
# Higher National Diploma in Data Science.

## Machine Learning.

### Module leader: Eng. Chameera De Silva.

COHNDDS23.1F

30th May 2023

Sonal Punchihewa - 002

# Acknowledgement

I would like to use this opportunity to convey my sincere gratitude to my lecturers, in particular to Mr. Chameera De Silva, who is the module leader in charge, for their important direction, criticism, and advice. I also appreciate the cooperation and encouragement I have received from my peers during this academic journey. Last but not least, I want to express my gratitude to my family, especially my parents, for their continuous support and encouragement while I worked toward achieving my academic objectives.

# Table of Contents

# Introduction

This report will showcase us the model building and prediction of whether the customer will subscribe to a term deposit based on their demographic information such as age, marital status, education, etc. and their previous banking history such as the number of contacts with the bank and the outcome of the previous marketing attempts.

Furthermore, this report will show us the step-by-step process of how we will be building the model and how it'll be utilized to predict the results we want. Moreover, the results obtained from this prediction model will be interpreted on the later chapters on this report along with an overall summary.

# Step 1: Data Exploratory Analysis

## 1.1 Data Overview

First and foremost, the dataset that is going to be used for this project should be cleaned and pre-processed.

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

The necessary libraries required for the initial stage are imported here as you can see from the above image.

```
In [2]:
1 df=pd.read_csv('/Users/sonalpunchihewa/Downloads/bank-full.csv')
```

```
In [3]:
1 df.head()
```

Out[3]:

| | age | job | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign | pdays | previous | poutcome | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 58 | management | married | tertiary | no | 2143 | yes | no | unknown | 5 | may | 261 | 1 | -1 | 0 | unknown | no |
| 1 | 44 | technician | single | secondary | no | 29 | yes | no | unknown | 5 | may | 151 | 1 | -1 | 0 | unknown | no |
| 2 | 33 | entrepreneur | married | secondary | no | 2 | yes | yes | unknown | 5 | may | 76 | 1 | -1 | 0 | unknown | no |
| 3 | 47 | blue-collar | married | unknown | no | 1506 | yes | no | unknown | 5 | may | 92 | 1 | -1 | 0 | unknown | no |
| 4 | 33 | unknown | single | unknown | no | 1 | no | no | unknown | 5 | may | 198 | 1 | -1 | 0 | unknown | no |

```
In [4]:
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        45211 non-null  int64
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64
 12  campaign   45211 non-null  int64
 13  pdays      45211 non-null  int64
 14  previous   45211 non-null  int64
 15  poutcome   45211 non-null  object
 16  Target     45211 non-null  object
dtypes: int64(7), object(10)
```

The above image shows us the necessary pre-processing steps taken to understand what kind of a dataset we are dealing with here. In this case, we have taken the ‘head()’ function to get a detailed break-up of the total columns in the dataset and the ‘info()’ function to get the data-types and the total count of each value available under every column.

```
In [5]: 1 df.shape
Out[5]: (45211, 17)

In [6]: 1 df.describe()
Out[6]:
```

| | age | balance | day | duration | campaign | pdays | previous |
|---|---|---|---|---|---|---|---|
| count | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 | 45211.000000 |
| mean | 40.936210 | 1362.272058 | 15.806419 | 258.163080 | 2.763841 | 40.197828 | 0.580323 |
| std | 10.618762 | 3044.765829 | 8.322476 | 257.527812 | 3.098021 | 100.128746 | 2.303441 |
| min | 18.000000 | -8019.000000 | 1.000000 | 0.000000 | 1.000000 | -1.000000 | 0.000000 |
| 25% | 33.000000 | 72.000000 | 8.000000 | 103.000000 | 1.000000 | -1.000000 | 0.000000 |
| 50% | 39.000000 | 448.000000 | 16.000000 | 180.000000 | 2.000000 | -1.000000 | 0.000000 |
| 75% | 48.000000 | 1428.000000 | 21.000000 | 319.000000 | 3.000000 | -1.000000 | 0.000000 |
| max | 95.000000 | 102127.000000 | 31.000000 | 4918.000000 | 63.000000 | 871.000000 | 275.000000 |

```
In [7]: 1 #checking for null values
        2 df.isnull()
Out[7]:
```

| | age | job | marital | education | default | balance | housing | loan | contact | day | month | duration | campaign | pdays | previous | poutcome | Target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| 1 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| 2 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| 3 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| 4 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 45206 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |
| 45207 | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False | False |

Furthermore, we have used the ‘shape()’ function to get the number of rows and columns in the dataset and the ‘describe()’ function allows us to get the statistical description of the dataset we are dealing with.

