Prediction of client subscription of term deposit using SVM-Kernal

Table of Contents

1. Data Loading and Importing Libraries

First of all, the following libraries which are necessary to the machine learning process are imported before starting the preprocessing of the dataset.

```
[1] # import python libraries
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import scipy.stats as stats
    import seaborn as sns
    from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error, mean_absolute_error
    from sklearn.neural_network import MLPClassifier
    from sklearn.preprocessing import FunctionTransformer, OneHotEncoder, StandardScaler
    from sklearn.model_selection import train_test_split
    from sklearn.decomposition import PCA
    from tensorflow.keras import Sequential
    from tensorflow.keras.layers import Dense
    from keras.optimizers import SGD
```

The dataset is imported using the following code and it was verified by the following outputs.

```
# load the dataset and show first 10 records
data_set = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/ML/data/banking.csv')
data_set.head(10)
```

housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
yes	no	cellular	aug	thu	210	1	999	0	nonexistent	1.4	93.444	-36.1	4.963	5228.1	0
no	no	cellular	nov	fri	138	1	999	0	nonexistent	-0.1	93.200	-42.0	4.021	5195.8	0
yes	no	cellular	jun	thu	339	3	6	2	success	-1.7	94.055	-39.8	0.729	4991.6	1
no	no	cellular	apr	fri	185	2	999	0	nonexistent	-1.8	93.075	-47.1	1.405	5099.1	0
yes	no	cellular	aug	fri	137	1	3	1	success	-2.9	92.201	-31.4	0.869	5076.2	1
yes	no	cellular	jul	tue	68	8	999	0	nonexistent	1.4	93.918	-42.7	4.961	5228.1	0
yes	no	cellular	may	thu	204	1	999	0	nonexistent	-1.8	92.893	-46.2	1.327	5099.1	0
yes	no	cellular	may	fri	191	1	999	0	nonexistent	-1.8	92.893	-46.2	1.313	5099.1	0
no	no	cellular	jun	mon	174	1	3	1	success	-2.9	92.963	-40.8	1.266	5076.2	1
yes	no	cellular	apr	thu	191	2	999	1	failure	-1.8	93.075	-47.1	1.410	5099.1	0

Columns in the dataset;

```
[5] data_set.columns

    Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
           'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
           'previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
           'cons_conf_idx', 'euribor3m', 'nr_employed', 'y'],
          dtype='object')
```

The shape of the dataset:

```
[6] data_set.shape

(41188, 21)
```

As it can be seen from the output data set has 41188 rows and 21 columns.

Description of the dataset:

```
# describe summary
data_set.describe()
```

	age	duration	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035911	0.112654
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251528	0.316173
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000	0.000000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000	0.000000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000	0.000000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000	0.000000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000	1.000000

Count of each column, mean and standard deviation, minimum value, maximum value, 1st quantile, 2nd quantile, third quantile values of each column are shown here From describe() function it is possible to get a clear idea about how the data is distributed for each attribute.

2. Data Preprocessing

Real-world data usually contains noise, missing values, and is in an unsuitable format that cannot be used directly in machine learning models. Data preprocessing is a necessary task for cleaning data and making it suitable for a machine learning model, which improves the model's accuracy and efficiency.

i. Checking duplicate rows.

During the process of preprocessing, first thing is to check whether there are any duplicate rows in the dataset. The following code is used to check whether there are any duplicates in the dataset or not.

```
[8] data_set.duplicated().value_counts()

False    41176
True        12
dtype: int64
```

As per the output it can be seen that there are 12 duplicate rows. Therefore, it is required to remove the duplicate rows before digging the dataset further. Following code is used to drop them.

```
data_set=data_set.drop_duplicates()
data_set=data_set.reset_index(drop=True)
```

ii. Handle Missing Values

To check whether the data set contain any missing values, the following code is used.

```
data_set.isnull().any()

age               False
job               False
marital           False
education         False
default           False
housing           False
loan              False
contact           False
month             False
day_of_week       False
duration          False
campaign          False
pdays             False
previous          False
poutcome          False
emp_var_rate      False
cons_price_idx    False
cons_conf_idx     False
euribor3m         False
nr_employed       False
y                 False
dtype: bool
```

As it can be seen as the output there are no missing values in the dataset.

iii. Dropping ‘duration’ column

When considering the duration attribute, it has a significant impact on the output target (for example, if duration=0, y='no'). However, the duration is unknown before a call is made. Also, y is known at the end of the call. Since the goal is to create a realistic predictive model, the duration column has been removed.

```
[97] data_set=data_set.drop(columns='duration',axis=1)
     data_set=data_set.reset_index(drop=True)
```

iv. Dividing Columns based on Categorical & Numerical data types

Hence dataset has more than 20 columns I have divided it into categorical column and numerical column so that data preprocessing will be easier to handle.

All the numerical columns are extracted from the dataset and added it to num_df data frame using following code:

```
# extract numerical columns from the dataset
num_df = features.select_dtypes(include=np.number)
# get the information about numerical columns
num_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41176 entries, 0 to 41175
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41176 non-null  int64
 1   campaign        41176 non-null  int64
 2   pdays           41176 non-null  int64
 3   previous        41176 non-null  int64
 4   emp_var_rate    41176 non-null  float64
 5   cons_price_idx  41176 non-null  float64
 6   cons_conf_idx   41176 non-null  float64
 7   euribor3m       41176 non-null  float64
 8   nr_employed     41176 non-null  float64
 9   y               41176 non-null  int64
dtypes: float64(5), int64(5)
memory usage: 3.1 MB
```

As per the output numerical columns - age campaign, pdays, previous, emp_var_rate, cons_price_idx, cons_conf_idx, euriborm3 ,nr_employed, y columns are added to num_df data frame

Description of the num_df data frame is as follows:

```
[100] num_df.describe()
```

	age	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
count	41176.00000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000	41176.000000
mean	40.02380	2.567879	962.464810	0.173013	0.081922	93.575720	-40.502863	3.621293	5167.034870	0.112663
std	10.42068	2.770318	186.937102	0.494964	1.570883	0.578839	4.627860	1.734437	72.251364	0.316184
min	17.00000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000	0.000000
25%	32.00000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000	0.000000
50%	38.00000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000	0.000000
75%	47.00000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000	0.000000
max	98.00000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000	1.000000

All the categorical columns are extracted from the dataset and added it to categorical _df data frame using following code:

```
[36] #getting categorical columns separately
     categorical_df = data_set.select_dtypes(exclude=np.number)
     categorical_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41176 entries, 0 to 41175
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   job          41176 non-null  object
 1   marital      41176 non-null  object
 2   education    41176 non-null  object
 3   default      41176 non-null  object
 4   housing      41176 non-null  object
 5   loan         41176 non-null  object
 6   contact      41176 non-null  object
 7   month        41176 non-null  object
 8   day_of_week  41176 non-null  object
 9   poutcome     41176 non-null  object
dtypes: object(10)
memory usage: 3.1+ MB
```

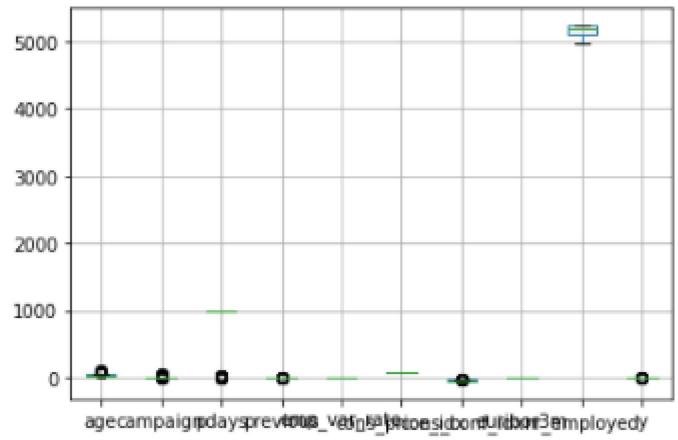
As per the output categorical columns – job, marital, education , default, housing, loan, contact, month, day_of_week, poutcome columns are added to categorical _df data frame

v. Handling outliers

the next thing to do is to check whether there are any outliers in the numerical dataset. Boxplot can be used to visualize the dataset and check whether there are any outliers.

```
[101] num_df.boxplot()
```

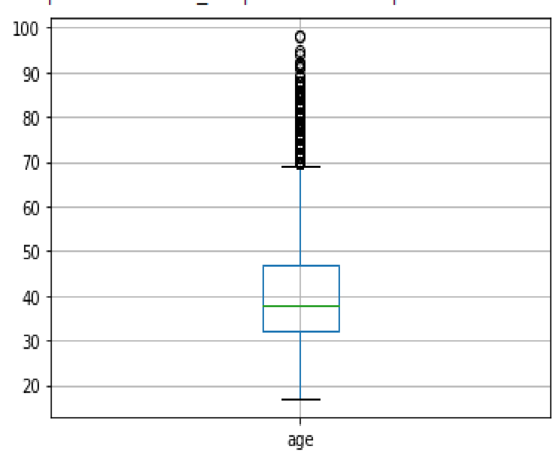
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a1ec391d0>



Since the graph is not clear I have constructed a boxplot for each column separately.

