Assignment

on

Customer Churn Prediction

Submitted to:
UniAcco
Data Science Internship

Submitted by:
Shikha Pathak

Table of Contents

Chapter 1

1.1 Customer Churn Prediction

Churn prediction is the process of identifying which customers are likely to terminate their service or subscription. This is a vital prediction for many businesses, as retaining existing customers is usually more cost-effective than acquiring new ones. To maximize the chances of keeping a customer, it is important to understand which marketing actions should be taken for each individual customer. As customers have different behaviours and preferences, they may cancel their subscriptions for a variety of reasons. Therefore, it is essential to engage proactively with each customer to try to retain them. Knowing the right marketing action to take for each customer, and when to take it, is key to success in customer retention.

In this assignment a dataset is provided which inclues the data of bank's customers.

1.2 Data Description

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution.

The attributes in the dataset are given below:

1. **age:** Age of the customer (numeric)
2. **job:** type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
3. **marital :** marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed)
4. **education:** Education level (categorical: 'primary', 'secondary', 'tertiary', 'unknown')
5. **default:** has credit in default? (categorical: 'no','yes','unknown')
6. **balance:** has housing loan? (numeric: outstanding amount)
7. **housing:** has housing loan? (categorical: 'no','yes','unknown')
8. **loan:** has personal loan? (categorical: 'no', 'yes', 'unknown')
9. **contact:** contact communication type (categorical: 'cellular','telephone')
10. **day:** last contact day of the month (numeric: day of the month)
11. **month:** last contact month of year (categorical: 'jan', 'feb', 'mar', …, 'nov', 'dec')
12. **duration:** last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, 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 if the intention is to have a realistic predictive model.
13. **campaign:** number of contacts performed during this campaign and for this client (numeric, includes last contact)
14. **pdays:** number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
15. **previous:** number of contacts performed before this campaign and for this client (numeric)
16. **poutcome:** outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')
17. **y:** has the client subscribed a term deposit? (binary: 'yes','no')**[Target Variable]**

1.3 Problem Statement

The problem is to create a predictive model that can predict the customer churn which will subscribe to the term deposit. It’s a classification problem i.e., whether the customer would continue subscribing the ter deposit or not. Prediction here is to predict the value of output variable ***‘y’***.

Chapter 2

2.1 Python

Python is an interpreted, high-level, general-purpose programming language. It was created by Guido van Rossum and first released in 1991. Python has a design philosophy which emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It is used in many application domains, including web and internet development, scripting, scientific computing, and artificial intelligence.

Environment used: Jupyter Notebook

Jupyter Notebook is an open-source web application and python IDE that allows you to create and share documents that contain live code, equations, visualizations and narrative text. It is used in data science, machine learning and scientific computing. It supports many programming languages such as Python, R, Julia, Scala and Haskell. It is a great tool for data exploration, prototyping and data analysis.

2.2 Libraries Used

Python offers a wide number of libraries which can perform various function as per the user requirement. In this project following libraries will be used:

I. **pandas** – Powerful Python library for data analysis and manipulation primarily using dataframes.
II. **numpy** – Used for efficient scientific computing in Python, providing powerful data structures and functions for fast numerical calculations.
III. **matplotlib** – Primarily used fro 2D plots and visualizations.
IV. **seaborn** – Used for making attractive and informative statistical graphics in Python with strong customisations.
V. **warnings** – It provides a way to handle warnings as exceptions, making it easier to write programs that respond to warnings.
VI. **sklearn** – Scikit-learn offers efficient tools for data analysis and predictive modeling, making it a great choice for any data science project.
VII. **xgboost** – It is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.
VIII. **scipy** – It is an incredibly powerful and versatile library for scientific computing in Python, providing a wide range of advanced mathematical tools for data analysis and visualization

Chapter 3

3.1 Idea

The idea is to use the powerful python libraries to meet the objective and create a model that will effectively predict the customer churn that are going to subscribe to a term deposit or not based on the data previously feeded to the ML model. The classification technique that will be used here is Gradient Boosting.

3.2 Steps Involved

Steps involved to meet the objective are the common steps to build any ML model for prediction:
Step 1: Data Exploration – Understanding the data based on central tendencies and visualisation for a better understanding of the features.

Step 2: Data Cleaning – Removing noise from the data is very essential before building any model to avoid underfitting and overfitting, hence appropriate cleaning is done on the dataset by removing the unnecessary and insignificant features from the dataset. It also involves filling the unknown values in significant features(columns).

Step 3: Data Preprocessing – After the data is cleaned it is stil not ready for model building, all the variables are needed to be converted to numerical type for model estimation. Hence, different techniques like ***One Hot Encoding*** is used to convert categorical variables to numerical values.

Step 4: Model Building – On the processed data ML model can be build easily, first by splitting the data into training and testing data and then fitting the model based on any kind of classification techique. Here I have used the Gradient Boosting Classifier.
Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.

Step 5: Best Parameter evaluation – to improve the accuracy of any model it becomes essential to provide the technique(classifier used) with best parameters. Hence, a set of values of parameters is provided to the ***RandomizedSearchCV*** function to get the best parameters by ***'fit'*** & ***'score'*** method. These parametrs vary upon the method used for model building.

Step 6: Final model and Performance Metric – Based on the best parameters obtaioned in the last step, the model is fitted once again to get better and more accurate predictions. And the performance of the model is evluated using various measures such as accuracy, precision, recall, F1-score and AUC-ROC.

