:")
        print(df_importance)
    return accuracy, decisionTree_Model_Features


34  def GausianNB_Model(x_train, x_test, y_train, y_test, X, yes_print=False):
    """This Function goes through the GausianNB model and calculates the accuracy, computes the confusion
    and then lists the Gaussian Model Features and their significance"""
    model2 = GaussianNB()
    model2.fit(x_train, y_train)
    gausianNB_predicted = model2.predict(x_test)
    accuracy = accuracy_score(y_test, gausianNB_predicted)
    imps = permutation_importance(model2, x_test, y_test)
    df_Gausian_Feature_Importance_List = []
    Gausian_Feature_Importance_List =  imps.importances_mean.tolist()
    for row in enumerate(Gausian_Feature_Importance_List):
        df_Gausian_Feature_Importance_List.append([X.columns[row[0]], row[1]])
    df_Gausian_Feature_Importance = pd.DataFrame(df_Gausian_Feature_Importance_List, columns=["Feature",
    df_Gausian_Feature_Importance.sort_values(by=["Significance"], ascending=False, inplace=True)
    GausianND_Features = df_Gausian_Feature_Importance["Feature"].tolist()
    if yes_print:
        print('\nconfusion_matrix from Gaussian naive bayes:')
        print(confusion_matrix(y_test, gausianNB_predicted))
        print('accuracy = ' + str(accuracy))
        print("gausianNB feature importance:")
        print(df_Gausian_Feature_Importance)
    return accuracy, GausianND_Features


35  def backward_selection(accuracy, features, selection, df, model):
    """This is a backward selection model that removes one feature at a time to see if we can improve the
    accuracy of the model. Once any improvement is reached the function returns which feature that was dr
    that improves the model. The features that were included and the accuracy of the new model"""
    features_loop = [word for word in features if word not in selection]
    accuracy_start = accuracy
    for feature in features_loop:
        x_train, x_test, y_train, y_test, X = train_data(df, [feature] + selection)  # Dropping selection
        if model == 1:  # 1 For decisionTree
            new_accuracy, new_features = decisionTree_Model(x_train, x_test, y_train, y_test, X, False)
        else:
            new_accuracy, new_features = GausianNB_Model(x_train, x_test, y_train, y_test, X, False)
        if new_accuracy >= accuracy_start:
            print("Accuracy:", new_accuracy, "\nFeatures:", new_features,"\nDropped Feature:", feature)
            return
    return "No Improvement"
```

36
```
def forward_selection(accuracy, features, selection, df, model):
    """This is a backward selection model that removes one feature at a time to see if we can improve the
    accuracy of the model. Once any improvement is reached the function returns which feature that was dr
    that improves the model. The features that were included and the accuracy of the new model"""
    features_loop = [word for word in features if word not in selection]
    best_accuracy = accuracy
    for feature in features_loop:
        drop_features = [word for word in features if word not in [feature] + selection]
        x_train, x_test, y_train, y_test, X = train_data(df, drop_features)  # Dropping selection and fea
        if model == 1:  # 1 For decisionTree
            new_accuracy, new_features = decisionTree_Model(x_train, x_test, y_train, y_test, X, False)
        else:
            new_accuracy, new_features = GausianNB_Model(x_train, x_test, y_train, y_test, X, False)
        if new_accuracy >= best_accuracy:
            best_accuracy = new_accuracy
            add_feature = feature
    if best_accuracy > accuracy:
            print("Accuracy:", best_accuracy, "\nAdd Feature:", add_feature, "\nModel Features", [add_fea
    else:
        return "No Improvement"
```

Now To loop through. Starting Backwards with the Decision Tree Models

37
```
# Running the data back through the trainer
x_train, x_test, y_train, y_test, X = train_data(df_b)

# Decision Tree Model
dt_a, dt_f = decisionTree_Model(x_train, x_test, y_train, y_test, X, True)
```

```
DecisionTree Accuracy:  0.8625456153931218
DecisionTree Confusion Matrix:
[[7315  665]
 [ 578  485]]
decision tree dtree feature importance:
        FName     Score
11   duration  0.296549
5     balance  0.123322
10      month  0.102925
0         age  0.095055
9         day  0.094529
14   poutcome  0.060507
1         job  0.048802
13   previous  0.036116
12   campaign  0.033984
3   education  0.027874
6     housing  0.024090
8     contact  0.022019
2     marital  0.021672
7        loan  0.010346
4     default  0.002211
```

This first look gives us [duration, balance, month, age, day] the most important deciding factors. To improve upon this model we will first step backwards to see if we can improve accuracy

38
```
features = df_b.columns.tolist()
features.pop(-1)
accuracy = 0.8625456153931218
selection = []
```

