MSc Big Data Analytics

CMM704 – Data Mining

IIT Student ID: **20200361**
RGU Student ID: **2016629**

Don Manula Ransika Udugahapattuwa

Table of Contents

(a)

We can start the project by setting up the prerequisites and introducing the data set.

Setup the pre-requists and introduce the data set

```
[3]
#Set Prerequisites
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

```
[4]
#Mount gdrive
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

```
#Import the dataset asa Pandas data frame
df = pd.read_csv('/content/gdrive/My Drive/DataMining/assignment/dataset/banking.csv')
df.head(10)
```

I. Handle missing values and outliers

Drop "Duration" column first: the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this Input should only be included for benchmark purposes and should be discarded.

(a)I. Handle Missing Values and Outliers if any

Drop "Duration" column first: the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this Input should only be included for benchmark purposes and should be discarded

```
[6]
df = df.drop(['duration'],axis=1)
df.head(5)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	campaign	pdays	previous	poutcom
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	1	999	0	nonexisten
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	1	999	0	nonexisten
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	3	6	2	succes
3	39	services	married	high.school	no	no	no	cellular	apr	fri	2	999	0	nonexisten
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	1	3	1	succes

Then we can proceed with checking for missing values:

```
#Check for missing values
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  campaign        41188 non-null  int64
 11  pdays           41188 non-null  int64
 12  previous        41188 non-null  int64
 13  poutcome        41188 non-null  object
 14  emp_var_rate    41188 non-null  float64
 15  cons_price_idx  41188 non-null  float64
 16  cons_conf_idx   41188 non-null  float64
 17  euribor3m       41188 non-null  float64
 18  nr_employed     41188 non-null  float64
 19  y               41188 non-null  int64
dtypes: float64(5), int64(5), object(10)
memory usage: 6.3+ MB
None
```

Even though there are no NULL values, there might be other custom values like "unknown" that suggest a missing values.

- To check this, we can get the unique values of the columns.

Checking for unique values in each of the categorical features to check for other cases of missing values such as unknown and nonexistant

```
[5]  df['job'].unique()

array(['blue-collar', 'technician', 'management', 'services', 'retired',
       'admin.', 'housemaid', 'unemployed', 'entrepreneur',
       'self-employed', 'unknown', 'student'], dtype=object)

[6]  df['marital'].unique()

array(['married', 'single', 'divorced', 'unknown'], dtype=object)

[7]  df['education'].unique()

array(['basic.4y', 'unknown', 'university.degree', 'high.school',
       'basic.9y', 'professional.course', 'basic.6y', 'illiterate'],
      dtype=object)

[8]  df['default'].unique()

array(['unknown', 'no', 'yes'], dtype=object)

[9]  df['housing'].unique()

array(['yes', 'no', 'unknown'], dtype=object)
```

```
[10] df['loan'].unique()

array(['no', 'yes', 'unknown'], dtype=object)

[11] df['contact'].unique()

array(['cellular', 'telephone'], dtype=object)

[12] df['month'].unique()

array(['aug', 'nov', 'jun', 'apr', 'jul', 'may', 'oct', 'mar', 'sep',
       'dec'], dtype=object)

[13] df['day_of_week'].unique()

array(['thu', 'fri', 'tue', 'mon', 'wed'], dtype=object)

[14] df['poutcome'].unique()

array(['nonexistent', 'success', 'failure'], dtype=object)
```

- Then we can check the missing value percentage in order to get an idea on handling the data.

```
#Check for missing values in "job"
print('Unique Values in job column: ',df['job'].unique())

#Get the unkownvalue percentage out of the total number of
#data in the column in order to decide how to handle the missing values.
print('unkown entries percentage: ', len(df[df['job'] == 'unknown'])/len(df['job'])*100)

Unique Values in job column:  ['blue-collar' 'technician' 'management' 'services' 'retired' 'admin.'
 'housemaid' 'unemployed' 'entrepreneur' 'self-employed' 'unknown'
 'student']
unkown entries percentage:  0.8012042342429834
```

As the missing value percentage is low in the feature "job" (0.8%), we can choose to remove the rows with the "unknown" entry.

```
#There are "unknown" fields which we might have to handle
#As the missing value percentage is low, we can choose to
#remove the rows with the "unknown" entry
housing_dropping_index = df[ df['job'] == 'unknown' ].index
df.drop(housing_dropping_index, inplace = True)
df.shape

(40858, 20)
```

It can be done for all the columns which show a lower percentage of missing value count.

Marital:0.1%

[10]

```
#Check for missing values in "marital"
print('Unique Values in job column: ',df['marital'].unique())

#Get the unkown value percentage out of the total number of
#data in the column in order to decide how to handle the missing values.
print('unkown entries percentage: ', len(df[df['marital'] == 'unknown'])/len(df['marital'])*100)
```

```
Unique Values in job column:  ['married' 'single' 'divorced' 'unknown']
unkown entries percentage:  0.17377257819766018
```

```
#As the missing value percentage is low, we can choose to remove the rows with the "unknown" entry
housing_dropping_index = df[ df['marital'] == 'unknown' ].index
df.drop(housing_dropping_index, inplace = True)
df.shape
```

```
(40787, 20)
```

Education: 3.9%

```
#Check for missing values in "education"
print('Unique Values in job column: ',df['education'].unique())

#Get the unkown value percentage out of the total number of data in the column in order to decide how to
print('unkown entries percentage: ', len(df[df['education'] == 'unknown'])/len(df['education'])*100)
```

```
Unique Values in job column:  ['basic.4y' 'unknown' 'university.degree' 'high.school' 'basic.9y'
 'professional.course' 'basic.6y' 'illiterate']
unkown entries percentage:  3.9130114987618607
```

[13]

```
#As the missing value percentage is low, we can choose to remove the rows with the "unknown" entry
housing_dropping_index = df[ df['education'] == 'unknown' ].index
df.drop(housing_dropping_index, inplace = True)
df.shape
```

```
(39191, 20)
```

Loan: 20%

20% is a considerable "unknown" percentage. Still, the percentage is not high enough to fully drop the column. How ever, removing the rows would result in a considerable effect on the final predictions. Thus, decided to keep the "default" column as it is.

[14]

```
#Check for missing values in "default"
print('Unique Values in job column: ',df['default'].unique())

#Get the unkown value percentage out of the total number of data in the column in order to decide how
print('unkown entries percentage: ', len(df[df['default'] == 'unknown'])/len(df['default'])*100)
```

```
Unique Values in job column:  ['unknown' 'no' 'yes']
unkown entries percentage:  20.320992064504605
```

```
[15]
#As the missing value percentage is low, we can choose to remove the rows with the "unknown" entry
housing_dropping_index = df[ df['housing'] == 'unknown' ].index
df.drop(housing_dropping_index, inplace = True)
df.shape

(38245, 20)
```

```
[16]
#Check for missing values in "loan"
print('Unique Values in job column: ',df['loan'].unique())

#Get the unkown value percentage out of the total number of data in the column in order to decide how to handle the missing values.
print('unkown entries percentage: ', len(df[df['loan'] == 'unknown'])/len(df['loan'])*100)

Unique Values in job column:  ['no' 'yes']
unkown entries percentage:  0.0
```

Poutcome: 86%

A very high *86% of missing values * "nonexistant" are there in the poutcome column. As the missing data is extremely high, it will affect the final outcome if we use this column as a feature. Therefore, we can decide to **drop** this column.

When the housing column dropped the unknown rows, loan column has dropped its rows as well

```
[17]
#Check for missing values in "poutcome"
print('Unique Values in job column: ',df['poutcome'].unique())

#Get the unkown value percentage out of the total number of data in the column in order to decide how to handle the m
print('unkown entries percentage: ', len(df[df['poutcome'] == 'nonexistent'])/len(df['poutcome'])*100)

Unique Values in job column:  ['nonexistent' 'success' 'failure']
unkown entries percentage:  86.45836057000915
```

A very high **86% of missing values** "nonexistant" are there in the poutcome column. As the missing data is extremely high, it will affect the final outcome if we use this column as a feature. Therefore, we can decide to drop this column.

