BANKING CAMPAIGN PREDICTIVE ANALYTICS

Haveeshya Kandhiraju
Department of Computer Science
Montclair State University
Montclair, NJ, USA
Kandhirajuh1@montclair.edu

Abstract — **This project explores the application of Machine Learning on Banking Data to perform predictive analytics. In this project, data is ingested, pre-processed, and analyzed, and build Machine Learning models to train the data such that the model generates certain predictions on the likelihood of a customer signing up for a Banking campaign. These predictions help the Banks to focus more on the customers who are more likely to take up the campaign, thus saving time and money for the Banks and increasing the efficiency of the campaign. For implementation, the AWS platform is used along with AWS S3 for storage, IAM for access and permissions and Sagemaker to run the models. In this project Supervised Learning classification Algorithms are used to train the model with Gaussian Naïve Bayes Classifier, Decision Trees, and XGBoost algorithm. Hyperparameter tuning is performed to find the best-fit parameters for each model. The accuracy and performance of the models are compared by generating a Confusion Matrix, Classification Report, and ROC-AUC Curve. In the end, the model that provides the most accurate results is considered as the best fit for this data.**

Keywords—Data mining; Banking Campaign; Machine Learning Models; Predictive analytics; Confusion Matrix: Classification Report

I. INTRODUCTION

There have been a lot of advancements in technology of late. With continuous generation of data, there has been a large amount of data accumulated which has been termed as 'Big Data'. The rise of Big Data has seen an increasing number of tools and techniques to process the data. With the ability to process that data, there have been several attempts made to apply the data for analysis. One such application of Big Data analytics is Machine Learning. Machine learning is a field in which, based on the existing data a Machine is trained to perform certain actions such as predictions, classification, and clustering. Machine learning can be classified into Supervised learning, Unsupervised Learning, and Reinforcement learning. Supervised learning is where there are independent variables and a dependent variable also known as the label. Predictions are performed using regression and classification techniques on the data. In Unsupervised learning there is independent data and techniques are applied to perform clustering on the data. In Reinforcement learning the model learns by trial-and-error method. For this project, the focus will be on Supervised learning models as the data includes independent variables and labels as well. The Machine Learning Algorithms that will be used as part of this project are Gaussian Naïve Bayes, Decision Trees, and XGBoost.

Python is a popular language used for Big Data processing and computations as well as Machine learning algorithms. Python has become the go-to language when it comes to Big Data analytics due to it having a large number of libraries for data analytics and visualization. Also, its libraries are designed to handle large datasets. Python also provides excellent libraries for running Machine Learning Algorithms. Some of the Python libraries being used as part of this project are NumPy, Pandas, and Scikit-learn. NumPy are libraries for data storage and manipulation whereas Scikit-learn is used for applying Machine Learning algorithms on the data.

Cloud computing is another one of the modern technologies that has become very popular, especially in the field of Big Data analytics. With the cloud providing both storage and processing platforms, cloud-based solutions have gained in popularity. The cloud platform also provides us with managed services where the cloud service provider takes care of the management of the various services with respect to availability, scalability, fault tolerance, and security. One such popular cloud platform that is being used in this project is Amazon Web Services (AWS). AWS is one of the leading cloud providers in the industry today. In this project, AWS S3 is used for storage, Identity, and Access Management (IAM) for accesses and permission. AWS Sagemaker is used for running the Machine Learning Models. AWS Sagemaker provides support for the algorithms that are being used and provides us with the required compute power, necessary to run the Machine Learning Algorithms on large datasets.

II. RELATED WORK

Machine learning for predictive analytics has gained in popularity over the years. Banking has started to slowly incorporate Machine Learning into its environment. There have been several prior cases of Machine Learning implementation on Banking data.
The author demonstrates the different Machine learning algorithms and how they can be applied to the data in [1]. The various classification Machine Learning algorithms such as Gaussian Naïve Bayes, Decision Trees and XGBoost are shown and explained by the author.
The author shows how Machine Learning algorithms can be run on the AWS Cloud Platform in [2]. How the AWS platform is setup and how the different Machine Learning algorithms can be run on it is shown by the author.
The author shows how the Machine learning Algorithms can be optimized in [4]. The optimization techniques used for the different Machine Learning algorithms to improve the performance of the algorithm is demonstrated by the author.
In this project the understandings from the research papers are taken to implement the Banking campaign Predictive analytics. It demonstrates the ability of Machine learning to make predictions on the Banking dataset. This project also integrates Machine Learning algorithms with the Cloud Platform. All the models in the project are built on the AWS Cloud, thus leveraging, and showcasing the power of cloud computing on the large datasets.

