Importing the packages

This is a Binary classification Problem.

In [1]:
```
import numpy as np
import pandas as pd
import copy
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
data = pd.read_excel(io = "Bank Data for case study assignment.xlsx", sheet_name="Data")
```

In [3]:
```
df = copy.copy(data)
```

In [4]:
```
data.head(6)
```

Out[4]:

	age	job	marital status	education	credit default?	housing loan?	Personal loan	y
0	30	unemployed	married	primary	no	no	no	no
1	33	services	married	secondary	no	yes	yes	no
2	35	management	single	tertiary	no	yes	no	no
3	30	management	married	tertiary	no	yes	yes	no
4	59	blue-collar	married	secondary	no	yes	no	no
5	35	management	single	tertiary	no	no	no	no

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1021 entries, 0 to 1020
Data columns (total 8 columns):
age                1021 non-null int64
job                1019 non-null object
marital status     1020 non-null object
education          1020 non-null object
credit default?    1020 non-null object
housing loan?      1019 non-null object
Personal loan      1019 non-null object
y                  1021 non-null object
dtypes: int64(1), object(7)
memory usage: 63.9+ KB
```

In [6]:
```
total_null = data.isnull().sum().sort_values(ascending=False)
print(total_null)
```

```
Personal loan      2
housing loan?      2
job                2
credit default?    1
education          1
marital status     1
y                  0
age                0
dtype: int64
```

There are 3 steps to proces further

1. Remove Missing Value
2. Replace missing value with Max occurance
3. Predict the Missing Value

We will go with Removing the Missing values. Take backup if needed

Dropping NA Values

In [14]:
```
data_proc = copy.copy(data)
```

In [15]:
```
len(data)
```

Out[15]: 1021

In [16]:
```
len(data_proc)
```

Out[16]: 1021

In [17]:
```
data_proc = data_proc.dropna(axis=0, inplace=False)
```

In [18]:
```
print(data_proc.head())
```

```
   age          job marital status   education credit default? housing loan?  \
0   30   unemployed         married    primary               no            no
1   33     services         married  secondary               no           yes
2   35   management          single   tertiary               no           yes
3   30   management         married   tertiary               no           yes
4   59  blue-collar         married  secondary               no           yes

  Personal loan   y
0            no  no
1           yes  no
2            no  no
3           yes  no
4            no  no
```

In [19]:
```
len(data_proc)
```

Out[19]: 1013

Replacing NA with Most occuring Value

In [20]:
```
data['Personal loan'].describe()
```

Out[20]:
```
count     1019
unique       2
top         no
freq       869
Name: Personal loan, dtype: object
```

In [21]:
```
for i in data.columns:
    print('-----{}-----'.format(i))
    print(data[i].describe())
    print()
```

```
-----age-----
count    1021.000000
mean       41.066601
std        10.400013
min        19.000000
25%        33.000000
50%        39.000000
75%        48.000000
max        84.000000
Name: age, dtype: float64

-----job-----
count            1019
unique             12
top       blue-collar
freq              217
Name: job, dtype: object

-----marital status -----
count        1020
unique          3
top       married
freq          617
Name: marital status , dtype: object

-----education-----
count          1020
unique            4
top       secondary
freq            524
Name: education, dtype: object

-----credit default?-----
count     1020
unique       2
top         no
freq       998
Name: credit default?, dtype: object

-----housing loan?-----
count     1019
unique       3
top        yes
freq       583
Name: housing loan?, dtype: object

-----Personal loan-----
count     1019
unique       2
top         no
freq       869
Name: Personal loan, dtype: object

-----y-----
count     1021
unique       2
top         no
freq       897
Name: y, dtype: object
```

In [22]:
```
data = data.fillna({"Personal loan": "no"})
data = data.fillna({"housing loan?": "yes"})
data = data.fillna({"job": "blue-collar"})
data = data.fillna({"credit default?": "no"})
data = data.fillna({"education": "secondary"})
data = data.fillna({"marital status ": "married"})
```

In [24]:
```
# Identifying no. of Unique Values

for i in data.columns:
    if i == 'age':
        pass
    else:
        print("------{}-----".format(i))
        print(data[i].value_counts())
        print()
```