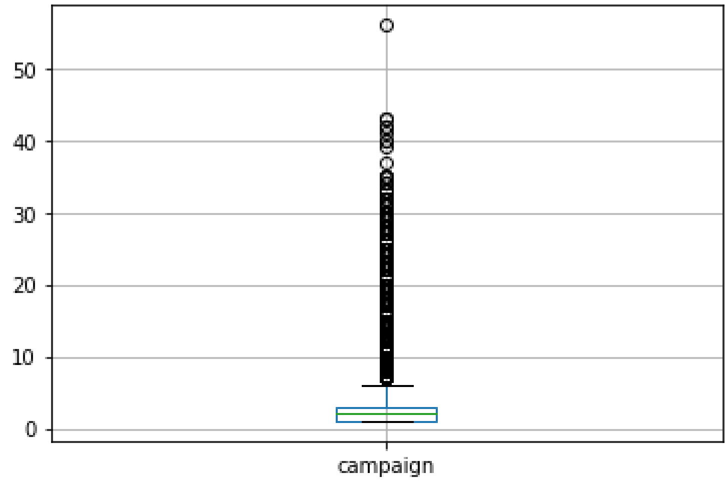
```
[14] num_df.boxplot(column='age')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc862702ad0>



```
[19] num_df.boxplot(column='campaign')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc86217a2d0>



As per the above boxplot campaign column has outliers. Outlier data points lies in the range above 50. I have removed outliers from the dataset using the following code.

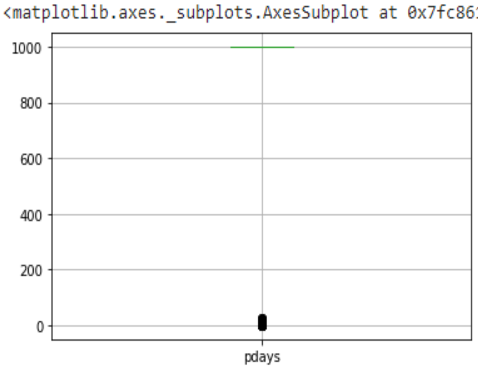
```
[105] df1=num_df.drop(num_df[num_df['campaign'] > 50].index)
      df1.shape
```

(41175, 10)

```
[106] num_df=df1;
      num_df=num_df.reset_index(drop=True)
```

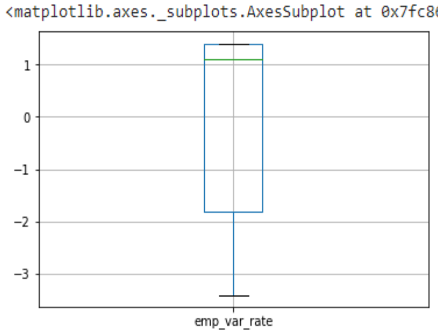
```
[22] num_df.boxplot(column='pdays')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc8616be610>



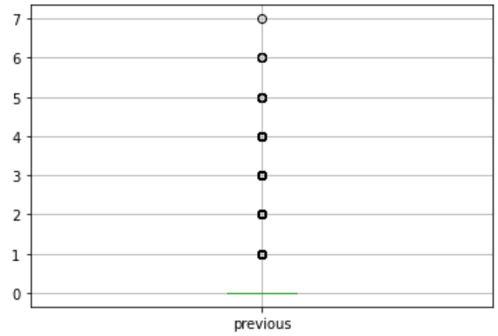

```
num_df.boxplot(column='emp_var_rate')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc86162ae90>



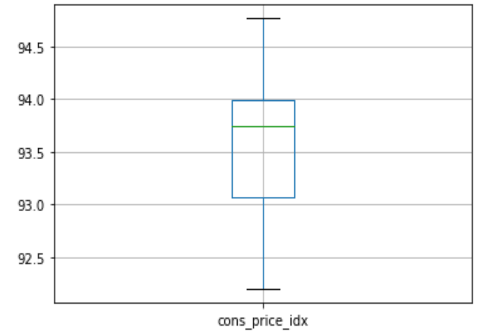
```
[24] num_df.boxplot(column='previous')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc86160d610>
```



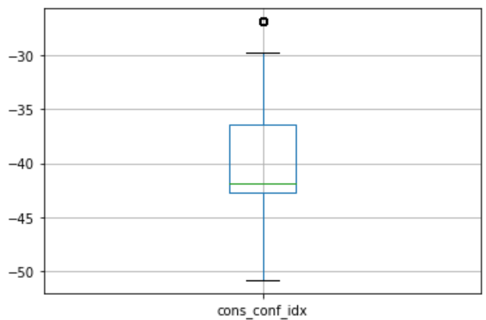

```
[25] num_df.boxplot(column='cons_price_idx')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc8627022d0>
```



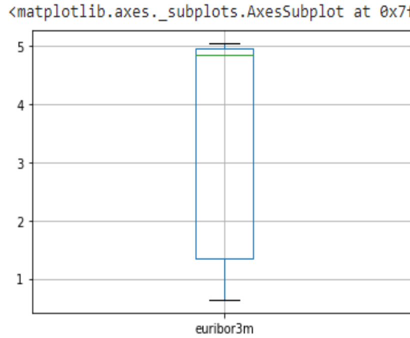

```
[26] num_df.boxplot(column='cons_conf_idx')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc8614d0cd0>
```



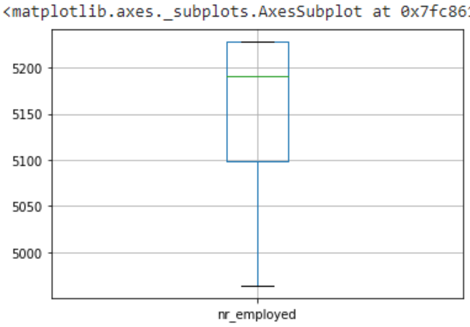

```
[27] num_df.boxplot(column='euribor3m')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc86150a2d0>
```



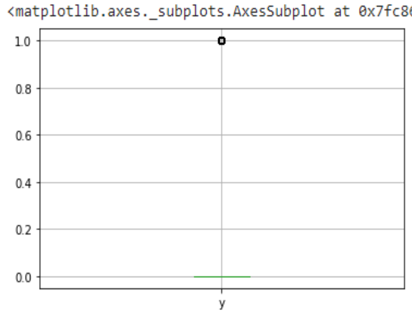

```
[28] num_df.boxplot(column='nr_employed')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc861427150>
```




```
num_df.boxplot(column='y')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc861391390>
```



3. Data transformation

Before applying any data transformation, Since num_df contains y column I have defined a separate data frame as target to add the y value and drop it from the num_df.