Since now we have somewhat of an understanding of the dataset, we could proceed ahead with finding the null values in the dataset to filter them out. As you can see above in the image, the ‘isnull()’ function allows us to find any null values within the dataset. In this case, there isn’t any.

```
In [8]:
#Checking for unique values
for col in df.select_dtypes(include='object').columns:
    print(col)
    print(df[col].unique())
```

```
job
['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']
marital
['married' 'single' 'divorced']
education
['tertiary' 'secondary' 'unknown' 'primary']
default
['no' 'yes']
housing
['yes' 'no']
loan
['no' 'yes']
contact
['unknown' 'cellular' 'telephone']
month
['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
poutcome
['unknown' 'failure' 'other' 'success']
Target
['no' 'yes']
```

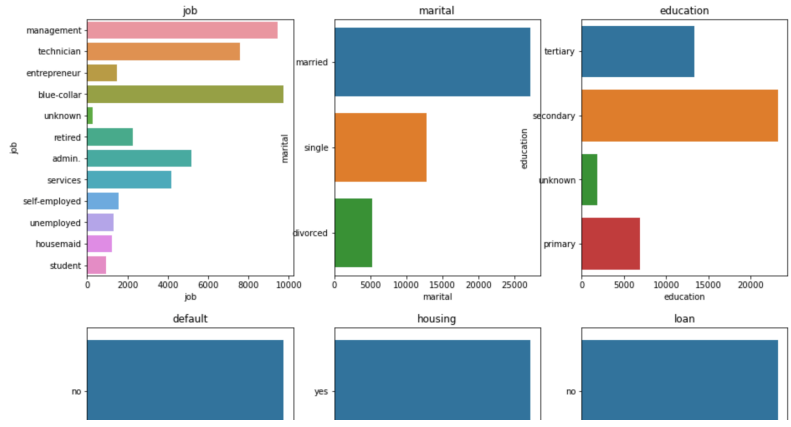
```
In [9]:
#finding variables with unique values
for column in df.columns:
    print(column,df[column].nunique())
```

```
age 77
job 12
marital 3
education 4
default 2
balance 7168
housing 2
loan 2
```

Further we could check other aspects of the dataset as well, such as the unique values and the total number of unique values within the dataset as shown above.

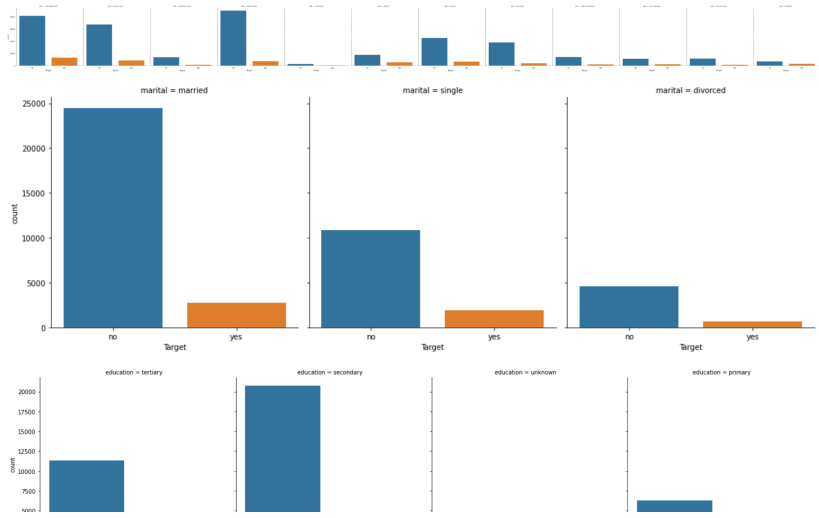
## 1.2 Data Visualization

```
In [12]:
#checking categorical variables distribution
plt.figure(figsize=(15,80), facecolor='white')
plotnumber =1
for categorical_variable in categorical_variables:
    ax = plt.subplot(12,3,plotnumber)
    sns.countplot(y=categorical_variable,data=df)
    plt.xlabel(categorical_variable)
    plt.title(categorical_variable)
    plotnumber+=1
plt.show()
```