```
backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.871171071547053
Features: ['duration', 'balance', 'month', 'day', 'poutcome', 'job', 'campaign', 'education', 'contact',
Dropped Feature: age
```

```
accuracy = 0.8625456153931218
selection = ['age']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8686276678093553
Features: ['duration', 'balance', 'month', 'day', 'poutcome', 'campaign', 'education', 'previous', 'marit
Dropped Feature: job
```

```
accuracy = 0.8678535884109255
selection = ['age','job']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8726086475727082
Features: ['duration', 'balance', 'day', 'month', 'poutcome', 'campaign', 'previous', 'marital', 'contact
Dropped Feature: education
```

```
accuracy = 0.8696229127501935
selection = ['age','job','marital']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.871171071547053
Features: ['duration', 'balance', 'day', 'month', 'poutcome', 'campaign', 'contact', 'housing', 'previous
Dropped Feature: education
```

```

accuracy = 0.871171071547053
selection = ['age','job','marital', 'education']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.872055733716687
Features: ['duration', 'balance', 'day', 'month', 'poutcome', 'campaign', 'contact', 'housing', 'previous
Dropped Feature: default
```

```
accuracy = 0.872055733716687
selection = ['age','job','marital', 'education', 'default']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8738250580559549
Features: ['duration', 'day', 'month', 'campaign', 'poutcome', 'previous', 'contact', 'housing', 'loan']
Dropped Feature: balance
```

```
accuracy = 0.8738250580559549
selection = ['age','job','marital', 'education', 'default', 'balance']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8764790445648568
```

```
Features: ['day', 'month', 'previous', 'campaign', 'poutcome', 'contact', 'housing', 'loan']
Dropped Feature: duration
```

```
accuracy = 0.8764790445648568
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration']
backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.879243613844963
Features: ['day', 'month', 'poutcome', 'campaign', 'previous', 'contact', 'loan']
Dropped Feature: housing
```

```
accuracy = 0.879243613844963
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8817870175826606
Features: ['day', 'month', 'poutcome', 'campaign', 'previous', 'contact']
Dropped Feature: loan
```

```
accuracy = 0.8817870175826606
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing', 'loan']

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8845515868627668
Features: ['day', 'month', 'poutcome', 'campaign', 'previous']
Dropped Feature: contact
```

```
accuracy = 0.8845515868627668
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing', 'loan', "cc

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.886873825058056
Features: ['poutcome', 'month', 'previous', 'campaign']
Dropped Feature: day
```

```
accuracy = 0.886873825058056
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing', 'loan',
             "contact", "day"]

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8905230565077961
Features: ['poutcome', 'previous', 'campaign']
Dropped Feature: month
```

```
accuracy = 0.8905230565077961
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing', 'loan',
             "contact", "day", "month"]

backward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8911865531350216
Features: ['poutcome', 'campaign']
Dropped Feature: previous
```

```
accuracy = 0.8905230565077961
selection = ['age','job','marital', 'education', 'default', 'balance', 'duration', 'housing', 'loan',
             "contact", "day", "month", "previous"]
```

```
backward_selection(accuracy, features, selection, df_b, 1)
```

```
'No Improvement'
```

Well that was anticlimatic. Of cources if they were subscribed before they are more likely to subscribe again. Lets run the forward model and see if we can find out anything that is more interesting...

```
selection= []
features.pop(-1)
accuracy = 0.8

forward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8911865531350216
Add Feature: campaign
Model Features ['campaign']
```

```
selection= ['poutcome']
accuracy = 0.8901913081941834

forward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8911865531350216
Add Feature: campaign
Model Features ['campaign', 'poutcome']
```

```
selection= ['poutcome', "campaign"]
accuracy = 0.8911865531350216

forward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8915183014486343
Add Feature: loan
Model Features ['loan', 'poutcome', 'campaign']
```

```
selection= ['poutcome', "campaign", "loan"]
accuracy = 0.8915183014486343

forward_selection(accuracy, features, selection, df_b, 1)
```

```
Accuracy: 0.8916288842198385
Add Feature: contact
Model Features ['contact', 'poutcome', 'campaign', 'loan']
```

```
selection= ['poutcome', "campaign", "loan", "contact"]
accuracy = 0.8916288842198385

forward_selection(accuracy, features, selection, df_b, 1)
```

```
'No Improvement'
```

Now to look at the GaussianNB model with forward model selection.

