Question 2

In this report, the *bank-full.csv* dataset will be used to make a predictive analysis to describe the data and to explain the relationship between one or more nominal, ordinal, interval, or ratio-level independent variables and one dependent binary variable.

Different classifications are used in this report, also the results are will be described as

Uploading File:

To upload dataset file we used this command, (sep=';') is for reading dataset and to separate data from each other if ';' symbol is indicated.

```
data=pandas.read_csv("bank-full.csv",sep=';')
```

The dataset includes 17 which are:

['age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'y']

'y' will be considered as the output of the classification and the other columns will be the classification inputs.

Data processing:

To ensure better results when performing the classification process, it must be ensured that the data does not contain missing data, as well as that the data are numbers.

By using command we can check if the data has a Nan values: data.isnull().sum()

The output is shown in the figure there is no missing data.

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
```

Figure 1: Nan Elements Numbers

By displaying the data, it was found that there are data that do not number. The classification process cannot be done by using it, and it cannot be dispensed, because of its impact on the classification process. So we re-represented these variables into numbers so that every non-numerical variable is represented by a number, and this number is uniform if the same variables are found in other places.

The columns: categ =
['job','marital','education','default','balance','housing','loan','contact','month','poutcome','y']

are just non-numerical elements therefore we used *LabelEncoder()* to solve this issue.

The pictures below show the same data before and after label encoding

Befor labelEncoder()

```
In [8]: data.head(1)
Out[8]:
   age         job  marital education  ... pdays  previous poutcome   y
0   58  management  married  tertiary  ...    -1         0  unknown  no
```

After labelEncoder()

```
In [11]: data.head(1)
Out[11]:
   age  job  marital  education  ...  pdays  previous  poutcome  y
0   58    4        1          2  ...     -1         0         3  0
```

Figure 2: Before And After *Labelencoder*

Splitting Dataset:

The dataset shape is (45211, 17) which means 45211 rows and 17 columns. The output is the last column and the others are features columns.

To split data first we have to convert the dataset type from DataFrame type to Numpy type:

```
data=data.to_numpy()
```

After that, select from 1. To 16. Columns and store it as X. The remained column will be stored as y as shown in the command below:

```
X, y = data[:, :16], data[:, 16:]
```

Before training we used the *train_test_split* function to divide the data to train and test data with a ratio of 75% training data and 25% testing data as shown in the command below:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1, stratify=y)
```

Now the training data shape is (33908, 16) with labeling output data with shape (33908, 1)

The test data shape is (11303, 16) and the labeling output data is (11303, 1).

This data will be used by all classification methods.

1) Logistic Regression

We used a ***linear_model.LogisticRegression*** classifier with solver 'liblinear' and the Inverse of regularization strength is 1 as shown in command below:

```
classifier_logisticREG = linear_model.LogisticRegression(solver='liblinear', C=1)
```

To train the data with the classifier shown in the command below:

```
classifier_logisticREG.fit(X_train, y_train)
```

For testing classifier performance with testing data shown in command below:

```
y_test_pred = classifier_logisticREG.predict(X_test)
```

To test the performance of the classifier we used *classification_report* function as shown in command:

```
classification_report(y_test, y_test_pred, target_names=class_names)
```

classifier_logisticREG performance on test dataset

	precision	recall	f1-score	support
Class-0	0.90	0.98	0.94	9981
Class-1	0.56	0.21	0.31	1322
accuracy			0.89	11303
macro avg	0.73	0.60	0.62	11303
weighted avg	0.86	0.89	0.87	11303

Figure 3: precision, recall,f1-score and Accuracy for Logistic Regression

Confusion matrix logistic REG Classifier

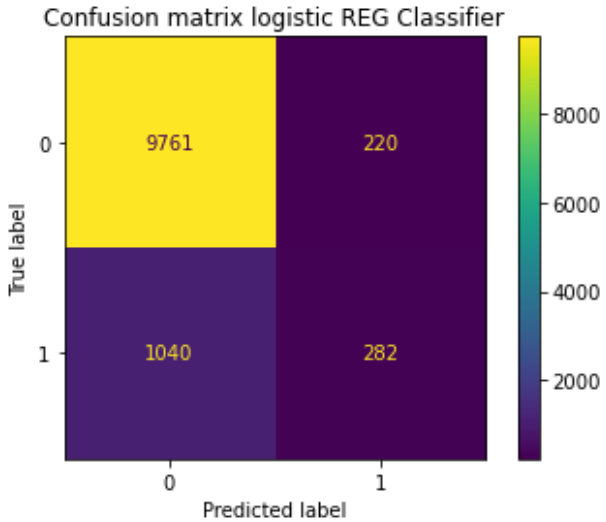

Figure 4: Confusion matrix Logistic Regression Classifier

2) Naïve Bayes:

We used a **GaussianNB** classifier as shown in the command below:
classifier_NaiveBayed = GaussianNB()

To train the data with the classifier shown in the command below:
classifier_NaiveBayed.fit(X_train, y_train)

For testing classifier performance with testing data shown in command below:
y_test_pred = classifier_NaiveBayed.predict(X_test)

To test the performance of the classifier we used *classification_report* function as shown in command:
classification_report(y_test, y_test_pred, target_names=class_names)

classifier Naive Bayed performance on test dataset

	precision	recall	f1-score	support
Class-0	0.93	0.88	0.90	9981
Class-1	0.35	0.49	0.41	1322
accuracy			0.84	11303
macro avg	0.64	0.69	0.66	11303
weighted avg	0.86	0.84	0.85	11303

Figure 5: precision, recall,f1-score and Accuracy for Naïve Bayes

Confusion matrix for Naïve Bayes Classifier:

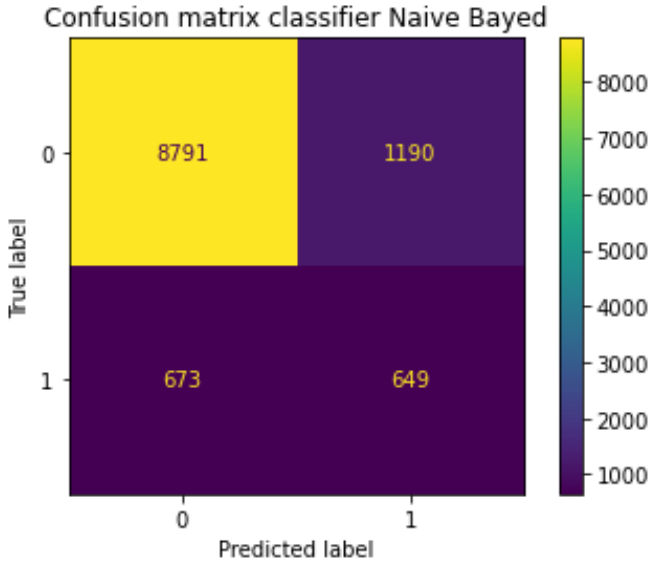

Figure 6: Confusion matrix for Naïve Bayes Classifier

3) Support Vector Machines-SVM:

We used a **OneVsOneClassifier** classifier as shown in command below:
classifier_SVM = OneVsOneClassifier(LinearSVC(random_state=0))

For training the classifier
classifier_SVM.fit(X_train, y_train)

To test the performance of the classifier we used *classification_report* function as shown in command:
y_test_pred = classifier_SVM.predict(X_test)

classifier SVM performance on test dataset

	precision	recall	f1-score	support
Class-0	0.88	1.00	0.94	9981
Class-1	0.29	0.01	0.01	1322
accuracy			0.88	11303
macro avg	0.59	0.50	0.47	11303
weighted avg	0.81	0.88	0.83	11303

Figure 7: precision, recall,f1-score, and Accuracy for SVM

Confusion matrix for SVM Classifier:

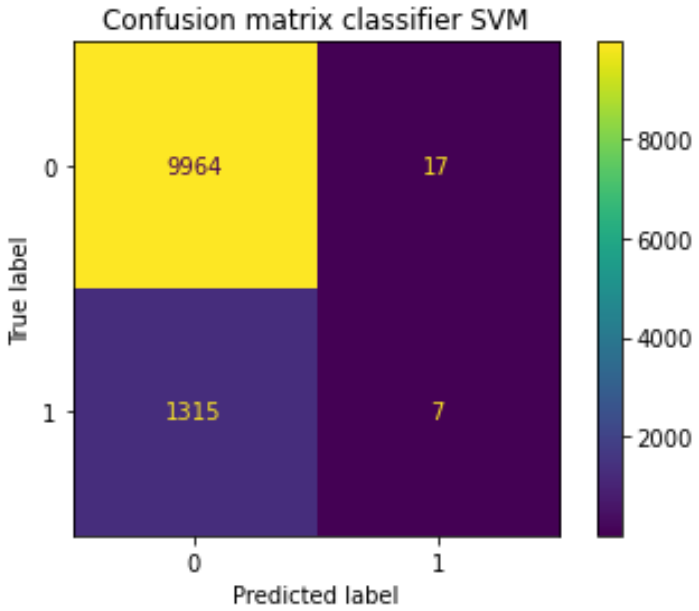

Figure 8: Confusion matrix for SVM Classifier

4) kNN

In this method *neighbors.KNeighborsClassifier* classifier is used with step size 10 and the distance weight type as shown in the command below:

```
KNeighbors_Classifier = neighbors.KNeighborsClassifier(10, weights='distance')
```

For training we trained the input data with its labels by command shown below:

```
KNeighbors_Classifier.fit(X_train, y_train)
```

Testing performance of the classifier by predicting the test data as shown in the command below:

```
y_test_pred = KNeighbors_Classifier.predict(X_test)
```

```
classifier K Neighbors performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.91      0.97      0.94      9981
     Class-1       0.50      0.25      0.33      1322

    accuracy                           0.88     11303
   macro avg       0.70      0.61      0.63     11303
weighted avg       0.86      0.88      0.87     11303
```

Figure 9: precision, recall,f1-score, and Accuracy for kNN

Confusion matrix for SVM Classifier:

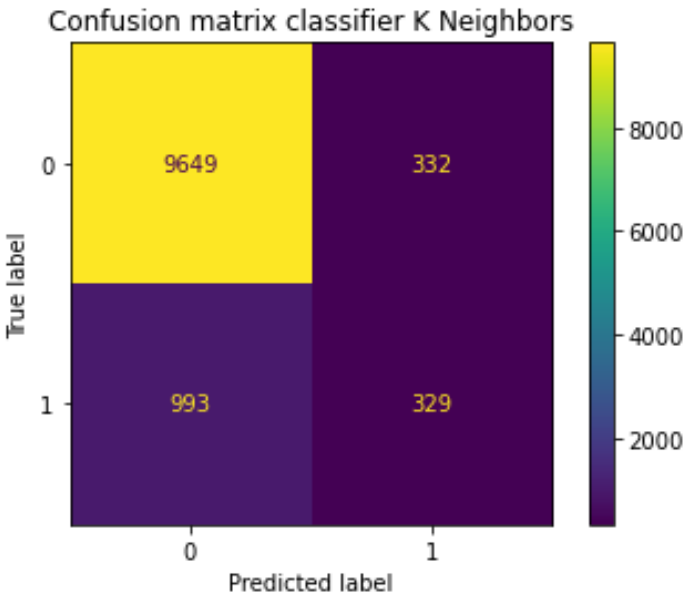

Figure 10: Confusion matrix for kNN Classifier

5) Neural Network:

At this question, I will use a deep neural network to make the classification of the bank dataset.

Loading dataset:
By comand lines below, I *bank-full.csv* loaded dataset which has shape (45211 , 17)
data=pandas.read_csv("bank-full.csv",sep=';')

For checking if there are NaN values I used the command line below, with its output as shown in figure 11:

data.isnull().sum()

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
```

Figure 11: NaN values Existance

Because there are some string columns, and the input dataset to the network must be numbered, I used LabelEncoder() function to represent each string element with numbers. I used the command lines below to re-encode the dataset strongly elements.

```
categ =
['job','marital','education','default','balance','housing','loan','contact','month','poutcom
e','y']
le = LabelEncoder()
data[categ] = data[categ].apply(le.fit_transform)
data.head(10)
```

This dataset can be considered as unbalanced data. Because the number of first-class is more than the other class by almost eight times. Therefore, I will use the same dataset but with different shapes.

The first shape is (45211, 17), here I did not change anything in the dataset the commands lines below used to split the dataset to train, validate and test input data (33908,16), (22605,16) and (11303,16) respectively, and for train, validation and test output data (33908,1), (22605,1) and (11303,1) respectively.