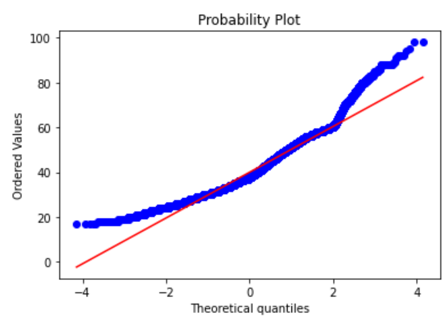
```
target=pd.DataFrame(num_df,columns=['y'])
```

```
[31] num_df = num_df.drop('y', axis=1)
     num_df=num_df.reset_index(drop=True)
```

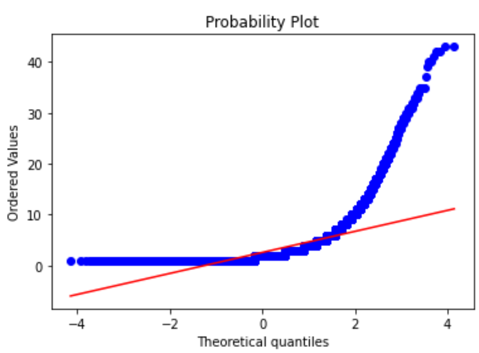
i. Q-Q Plots and Histograms of the features

The following code is used to construct the QQ plots of each feature, and the Histogram as follows.

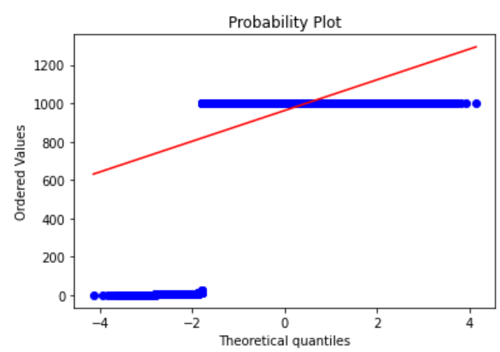
```
[ ] stats.probplot(num_df["age"], dist="norm", plot=plt)
    plt.show()
```



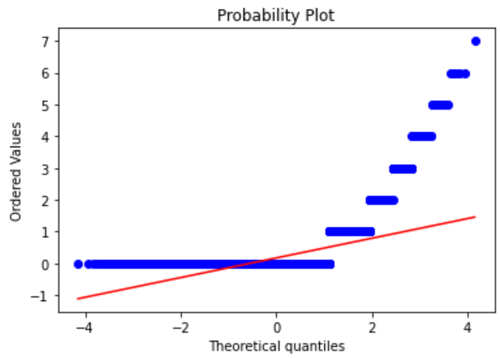

```
stats.probplot(num_df["campaign"], dist="norm", plot=plt)
plt.show()
```



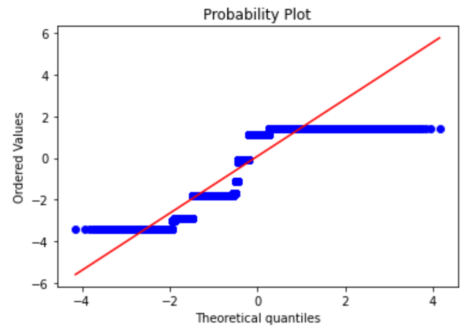

```
stats.probplot(num_df["pdays"], dist="norm", plot=plt)
plt.show()
```



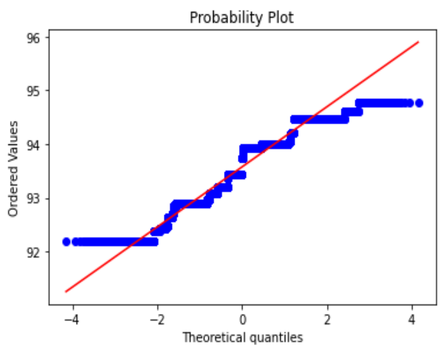
```
stats.probplot(num_df["previous"], dist="norm", plot=plt)
plt.show()
```



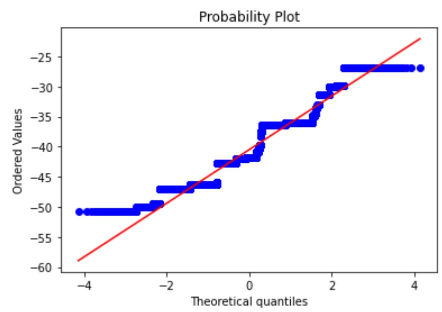

```
stats.probplot(num_df["emp_var_rate"], dist="norm", plot=plt)
plt.show()
```



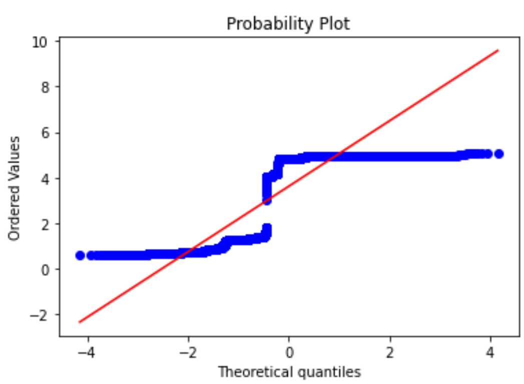

```
stats.probplot(num_df["cons_price_idx"], dist="norm", plot=plt)
plt.show()
```



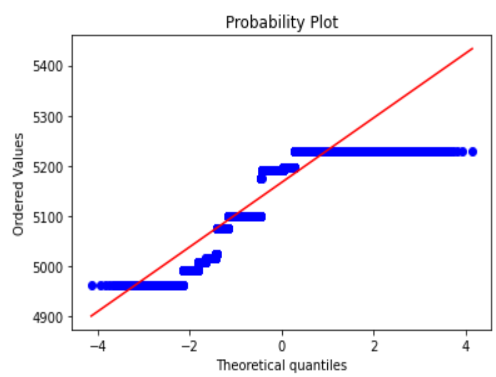

```
stats.probplot(num_df["cons_conf_idx"], dist="norm", plot=plt)
plt.show()
```



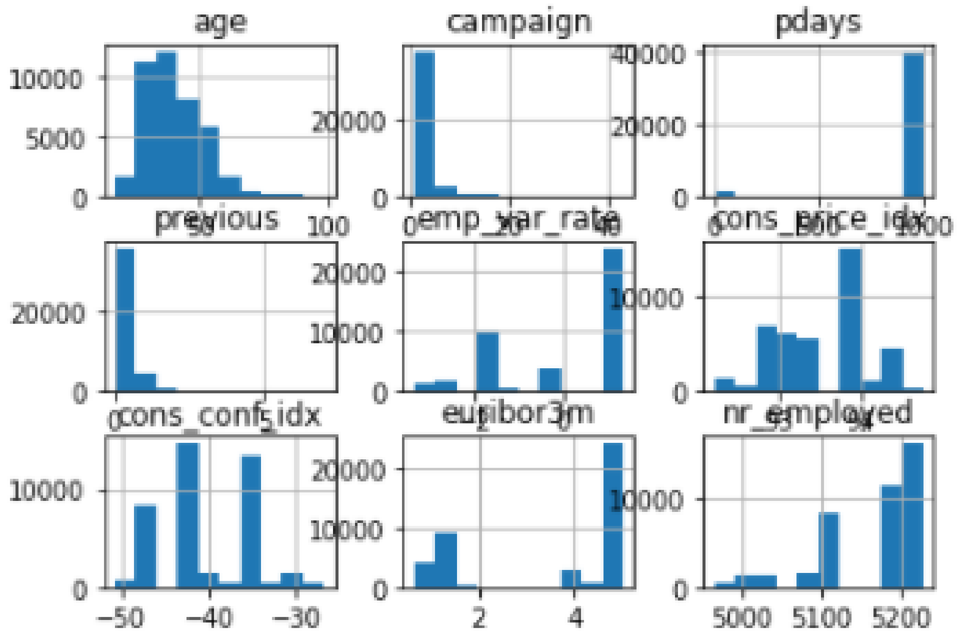

```
stats.probplot(num_df["euribor3m"], dist="norm", plot=plt)
plt.show()
```




```
stats.probplot(num_df["nr_employed"], dist="norm", plot=plt)
plt.show()
```



If the variable has a normal distribution, the values should fall in a 45-degree line when plotted against the theoretical quantiles in the Q-Q plots. As it can be seen from the diagrams except for age, campaign, duration, previous columns, other columns data fall in 45-degree line which shows normal distribution. To observe further histogram is constructured as follows.