```
x_train, x_test, y_train, y_test, X = train_data(df_b)
gm_a, gm_f = GausianNB_Model(x_train, x_test, y_train, y_test, X, True)
```

```
confusion_matrix from Gaussian naive bayes:
[[7374  606]
 [ 610  453]]
accuracy = 0.8655313502156364
gausianNB feature importance:
      Feature  Significance
11   duration      0.046754
0         age      0.006259
6     housing      0.002389
2     marital      0.000752
1         job      0.000641
12   campaign      0.000553
3   education      0.000487
7        loan      0.000288
4     default     -0.000088
9         day     -0.000288
8     contact     -0.000487
13   previous     -0.000487
14   poutcome     -0.000730
10      month     -0.001039
5     balance     -0.001902
```

At first glance to speed things up a bit, we will remove the negative significance values

```
58 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance']
   accuracy = 0.8655313502156364
   backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8820081831250691
Features: ['duration', 'housing', 'education', 'job', 'marital', 'campaign']
Dropped Feature: age
```

```
59 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance',
                'age']
   accuracy = 0.8820081831250691
   backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8820081831250691
Features: ['duration', 'housing', 'education', 'marital', 'campaign']
Dropped Feature: job
```

```
60 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance',
                'age', 'job']
   accuracy = 0.8820081831250691
   backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8821187658962734
Features: ['duration', 'housing', 'education', 'campaign']
Dropped Feature: marital
```

```
61 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance',
                'age', 'job', 'marital']
   accuracy = 0.8821187658962734
   backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8824505142098861
Features: ['education', 'housing', 'campaign']
Dropped Feature: duration
```

```
62 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance',
                'age', 'job', 'marital','duration']
```

```
accuracy = 0.8824505142098861
backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8824505142098861
Features: ['housing', 'campaign']
Dropped Feature: education
```

```
63 selection = ['contact','default','loan','day', 'previous','month', 'poutcome', 'balance',
                'age', 'job', 'marital','duration', "education"]
   accuracy = 0.8824505142098861
   backward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8824505142098861
Features: ['campaign']
Dropped Feature: housing
```

Now to look at the forward selection with GaussianNB

```
64 accuracy = 0.8824505142098861
   selection = []
   forward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8828928452947031
Add Feature: duration
Model Features ['duration']
```

```
65 accuracy = 0.8828928452947031
   selection = ['duration']
   forward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.8847727524051753
Add Feature: housing
Model Features ['housing', 'duration']
```

```
66 accuracy = 0.8847727524051753
   selection = ['housing', 'duration']
   forward_selection(accuracy, features, selection, df_b, 0)
```

```
Accuracy: 0.885546831803605
Add Feature: age
Model Features ['age', 'housing', 'duration']
```

```
67 accuracy = 0.885546831803605
   selection = ['housing', 'duration', "age"]
   forward_selection(accuracy, features, selection, df_b, 0)
```

```
67 'No Improvement'
```

Now that we have run the decision tree and Gaussian models both forward and backwards it's time to lookinto DBscan and K-Means

K-Means

First we will run an inertias test with K-means to see if we can find the knee for the amount of clusters We are going to look at the PCA for 2 principle components because we are trying to discover if a success is signing up or not subscribing

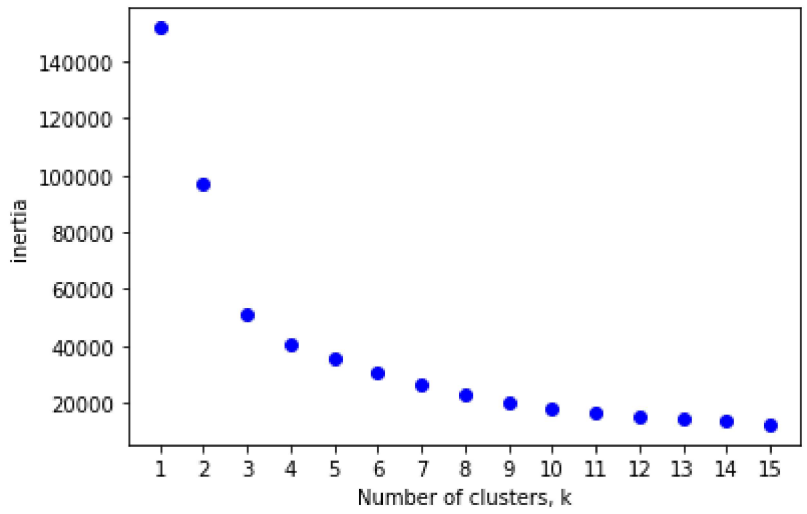
103
```
y = df_b["y"]
X = df_b.drop("y", axis=1)