We could also visualize the distribution of the categorical variables in the dataset to better understand the data using the above code shown. This shows the various distributions using bar plots.

```
In [13]:
#Find out the relationship between categorical variable and dependent variable
for categorical_variable in categorical_variables:
    sns.catplot(x='Target', col=categorical_variable, kind='count', data= df)
plt.show()
```



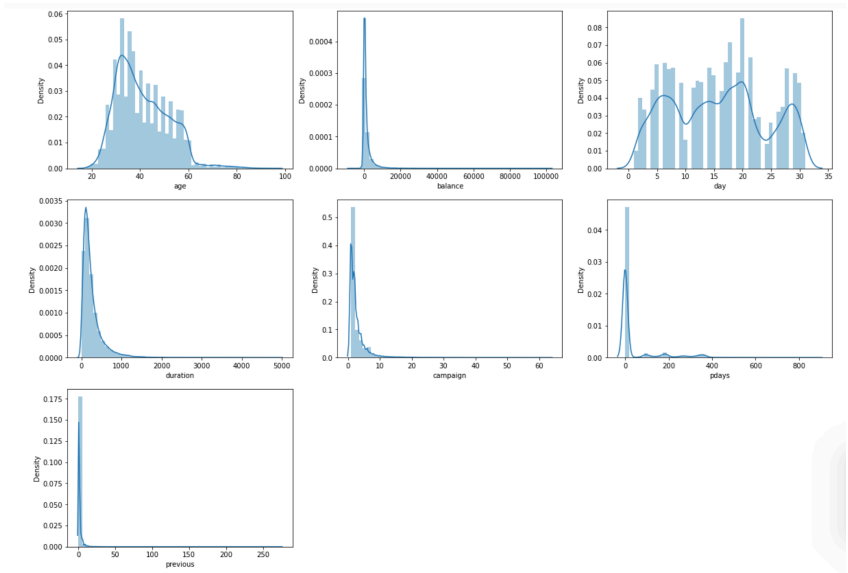

We could also check the relationship of the categorical variable against the dependent variable, which is ‘Target’ in this case. This would enable us to further understand how it’s distributed among the dependent variable and its relationship. The graphs are better visualized in the python file.

```
In [25]:
#finding outliers in numerical features using boxplot
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for numerical_feature in numerical_variables:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(df[numerical_feature])
    plt.xlabel(numerical_feature)
    plotnumber+=1
plt.show()
```

Here we are plotting a boxplot to spot the outliers in all the numerical variables/features in the dataset. Features in this context also means variables or values.

```
In [20]:
#plot a univariate distribution of continues observations
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for continuous_feature in continuous_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.distplot(df[continuous_feature])
    plt.xlabel(continuous_feature)
    plotnumber+=1
plt.show()
```

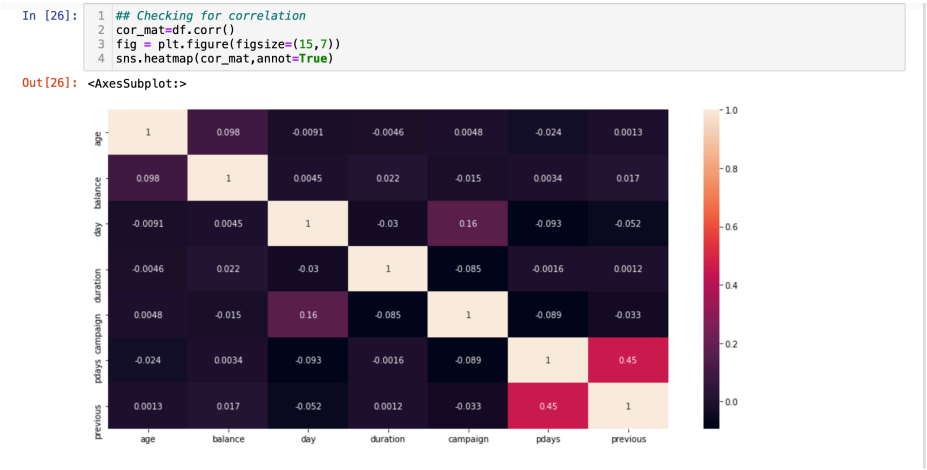
The above code is used to plot the univariate distributions of continuous numerical observations. The detailed graphs can be viewed in the python file uploaded along with this.