Chapter 4

4.1 Describing and Visualising the data

Firstly, we will explore the data and visualize essential items of the dataset
Importing the libraries required for the programming:

```
#Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings
```

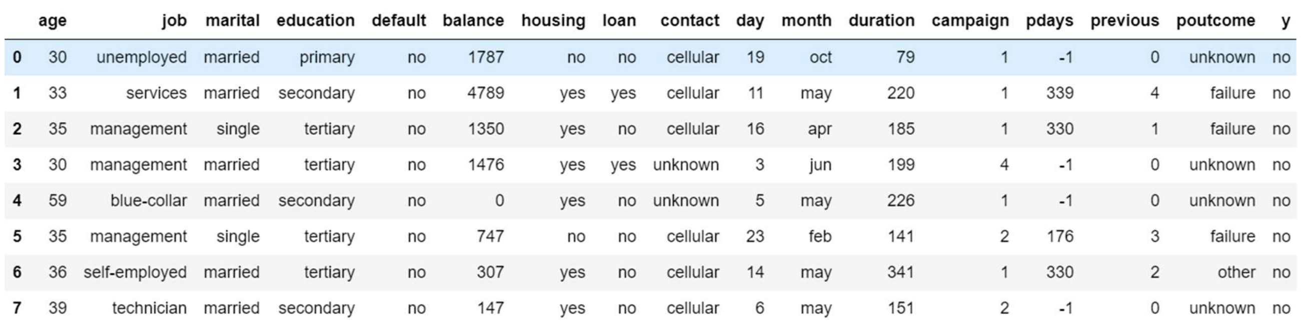
Then importing the dataset in Python:

```
df = pd.read_csv("bank.csv", sep=";")
#ensure that the file is saved in the working directory
```

Viewing first few observations:

```
df.head(8)
```

Output:



	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no
5	35	management	single	tertiary	no	747	no	no	cellular	23	feb	141	2	176	3	failure	no
6	36	self-employed	married	tertiary	no	307	yes	no	cellular	14	may	341	1	330	2	other	no
7	39	technician	married	secondary	no	147	yes	no	cellular	6	may	151	2	-1	0	unknown	no

Describing the dataset with shape, central tendencies and information:

```
df.shape
```

Output:

```
(4521, 17)
```

```
df.describe()
```

Output:

	age	balance	day	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4521 non-null   int64 
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64 
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64 
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64 
 12  campaign   4521 non-null   int64 
 13  pdays      4521 non-null   int64 
 14  previous   4521 non-null   int64 
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

Checking for null and duplicated values:

```
df.isnull().sum()
```

Output:

```
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
dtype: int64
```

There are no null values in the dataset.

```
df.duplicated().sum()
```

Output:

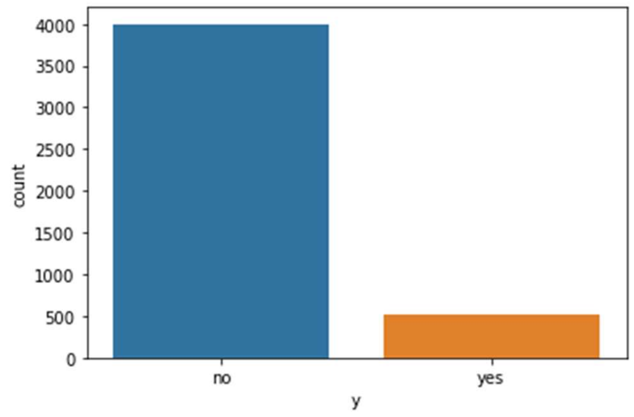
```
0
```

There are no duplicated values in the dataset.

Plotting the target variable ***"y"*** which defines wheter a cutomer has subscribed to a term deposit or not:

```
sns.countplot(x='y', data=df)
```

Output:



Checking the imbalancy in the dataset i.e., what percent of target values are ***yes*** or ***no***:

```
y_im = 100*df['y'].value_counts()/len(df['y'])
y_im
```

Output:

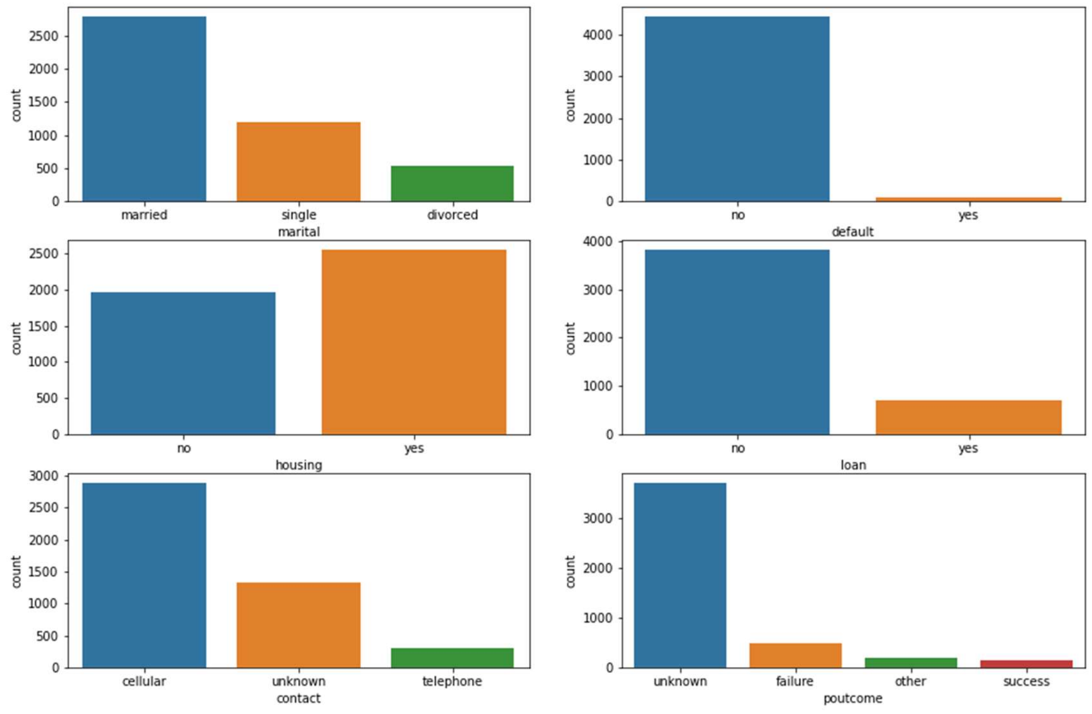
```
no     88.476001
yes    11.523999
Name: y, dtype: float64
```