```
[24]
df = df.drop(['poutcome'],axis=1)
df.head(5)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	campaign	pdays	previous	emp
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	1	999	0	
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	3	6	2	
3	39	services	married	high.school	no	no	no	cellular	apr	fri	2	999	0	
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	1	3	1	
5	30	management	divorced	basic.4y	no	yes	no	cellular	jul	tue	8	999	0	

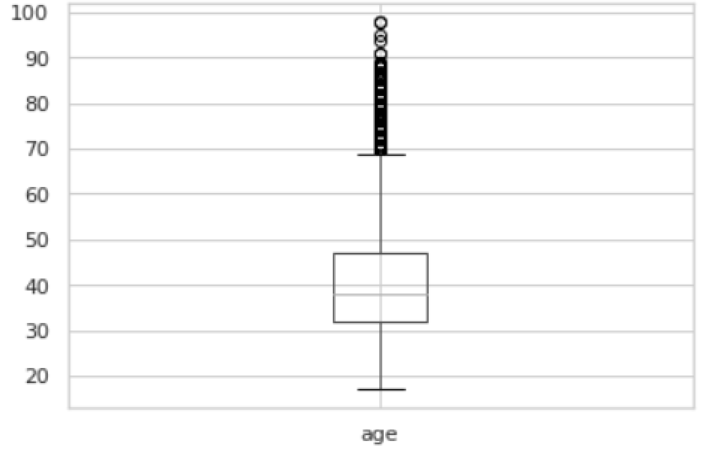
Checking and handling Outliers

Boxplots can identify the outliers efficiently.

Checking for Outliers and handling them.

```
[ ]
#Checking for outliers in "age"
df.boxplot(column=['age'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c5eb91110>
```

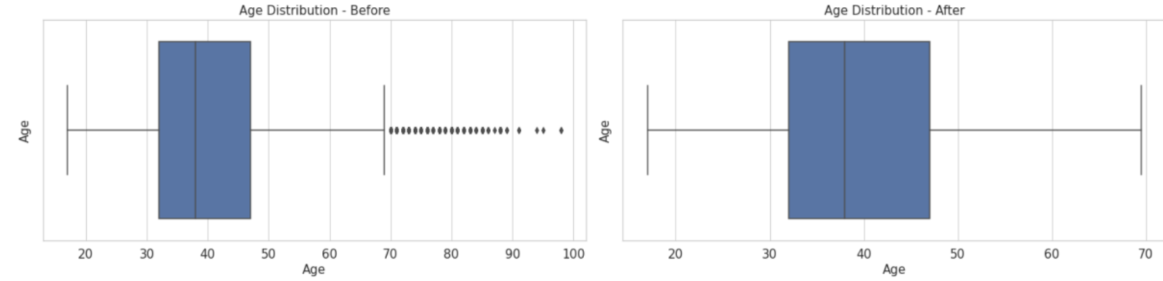


Removing the outliers in age would affect the overall result as the outlier count is almost half the total dataset. Thus, we can decide to replace the missing values using the Inter Quartile Range.

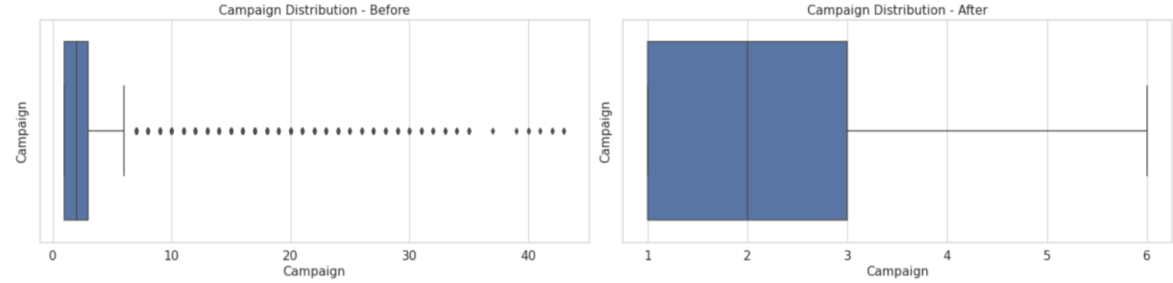
```
import warnings
warnings.filterwarnings("ignore")

numeric_col = df.select_dtypes(include=['int', 'float']).columns
print("Previous Shape With Outlier: ",df.shape)
Q1 = df.age.quantile(0.25)
Q3 = df.age.quantile(0.75)
print('Q1 & Q3',Q1,Q3)
IQR = Q3-Q1
print(IQR)

lower_limit = Q1 - 1.5*IQR
upper_limit = Q3 + 1.5*IQR
print(lower_limit,upper_limit)

df_copy = df.copy()
df_copy['age'] = np.where(df_copy['age']>upper_limit,upper_limit,df_copy['age'])
df_copy['age'] = np.where(df_copy['age']<lower_limit,lower_limit,df_copy['age'])
print("Shape After Removing Outliers:", df_copy.shape)
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))

#Manipulating the boxplot views
sns.boxplot(x = 'age', data = df[numeric_col], orient = 'v', ax = ax1)
ax1.set_xlabel('Age', fontsize=15)
ax1.set_ylabel('Age', fontsize=15)
ax1.set_title('Age Distribution - Before', fontsize=15)
ax1.tick_params(labelsize=15)

sns.boxplot(x = 'age', data = df_copy[numeric_col], orient = 'v', ax = ax2)
ax2.set_xlabel('Age', fontsize=15)
ax2.set_ylabel('Age', fontsize=15)
ax2.set_title('Age Distribution - After', fontsize=15)
ax2.tick_params(labelsize=15)


plt.subplots_adjust(wspace=0.5)
plt.tight_layout()

```

This is how the Before and after boxplots look like.

```
Previous Shape With Outlier:  (38245, 19)
Q1 & Q3 32.0 47.0
15.0
9.5 69.5
Shape After Removing Outliers: (38245, 19)
```



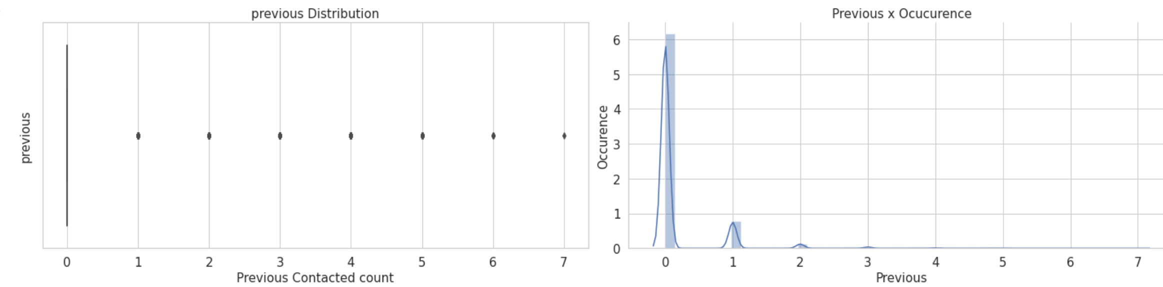

```
Previous Shape With Outlier:  (38245, 19)
Q1 & Q3 1.0 3.0
2.0
-2.0 6.0
Shape After Removing Outliers: (38245, 19)
```