Age, Campaign, Previous and nr.employed columns data are skewed. Therefore, it is a must to transform them into a normal distribution. As per the histograms of each feature, Age, Campaign, and Previous columns are left-skewed while the nr.employed feature is right-skewed. Therefore, Square root transformation is applied to the Age, Campaign, and Previous features and Exponential or power transformation is applied to the nr.employed column. I have applied transformation to both testing and training sets. Code is as below.

Square root transformation:

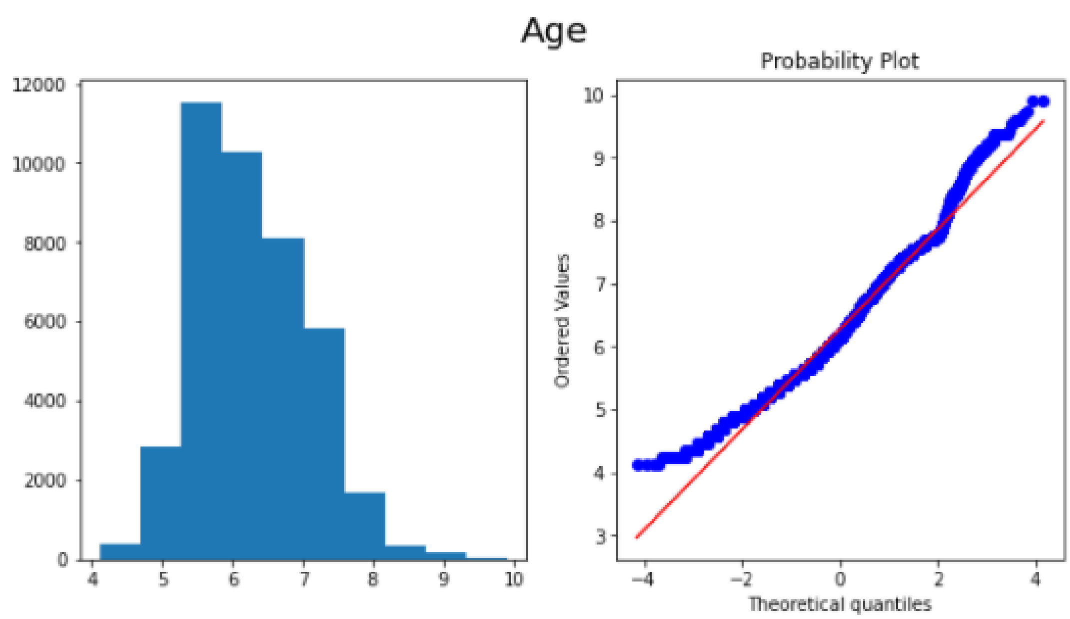
```
# transformations for right skewed features
sqrt_transformer = FunctionTransformer(np.sqrt, validate=True)

columns = ['age', 'campaign', 'previous']

data = sqrt_transformer.transform(num_df[columns])

num_df[columns]=data
```

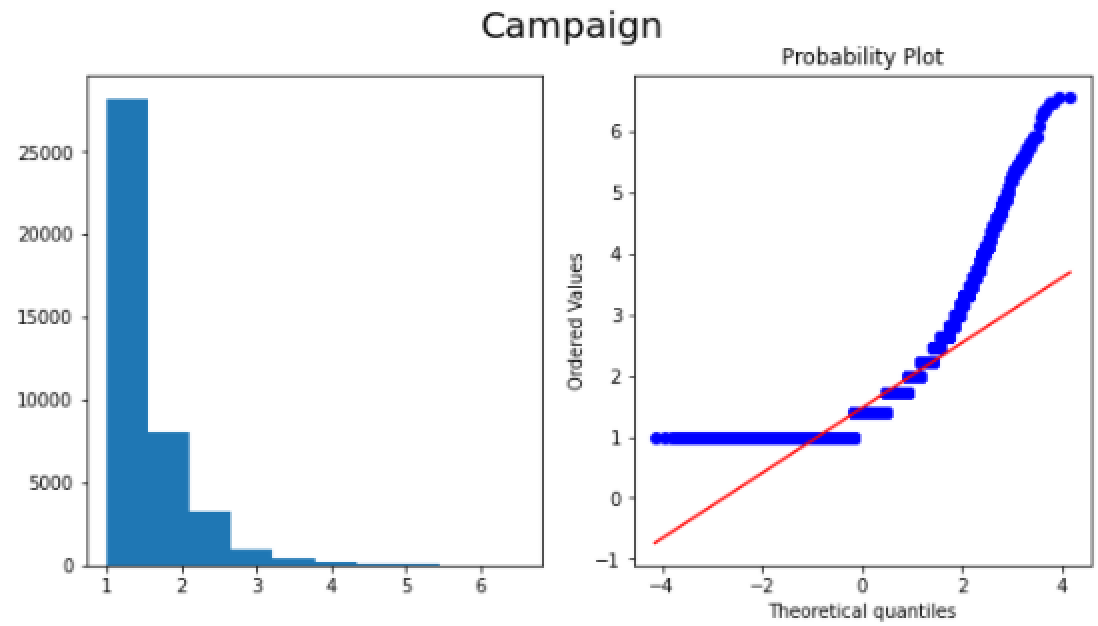
Exponential or power transformation:

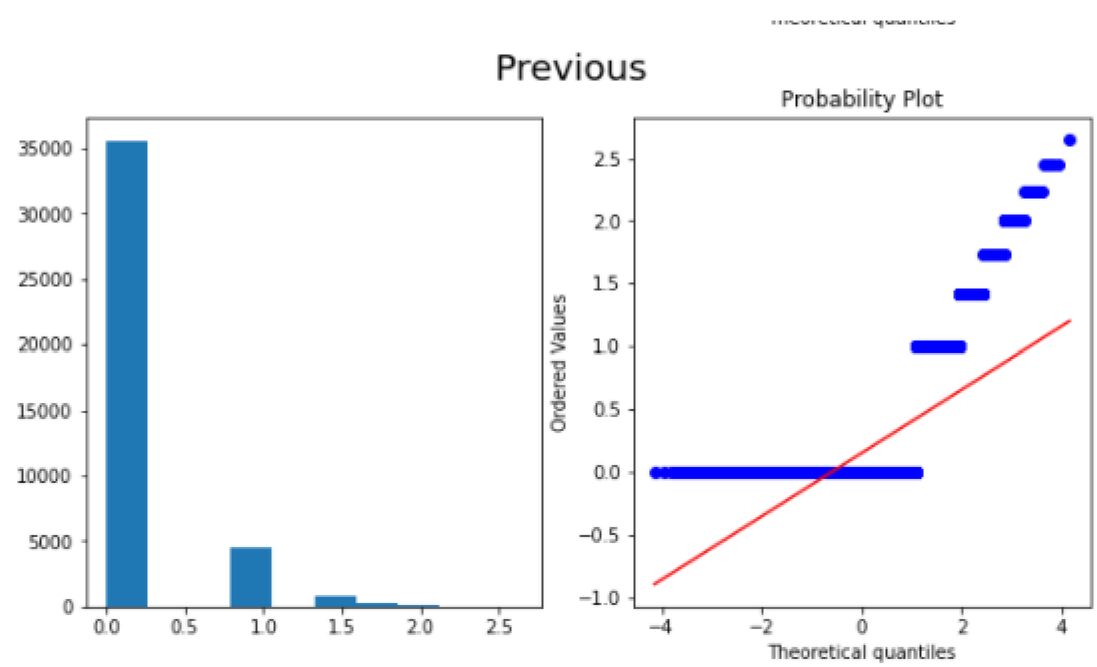
```
[232] # do the transformations for left skewed features
      squared_transformer = FunctionTransformer(lambda x: x**2, validate=True)

      columns = ['nr_employed']

      data = squared_transformer.transform(num_df[columns])

      num_df[columns]=data
```