```
data=data.to_numpy()
X, y = data[:, :16], data[:, 16:]
Y_Class = np_utils.to_categorical(y, 2)
X_train, X_test, y_train, y_test = train_test_split(X, Y_Class, test_size=0.25,
random_state=1, stratify=y)
X_val, X_test1, y_val, y_test1 = train_test_split(X, Y_Class, test_size=0.5,
shuffle=True, random_state=10)
```

In the second shape, the number of samples will be equal. By using the command line below, we can know the numbers of each class:

```
from collections import Counter
Counter(data[:,16])
```

The output was Counter({0: 39922, 1: 5289}), which means I have to select from class 0 just 5289 samples.
To achieve that I regenerate new dataset by using the original dataset, as shown in the command lines below:

```
for i in range(len(data)-1):
    if ((data[i,16] == 0) and (j < 5289)):
        Rawdara.append(data[i])
        j = j+1
    elif (data[i,16] == 1):
        Rawdara.append(data[i])
Rawdara = numpy.array(Rawdara)
```

After that I split the data by command lines below:

```
X, y = Rawdara [:, :16], Rawdara [:, 16:]
Y_Class = np_utils.to_categorical(y, 2)
X_train, X_test, y_train, y_test = train_test_split(X, Y_Class, test_size=0.25,
random_state=1, stratify=y)
X_val, X_test1, y_val, y_test1 = train_test_split(X, Y_Class, test_size=0.5,
shuffle=True, random_state=10)
```

The network data shapes were for a training input (7923,16) output (7923,1), validation input (5289,16) output(5289,1), and test input (2654,16) output (2654,1)

Creating Model:

For creating a model that the network will be trained by, we will re-call sequential function, this function is the easiest way to build a model in Keras. It allows us to build a model layer by layer. The sequential function can recall by the command line below:

```
model = Sequential()
```

The first layer will be the input layer. Layer hyperparameter is 8 neurons as inputs, the activation will be the *'real'* function and the input will be 16. 40% of data output from 8 neurons will be dropped out, input layer can be described by command lines below:

```
model.add(Dense(8, activation='relu', input_dim=16))
model.add(Dropout(0.4))
```

The output from the input layer will represent the input for the first hidden layer. This layer has hyperparameter 4 neurons, 30% of the output of this layer will be dropped out.
The commands below show hidden layers of code.

```
model.add(Dense(4, activation='relu'))
model.add(Dropout(0.3))
```

Due to the variety of the output data, the output layer will have just 2 neurons. Every neuron represents one output data type. This model has 2 different types, which are 0 and 1, the activation function will be *'sigmoid'*. The output layer can be written by command line below:

```
model.add(Dense(2, activation='sigmoid'))
```

The model compiler will be *'binary_crossentropy'* for losses and *'adam'* as optimizer.

```
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer='adam')
```

For training, we used 64 samples as batch size and the number of epochs is 200.

```
model.fit(X_train, y_train, batch_size=64, epochs=400, validation_data = (X_val, y_val))
```

Predicting test data: For predictng I used predict() keras funcation that allow to predict the model output by passing data, the command lines below are predication commands:

```
y_pred = model.predict(X_test)
y_pred1 = numpy.argmax(y_pred, axis = 1)
y_test1 = numpy.argmax(y_test, axis = 1)
acct=y_pred1==y_test1
acct=acct*1
```

```
acc=(sum(acct)/len(acct))*100
acc
```

For Unbalanced data, we got 88.3% accuracy. the precision, recall, f1-score, and support I used the commands line below:

```
class_names = ['Class-0', 'Class-1']
print("#"*40)
print("\nclassifier DNN performance on test dataset\n")
print(classification_report(y_test1, y_pred1, target_names=class_names))
print("#"*40 + "\n")
```

the output as shown in figure 12:

```
lassifier DNN performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.88      1.00      0.94      9981
     Class-1       0.00      0.00      0.00      1322

    accuracy                           0.88     11303
   macro avg       0.44      0.50      0.47     11303
eighted avg       0.78      0.88      0.83     11303