III. DATASET

The dataset for this project is collected from the UCI website. The data is related to the marketing campaign of a Bank. The dataset contains 45211 records with 17 different variables. The dataset consists of the following fields:

- Age: Age of the customer
- Marital: Marital status of the customer
- Education: Education of the customer
- Job: Job type of the customer
- Balance: Amount Balance of the customer
- Housing: Whether customer has Housing Loan
- Loan: Whether the customer has Personal Loan
- Contact: Type of contact made with the customer
- Day: last day of contact
- Month: last month of contact
- Duration: Duration of contact
- Campaign: Number of contacts during campaign
- Target: Campaign Outcome

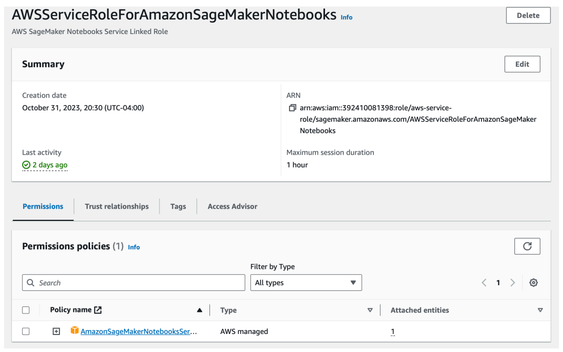
IV. PLATFORM SETUP

The project uses Cloud Computing to process the data and run the Machine Learning Algorithms. The cloud platform used in this project is the Amazon Web Services (AWS). To run the jobs on AWS initially setup of the platform needs to be done. Platform Setup consists of the following steps:

- Create an IAM Role to provide the required access and permissions.
- Setup AWS S3 Bucket and upload the dataset.
- Setup the Sagemaker domain and create a user.
- Start the Sagemaker Studio and execute the jobs.

A) IAM Role

IAM stands for Identity and Access Management. IAM role is used to provide the required accesses and permissions. IAM provides a layer of security with access restrictions as only the necessary users and services are provided access. In this project configuration of the IAM role is done such that the access is restricted between AWS S3 and AWS Sagemaker.


AWSServiceRoleForAmazonSageMakerNotebooks Info
AWS SageMaker Notebooks Service Linked Role

Summary

Creation date
October 31, 2023, 20:30 (UTC-04:00)

ARN
arn:aws:iam::392410081398:role/aws-service-role/sagemaker.amazonaws.com/AWSServiceRoleForAmazonSageMakerNotebooks

Last activity
2 days ago

Maximum session duration
1 hour

Permissions policies (1) Info

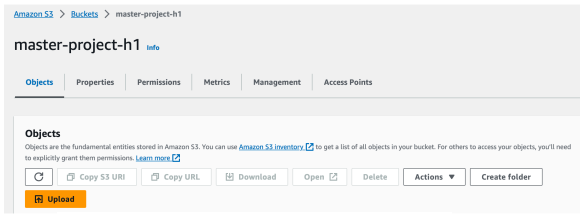
Filter by Type: All types

Policy name	Type	Attached entities
AmazonSageMakerNotebooksSer...	AWS managed	1

Fig. 1. Creation of IAM Role

B) S3 Bucket

S3 stands for Simple Storage Service. AWS S3 stores data as objects in a bucket. The data can be stored in a file and folder format. S3 supports storing data with different file formats. S3 provides data encryption so that the data that is stored in S3 is secure. For this project an S3 bucket is created, and the dataset is stored there so that the data can be accessed in Sagemaker to run the Machine Learning models.