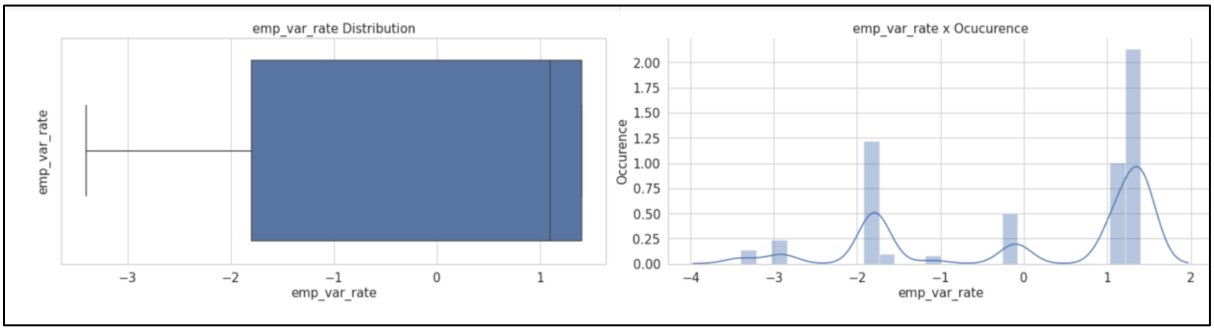
Number of previous contacts are distributed between 0 times and 7 times. Number of previous contacts are distributed between 0 times and 7 times.

This would not be manipulated as that might make it loose almost all the data in previous column.

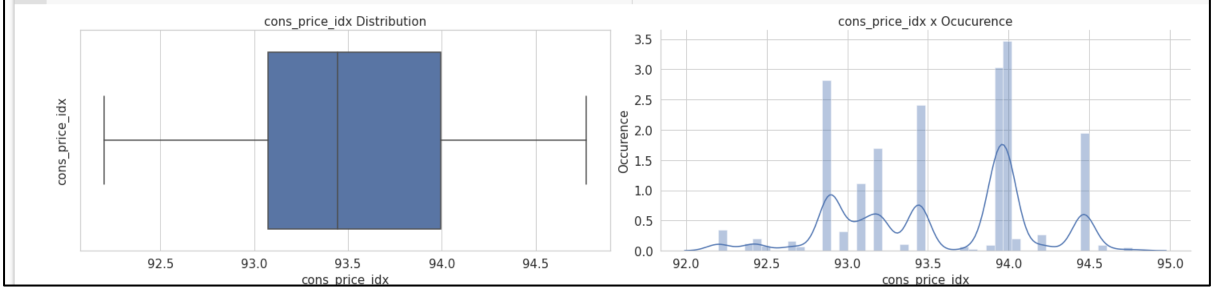
```
#Checking for outliers in "previous"
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))
cols_numerical = df_copy.select_dtypes(include=['int', 'float']).columns
sns.boxplot(x = 'previous', data = df_copy[cols_numerical], orient = 'v', ax = ax1)
ax1.set_xlabel('Previous Contacted count', fontsize=15)
ax1.set_ylabel('previous', fontsize=15)
ax1.set_title('previous Distribution', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(df_copy[cols_numerical]['previous'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('Previous', fontsize=15)
ax2.set_ylabel('Occurence', fontsize=15)
ax2.set_title('Previous x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



emp_var_rate and cons_price_idx has no visible outliers to be removed:

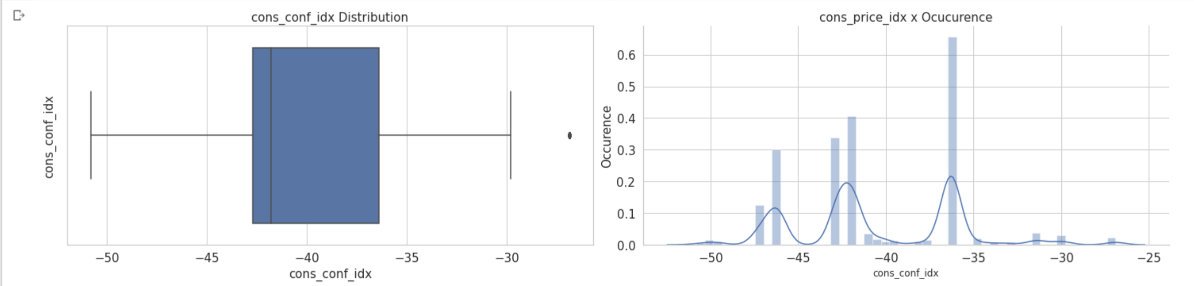



```
#Checking for outliers in "cons_price_idx"
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))
cols_numerical = df_copy.select_dtypes(include=['int', 'float']).columns
sns.boxplot(x = 'cons_price_idx', data = df_copy[cols_numerical], orient = 'v', ax = ax1)
ax1.set_xlabel('cons_price_idx', fontsize=15)
ax1.set_ylabel('cons_price_idx', fontsize=15)
ax1.set_title('cons_price_idx Distribution', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(df_copy[cols_numerical]['cons_price_idx'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('cons_price_idx', fontsize=15)
ax2.set_ylabel('Occurence', fontsize=15)
ax2.set_title('cons_price_idx x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



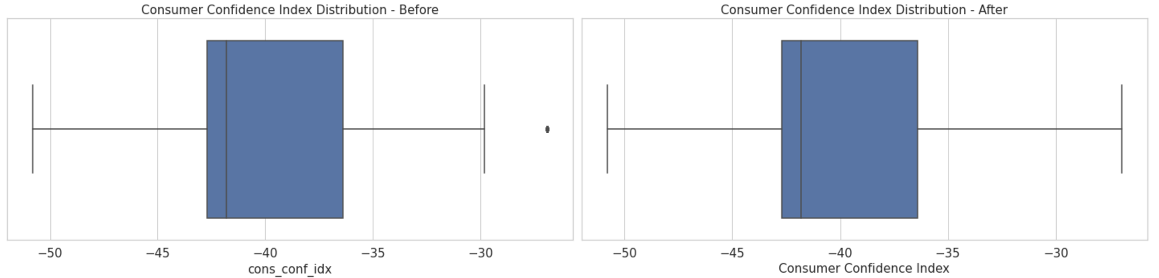
Checking for outliers in "cons_conf_idx" ehich will then be replaced using the IQR

```
#Checking for outliers in "cons_conf_idx"
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))
cols_numerical = df_copy.select_dtypes(include=['int', 'float']).columns
sns.boxplot(x = 'cons_conf_idx', data = df_copy[cols_numerical], orient = 'v', ax = ax1)
ax1.set_xlabel('cons_conf_idx', fontsize=15)
ax1.set_ylabel('cons_conf_idx', fontsize=15)
ax1.set_title('cons_conf_idx Distribution', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(df_copy[cols_numerical]['cons_conf_idx'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_ylabel('Occurence', fontsize=15)
ax2.set_title('cons_price_idx x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```



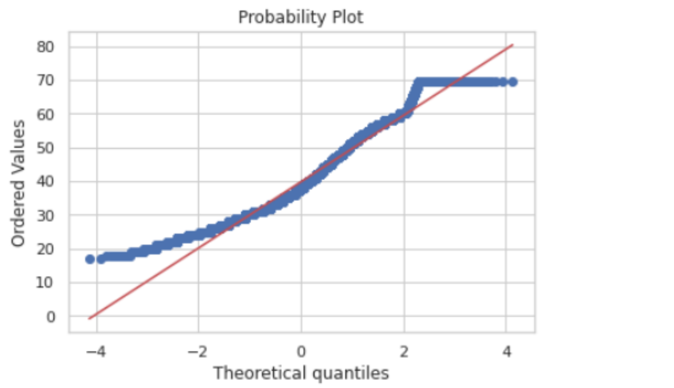
```python
#Handling outliers for Consumer Confidence Index