Plotting all the variables in the dataset to get an idea of their significance.

Firstly, all categorical variables:

```
fig, axes = plt.subplots(3,2,figsize=(15,10))
sns.countplot(df['marital'],ax=axes[0,0])
sns.countplot(df['default'],ax=axes[0,1])
sns.countplot(df['housing'],ax=axes[1,0])
sns.countplot(df['loan'],ax=axes[1,1])
sns.countplot(df['contact'],ax=axes[2,0])
sns.countplot(df['poutcome'],ax=axes[2,1])
```

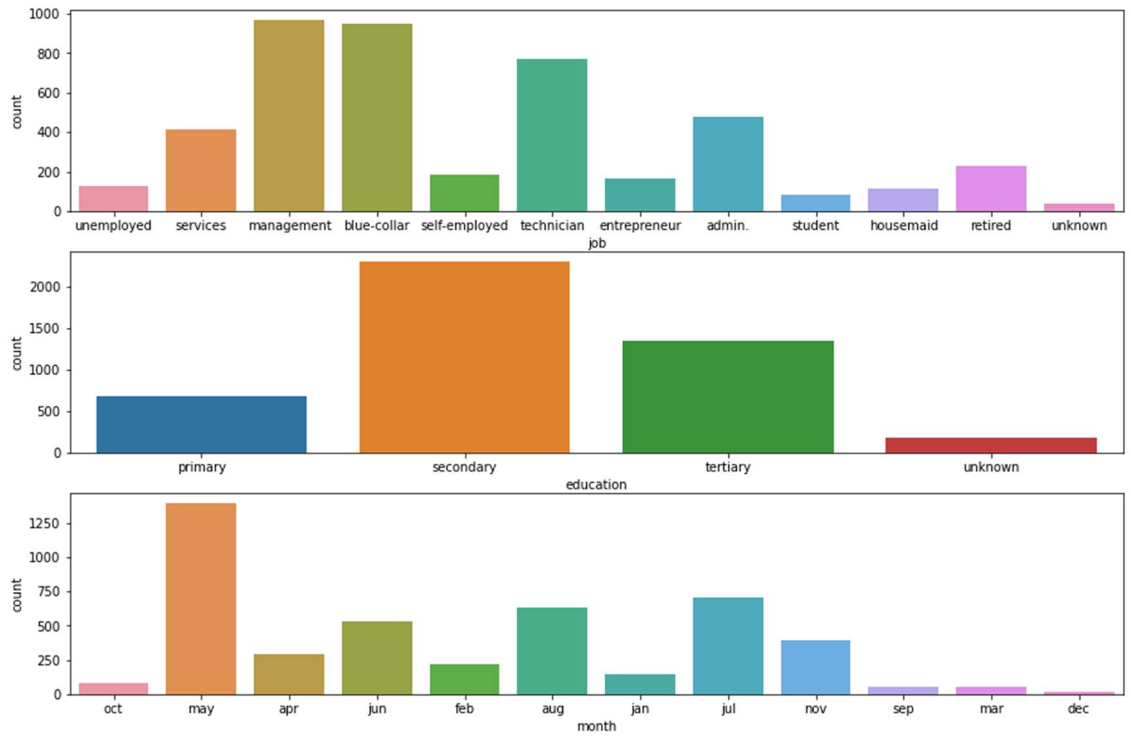
Output:



Now, all the ordinal variables:

```
fig, axes = plt.subplots(3,1,figsize=(15,10))
sns.countplot(df['job'],ax=axes[0])
sns.countplot(df['education'],ax=axes[1])
sns.countplot(df['month'],ax=axes[2])
```