Amazon S3 > Buckets > master-project-h1

master-project-h1 Info

Objects

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Fig. 2. Creation of S3 Bucket

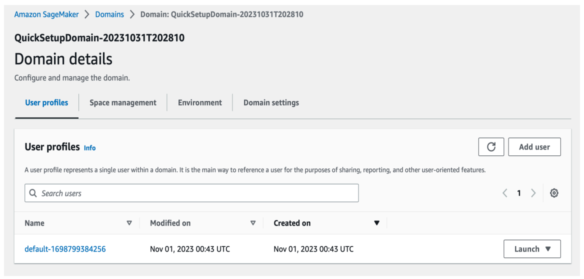
C) Sagemaker

AWS Sagemaker is a service that supports running the various Machine Learning models. Sagemaker has its own set of libraries to run the different Machine Learning models and it also supports the various Python libraries including Scikit-Learn which are being used to train the models. Sagemaker setup requires the following:

Creating the Sagemaker Domain: First create a Sagemaker domain.

Creating a user profile: Create a user profile and attach the user to the Sagemaker Domain.

Creating a cluster: Depending on the size of the dataset being used, and the complexity of operations required the correct sized cluster is created that can support running the algorithms.


Amazon SageMaker > Domains > Domain: QuickSetupDomain-20231031T202810

QuickSetupDomain-20231031T202810

Domain details

Configure and manage the domain.

User profiles | Space management | Environment | Domain settings

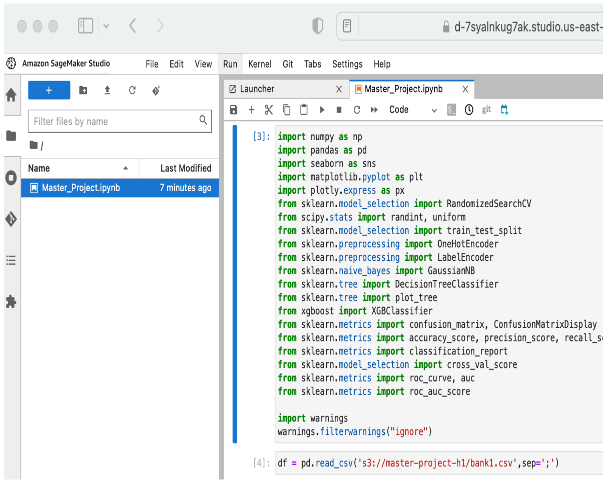
User profiles Info

A user profile represents a single user within a domain. It is the main way to reference a user for the purposes of sharing, reporting, and other user-oriented features.

Name	Modified on	Created on
default-1698799384256	Nov 01, 2023 00:43 UTC	Nov 01, 2023 00:43 UTC

Fig. 3. Creation of AWS Sagemaker Domain and user

Once the AWS Sagemaker service is setup with the correct Domain, User profile and Cluster size the Various Machine Learning Algorithms can be run to train the models.

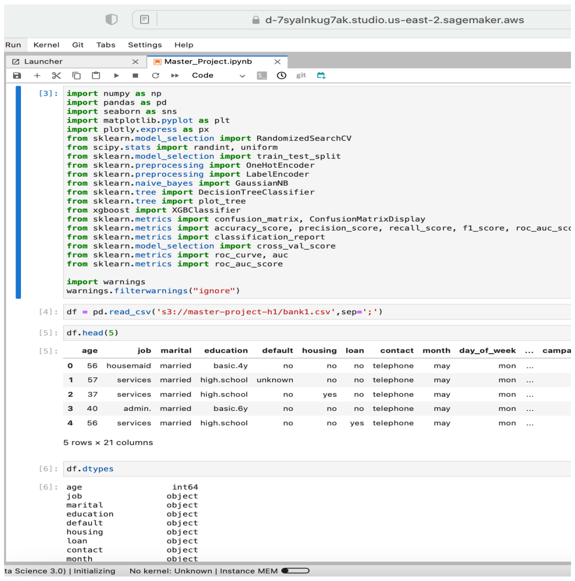


```
[3]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import plotly.express as px
     from sklearn.model_selection import RandomizedSearchCV
     from scipy.stats import randint, uniform
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.preprocessing import LabelEncoder
     from sklearn.naive_bayes import GaussianNB
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.tree import plot_tree
     from xgboost import XGBClassifier
     from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
     from sklearn.metrics import accuracy_score, precision_score, recall_s
     from sklearn.metrics import classification_report
     from sklearn.model_selection import cross_val_score
     from sklearn.metrics import roc_curve, auc
     from sklearn.metrics import roc_auc_score

     import warnings
     warnings.filterwarnings("ignore")

[4]: df = pd.read_csv('s3://master-project-h1/bank1.csv',sep=';')
```