You could see the various distributions among various variables in the above image. The variables used in the above image to visualize the graphs are age, balance, day, duration, campaign, pdays and previous. From the above image, we could take away some key information, for instance, the balance, duration, compaign, pdays and previous are heavily skewed towards left. This could indicate that there seems to be some outliers in these variables. In the next step we will be further looking into it and remove the unnecessary values to help the model predict better. However the day variable seems to have distributed normally.

# Step 2: Correlation

Before we move into the Data cleaning section, we need to check whether any of the variable in the dataset has any significant correlation between each other. To conduct this, we could use a heat map to better understand it as it's visualized instead of providing raw numbers.




The above image shows the code and the heat map as well. We could take away the fact that there isn't a major correlation between the variables in this dataset. The darker colors mean that there isn't a correlation between the two variables, and when it is lighter it means there is.

# Step 3: Data Pre-Processing and Cleaning

## 3.1 Removing Outliers

This is a crucial step when it comes to creating a predictive model. Since we identified a bunch of outliers using box plots, we could remove some of them by analyzing each variable individually.

```
In [32]: #default features does not play imp role
         df2.groupby(['Target','default']).size()

Out[32]: Target  default
         no      no         39159
                 yes          763
         yes     no          5237
                 yes           52
         dtype: int64

In [33]: df2.drop(['default'],axis=1, inplace=True)
```

As shown in the image above, the 'groupby' function shows us how many values of the 'default' variable are there in 'Target'. In this case, the 'default' variable does not play an important role in this predictive model. As you can see above, we dropped the 'default' variable in the following code using the 'drop' function.

```
In [34]: df2.groupby(['Target','pdays']).size()

Out[34]: Target  pdays
         no      -1       33570
                  1           9
                  2          35
                  3           1
                  4           1
                            ...
         yes      804         1
                  805         1
                  828         1
                  842         1
                  854         1
         Length: 914, dtype: int64

In [35]: # drop pdays as it has -1 value for around 40%+
         df2.drop(['pdays'],axis=1, inplace=True)
```

Next, we check the other variable 'pdays', which has a negative number which is -1. And there are around 33,570 values of 'Target'; 'no' variable. This will be useless to our prediction model so therefore we could go ahead and drop the entire variable from our dataset.

```
In [36]:
# remove outliers in feature age...
df2.groupby('age',sort=True)['age'].count()
# these can be ignored and values lies in between 18 to 95

Out[36]:
age
18    12
19    35
20    50
21    70
```

The ‘Age’ variable seems alright despite there are 1 or 2 outliers since it could still be used to predict the term deposit value. Therefore, we won’t be dropping this variable and we will be proceeding ahead with all the other variables.

```
In [39]:
# remove outliers in feature campaign...
df2.groupby(['Target','campaign'],sort=True)['campaign'].count()

Out[39]:
Target  campaign
no      1           14983
        2           11104
        3            4903
        4            3205
        5            1625
                    ...
yes     20              1
        21              1
        24              1
        29              1
        32              1
Name: campaign, Length: 70, dtype: int64

In [40]:
df3 = df2[df2['campaign'] < 33]

In [41]:
df3.groupby(['Target','campaign'],sort=True)['campaign'].count()
```

The ‘campaign’ variable has values more than 32 in the ‘Target’ variable under ‘no’. whereas most values are within the range of less then 32. Therefore, we could create a new dataframe which includes ‘campaigns’ of less than 32 and not more than that since they are the outliers and won’t really help with the prediction model.

Thereafter, we could re-check it again after filtering the values by re-running the same ‘groupby’ ‘count’ function to the new dataframe.

```
In [42]:
# remove outliers in feature previous...
df3.groupby(['Target','previous'],sort=True)['previous'].count()

Out[42]: Target  previous
no      0           33532
        1            2189
        2            1650
        3             848
        4             543
                     ...
yes     26              1
        29              1
        30              1
        55              1
        58              1
Name: previous, Length: 66, dtype: int64

In [43]:
df4 = df3[df3['previous'] < 31]
```

In this above screenshot, we have taken the 'previous' variable to sort out the outliers. In this case, there were some values above 31 inside the 'Target' variable under the 'no' category. Since they weren't available under the 'yes' category, we decided to filter them out by creating a new dataframe with all values less than 31 in the 'previous' variable.