########################################
```

Figure 12: Unbalanced Dataset Precision, Recall, F1-Score, And Support Results

For the Confusion matrix, I used the command lines below:

```
matrix = metrics.confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
sns.heatmap(matrix, annot=True,fmt='.4g')
```

The confusion matrix is shown in figure 13:

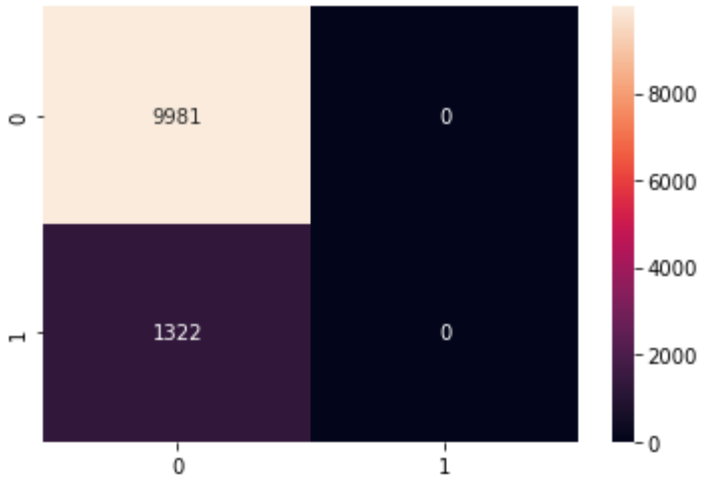

Figure 13: Unbalanced dataset Confusion Matrix

As shown in figure 13, the prediction is just in class 0 because of the significant difference in the number of samples.

For the balanced dataset, the accuracy is 95.39% , . the precision, recall, f1-score, and support I used the commands line below:

```
class_names = ['Class-0', 'Class-1']
print("#"*40)
print("\nclassifier DNN performance on test dataset\n")
print(classification_report(y_test1, y_pred1, target_names=class_names))
print("#"*40 + "\n")
```

the output as shown in figure 14:

```
classifier DNN performance on test dataset

              precision    recall  f1-score   support

     Class-0       0.94      0.97      0.95      1322
     Class-1       0.97      0.94      0.95      1323

    accuracy                           0.95      2645
   macro avg       0.95      0.95      0.95      2645
weighted avg       0.95      0.95      0.95      2645
```

Figure 14: Balanced Dataset Precision, Recall, F1-Score, And Support Results

For the Confusion matrix, I used the command lines below:

```
matrix = metrics.confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
sns.heatmap(matrix, annot=True,fmt='.4g')
```

The confusion matrix is shown in figure 14:

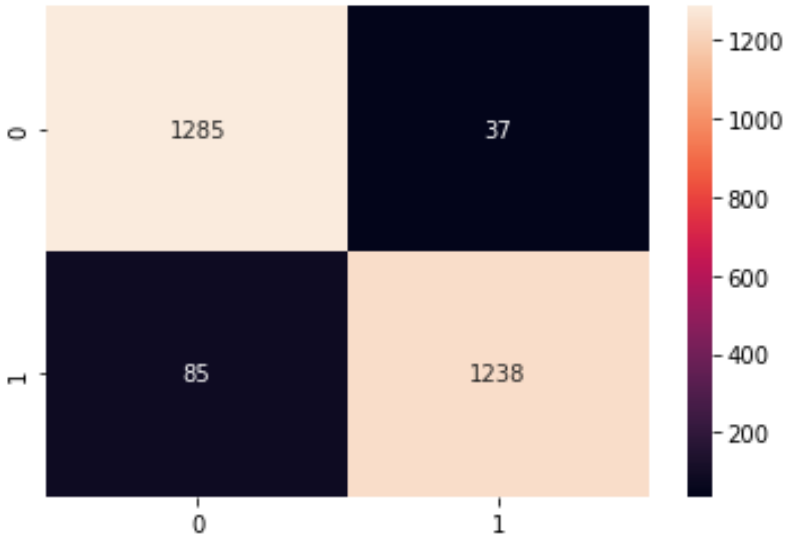

Figure 14: Balanced dataset Confusion Matrix