Fig. 4. Shows AWS Sagemaker Studio

V. DATA LOADING

Data Loading is a step where the data is loaded into the environment. In this project the dataset is loaded from S3. Also, all the required libraries are imported.



```
[3]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import plotly.express as px
     from sklearn.model_selection import RandomizedSearchCV
     from scipy.stats import randint, uniform
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.preprocessing import LabelEncoder
     from sklearn.naive_bayes import GaussianNB
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.tree import plot_tree
     from xgboost import XGBClassifier
     from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_sc
     from sklearn.metrics import classification_report
     from sklearn.model_selection import cross_val_score
     from sklearn.metrics import roc_curve, auc
     from sklearn.metrics import roc_auc_score

     import warnings
     warnings.filterwarnings("ignore")

[4]: df = pd.read_csv('s3://master-project-h1/bank1.csv',sep=';')

[5]: df.head(5)

[5]:   age        job  marital    education  default  housing  loan    contact  month  day_of_week  ...  campa
     0  56  housemaid  married     basic.4y       no       no    no  telephone    may          mon  ...
     1  57   services  married  high.school  unknown       no    no  telephone    may          mon  ...
     2  37   services  married  high.school       no      yes    no  telephone    may          mon  ...
     3  40     admin.  married     basic.6y       no       no    no  telephone    may          mon  ...
     4  56   services  married  high.school       no       no   yes  telephone    may          mon  ...

     5 rows × 21 columns

[6]: df.dtypes

[6]: age                int64
     job               object
     marital           object
     education         object
     default           object
     housing           object
     loan              object
     contact           object
     month             object
```

ta Science 3.0) | Initializing   No kernel: Unknown | Instance MEM

Fig. 5. Shows libraries imported and data loading from S3.

VI. DATA PRE-PROCESSING

Data pre-processing is an important step in the Machine Learning process. It is the next step after loading the data. Pre-processing helps in transforming the data in such a way, that there is a cleaner dataset. Pre-processing the dataset also enables us to train the models in a more efficient manner. The following operations are performed as part of data pre-processing:

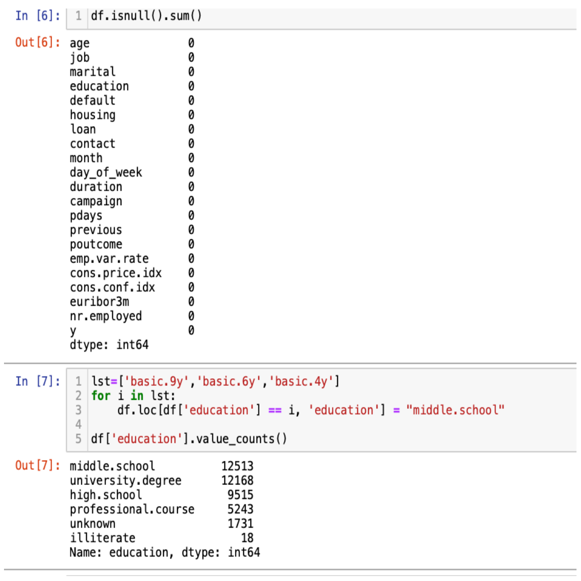
A) Null Check
B) Data Grouping
C) Data Type Casting
D) Near Zero variance

A) Null Check

This project looks to handle Null values in the dataset. Null Values can be unnecessary noise in the data and can strongly influence the models. Hence, null value processing is performed in the dataset. There are few ways to handle null values. If the column has more than 50 percent of null values, then drop the column from the dataset and do not consider the column for this model. There are other ways to handle the null values. Some of the other ways to handle null values are to fill the null records with similar values. Null values can be filled by computing the mean of entire column and filling the null records with the mean value of the column. Another way to handle null values is using the median of the column. Column median values can be filled the missing records. Another such way to handle null values is to fill the null records with the nearest non-null value. Handling nulls using the above techniques allows us to have a more balanced dataset to train the models on.