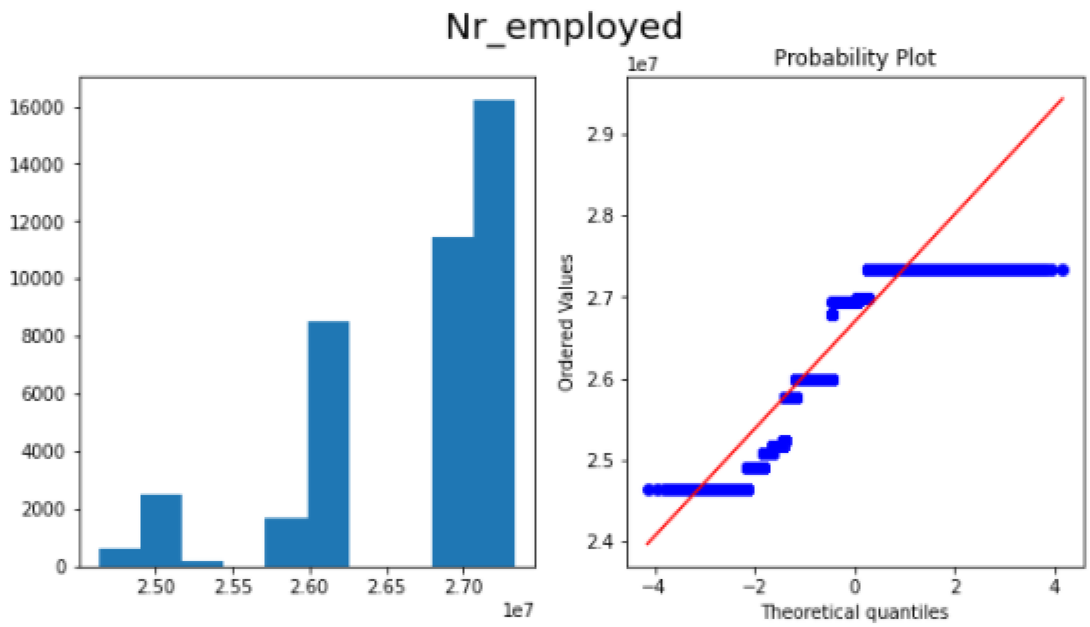
After transformation histogram and QQ plots of the age , campaign, previous, nr_employed columns are as follows.

```
columns = ['age', 'campaign', 'previous', 'nr_employed']

# code to get the histograms and Q-Q plots after applying transformations
for index, col in enumerate(columns):
    fig, axes = plt.subplots(1,2, figsize=(10,5))
    fig.suptitle(col.capitalize(), fontsize=20, color='black')
    axes[0].hist(num_df[col])
    stats.probplot(num_df[col], dist="norm", plot=axes[1])
    plt.show()
```

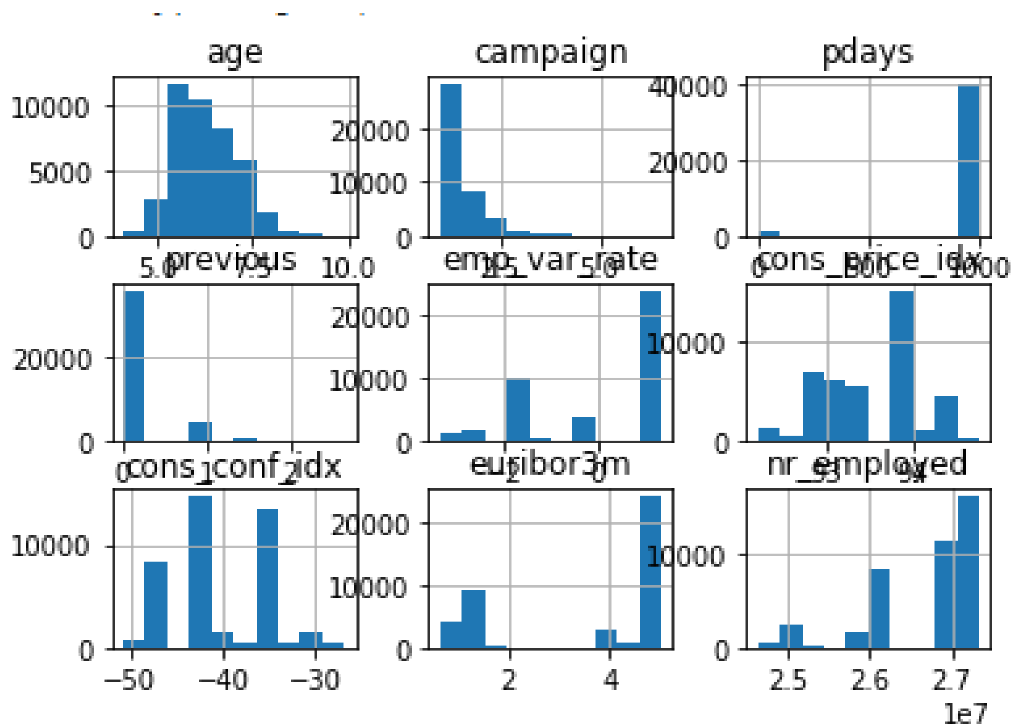



Campaign
Probability Plot
Ordered Values
Theoretical quantiles


Previous
Probability Plot
Ordered Values
Theoretical quantiles



Now all the features including age, nr_employed, campaign, previous features data have a normal distribution.



ii. Applying feature encoding technique for categorical data

Following code shows the columns of the categorical_df and data type of each column.

```
[126] categorical_df.dtypes

job            object
marital        object
education      object
default        object
housing        object
loan           object
contact        object
month          object
day_of_week    object
poutcome       object
dtype: object
```

These categorical data is in an object format. In order to convert them to numerical values feature encoding is used.

There are 2 types of feature encoding techniques.

a. Label Encoding
b. One hot encoding

The Label Encoder encodes classes with values ranging from 0 to n-1, where n is the number of distinct classes.

One Hot Encoding takes a column with categorical data and divides it into multiple columns. Depending on which column has a particular class, the classes are replaced by binaries (1s and 0s).

Since there are 10 categorical columns, if one hot encoding is used it will lead to have a complex data frame due to multiple columns. Therefore, to reduce the complexity after encoding label encoding is used.

To apply label encoding following code is used.

```python
categorical_df['job']=categorical_df['job'].astype('category').cat.codes
categorical_df['marital']=categorical_df['marital'].astype('category').cat.codes
categorical_df['education']=categorical_df['education'].astype('category').cat.codes
categorical_df['default']=categorical_df['default'].astype('category').cat.codes
categorical_df['housing']=categorical_df['housing'].astype('category').cat.codes
categorical_df['loan']=categorical_df['loan'].astype('category').cat.codes
categorical_df['contact']=categorical_df['contact'].astype('category').cat.codes
categorical_df['month']=categorical_df['month'].astype('category').cat.codes
categorical_df['day_of_week']=categorical_df['day_of_week'].astype('category').cat.codes
categorical_df['poutcome']=categorical_df['poutcome'].astype('category').cat.codes
```

I have created a new data frame to combine num_df and categorical_df.

```python
# final datadframe to join numerical data columns with categorical data columns
final_df = num_df.join(categorical_df)
final_df.info()
```

iii. Scale and/or standardized the features.

As the scaling method standardization is used. Data is centered using standardization. Since only continuous data can be standardized, StandardScaler() must be only used for continuous data columns. Following code is used for the standardization. As it can be seen from the below picture I have extracted continuous data columns to a new variable called column features and only used these columns to fit into the scaler for standardization.