## 3.2 Creating Dummy Variables

```
In [44]:
cat_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']
for col in  cat_columns:
    df4 = pd.concat([df4.drop(col, axis=1),pd.get_dummies(df4[col], prefix=col, prefix_sep='_',drop_first=True,

In [45]:
bool_columns = ['housing', 'loan', 'Target']
for col in  bool_columns:
    df4[col+'_new']=df4[col].apply(lambda x : 1 if x == 'yes' else 0)
    df4.drop(col, axis=1, inplace=True)

In [46]:
df4.head()
```

For the final data pre-processing and cleaning process, we should create dummy variables and columns for all the columns in the dataset. To do this, we've used the following code shown above, to create the dummy variables to the categorical columns and concatenate them with the original dataframe, which is done in the 'cat_columns'. The 'bool_columns' function is used to create dummy variables for the Boolean value variables, which in this case is 'yes' or 'no' values. This process is mainly done to change the categorical variables to numerical variables which is suitable for machine learning algorithms which requires numeric input.

Furthermore, with the 'head()' function we could check whether the dummy values have been created and the values are all being converted to numerical values.

# Step 4: Model Building and Conclusion

## 4.1 Splitting the dataset into Training and Test Sets

Splitting the dataset into the training set and test set is a crucial step when it comes to machine learning. This is mainly done to evaluate the accuracy or the performance of the machine learning model on unseen data. This also prevents overfitting in the machine learning model. This is where the model predicts accurate information for the existing data but not for the additional new data.

```
In [47]:
X = df4.drop(['Target_new'],axis=1)
y = df4['Target_new']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=0)

In [48]:
len(X_train)
Out[48]: 36128

In [49]:
len(X_test)
Out[49]: 9033
```

The above code is used to split the dataset into 4 sets which is, X_train, X_test, y_train, and y_test. X variable which is the independent variable includes all the variables of the dataframe ‘df4’ whereas Y variable, the dependent variable, includes only the ‘Target_new’ variable in the df4 dataframe. The ‘test_size’ parameter is used to set the test size percentage of the dataset, which is 20% in this case, which means 80% of the dataset is in the training set. The ‘len’ function shows us how many total values are in the X_train set and the X_test set.

## 4.2 Model Selection

To pick the best model from a group of possible models for a particular task, model selection is a crucial step in machine learning. The chosen model should be able to make precise predictions and generalize effectively to new data.

```
In [52]:
from sklearn.model_selection import cross_val_score
model_score =cross_val_score(estimator=XGBClassifier(),X=X_train, y=y_train, cv=5)
print(model_score)
print(model_score.mean())

[0.90783283 0.90617216 0.9017437  0.90906574 0.90311419]
0.9055857232197839
```

For this project, XGBClassifier seems to be a suitable model to use to predict the term deposits. In this instance, we have used the cross-validation score method to check the accuracy of the model's performance. This is imported via the scikit-learn library and it computes the cross validated scores for XGB Classifier using the training data (X_train and y_train). CV code means it's going to execute the code using 5-fold cross-validation.

The 'model_score' function gives us all the 5 results whereas the 'model_score.mean'gives us the mean score from the 5 fold cross-validation.

The average accuracy of the XGBoost classifier model, as measured using 5-fold cross-validation on the training data, is roughly 90.56%, according to the mean cross-validated score of about 0.9056.

The model's performance on the training set during cross-validation is represented by this score. It gives an estimate of the model's potential performance on hypothetical data. A higher cross-validated score often denotes a model with more generalization and predictive power.

Since there is a higher cross-validated score, we will proceed ahead with this model.

## 4.3 Model Building

In machine learning, the process of generating a predictive model using a given dataset is referred to as model building. In this case, we will be utilizing XGB Classifier model since it's accuracy score is 90.56%. More information about the XGB Classifier is further discussed below.