B) Data Grouping

Data grouping is a technique where similar data records are grouped together for better analysis. Customers with a similar salary range can be grouped together. Similarly, customers that have a similar education background can form one group. Customers who age lies within a specific range can be put into another group. Grouping data in such a manner can help us analyze the data better and train the models in a more efficient manner.



```
In [6]: df.isnull().sum()
```

```
Out[6]: age               0
        job               0
        marital           0
        education         0
        default           0
        housing           0
        loan              0
        contact           0
        month             0
        day_of_week       0
        duration          0
        campaign          0
        pdays             0
        previous          0
        poutcome          0
        emp.var.rate      0
        cons.price.idx    0
        cons.conf.idx     0
        euribor3m         0
        nr.employed       0
        y                 0
        dtype: int64
```

```
In [7]: lst=['basic.9y','basic.6y','basic.4y']
        for i in lst:
            df.loc[df['education'] == i, 'education'] = "middle.school"

        df['education'].value_counts()
```

```
Out[7]: middle.school          12513
        university.degree      12168
        high.school             9515
        professional.course     5243
        unknown                 1731
        illiterate                18
        Name: education, dtype: int64
```

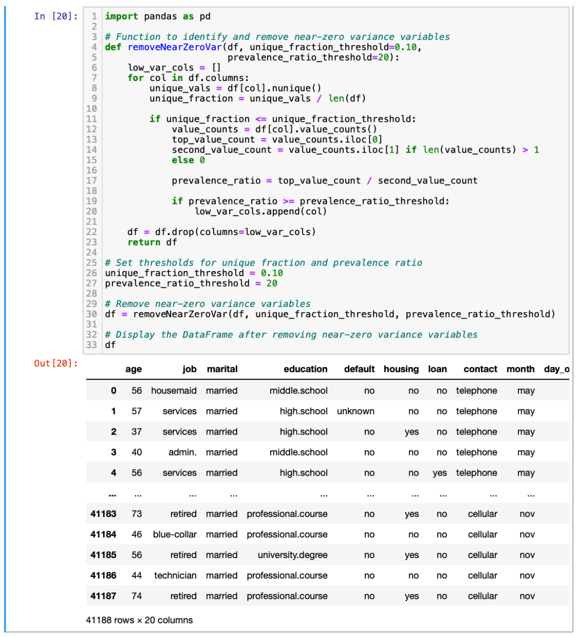
Fig. 6. Shows null check and data grouping.

C) Data Type Casting

Type casting is a method where data is casted to the correct datatype. Numerical data is casted to be an Integer or a Long. Similarly, data with decimal points is casted to be a Double or a Float. Data with Categorical data is type casted to String datatype. Type Casting the data in this manner allows us to analyze the data better.

D) Near Zero Variance

Near zero variance means there is a low number of unique values in the dataset. If the ratio of the highest occurring value to the second highest occurring value is large, then it is considered to have a near zero variance. If a value is heavily influencing the data, then such values are removed from the dataset. In this data the records are dropped, that have a Near Zero Variance.



```
In [20]: import pandas as pd

         # Function to identify and remove near-zero variance variables
         def removeNearZeroVar(df, unique_fraction_threshold=0.10,
                               prevalence_ratio_threshold=20):
             low_var_cols = []
             for col in df.columns:
                 unique_vals = df[col].nunique()
                 unique_fraction = unique_vals / len(df)

                 if unique_fraction <= unique_fraction_threshold:
                     value_counts = df[col].value_counts()
                     top_value_count = value_counts.iloc[0]
                     second_value_count = value_counts.iloc[1] if len(value_counts) > 1
                     else 0

                     prevalence_ratio = top_value_count / second_value_count

                     if prevalence_ratio >= prevalence_ratio_threshold:
                         low_var_cols.append(col)

             df = df.drop(columns=low_var_cols)
             return df

         # Set thresholds for unique fraction and prevalence ratio
         unique_fraction_threshold = 0.10
         prevalence_ratio_threshold = 20

         # Remove near-zero variance variables
         df = removeNearZeroVar(df, unique_fraction_threshold, prevalence_ratio_threshold)

         # Display the DataFrame after removing near-zero variance variables
         df
```

Out[20]:

	age	job	marital	education	default	housing	loan	contact	month	day_o
0	56	housemaid	married	middle.school	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	middle.school	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	
...	...	...	...	...	...	...	...	...	...	
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	

41188 rows × 20 columns

Fig. 7. Shows removing of near zero variance.