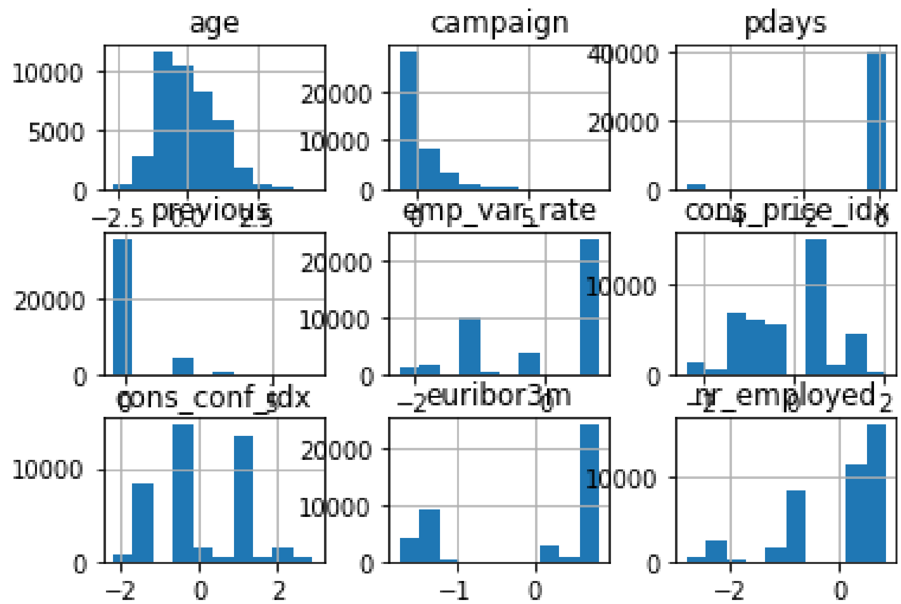
```python
[589] columnfeatures=['age','campaign','pdays','previous','emp_var_rate','cons_price_idx','cons_conf_idx','euribor3m','nr_employed' ]
      scaler = StandardScaler()
      scaler.fit(final_df[columnfeatures])

      final_df[columnfeatures]= scaler.transform(final_df[columnfeatures])
```

After standardization dataset is as follows.

```python
[592] final_df.head()
```

	age	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	job	marital	education	default
0	0.445094	-0.777914	0.195446	-0.388096	0.839089	-0.227543	0.951413	0.773591	0.849370	1	1	0	1
1	1.248893	-0.777914	0.195446	-0.388096	-0.115793	-0.649079	-0.323487	0.230472	0.395127	9	1	7	0
2	-1.222184	0.410968	-5.116504	3.258902	-1.134334	0.828025	0.151900	-1.667562	-2.411437	4	2	6	0
3	-0.037355	-0.105214	0.195446	-0.388096	-1.197993	-0.865030	-1.425519	-1.277808	-0.947961	7	1	3	0
4	1.417999	-0.777914	-5.132552	2.190721	-1.898240	-2.374958	1.967011	-1.586844	-1.262329	5	1	0	0

Histogram after standardization.



4. Perform Feature Engineering

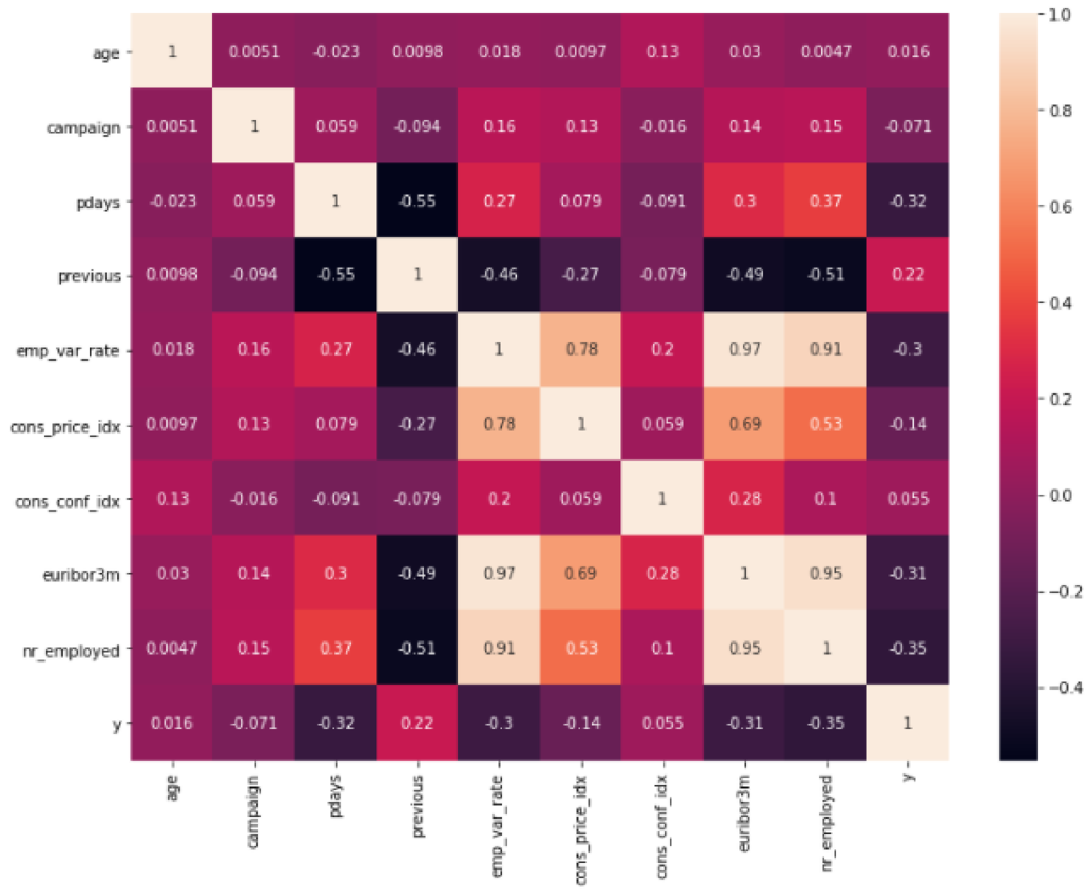
i. Identifying significant and independent features

A feature is an individual attribute or characteristic of a process under study in machine learning. The quality of features in a dataset has a significant impact on the quality of machine learning algorithms' output. Choosing relevant features is a critical step in efficiently training and calibrating algorithms.

In order to capture the significance of the features, a correlation matrix is constructed using the following code and the following output was obtained.

```
[593] import seaborn as sns
      # correlation matrix
      import matplotlib.pyplot as plt

      plt.figure(figsize=(12, 9))
      correlation_matrix = pd.concat([final_df.iloc[:,:9], y_true], axis=1).corr()
      sns.heatmap(correlation_matrix,annot=True)
      correlation_matrix
```

	age	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y
age	1.000000	0.005127	-0.022693	0.009771	0.018136	0.009690	0.125121	0.030102	0.004690	0.016245
campaign	0.005127	1.000000	0.058675	-0.093991	0.160024	0.130872	-0.015750	0.141810	0.153066	-0.071138
pdays	-0.022693	0.058675	1.000000	-0.552779	0.271061	0.078917	-0.091379	0.296945	0.371042	-0.324947
previous	0.009771	-0.093991	-0.552779	1.000000	-0.464156	-0.267540	-0.078784	-0.488842	-0.512821	0.217486
emp_var_rate	0.018136	0.160024	0.271061	-0.464156	1.000000	0.775291	0.196246	0.972244	0.908811	-0.298285
cons_price_idx	0.009690	0.130872	0.078917	-0.267540	0.775291	1.000000	0.059156	0.688176	0.525197	-0.136129
cons_conf_idx	0.125121	-0.015750	-0.091379	-0.078784	0.196246	0.059156	1.000000	0.277853	0.102507	0.054810
euribor3m	0.030102	0.141810	0.296945	-0.488842	0.972244	0.688176	0.277853	1.000000	0.946382	-0.307737
nr_employed	0.004690	0.153066	0.371042	-0.512821	0.908811	0.525197	0.102507	0.946382	1.000000	-0.353604
y	0.016245	-0.071138	-0.324947	0.217486	-0.298285	-0.136129	0.054810	-0.307737	-0.353604	1.000000



As it can be seen from the output,

- Previous and y have a positive correlation of 0.22.
- Pdays and y are negatively correlated of 0.32
- Emp_var_rate and y are negatively correlated of -0.3
- Duration and y have strong positive correlation of 0.41.