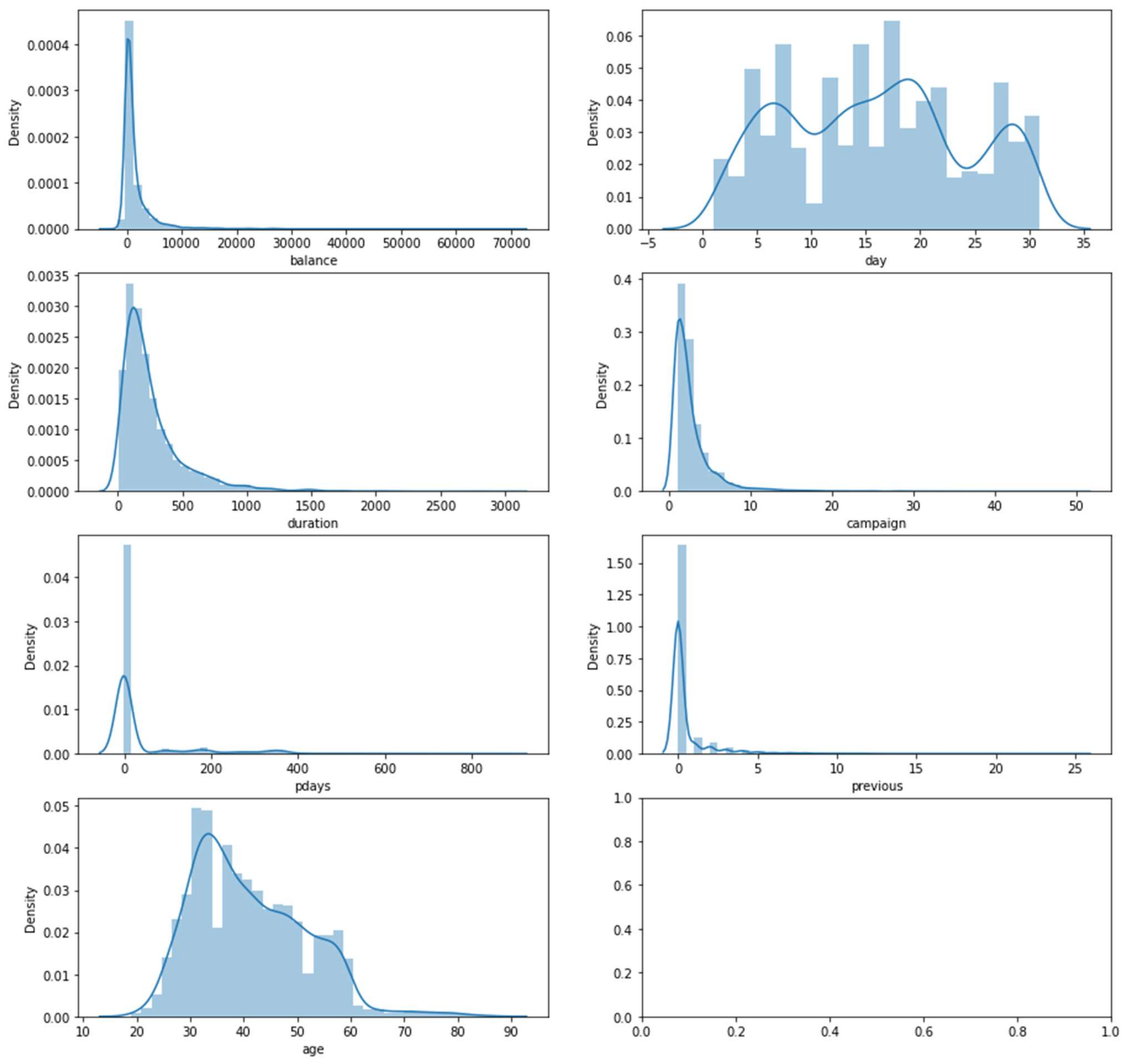
Output:



At last, plotting numerical variables with their distrbution:

```
fig, axes = plt.subplots(4,2,figsize=(15,15))
sns.distplot(df['balance'],ax=axes[0,0])
sns.distplot(df['day'],ax=axes[0,1])
sns.distplot(df['duration'],ax=axes[1,0])
sns.distplot(df['campaign'],ax=axes[1,1])
sns.distplot(df['pdays'],ax=axes[2,0])
sns.distplot(df['previous'],ax=axes[2,1])
sns.distplot(df['age'],ax=axes[3,0])
```

Output:



4.2 Data Cleaning

Now, proceeding with the data cleaning process:

Initially we will create a copy of the dataset to avoid any kind of loss:

```
bank_df = df.copy()
bank_df.head()
```

Output:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no

Now removing the unnecessary columns from the dataset i.e., the ***'day**'* and ***'month'*** columns can be omitted since the ***'pdays'*** column provides the number of days that have elapsed since the client's last contact from a prior campaign.

```
bank_df.drop(['day','month'], axis = 1, inplace = True)
```

Replacing all the ***'unknown'*** values in the dataset by ***'nan'*** and checking the number of those values in each column as it will be easier for the compiler to understand the data for further processing:

```
for i in bank_df.columns:
    bank_df[i] = np.where(bank_df[i] == "unknown", np.nan, bank_df[i])
bank_df.isna().sum()
```

Output:

```
age            0
job           38
marital        0
education    187
default        0
balance        0
housing        0
loan           0
contact     1324
duration       0
campaign       0
pdays          0
previous       0
poutcome    3705
y              0
dtype: int64
```

Also removing the ***'contact'*** & ***'poutcome'*** column as they have too many unknown values and these features are of littile to no signinficance for the churn prediction:

```
bank_df.drop("poutcome", inplace = True, axis = 1)
bank_df.drop("contact", inplace = True, axis = 1)
bank_df.head()
```