print("Previous Shape With Outlier: ",df_copy.shape)
Q1 = df_copy.cons_conf_idx.quantile(0.25)
Q3 = df_copy.cons_conf_idx.quantile(0.75)
print('Q1 & Q3',Q1,Q3)
IQR = Q3-Q1
print(IQR)

lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
print(lower_limit,upper_limit)


df_copy['cons_conf_idx'] = np.where(df_copy['cons_conf_idx']>upper_limit,upper_limit,df_copy['cons_conf_i
df_copy['cons_conf_idx'] = np.where(df_copy['cons_conf_idx']<lower_limit,lower_limit,df_copy['cons_conf_i
print("Shape After Removing Outliers:", df_copy.shape)
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))

sns.boxplot(x = 'cons_conf_idx', data = df[numeric_col], orient = 'v', ax = ax1)
ax1.set_xlabel('cons_conf_idx', fontsize=15)
ax1.set_title('Consumer Confidence Index Distribution - Before', fontsize=15)
ax1.tick_params(labelsize=15)

sns.boxplot(x = 'cons_conf_idx', data = df_copy[numeric_col], orient = 'v', ax = ax2)
ax2.set_xlabel('Consumer Confidence Index', fontsize=15)
ax2.set_title('Consumer Confidence Index Distribution - After', fontsize=15)
ax2.tick_params(labelsize=15)


plt.subplots_adjust(wspace=0.5)
```

```
Previous Shape With Outlier:  (38245, 19)
Q1 & Q3 -42.7 -36.4
6.300000000000004
-52.150000000000006 -26.949999999999992
Shape After Removing Outliers: (38245, 19)
```

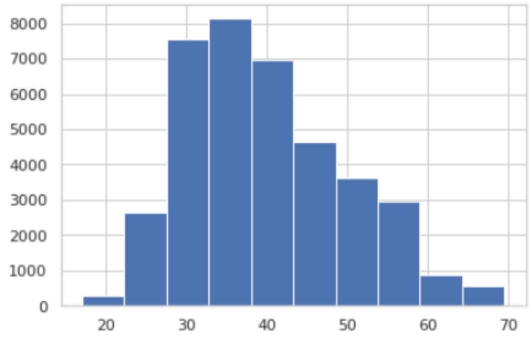


II. Produce **Q-Q Plots** and Histograms of the features, and apply the transformations if required

```
[ ]
#Q-Q Plots for the feature "age"
# import the libraries
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd

#Q-Q Plot
stats.probplot(df_copy["age"], dist="norm", plot=plt)
plt.show()

#Histogram
df_copy['age'].hist()
```

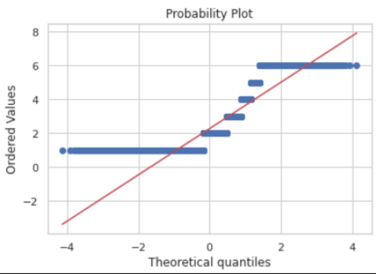



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c5dfbb590>
```

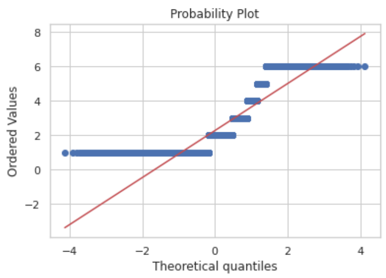


The feature "age" is right skewed when we take a look at the histogram. Thus, we can apply the technique of Transforming **Right Skewed Data using logarithm**

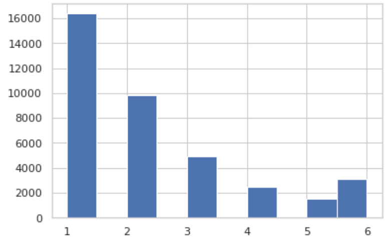
```
[ ]
#Q-Q Plots for the feature "age"

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

#Histogram
df_copy['campaign'].hist()
```



```
#Q-Q Plots for the feature "age"

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

#Histogram
df_copy['campaign'].hist()
```

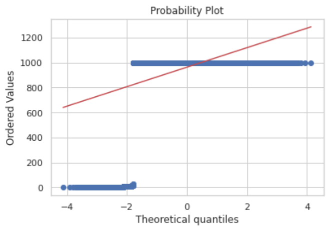



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c550e9ad0>
```

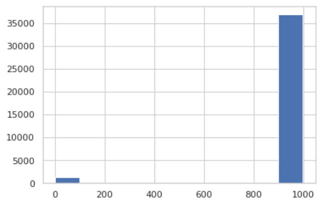


The feature "campaign" is also right skewed when we take a look at the histogram. Thus, we can apply the technique of **Transforming** Right Skewed Data using logarithm

```
#Q-Q Plots for the feature "pdays"

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

#Histogram
df_copy['pdays'].hist()

```

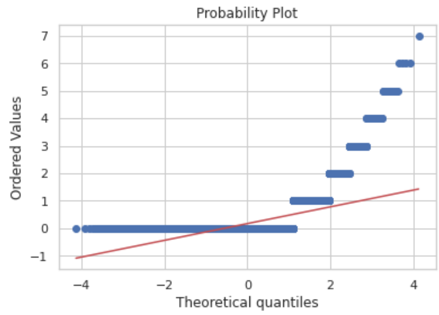



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c54c60250>
```

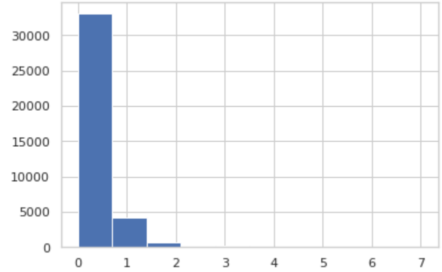


Pdays is heavily left skewed. Exponential transformation can be used to transform for left skewed data.

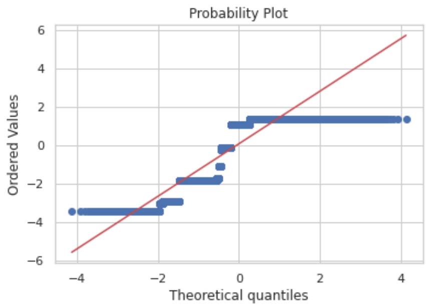
```
#Q-Q Plots for the feature "previous"

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

#Histogram
df_copy['previous'].hist()
```



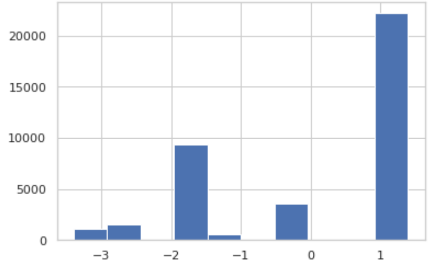

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c552dd510>
```



```
#Q-Q Plots for the feature "emp_var_rate"