# Standardizing the feature data
X[X.columns] = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X)
PCA_components = pd.DataFrame(principalComponents, columns=['pc1','pc2'])

inertias = []
for k in range(1,16):
    """Creating a K means Clustering"""
    model = KMeans(n_clusters=k)
    model.fit(PCA_components.iloc[:,:2])
    inertias.append(model.inertia_)
```

104
```
x = [i for i in range(1,16)]
plt.plot(x, inertias, 'o', color='b')
plt.xlabel("Number of clusters, k")
plt.ylabel("inertia")
plt.xticks(x)
plt.show()
```



The Knee Appears to be at three. Because of that we will set the clusters to three

124
```
k_means = KMeans(n_clusters=3, random_state=0).fit(principalComponents)
y_pred = k_means.fit_predict(principalComponents)
k_labels = k_means.labels_
y_predicted = sum(y != k_labels)
y_predicted2 = sum(y == k_labels)
k_means_Accuracy = max(y_predicted, y_predicted2)

# print("Samples Correctly Labeled: {}".format(y_predicted))
print("Accuracy: {}".format(k_means_Accuracy/float(y.size)))
print(f'Homogenity Score: {homogeneity_score(df_b["y"].tolist(),y_pred)}')
print(f'Completeness Score: {completeness_score(df_b["y"].tolist(), y_pred)}')
print(f'Adjusted Mutual Info Score: {adjusted_mutual_info_score(df_b["y"].tolist(), y_pred)}')
```