Output:

	age	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	y
0	30	unemployed	married	primary	no	1787	no	no	19	oct	79	1	-1	0	no
1	33	services	married	secondary	no	4789	yes	yes	11	may	220	1	339	4	no
2	35	management	single	tertiary	no	1350	yes	no	16	apr	185	1	330	1	no
3	30	management	married	tertiary	no	1476	yes	yes	3	jun	199	4	-1	0	no
4	59	blue-collar	married	secondary	no	0	yes	no	5	may	226	1	-1	0	no

Checking for left over **NaN** values and filling them ***'ffill'*** method provided by ***'pandas'*** library:

```
bank_df.isna().sum()
```

Outcome:

```
age            0
job           38
marital        0
education    187
default        0
balance        0
housing        0
loan           0
duration       0
campaign       0
pdays          0
previous       0
y              0
dtype: int64
```

```
bank_df["job"].fillna(method = "ffill",inplace=True)
bank_df["education"].fillna(method = "ffill",inplace= True)
bank_df.isna().sum()
```

Output:

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
duration     0
campaign     0
pdays        0
previous     0
y            0
dtype: int64
```

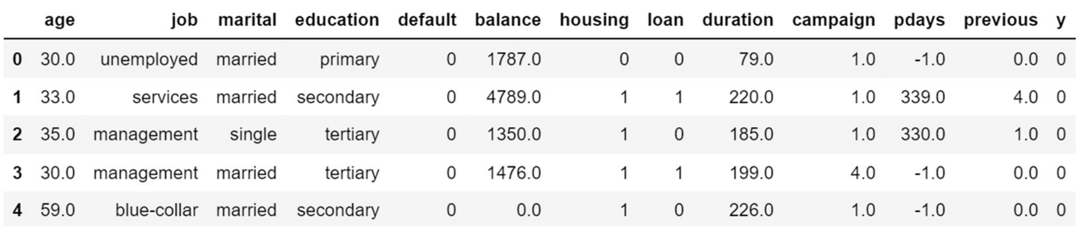
4.3 Data Preprocessing

Now proceeding with the data preprocessing part for model building as all the categorical and ordinal variable are needed to be converted to numeric before buildinga model.

Starting with the binary(***yes*** or ***no***) categorical variables and converting them to numeric by assigning ***1*** or ***0*** value to them:

```
bank_df['y']= bank_df['y'].replace(['yes','no'],[1, 0])
bank_df['default']= bank_df['default'].replace(['yes','no'],[1, 0])
bank_df['housing']= bank_df['housing'].replace(['yes','no'],[1, 0])
bank_df['loan']= bank_df['loan'].replace(['yes','no'],[1, 0])
bank_df.head()
```

Output:



	age	job	marital	education	default	balance	housing	loan	duration	campaign	pdays	previous	y
0	30.0	unemployed	married	primary	0	1787.0	0	0	79.0	1.0	-1.0	0.0	0
1	33.0	services	married	secondary	0	4789.0	1	1	220.0	1.0	339.0	4.0	0
2	35.0	management	single	tertiary	0	1350.0	1	0	185.0	1.0	330.0	1.0	0
3	30.0	management	married	tertiary	0	1476.0	1	1	199.0	4.0	-1.0	0.0	0
4	59.0	blue-collar	married	secondary	0	0.0	1	0	226.0	1.0	-1.0	0.0	0

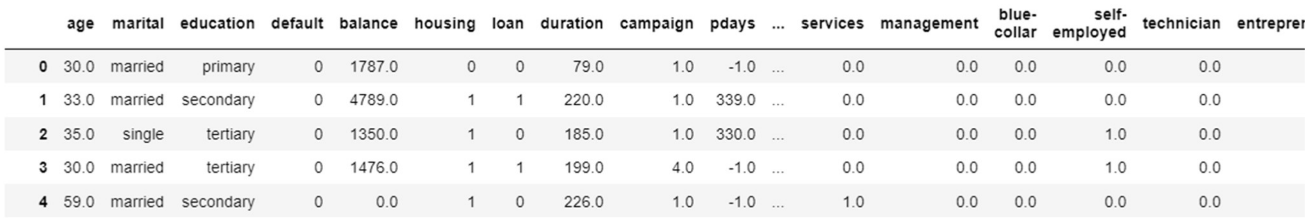
Now, moving ahead with the ***'job'*** column. As it has more then 2 categories 11 to be specific. ***One Hot Encoding*** will be used here

One-hot encoding is the process by which categorical data are converted into numerical data for use in machine learning.
Categorical features are turned into binary features that are "one-hot" encoded, meaning that
if a feature is represented by that column, it receives a 1. Otherwise, it receives a 0.

OneHotEncoder library will be used and ***fit_transform*** attribute will be used to create and add new columns to the dataset with unique values from the ***'job'*** columns in the dataset and assigning ***0*** or ***1*** to the respective observation as:

```
from sklearn.preprocessing import OneHotEncoder
one = OneHotEncoder()
bank_df[list(bank_df["job"].unique())] = one.fit_transform(bank_df[["job"]]).A
bank_df.drop("job",axis = 1, inplace = True)
bank_df
```

Output:

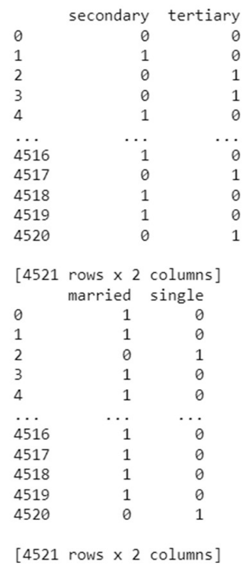


	age	marital	education	default	balance	housing	loan	duration	campaign	pdays	...	services	management	blue-collar	self-employed	technician	entreprer
0	30.0	married	primary	0	1787.0	0	0	79.0	1.0	-1.0	...	0.0	0.0	0.0	0.0	0.0	
1	33.0	married	secondary	0	4789.0	1	1	220.0	1.0	339.0	...	0.0	0.0	0.0	0.0	0.0	
2	35.0	single	tertiary	0	1350.0	1	0	185.0	1.0	330.0	...	0.0	0.0	0.0	1.0	0.0	
3	30.0	married	tertiary	0	1476.0	1	1	199.0	4.0	-1.0	...	0.0	0.0	0.0	1.0	0.0	
4	59.0	married	secondary	0	0.0	1	0	226.0	1.0	-1.0	...	1.0	0.0	0.0	0.0	0.0	

Creating dummy dataframes of ***'education'*** & ***'marital'*** columns' unique values and dropping the first value as it will be an obvious observation based on the remaining 2 i.e., ***'0'*** if there is ***'1'*** in any of the remaining values and ***'1'*** if both the remaining values are ***'0'.*** As:

```
education_dum = pd.get_dummies(bank_df['education'],drop_first=True)
marital_dum = pd.get_dummies(bank_df['marital'],drop_first=True)
print(education_dum)
print(marital_dum)
```

Output:

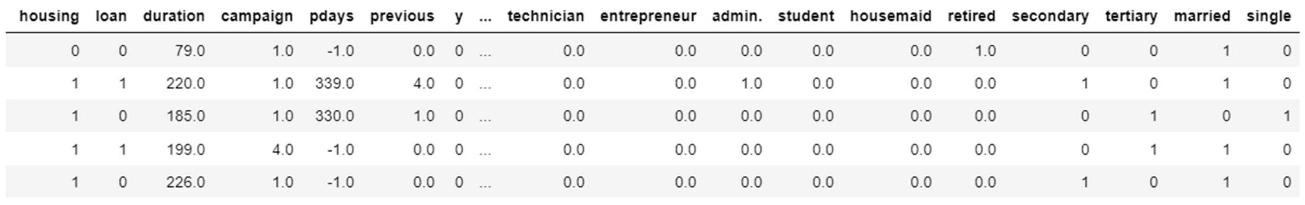


```
      secondary  tertiary
0             0         0
1             1         0
2             0         1
3             0         1
4             1         0
...         ...       ...
4516          1         0
4517          0         1
4518          1         0
4519          1         0
4520          0         1

[4521 rows x 2 columns]
      married  single
0           1       0
1           1       0
2           0       1
3           1       0
4           1       0
...       ...     ...
4516        1       0
4517        1       0
4518        1       0
4519        1       0
4520        0       1

[4521 rows x 2 columns]
```

Concatinating the dummy dataframes with main dataset and dropping the ***'education'*** & ***'marital'*** columns as they are no longer needed:

```
bank_df=pd.concat([bank_df,education_dum,marital_dum],axis=1)
bank_df.drop("marital",axis = 1, inplace = True)
bank_df.drop("education",axis = 1, inplace = True)
bank_df
```

Output:



housing	loan	duration	campaign	pdays	previous	y	...	technician	entrepreneur	admin.	student	housemaid	retired	secondary	tertiary	married	single
0	0	79.0	1.0	-1.0	0.0	0	...	0.0	0.0	0.0	0.0	0.0	1.0	0	0	1	0
1	1	220.0	1.0	339.0	4.0	0	...	0.0	0.0	1.0	0.0	0.0	0.0	1	0	1	0
1	0	185.0	1.0	330.0	1.0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0	1	0	1
1	1	199.0	4.0	-1.0	0.0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	0	1	1	0
1	0	226.0	1.0	-1.0	0.0	0	...	0.0	0.0	0.0	0.0	0.0	0.0	1	0	1	0

The obtained data frame completely consists of numerical values and a ML model can be easily created from this for predicting customer churns.

4.4 Model Builiding

Moving ahead with model building and importing necesssary libraries as:

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from xgboost import XGBClassifier
```

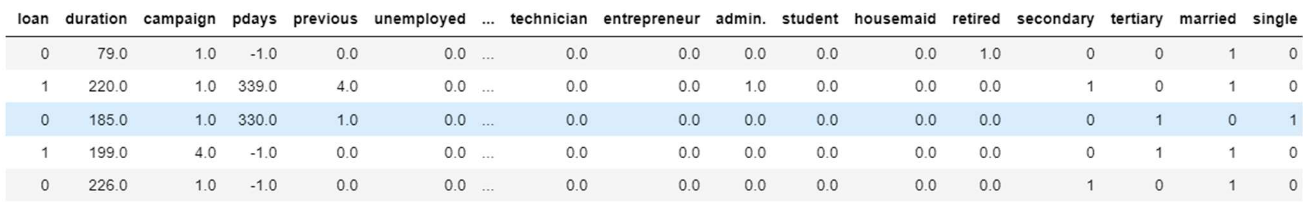
Breaking the dataset for easier navigation of input and output features:

```
x=bank_df.drop('y',axis=1)
y=bank_df['y']
```