```
------job-----
blue-collar      219
management       212
technician       178
admin.           107
services          93
self-employed     52
retired           46
entrepreneur      32
unemployed        29
student           23
housemaid         20
unknown           10
Name: job, dtype: int64

------marital status -----
married     618
single      281
divorced    122
Name: marital status , dtype: int64

------education-----
secondary    525
tertiary     303
primary      151
unknown       42
Name: education, dtype: int64

------credit default?-----
no     999
yes     22
Name: credit default?, dtype: int64

------housing loan?-----
yes      585
no       435
xxxyy      1
Name: housing loan?, dtype: int64

------Personal loan-----
no     871
yes    150
Name: Personal loan, dtype: int64

------y-----
no     897
yes    124
Name: y, dtype: int64
```

Housing Loan has extra variable as 'xxxyy', we replace it to max result 'yes'

In [25]:
```
data['housing loan?'].value_counts()
```

Out[25]:
```
yes      585
no       435
xxxyy      1
Name: housing loan?, dtype: int64
```

In [26]:
```
data['housing loan?'] = data['housing loan?'].str.replace('xxxyy','yes')
```

In [27]:
```
data['housing loan?'].value_counts()
```

Out[27]:
```
yes    586
no     435
Name: housing loan?, dtype: int64
```

In [28]:
```
# Find total Num after processing
total_null_1 = data.isnull().sum().sort_values(ascending=False)
print(total_null_1)
```

```
y                  0
Personal loan      0
housing loan?      0
credit default?    0
education          0
marital status     0
job                0
age                0
dtype: int64
```

In [30]:
```
dff = pd.DataFrame(data)
```

In [31]:
```
dff.head()
```

Out[31]:

	age	job	marital status	education	credit default?	housing loan?	Personal loan	y
0	30	unemployed	married	primary	no	no	no	no
1	33	services	married	secondary	no	yes	yes	no
2	35	management	single	tertiary	no	yes	no	no
3	30	management	married	tertiary	no	yes	yes	no
4	59	blue-collar	married	secondary	no	yes	no	no

In [32]:
```
dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1021 entries, 0 to 1020
Data columns (total 8 columns):
age                1021 non-null int64
job                1021 non-null object
marital status     1021 non-null object
education          1021 non-null object
credit default?    1021 non-null object
housing loan?      1021 non-null object
Personal loan      1021 non-null object
y                  1021 non-null object
dtypes: int64(1), object(7)
memory usage: 63.9+ KB
```

Use one-hot-encoder or Manual inner-variable Conversion

Trying Label Encoding

In [33]:
```
from sklearn.preprocessing import LabelEncoder
```

In [34]:
```
labelencoder = LabelEncoder()
```

In [35]:
```
dff.head()
```

Out[35]:

	age	job	marital status	education	credit default?	housing loan?	Personal loan	y
0	30	unemployed	married	primary	no	no	no	no
1	33	services	married	secondary	no	yes	yes	no
2	35	management	single	tertiary	no	yes	no	no
3	30	management	married	tertiary	no	yes	yes	no
4	59	blue-collar	married	secondary	no	yes	no	no

In [36]:
```
dff.iloc[:,1] = labelencoder.fit_transform(dff.iloc[:,1])
dff.iloc[:,2] = labelencoder.fit_transform(dff.iloc[:,2])
dff.iloc[:,3] = labelencoder.fit_transform(dff.iloc[:,3])
dff.iloc[:,4] = labelencoder.fit_transform(dff.iloc[:,4])
dff.iloc[:,5] = labelencoder.fit_transform(dff.iloc[:,5])
dff.iloc[:,6] = labelencoder.fit_transform(dff.iloc[:,6])
dff.iloc[:,7] = labelencoder.fit_transform(dff.iloc[:,7])
```

In [37]:
```
dff.head(10)
```

Out[37]:

	age	job	marital status	education	credit default?	housing loan?	Personal loan	y
0	30	10	1	0	0	0	0	0
1	33	7	1	1	0	1	1	0
2	35	4	2	2	0	1	0	0
3	30	4	1	2	0	1	1	0
4	59	1	1	1	0	1	0	0
5	35	4	2	2	0	0	0	0
6	36	6	1	2	0	1	0	0
7	39	9	1	1	0	1	0	0
8	41	2	1	2	0	1	0	0
9	43	7	1	0	0	1	1	0

In [38]:
```
Y_data = dff['y']
```

In [39]:
```
Y_data.value_counts()
```

Out[39]:
```
0    897
1    124
Name: y, dtype: int64
```

In [40]:
```
X_data = dff.drop('y', axis = 1)
```

In [41]:
```
X_data.head()
```

Out[41]:

	age	job	marital status	education	credit default?	housing loan?	Personal loan
0	30	10	1	0	0	0	0
1	33	7	1	1	0	1	1
2	35	4	2	2	0	1	0
3	30	4	1	2	0	1	1
4	59	1	1	1	0	1	0

In [42]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, Y_data, test_size = 0.25, random_state =
0)
```