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

#Histogram
df_copy['emp_var_rate'].hist()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c54b5c490>
```

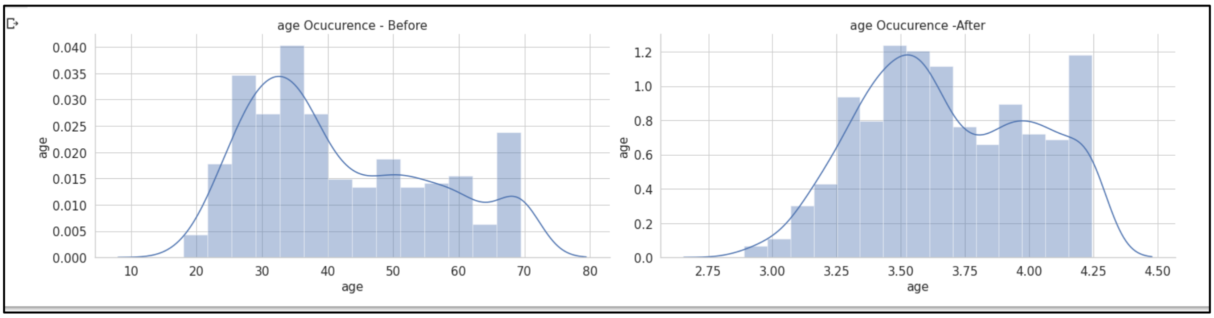


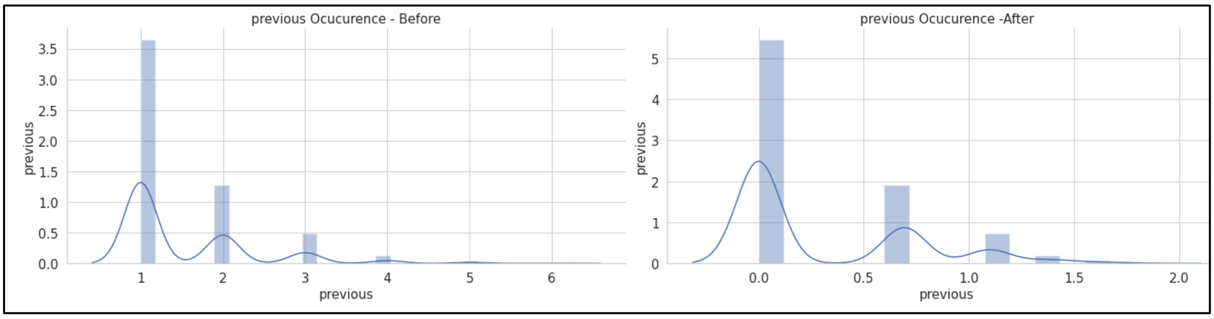
emp_var_rate is left skewed. Exponential transformation can be used to transform for left skewed data.

- emp_var_rate is left skewed. Exponential transformation can be used to transform for left skewed data.
- exponential_transformation can be applied for left skewed data

All Right Skewed features will be transformed using Logarithmic transformation method

```
from sklearn.preprocessing import FunctionTransformer
# load your data
data = df_copy
data = data.replace(0, np.nan) #replace 0 s with nan because ln
data = data.dropna() #drop all nan
columns = ['age','campaign', 'previous', 'cons_price_idx']

# create the function transformer object with logarithm transformation
logarithm_transformer = FunctionTransformer(np.log, validate=True)

# apply the transformation to the data
data_new = logarithm_transformer.transform(data[columns])
df_new = pd.DataFrame(data_new, columns=columns)

#histograms for 'age', 'campaign' Before & After the transformation
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))

sns.distplot(data[numeric_col]['age'], ax = ax1)
sns.despine(ax = ax1)
ax1.set_xlabel('age', fontsize=15)
ax1.set_ylabel('age', fontsize=15)
ax1.set_title('age Ocucurence - Before', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(df_new[columns]['age'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('age', fontsize=15)
ax2.set_ylabel('age', fontsize=15)
ax2.set_title('age Ocucurence -After', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```






```
#exponential_transformation for left skewed data
exponential_transformer = FunctionTransformer(np.exp)
columns2 = ['pdays', 'emp_var_rate', 'nr_employed']

data2 = df_copy

data_new2 = exponential_transformer.transform(data2[columns2])
df_new2 = pd.DataFrame(data_new2, columns=columns2)
df_new2.head(2)

```

	pdays	emp_var_rate	nr_employed
0	inf	4.055200	inf
2	403.428793	0.182684	inf

Now we can combine the transformed features with the complete dataset we have:

```
[46]
#replacing age & campaign column data with transformed data
data['age'] = df_new['age']
data['campaign'] = df_new['campaign']
```

```
[48]
#replacing 'pdays', 'emp_var_rate' column data with transformed data
data['pdays'] = df_new2['pdays']
data['emp_var_rate'] = df_new2['emp_var_rate']
```

```
data.head(10)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	campaign	pdays	previous
2	3.583519	management	single	university.degree	no	yes	no	cellular	jun	thu	0.000000	403.428793	2.0

III. Feature Coding

Integer (Label) Encoding is used as it will not add new columns and does not expand feature space

```
[50]
#Analyze the data
data.dtypes
```

```
age                float64
job                 object
marital             object
education           object
default             object
housing             object
loan                object
contact             object
month               object
day_of_week         object
campaign           float64
pdays              float64
previous           float64
emp_var_rate       float64
cons_price_idx     float64
cons_conf_idx      float64
euribor3m          float64
nr_employed        float64
y                  float64
dtype: object
```

```
data['job_code'] = data['job'].astype('category').cat.codes
data['marital_code'] = data['marital'].astype('category').cat.codes
data['education_code'] = data['education'].astype('category').cat.codes
data['default_code'] = data['default'].astype('category').cat.codes
data['housing_code'] = data['housing'].astype('category').cat.codes
data['loan_code'] = data['loan'].astype('category').cat.codes
data['contact_code'] = data['contact'].astype('category').cat.codes
data['month_code'] = data['month'].astype('category').cat.codes
data['day_of_week_code'] = data['day_of_week'].astype('category').cat.codes
data.head(10)
```

After data encoding:

```
1 #Verify the new categorical columns exist
2 data.dtypes
```

```
age                float64
job                 object
marital             object
education           object
default             object
housing             object
loan                object
contact             object
month               object
day_of_week         object
campaign           float64
pdays              float64
previous           float64
emp_var_rate       float64
cons_price_idx     float64
cons_conf_idx      float64
euribor3m          float64
nr_employed        float64
y                  float64
job_code              int8
marital_code          int8
education_code        int8
default_code          int8
housing_code          int8
loan_code             int8
contact_code          int8
month_code            int8
day_of_week_code      int8
dtype: object
```

Now we can drop all the unencoded textual columns from the dataframe:

Drop the textual columns as we have already encoded them and added as new columns