In order to get better idea to identify what features can be drop PCA is performed.

ii. PCA (Principal Component Analysis) for feature reduction

Next, it is required to identify what are the important variables in the dataset and extract them. To identify it, PCA is used. Therefore, explained variance ratio is calculated in order to identify the number of components to apply to PCA so that to capture maximum of 95% variance to capture while dropping 5% of variance. The proportion of each principal component axis' contribution to the variance of the entire dataset is represented by the explained variance ratio.

```
[595] pca = PCA()

      final_Scaled_pca = pca.fit_transform(final_df)
```

```
[596] pca.explained_variance_ratio_

      array([3.60669973e-01, 1.53764511e-01, 1.21669181e-01, 1.06012469e-01,
             5.39087404e-02, 3.59319010e-02, 2.95925809e-02, 2.68749697e-02,
             2.68010681e-02, 2.35982824e-02, 1.43962188e-02, 1.28714353e-02,
             1.14189365e-02, 9.56540712e-03, 5.37729685e-03, 4.06675000e-03,
             2.54687399e-03, 6.61451887e-04, 2.71953598e-04])
```

To select no of components for PCA following code is used,

```
[597] pca = PCA(.95)
      pca.fit(final_df)
      pca.n_components_

      11
```

As it can be seen in the output. 1st variable shows 0.3606 variance to the dataset. I have selected 11 components to capture 95% of the variance.

I have assigned the feature set to a new data frame principleDf and the snapshot of dataset is as follows.

```
[598] pca = PCA()
      final_Scaled_pca = PCA(n_components=11).fit_transform(final_Scaled_pca)
      principalDf=pd.DataFrame(data =final_Scaled_pca)
      principalDf
```

	0	1	2	3	4	5	6	7	8	9	10
0	-3.022681	-2.344430	-4.299381	0.775681	-0.067002	0.516810	-1.074918	-0.546709	-1.068702	0.342478	-0.400924
1	5.512558	1.934199	2.948065	-1.583600	2.032565	-0.174208	-1.248843	-0.380308	1.066050	-1.124997	-0.264299
2	0.625998	-0.190853	3.648622	3.258147	-0.191875	4.042346	3.453159	0.245416	-1.357098	0.249373	-0.409749
3	3.333365	-3.264368	-1.275992	3.204110	1.919122	-1.374249	0.124298	-0.180542	1.219428	-0.564511	-0.250936
4	1.024487	-1.173128	-2.340678	6.481074	1.756217	4.595884	-0.661318	-0.075005	-1.103121	0.916779	-0.422521
...	...	...	...	...	...	...	...	...	...	...	...
41170	-3.870888	1.338071	-0.575030	-1.842146	0.077366	0.601394	-0.967784	0.280289	-0.976957	-1.785352	-0.399239
41171	1.167535	-0.041196	-1.356550	-0.982353	0.029565	0.332613	0.390193	-0.960583	0.694309	0.780719	1.745849
41172	-1.265838	2.569651	-3.415688	-0.616796	0.142931	0.661707	-0.162393	-0.115295	1.058185	0.520681	-0.240382
41173	-3.456744	1.238292	3.962325	1.513719	-2.047889	0.345900	-2.974768	0.441656	-0.428456	2.495094	1.654249
41174	5.212600	3.409827	0.963566	-2.613814	-0.865604	-0.036974	1.083997	-0.327730	0.832922	1.396734	1.762218

41175 rows × 11 columns

5. Splitting the dataset

The training set is where we train and fit our model to fit the parameters, whilst test data is only used to evaluate the model's performance. The output of training data is accessible to model, but testing data is unobserved data for which predictions must be generated.

I have split my training and testing data to the 80% and 20% ratio. principleDf includes features to predict the y column and the y data set includes y column, The indexes are reset as follows in order to avoid further confusion.

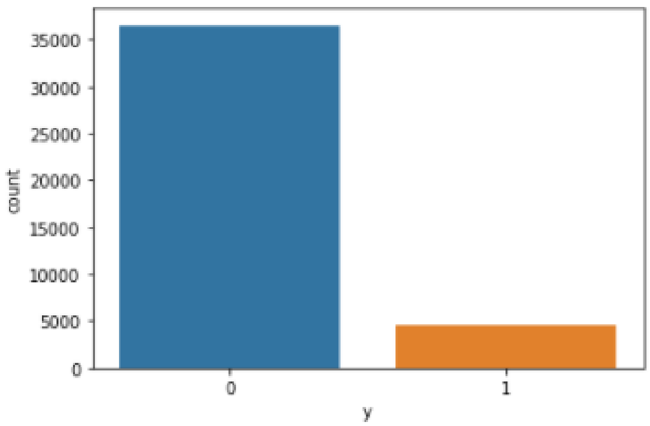
```
[148] # split into train and test datasets
      X_train, X_test, Y_train, Y_test = train_test_split(principalDf, y, test_size = 0.2, random_state = 100)
      X_train=X_train.reset_index(drop=True)
      X_test=X_test.reset_index(drop=True)
      Y_train=Y_train.reset_index(drop=True)
      Y_test=Y_test.reset_index(drop=True)
```

6. SMOTE

when considering y counts in the dataset.

```
[280] y.value_counts()

y
0    36535
1     4639
dtype: int64
```

```
sns.countplot(y['y'])

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fol
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f7087129b90>
```



As it can be seen from the above diagram there is an imbalance in the class column. Number of 0 value counts are far more than number of 1 value counts. If the class imbalance not handled properly, it can have a significant impact on model performance.T herefore to address class imbalance issue SMOTE is used. SMOTE increases the number of samples in the minority class by creating new points within the range of possibility, rather than replicating existing data points. To put it more simply, it adds new data points to the existing data. I have applied SMOTE to the training dataset.

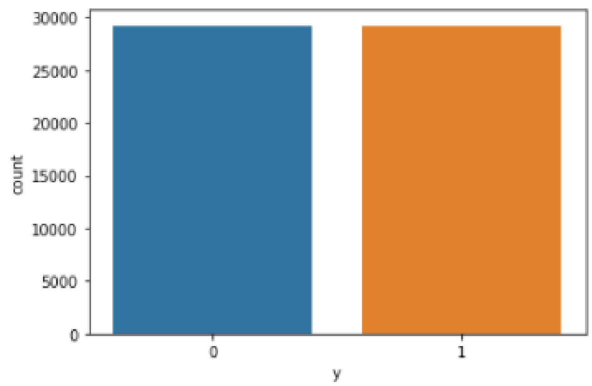
Code is as follows.

```
[605] from imblearn.over_sampling import SMOTE
      smote = SMOTE(random_state = 100)
      X, Y = smote.fit_resample(X_train, Y_train)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
  warnings.warn(msg, category=FutureWarning)
```

Y column values after applying SMOTE is as follows.

```
[609] finalD['y'].value_counts()

1    29236
0    29236
Name: y, dtype: int64
```

```
[610] sns.countplot(finalD['y'])

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following var
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fc9ad48a7d0>
```



As it can be seen from the above diagram count of 1s and 0s are equal which indicate that the y class in the training set is well balanced now.

7. SVM kernal Model

I have used SVM kernal model in order to create the model. Tuning the values of the parameters for SVM effectively improves the model performance. Therefore I have applied GridSearchCV to find the c and gamma values which gives the optimal performance.

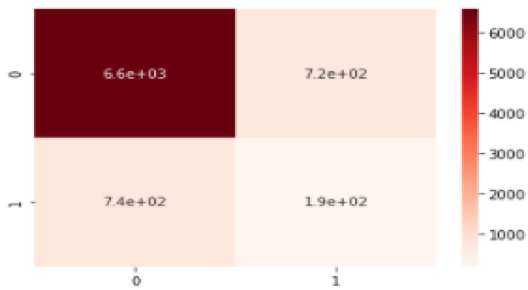
Classification report and confusion metrix of each c and gamma is as follows.