In [43]:
```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
765 256 765 256
```

In [44]:
```
len(y_test)
```

Out[44]: 256

In [45]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [49]:
```
print(X_train[:2, :])
```

```
[[-0.86139886  0.75242465  1.36016339 -0.28501424 -0.14615682 -1.1760857
   2.44019373]
 [ 1.27874623 -1.05575815 -0.26059205 -0.28501424 -0.14615682  0.85027817
  -0.40980353]]
```

In [ ]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
dtclassifier = DecisionTreeClassifier(criterion = 'gini', random_state =0)
dtclassifier.fit(X_train,y_train)
```

In [ ]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
rfclassifier = RandomForestClassifier(n_estimators= 100, criterion = 'gini', random_state =0)
rfclassifier.fit(X_train,y_train)
```

In [ ]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [ ]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [ ]:
```
cm
```

In [ ]:
```
acc_random_forest = round(classifier.score(X_train, y_train) * 100, 2)
print(acc_random_forest)
```

In [ ]:
```
logreg = LogisticRegression()
```

In [ ]:
```
logreg.fit(X_train, y_train)
```

In [ ]:
```
Y_pred = logreg.predict(X_test)
```

In [ ]:
```
acc_log = round(logreg.score(X_train, y_train)*100, 2)
print(acc_log)
```

In [ ]:
```
# Fitting Knn classifier to the Training set
from sklearn.neighbors import KNeighborsClassifier
knnclassifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =2)
knnclassifier.fit(X_train, y_train)
```

In [ ]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [ ]:
```
cmm = confusion_matrix(y_test, y_pred)
```

In [ ]:
```
cmm
```

In [ ]:
```
# Fitting SVM to the Training set
from sklearn.svm import SVC
svmclassifier = SVC(kernel = 'linear',random_state = 0)
svmclassifier.fit(X_train,y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

In [ ]:
```
cmm = confusion_matrix(y_test, y_pred)
cmm
```

Performing ML

In [ ]:
```
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

In [ ]:
```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
```

In [ ]:
```
sgd.fit(X_train, Y_train)
```

In [ ]:
```
y_pred = sgd.predict(X_test)
```

In [ ]:
```
sgd.score(X_train, y_train)
```

In [ ]:
```
# Merging the Values
```

In [ ]:
```
data[data['credit default?'].isna()]

# Finding the NA Rows

for i in data.columns:
    print(data[data[i].isna()])
```

In [ ]:
```
# Identifying the Numerical stats
data.age.describe()
```

In [ ]:
```
data["age"].value_counts()
```

In [ ]:
```
# Binning the data age to make it Categorical

data['agebin'] = pd.cut(data['age'], [10, 20, 30,40, 50, 60, 70, 80, 90], labels=['11-20', '21-30',
'31-40', '41-50', '51-60', '61-70', '71-80', '81-90'])
```

In [ ]:
```
print(data[data['age'] <= 20])
```

In [ ]:
```
data['agebin'].value_counts()
```

In [ ]:
```
# Finding the Unique value counts of data columns

for i in data.columns:
    if i == "age":
        pass
    else:
        print("---------------{}----------------".format(i))
        print(data[i].value_counts())
        print('')
```

In [ ]:
```
data["job"].value_counts()
```

In [ ]:
```
for i in data.columns:
    print(i)
```

In [ ]:
```
for i in data.columns:
    if i == "age":
        pass
    else:
        print("---------------{}----------------".format(i))
        print(data[i].value_counts())
        print('')
```

In [ ]:
```
data.apply(pd.value_counts)
```

In [ ]:
```
data["job"] = data["job"].astype('category')
```

In [ ]:
```
data["job"].describe()
```