```
1 data = data.drop(['job','marital','education','default','housing','loan','contact','month','day_of_week'], axis=1)
2 data.head(5)
```

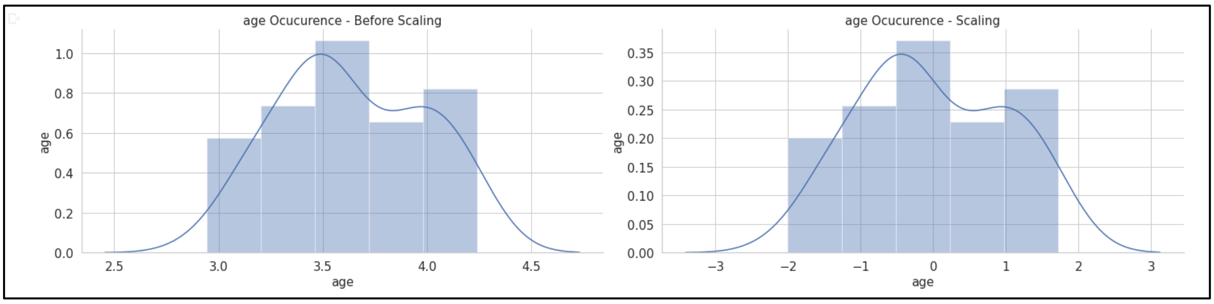
	age	campaign	pdays	previous	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	y	job_cc
2	3.583519	0.000000	403.428793	2.0	0.182684	94.055	-39.8	0.729	4991.6	1.0	
4	3.178054	0.000000	20.085537	1.0	0.055023	92.201	-31.4	0.869	5076.2	1.0	
8	3.713572	1.098612	20.085537	1.0	0.055023	92.963	-40.8	1.266	5076.2	1.0	
96	3.218876	0.693147	403.428793	1.0	0.165299	93.749	-34.6	0.659	5008.7	1.0	
117	3.583519	0.000000	inf	2.0	0.182684	94.027	-38.3	0.905	4991.6	1.0	

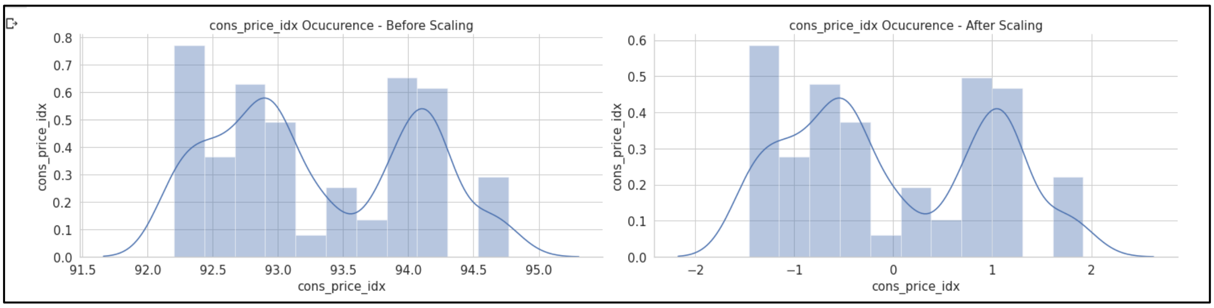
IV. Scaling data

Scaling features

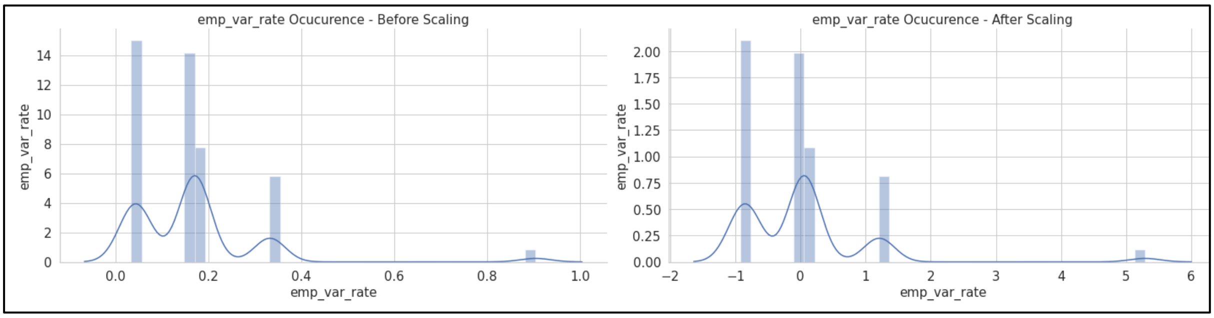
```python
# Appliying scaling

from sklearn.preprocessing import StandardScaler

# create the scaler object
scaler = StandardScaler()
scaling_columns=['age', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx' ,'euribor3m', 'nr_employed']
scaled = pd.DataFrame(data, columns = ['age', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx' ,'euribor3m', 'nr_employed'])
# fit the scaler to the  data
scaler.fit(scaled)

train_scaled = scaler.transform(scaled)
scaled_df = pd.DataFrame(train_scaled)
# scaled_df.info()


fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (20, 5))

sns.distplot(data[numeric_col]['age'], ax = ax1)
sns.despine(ax = ax1)
ax1.set_xlabel('age', fontsize=15)
ax1.set_ylabel('age', fontsize=15)
ax1.set_title('age Ocucurence - Before Scaling', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(scaled_df[0], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('age', fontsize=15)
ax2.set_ylabel('age', fontsize=15)
ax2.set_title('age Ocucurence - Scaling', fontsize=15)
ax2.tick_params(labelsize=15)


plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```

This is how the scaled age column looks like:





V. Feature discretization

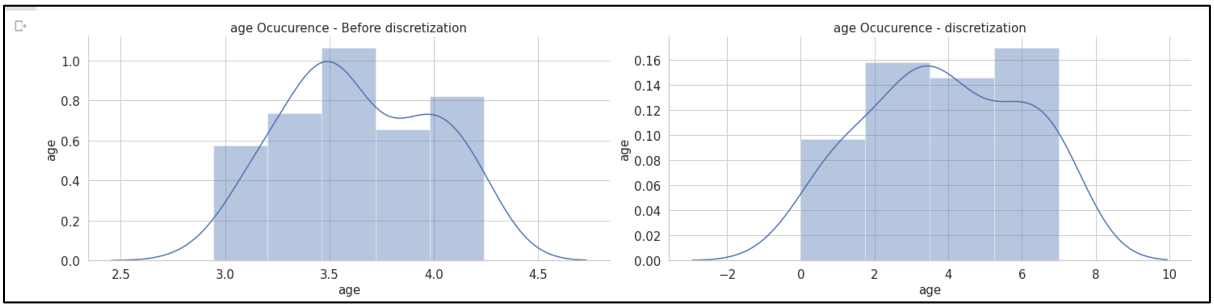


KBinsDiscretizer was used on age to discretize the feature:

(a)V. Feature discretization

[59]

```
import pandas as pd
from sklearn.preprocessing import KBinsDiscretizer


# create the scaler object
scaler = StandardScaler()


data5 = pd.DataFrame(data, columns=['age'])
data5 = data5.dropna()
# fit the scaler to the  data
discretizer = KBinsDiscretizer(n_bins=8, encode='ordinal', strategy='kmeans')
discretizer.fit(data5)
_discretize = discretizer.transform(data5)
df_discret = pd.DataFrame(_discretize)
df_discret
```



(b)

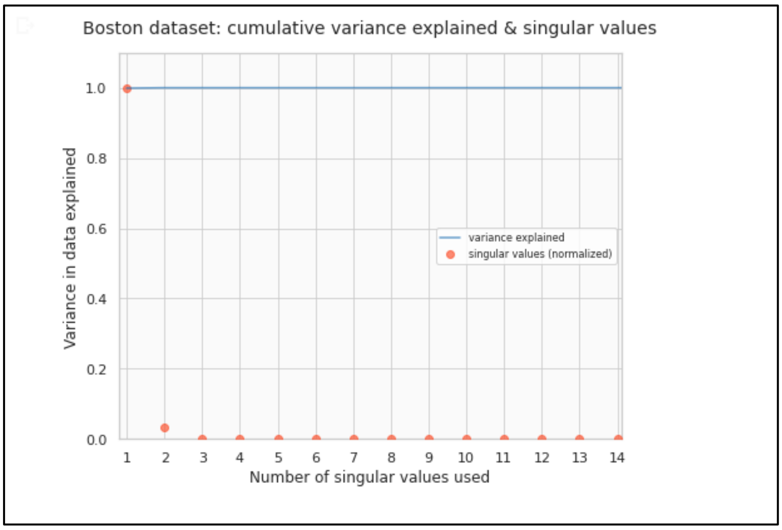
I. SVD (Singular Value Decomposition) for feature reduction.

```
# calculate the  SVD for the data X
U_boston,S_boston,Vt_boston = np.linalg.svd(df_copy, full_matrices=False)

#numpy.linalg.svd actually returns a Σ that is not a diagonal matrix, but a list of the entries on the diagonal.
num_sv_boston = np.arange(1, S_boston.size+1)
cum_var_explained_boston = [np.sum(np.square(S_boston[0:n])) / np.sum(np.square(S_boston)) for n in num_sv_boston]
print(S_boston)
print(S_boston.size)
```

```
[1.02834811e+06 3.45687932e+04 1.96412421e+03 9.39371486e+02
 6.97669711e+02 4.83694968e+02 4.17055250e+02 3.64590642e+02
 2.99496657e+02 2.70172453e+02 1.45441507e+02 1.07597114e+02
 9.72502617e+01 7.60077278e+01 7.39901478e+01 7.07496323e+01
 6.61331454e+01 5.48248662e+01 3.10460592e+01]
19
```

SVD gives a weighted value for each feature so that we can choose the most suitable ones.