XGBoost Classifier comes from the gradient boosting technique family which includes machine learning models. Due to its strong capabilities and effective performance, it has significantly increased in popularity in a variety of machine learning challenges and real-world applications.

At its foundation, XGBoost creates a strong learner capable of precisely predicting the target variable by combining the predictions of several weak learners, often decision trees. It uses the collective intelligence of a group of weak learners who are progressively trained to increase the overall forecast accuracy.

The use of Extreme Gradient Boosting is one of the main characteristics of XGBoost. This improvement uses a number of optimization and regularization approaches to speed up and improve the process. XGBoost delivers improved performance and generalization abilities by combining gradient boosting with advanced regularization techniques.

As weak learners, decision trees are fundamental to XGBoost. These trees are constructed by building a hierarchical structure by recursively splitting the data depending on feature values. By maximizing a certain loss function, such as log-loss for classification or mean square error for regression, the computer learns to make judgments.

XGBoost applies a gradient descent optimization technique to the model parameters. It use both first-order derivatives (gradients) and second-order derivatives (Hessian) to iteratively

update the model, choosing the best step size and direction. Faster convergence and effective model updates are guaranteed by this method.

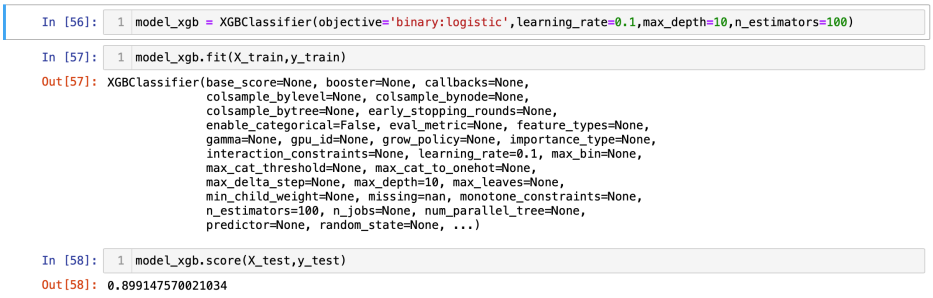
Regularization methods are also used in XGBoost to reduce overfitting and improve generalization. The model may manage complexity and lessen variation by adjusting variables including learning rate, tree depth, subsampling ratios, and regularization terms. To get the best performance, it is essential to be able to balance model complexity with generalization.

The capacity of XGBoost to offer insights regarding feature relevance is a noteworthy feature. Each feature's gain or improvement in the loss function is calculated by the model, and it then ranks the features in accordance with that result. This feature significance analysis may help in the selection of features, in determining the factors that have the greatest influence, and in obtaining understanding of the underlying relationships in the data.

Additionally, XGBoost is intended to be extremely scalable and able to handle enormous datasets. It uses multi-threading, takes use of parallel computing, and may be deployed over several machines. This scalability makes sure that the model can handle huge data processing scenarios efficiently without sacrificing speed.

With the large variety of configurable parameters provided by XGBoost, customers may adapt the model to their own requirements. To adapt to different problem domains, it also offers possibilities for customized assessment metrics and loss functions.

Overall, the XGBoost Classifier is a robust and adaptable machine learning model, to sum up. With regard to a variety of machine learning tasks, it is the go-to option due to its superior performance, speed, and scalability. XGBoost routinely produces cutting-edge results for classification, regression, or ranking issues and has established itself as a crucial tool for data scientists.




The above image shows us the model building code for this project of predicting the term deposit subscriptions. We have used the XGBClassifier class from the XGBoost library to build a gradient boosting model which, as discussed above, is the most suitable model for binary classification problems.

The goal function that will be optimized throughout the training phase is specified by the "objective='binary:logistic'" option. The model will maximize the logistic loss for binary classification in this scenario as it is set to "binary:logistic."

'learning_rate=0.1': Each boosting iteration's step size is based on the learning rate. The model becomes more cautious by making smaller moves when the learning rate is lower.

The maximum depth of each tree during the boosting process is controlled by the option 'max_depth=10'. A greater number increases the likelihood of overfitting while also allowing the model to capture more intricate interactions.

'n_estimators=100' specifies how many decision trees or boosting iterations will be created. Although adding more estimators often enhances model performance, it also lengthens calculation time.