Viewing the resulting dataframes:

```
x.head()
```

Output:



loan	duration	campaign	pdays	previous	unemployed	...	technician	entrepreneur	admin.	student	housemaid	retired	secondary	tertiary	married	single
0	79.0	1.0	-1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0	0	1	0
1	220.0	1.0	339.0	4.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1	0	1	0
0	185.0	1.0	330.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0	1	0	1
1	199.0	4.0	-1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0	1	1	0
0	226.0	1.0	-1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1	0	1	0

```
y.head()
```

Output:

```
0    0
1    0
2    0
3    0
4    0
Name: y, dtype: int64
```

Splitting the dataset for training and testing(80% for training and 20% for testing) then fitting the model based on Gradient Boosted Classifier and then predicting the values of target variable ***'y'*** on the model fitted:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

```
xg_mod = XGBClassifier(n_estimators=100, random_state = 100)
xg_mod.fit(x_train,y_train)
y_pred = xg_mod.predict(x_test)
y_pred
```

Output:

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0])
```

Checking the accuracy of the model using R-squared method(default):

```
r2_score = xg_mod.score(x_test, y_test)
print(r2_score)
```

Output:

```
0.8740331491712707
```

From the accuracy score obtained it can be easily said that model predicted 87.4% correct values which is a good prediction.

4.5 Best Parameters Evaluation

Now, moving ahead with finding the best parameters for the model fitted using ***'RandomizedSearchCV'*** method which does nothing but fits the model again and again based on the set of values of parameters defined already and calculates the accuracy and returns the model with parameters that gives best accuracy.
In the following code snippet the library is imported, the parameter are defined and the function is used to return the best model. As:

```
from scipy.stats import uniform, randint
from sklearn.model_selection import RandomizedSearchCV
```

```
params = {
    "n_estimators": randint(50, 500),
    "max_depth": randint(2, 10),
    "learning_rate": uniform(0.01, 0.3),
    "colsample_bytree": uniform(0.3, 0.7),
    "subsample": uniform(0.3, 0.7),
    "gamma": uniform(0, 0.5),
    "reg_lambda": uniform(0, 2),
}
```

```
rand_search = RandomizedSearchCV(xg_mod, params, cv=5, random_state=42, n_jobs=-
1)
```

Fitting the model and predicting the values:

```
rand_search.fit(x_train,y_train)
```

```
y_pred_1 = rand_search.predict(x_test)
```

Getting the new accuracy and best hyperparameters:

```
from sklearn.metrics import accuracy_score
```

```
accuracy = accuracy_score(y_test, y_pred_1)
print("Best hyperparameters: ", rand_search.best_params_)
print("Accuracy: ", accuracy)
```

Output:

```
Best hyperparameters:  {'colsample_bytree': 0.48114598712001183, 'gamma': 0.331261142176991, 'learning_rate': 0.103513322826823
28, 'max_depth': 7, 'n_estimators': 103, 'reg_lambda': 1.0934205586865593, 'subsample': 0.42939811886786894}
Accuracy:  0.8751381215469614
```

The hyperparameters obtained will result in a better model.

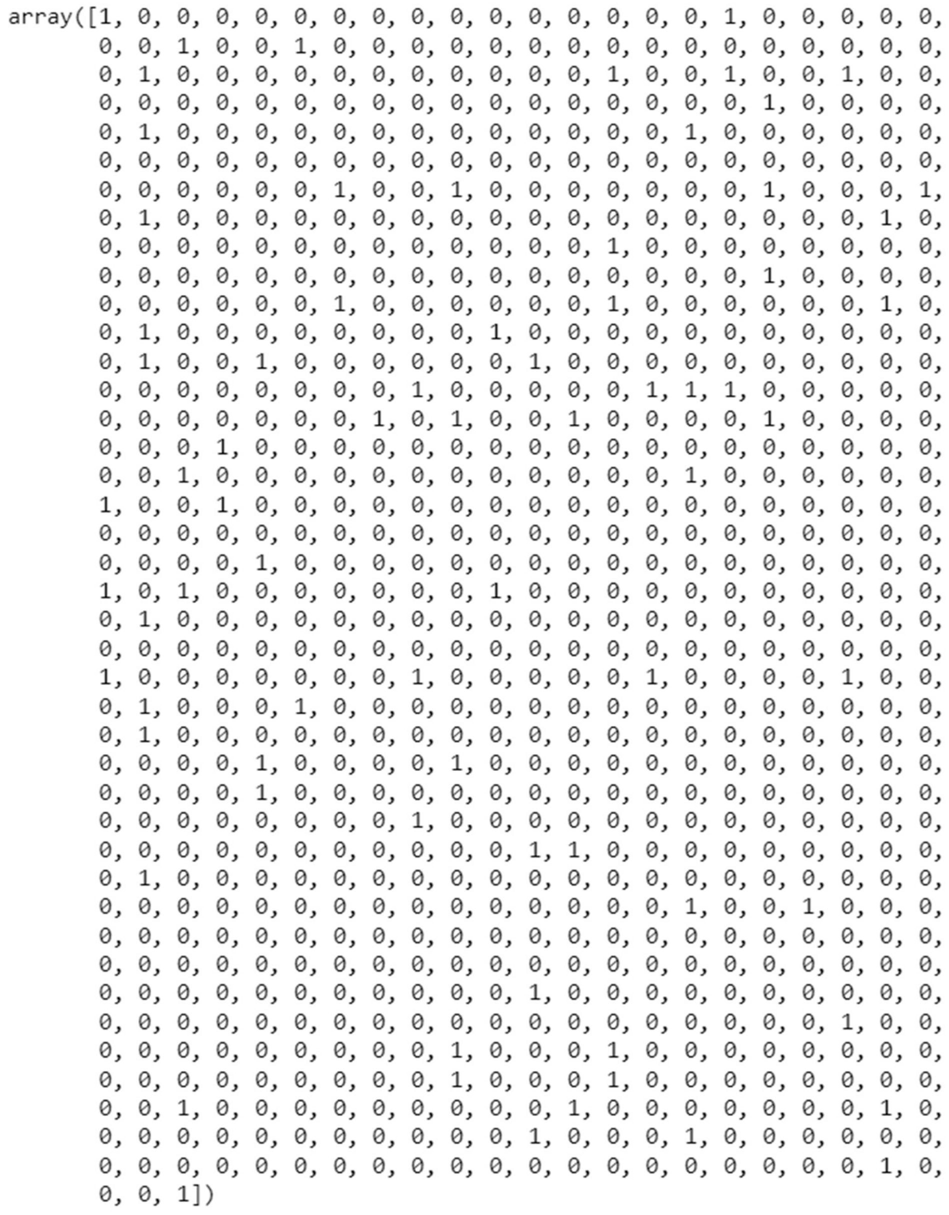
4.6 Final Model and Performance Metrics

Now, fitting the final model with obtained hyperparameters for best prediction:

```
xg_mod_fin = XGBClassifier(colsample_bytree = 0.602361513049481, gamma = 0.14561457009902096,learning_rate = 0.19355586841671385, max_depth = 3, n_estimators = 237,reg_lambda = 0.7327236865873834, subsample = 0.619248988951925)
xg_mod_fin.fit(x_train,y_train)
```

```
y_pred_fin = xg_mod_fin.predict(x_test)
y_pred_fin
```

Output:



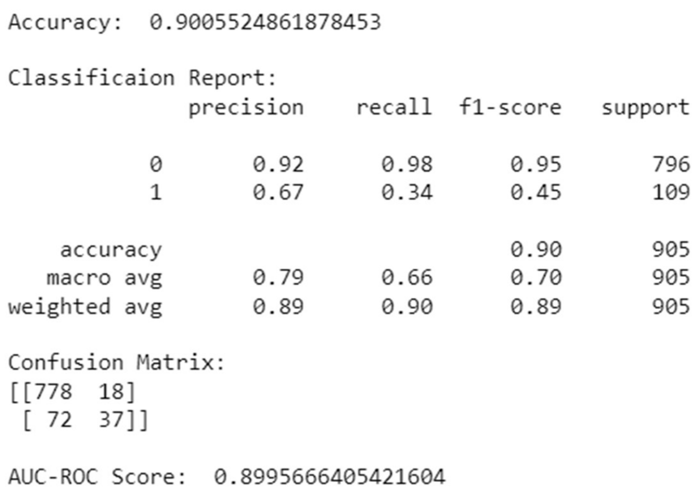
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1])
```

Evaluating the final model's performance using different metrices like accuracy, precision, recall, F1-score and AUC – ROC score:

```
from sklearn.metrics import roc_auc_score,roc_curve
```

```
acur_sc = xg_mod_fin.score(x_test, y_test)

y_pred_prob = xg_mod_fin.predict_proba(x_test)[:,1]#for auc-roc score
auc_roc = roc_auc_score(y_test, y_pred_prob)

print("Accuracy: ",acur_sc)
print(" ")
print("Classificaion Report:")
print(metrics.classification_report(y_test, y_pred_fin))
print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred_fin))
print(" ")
print("AUC-ROC Score: ", auc_roc)
```

Output:



```
Accuracy:  0.9005524861878453

Classificaion Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       796
           1       0.67      0.34      0.45       109

    accuracy                           0.90       905
   macro avg       0.79      0.66      0.70       905
weighted avg       0.89      0.90      0.89       905

Confusion Matrix:
[[778  18]
 [ 72  37]]

AUC-ROC Score:  0.8995666405421604
```

Conclusion

The final model created above is able to predict the target variable with ***90.5%*** accuracy. The dataset was cleaned by removing unnecessary columns and filling the unknown values in different colums using the ***ffill*** method in pandas library. After that appropriate preprocessing is done on data by converting the catergorical values to numeric values. Then a strong predictive model is built. The model is cross validated and corrected using the updated parameters for model building using appropriate parameter selection process. The classification technique used here is ***GB Classifier(Gradient Boosting Classifier)*** which basically uses multiple weak predictive models to build a significantly strong and effective prediction model. The initial model predicted the output with an accuracy of just ***87.4%*** but after re – evaluating the paraeters using ***RandomizedSearchCV*** method and fitting the model again with updated parameters the accuracy improved to ***90.5%***.

References

1. https://www.kaggle.com/competitions/bank-marketing-uci/overview
2. https://www.avaus.com/blog/predicting-customer-churn/#:~:text=Churn%20prediction%20means%20detecting%20which,more%20than%20retaining%20existing%20ones
3. https://medium.datadriveninvestor.com/python-programming-language-ac762a3b5977#:~:text=Python%20is%20an%20interpreted%2C%20high%20level%2C%20general%2Dpurpose%20programming,both%20small%20and%20large%20scale.
4. https://towardsdatascience.com/everything-you-need-to-know-about-jupyter-notebooks-10770719952b