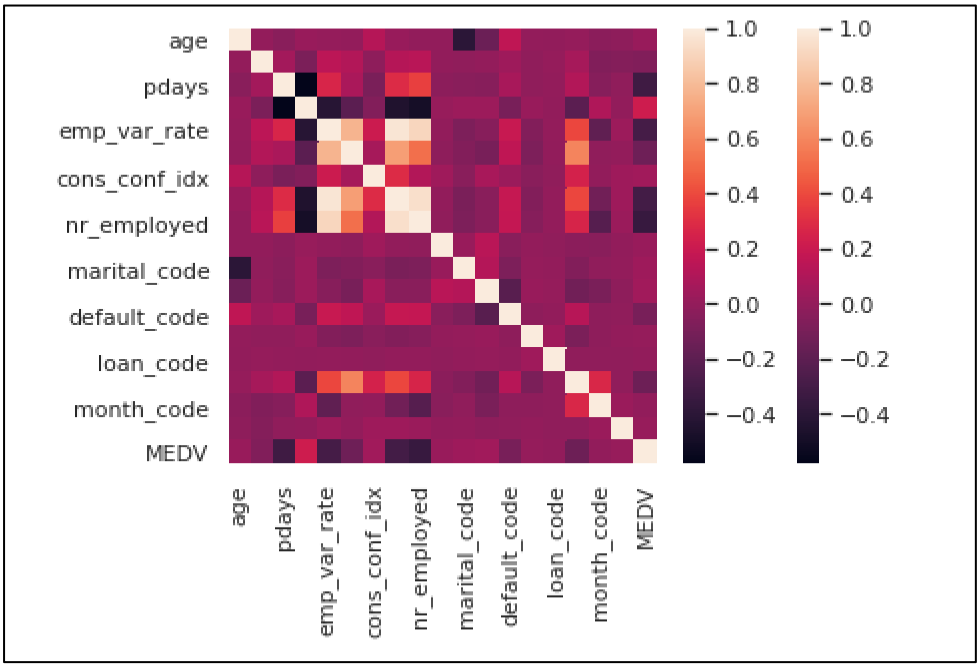
```
import matplotlib.pyplot as plt
import sklearn.datasets
import sklearn.preprocessing
fig = plt.figure(figsize=(7.0,5.5))
ax = fig.add_subplot(111)

plt.plot(num_sv_boston,
         cum_var_explained_boston,
         color='#2171b5',
         label='variance explained',
         alpha=0.65,
         zorder=1000)

plt.scatter(num_sv_boston,
            sklearn.preprocessing.normalize(S_boston.reshape((1,-1))),
            color='#fc4e2a',
            label='singular values (normalized)',
            alpha=0.65,
            zorder=1000)

plt.legend(loc='center right', scatterpoints=1, fontsize=8)

ax.set_xticks(num_sv_boston)
ax.set_xlim(0.8, 14.1)
ax.set_ylim(0.0, 1.1)
ax.set_xlabel('Number of singular values used')
ax.set_ylabel('Variance in data explained')
ax.set_title('Boston dataset: cumulative variance explained & singular values',
             fontsize=14,
             y=1.03)

ax.set_facecolor('0.98')

plt.grid(alpha=0.8, zorder=1)
plt.tight_layout()
```



II. Significant and independent features.

```
#test the signifncance of the features
import seaborn as sns
sns.heatmap(X.corr()); #Seems they can be assuemed as independent

d_data = X.copy()
d_data['MEDV'] = y_true
d_data.head(10)
print(d_data.corr())
sns.heatmap(d_data.corr()) #Seems we are ok with the signifncancy
```

```
                     age  campaign  ...  day_of_week_code      MEDV
age             1.000000  0.004124  ...         -0.018721  0.021775
campaign        0.004124  1.000000  ...         -0.053519 -0.066827
pdays          -0.031600  0.057497  ...         -0.008175 -0.319351
previous        0.020978 -0.081660  ...         -0.004482  0.221159
emp_var_rate    0.008260  0.148144  ...          0.032926 -0.292265
cons_price_idx  0.004577  0.115014  ...          0.002583 -0.133084
cons_conf_idx   0.125476 -0.015081  ...          0.040086  0.051331
euribor3m       0.020149  0.128578  ...          0.038921 -0.300580
nr_employed    -0.006760  0.140767  ...          0.029894 -0.347830
job_code       -0.004145 -0.005976  ...         -0.002870  0.020569
marital_code   -0.397646 -0.011171  ...          0.003796  0.041535
education_code -0.146225  0.003764  ...         -0.013620  0.054755
default_code    0.166061  0.038111  ...         -0.009527 -0.096116
housing_code   -0.000294 -0.010134  ...          0.001921  0.009996
loan_code      -0.006548  0.011173  ...         -0.010177 -0.005603
contact_code    0.009825  0.073329  ...         -0.011993 -0.140866
month_code     -0.025781 -0.063189  ...          0.025514 -0.002884
day_of_week_code -0.018721 -0.053519  ...        1.000000  0.014957
MEDV            0.021775 -0.066827  ...          0.014957  1.000000

[19 rows x 19 columns]
```



Darker the color the higher the correlation between the two variables.

(c). Applying Logistic Regression and Support Vector Machine techniques

First drop the y column from the data set for testing:

dropping y column from the data set for testing

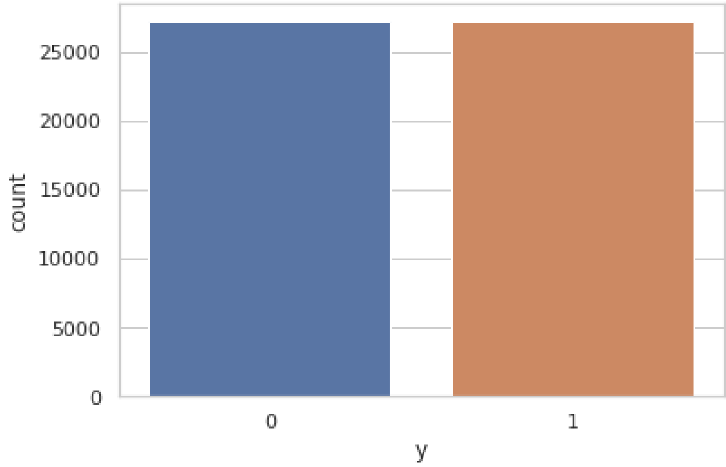
```
y_true = df_copy['y']
X = df_copy.drop('y', axis=1)
```

Split the dataset to **80% and 20%**

Splitting test and training data sets

```
from sklearn.model_selection import train_test_split
#Dont apply smote on test data
from imblearn.over_sampling import SMOTE

os = SMOTE(random_state=0)
X_class_train, X_test, y_class_train, y_test = train_test_split(X, y_true, test_size=0.2, random_state=0)
columns = X_class_train.columns

data_X, data_y = os.fit_sample(X_class_train, y_class_train)

smoted_X = pd.DataFrame(data=data_X,columns=columns )
smoted_y= pd.DataFrame(data=data_y,columns=['y'])
```

```
[ ]   1 sns.countplot(x='y', data=smoted_y)
      2 plt.show()
```



Training the dataset **with Logistic Regression**

```
[ ]   1 from sklearn.linear_model import LogisticRegression
      2 logreg = LogisticRegression()
      3 model = logreg.fit(X_train, y_train)
      4 y_pred = logreg.predict(X_test)
      5
      6 from sklearn.metrics import classification_report, confusion_matrix
      7 print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.75      0.84      6794
           1       0.26      0.68      0.37       855

    accuracy                           0.74      7649
   macro avg       0.60      0.72      0.61      7649
weighted avg       0.87      0.74      0.79      7649
```