These are the parameters set up to perform the binary logistic regression via XGBoost.

Furthermore, the XGBClassifier model ‘model_xgb’ is fitted (trained) based on the training data X_train and associated target labels y_train using the ‘model_xgb.fit(X_train, y_train)’ function.

In order to generate predictions on fresh, unobserved data, the model learns the basic trends and correlations in the training data throughout the fitting phase. An ensemble of decision trees is built iteratively via the XGBoost method, with each tree learning to fix the errors committed by the preceding trees. The provided hyperparameters and the stated objective function (in this example, binary logistic loss) are used to optimize the model.

The ‘model_xgb.score(X_test, y_test)’ function determines the model_xgb's accuracy on the basis of the test data X_test and associated target labels y_test.

The ‘score()’ function compares the model's predictions on the test data with the actual labels in order to assess the model's performance. It gives back accuracy, which is the percentage of accurate predictions among all samples.

In this case, the accuracy percentage returned as 89.91%. Better performance is indicated by a higher accuracy since it means the model is producing more accurate predictions where in this instance, 89.91% seems to be a pretty good accuracy rate.

### 4.4 Confusion Matrix

A table called a confusion matrix is frequently used to assess how well a classification model is working. It lists the model's predictions on a set of test data and contrasts them with the true labels.

Four key elements make up a confusion matrix:

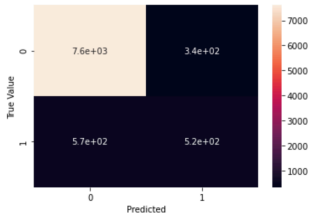
- True Positives (TP): The number of occurrences of positivity that the model accurately identified as positive.
- True Negatives (TN): The number of occurrences of negativity that the model properly identified as negativity.
- False Positives (FP): The number of negative occurrences that the model misinterpreted as positive.
- False Negatives (FN): The proportion of positive cases that the model misinterpreted as negative.

When it comes to successfully and erroneously categorizing cases, the model's performance is clearly displayed via the confusion matrix. It is used to determine several assessment measures including accuracy, precision, recall, and F1 score.

```
In [60]:
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,model_xgb.predict(X_test))
cm

Out[60]: array([[7606,  344],
                [ 567,  516]])

In [61]:
#plot the graph
from matplotlib import pyplot as plt
import seaborn as sn
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('True Value')
plt.show()
```



As shown above, the confusion matrix is calculated using the predicted labels and real labels using the ‘confusion_matrix()’ function from sklearn.metrics. When using the test data X_test and the accompanying true labels y_test, the code ‘cm = confusion_matrix(y_test, model_xgb.predict(X_test))’ creates the confusion matrix for the XGBClassifier model model_xgb.

Interpreting the confusion matrix's values from the above image:

- 7606 True Negatives (TN)
- 344 False positives (FP)
- 567 False Negatives (FN)
- 516 True Positives (TP)

The following is suggested by this confusion matrix:

- 7606 cases belonging to the negative class (true negatives) were properly predicted by the model.
- 344 occurrences were falsely classified as positive when the model should have indicated that they belonged to the negative class (false positives).
- 567 occurrences were falsely predicted by the model as being in the negative class when they were really in the positive class (false negatives).
- 516 cases belonging to the positive class (true positives) were properly predicted by the model.

You could evaluate the accuracy, precision, recall, and other assessment parameters of the model's performance in greater detail using the confusion matrix.

The confusion matrix could also be visualized using a heat map. Each colored column in the confusion matrix's grid reflects the number or percentage of predicted labels for a certain set of actual and predicted values.

## 4.5 Conclusion

These findings make it clear that the model's overall performance is fairly strong, with a high accuracy score. To assess the impact of false positives and false negatives, it is crucial to examine the unique criteria and objectives of the classification problem. Further enhancing the performance can involve adjusting the model's parameters or experimenting with various methods.

Overall, the confusion matrix's insights and the accuracy score show how well the XGBoost classifier performs at predicting the target variable, but further research and testing may be required to fully comprehend the model's efficiency.

# References


- Chen, T. and Guestrin, C., 2016. XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794). ACM.
- XGBoost Documentation. [Online]. Available at: https://xgboost.readthedocs.io/ [30th May 2023].
- Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), pp.1189-1232.