C=1, gamma=0.5

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.90      0.90      7300
           1       0.21      0.21      0.21       935

    accuracy                           0.82      8235
   macro avg       0.56      0.55      0.55      8235
weighted avg       0.82      0.82      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```

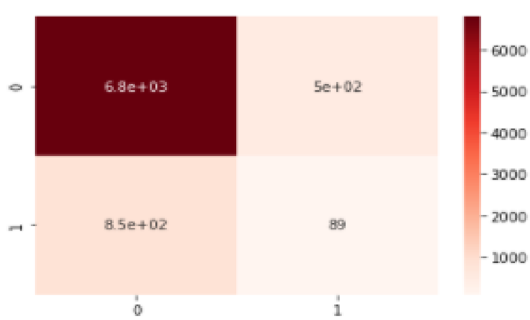


C=1, gamma=1

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      7300
           1       0.15      0.10      0.12       935

    accuracy                           0.84      8235
   macro avg       0.52      0.51      0.51      8235
weighted avg       0.81      0.84      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```

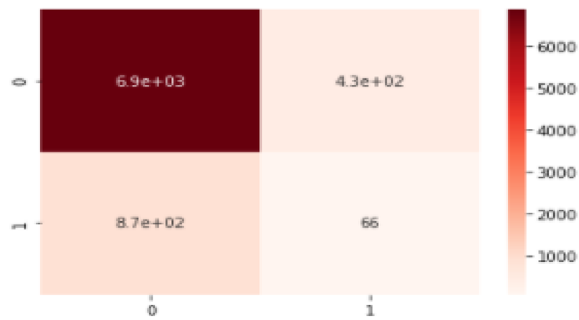


C=1, gamma=1.5

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      7300
           1       0.13      0.07      0.09       935

    accuracy                           0.84      8235
   macro avg       0.51      0.51      0.50      8235
weighted avg       0.80      0.84      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```

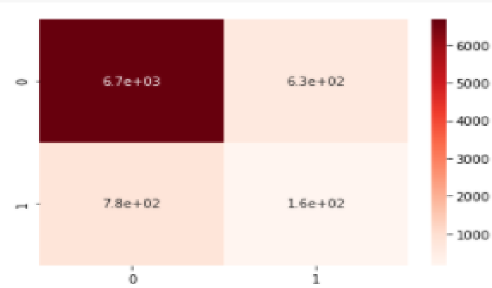


C=10, gamma=0.5

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.91      0.90      7300
           1       0.20      0.17      0.18       935

    accuracy                           0.83      8235
   macro avg       0.55      0.54      0.54      8235
weighted avg       0.82      0.83      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```

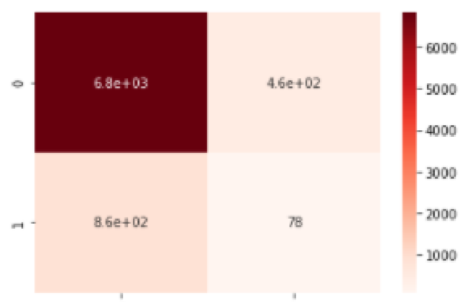


C=10,gamma=1

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      7300
           1       0.14      0.08      0.11       935

    accuracy                           0.84      8235
   macro avg       0.52      0.51      0.51      8235
weighted avg       0.80      0.84      0.82      8235
```

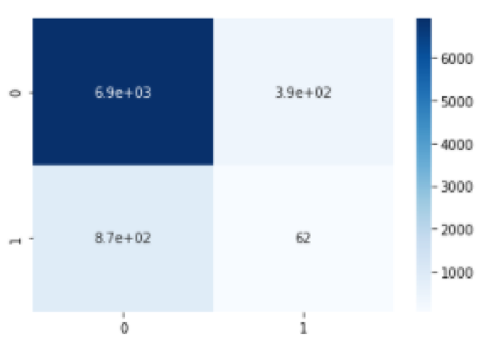
```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```



C=10, gamma=1.5

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      7300
           1       0.14      0.07      0.09       935

    accuracy                           0.85      8235
   macro avg       0.51      0.51      0.50      8235
veighted avg       0.80      0.85      0.82      8235
```

```
:nf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Blues, annot=True)
olt.show()
```

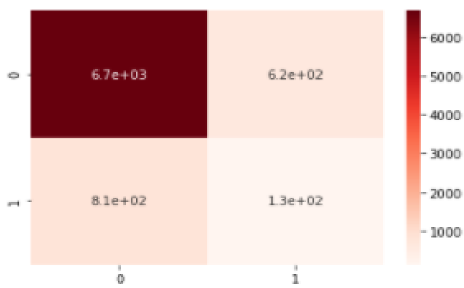


C=100, gamma=0.5

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      7300
           1       0.17      0.14      0.15       935

    accuracy                           0.83      8235
   macro avg       0.53      0.53      0.53      8235
weighted avg       0.81      0.83      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```

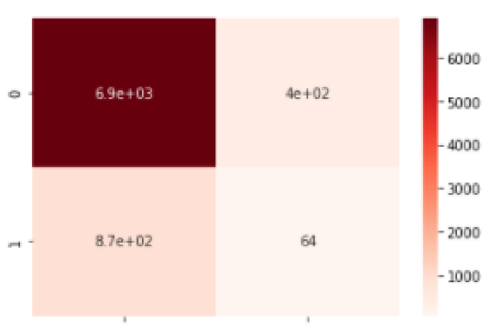


C=100, gamma=1.5

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.95      0.92      7300
           1       0.14      0.07      0.09       935

    accuracy                           0.85      8235
   macro avg       0.51      0.51      0.50      8235
weighted avg       0.80      0.85      0.82      8235
```

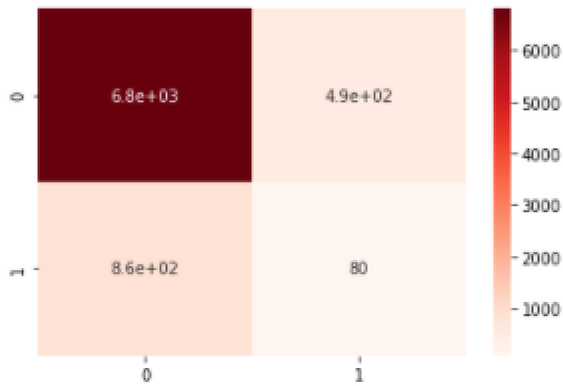
```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```



C=100, Gamma=1

```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      7300
           1       0.14      0.09      0.11       935

    accuracy                           0.84      8235
   macro avg       0.51      0.51      0.51      8235
weighted avg       0.80      0.84      0.82      8235
```

```
cnf_matrix = confusion_matrix(Y_test,Y_pred)
fig, ax = plt.subplots(1)
ax = sns.heatmap(cnf_matrix, ax=ax, cmap=plt.cm.Reds, annot=True)
plt.show()
```



As it can be seen from the output when c=10 and gamma=1.5 rbf kernel gives the optimal model.

```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', C = 10, gamma = 1.5)
classifier.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Pl
  y = column_or_1d(y, warn=True)
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1.5, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Y_pred is used to store the predicted value using the created model for the X_test set. Predicted values are as follows.

```
y=pd.DataFrame(Y_pred)
y.value_counts()
```

```
0    7787
1     448
dtype: int64
```

The mean error value of the model and the mean squared error value of the model is as follows.

```
from sklearn.metrics import mean_squared_error
mean_squared_error(Y_test, Y_pred)
```

```
0.15288403157255617
```

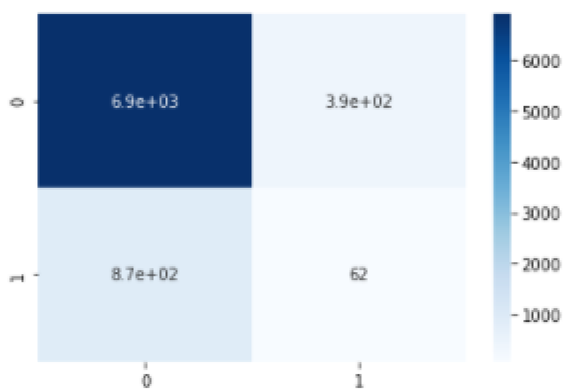
```
#Root Mean Sqaured Error
from math import sqrt
rmsq = sqrt(mean_squared_error(Y_test, Y_pred))
rmsq
```

```
0.3910038766720302
```

Classification report is as follows.

	precision	recall	f1-score	support
0	0.90	0.97	0.93	7365
1	0.22	0.07	0.11	873
accuracy			0.88	8238
macro avg	0.56	0.52	0.52	8238
weighted avg	0.83	0.88	0.85	8238

This created model shows 88% accuracy. Further confusion metrix is as follows.



As per the confusion metrix,

True Positive - 62

True Negative -6900

False Positive -8700

False Negative-3900