```
Accuracy: 0.5382760832540754
Homogenity Score: 0.033131795354886744
Completeness Score: 0.011710983905509098
Adjusted Mutual Info Score: 0.017273702933443352
```

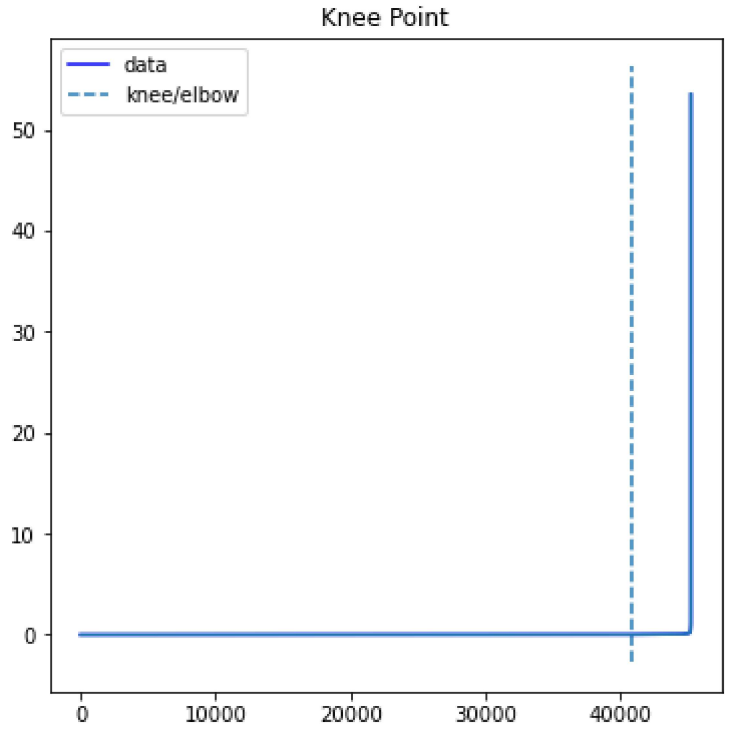
K-means is not really a prediction. But looking at the PCA for two components we were only able to classify 53 percent correctly. Additionally the Homogenity score, Completeness Score and Adjusted Mutual came out rather low.

DBScan

Now to start looking at the DBScan First we must Identify the best EPS. For this we will use

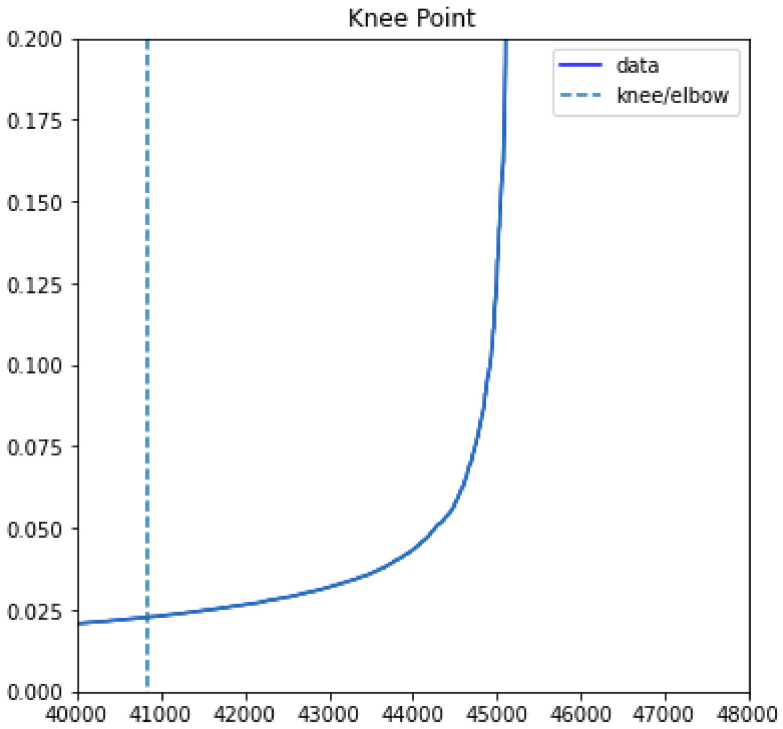
141
```
neigh = NearestNeighbors(n_neighbors=10)
nbrs = neigh.fit(principalComponents)
distances, indices = nbrs.kneighbors(principalComponents)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve="convex",
                   direction='increasing', interp_method="polynomial")
knee.plot_knee()
plt.plot(distances)
```

141
```
[<matplotlib.lines.Line2D at 0x1ba1dd03af0>]
```



142
```
knee.plot_knee()
plt.axis([40000, 48000, 0, .2])
plt.plot(distances)
```

142
```
[<matplotlib.lines.Line2D at 0x1ba1f29ebe0>]
```



145
```
db = DBSCAN(eps=.025, min_samples=6).fit(principalComponents)
pred_DB = db.fit_predict(principalComponents)
```

146
```
print('DB Scan')
print(f'Homogenity Score: {homogeneity_score(df_b["y"].to_list(),pred_DB)}')
print(f'Completeness Score: {completeness_score(df_b["y"].to_list(), pred_DB)}')
print(f'Adjusted Mutual Info Score: {adjusted_mutual_info_score(df_b["y"].to_list(), pred_DB)}')
```

```
DB Scan
Homogenity Score: 0.04437052185034992
Completeness Score: 0.0101899705266396
Adjusted Mutual Info Score: 0.011196499620374835
```

DBScan comes in a little bit better than the Kmeans in terms of looking at the Homogenity and completeness scores Just because my Knee locator looks off. I am going to run it again at .05

147
```
db = DBSCAN(eps=.05, min_samples=6).fit(principalComponents)
pred_DB = db.fit_predict(principalComponents)
```

148
```
print('DB Scan')
print(f'Homogenity Score: {homogeneity_score(df_b["y"].to_list(),pred_DB)}')
print(f'Completeness Score: {completeness_score(df_b["y"].to_list(), pred_DB)}')
print(f'Adjusted Mutual Info Score: {adjusted_mutual_info_score(df_b["y"].to_list(), pred_DB)}')
```

```
DB Scan
Homogenity Score: 0.028246856982262303
Completeness Score: 0.011799761714089613
Adjusted Mutual Info Score: 0.01384049379520564
```

The Knee locator was acctually a better predictor.