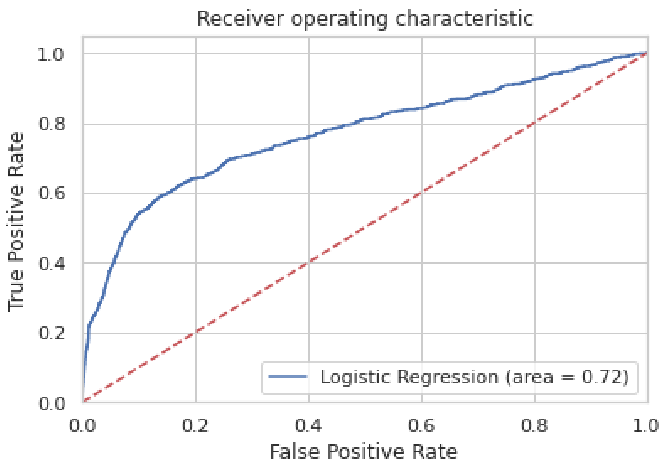
```
[ ]   1 print(confusion_matrix(y_test,y_pred))

[[5113 1681]
 [ 274  581]]
```

Confusion matrix is as follows:

```
[ ]   1 print(confusion_matrix(y_test,y_pred))

[[5113 1681]
 [ 274  581]]
```

ROC Curve is used to identify the diagnostic capability of a model.

```
#ROC CUrve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logreg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



Training the data **with SVM**

```
[157]
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.svm import SVC

model_svm = SVC(kernel='linear')
model_svm.fit(X_train,y_train)
preds = model_svm.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

```
0.8109556804811087
```

```
[158]
#confusion matrix , evaluation
print(confusion_matrix(y_test,preds))
```

```
[[5701 1093]
 [ 353  502]]
```

```
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.94      0.84      0.89      6794
           1       0.31      0.59      0.41       855

    accuracy                           0.81      7649
   macro avg       0.63      0.71      0.65      7649
weighted avg       0.87      0.81      0.83      7649
```

(d) applicability of SVM and LR

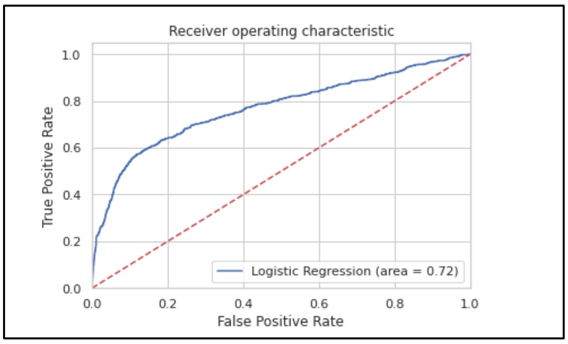
Logistic Regression:

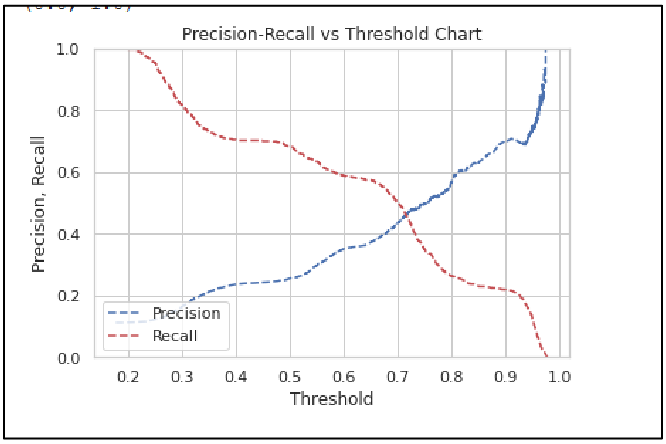
```
[ ]  from sklearn.linear_model import LogisticRegression
     logreg = LogisticRegression()
     model = logreg.fit(X_train, y_train)
     y_pred = logreg.predict(X_test)

     from sklearn.metrics import classification_report, confusion_matrix
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.75      0.84      6794
           1       0.26      0.68      0.37       855

    accuracy                           0.74      7649
   macro avg       0.60      0.72      0.61      7649
weighted avg       0.87      0.74      0.79      7649
```

```
[ ]  print(confusion_matrix(y_test,y_pred))
```

```
[[5113 1681]
 [ 274  581]]
```





Support Vector Machine

```
[158]  1 #confusion matrix , evaluation
       2 print(confusion_matrix(y_test,preds))

[[5701 1093]
 [ 353  502]]

       1 print(classification_report(y_test, preds))

              precision    recall  f1-score   support

           0       0.94      0.84      0.89      6794
           1       0.31      0.59      0.41       855

    accuracy                           0.81      7649
   macro avg       0.63      0.71      0.65      7649
weighted avg       0.87      0.81      0.83      7649
```

In Logistic Regression, the confusion matrix shows that negative-negative (true negative) count is 5113 while positive-positive count is 502. In SVM, negative-negative is 5701 and positive-positive (true-positive) count is 502 which is equal.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

According to the confusion matrix, SVM seems to be a better fit for our data set.

When comparing the 2 classification reports, LR shows an accuracy of 0.75 while SVM shows 0.84 which is a better score than LR.

Recall can be stated as the best metric to evaluate algorithms. This is because it takes the value of true positives over the total actual positives which gives a clear picture on the correctly predicted positive values.

In LR , recall value is 0.75 while in SVM it's 0.84, which suggests that SVM works better with the banking dataset.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Even in F1-score which is a good measure to check the balance between the two values, recall and precision, SVM has served our data set better.

$$\text{F1} = 2 \text{ x } \frac{Precision * Recall}{Precision + Recall}$$

Finally, we can conclude the fact that **SVM works best** with the banking dataset

(e) How significant your findings are

Logistic Regression

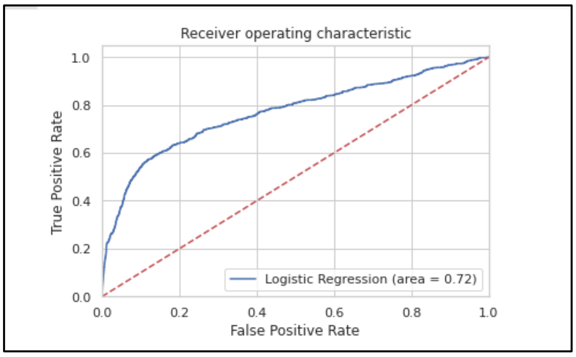
	precision	recall	f1-score	support
0	0.95	0.75	0.84	6794
1	0.26	0.68	0.37	855
accuracy			0.74	7649
macro avg	0.60	0.72	0.60	7649
weighted avg	0.87	0.74	0.79	7649

Support Vector Machine

	precision	recall	f1-score	support
0	0.94	0.84	0.89	6794
1	0.31	0.59	0.41	855
accuracy			0.81	7649
macro avg	0.63	0.71	0.65	7649
weighted avg	0.87	0.81	0.83	7649

The accuracy for SVM is 81% which is a considerable finding.

ROC Curve of svm is as follows.




```
[176]
#root mean squared error SVM
from sklearn.metrics import mean_squared_error
rms = np.sqrt(mean_squared_error(y_test,preds))
print(rms)

0.4347922716871717
```

RMSE shows a value of 0.43. An RMSE value lower than 0.5 reflects a good prediction rate of the model.

In Logistic Regression, recall value is 0.75 while in SVM it’s 0.84. Both the values are considerably good scores which signifies the findings.

*** END OF REPORT ***