VII. DATA VISUALIZATION

Data visualization is an important step in Machine Learning. It helps us to understand the different variables in the data. Data visualization is performed on this data to understand the unique values and the counts of different values in the data.

The following visualizations are performed:

A) Users by job type.
B) Users by marital status.
C) Users by education.
D) Users by contact type.
E) Users with housing Loans.
F) Users with Personal Loans.
G) Users by age.
H) Campaign Type.
I) Campaign Duration.

These charts help in getting a better understanding of data which will be further useful in selecting the relevant features for this project. Selecting the right features is important in Machine Learning as they can heavily influence the models.

Couple of ways visualization is performed in this project are through Bar Graphs and Histograms. Bar graphs are generated on some of the metrics above to get a good understanding of them.

A) Users by Job Type

Generate a count of users by their job type.

```
In [7]:  job_count=df.job.value_counts()
         job_count
         job_count.plot(kind="barh")
Out[7]:  <Axes: >
```

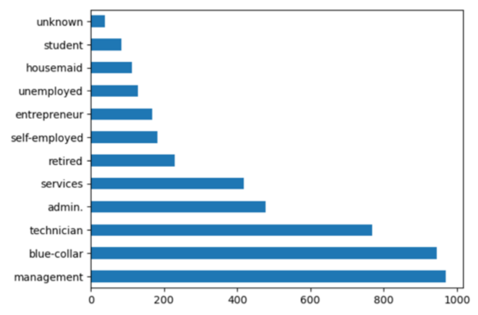


Fig. 8. Visualization of customers based on Job Type.

B) Users by Marital Status

Generate count of users by their marital status.

```
In [8]:  Marital_count=df.marital.value_counts()
         Marital_count
         Marital_count.plot(kind="barh")
Out[8]:  <Axes: >
```

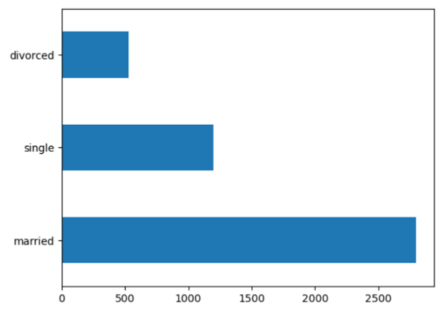


Fig. 9. Visualization of customers based on Marital Status

C) Users by Education

Generate counts of users by their education.

```
In [9]:  Education_count=df.education.value_counts()
         Education_count
         Education_count.plot(kind="barh")
Out[9]:  <Axes: >
```

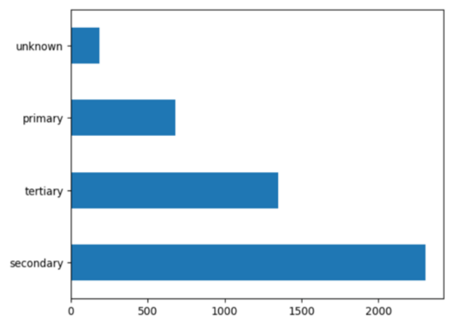


Fig. 10. Visualization of customers based on Education.

D) Users by Contact Type

Generate count of users by the contact type established with them.

```
In [10]:  Contact_count=df.contact.value_counts()
          Contact_count
          Contact_count.plot(kind="barh")
Out[10]:  <Axes: >
```

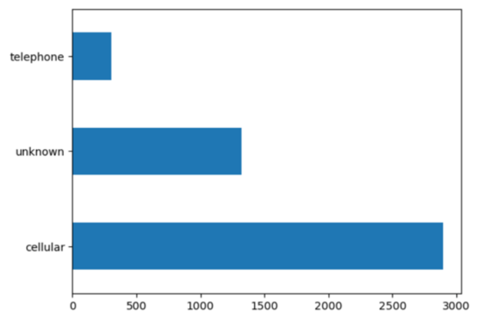


Fig. 11. Visualization of customers based on Contact Type.

E) Users with Housing Loans

Generate count of users on whether they have a housing loan.

```
In [12]:  Housing_loan_count=df.housing.value_counts()
          Housing_loan_count
          Housing_loan_count.plot(kind="barh")
Out[12]:  <Axes: >
```

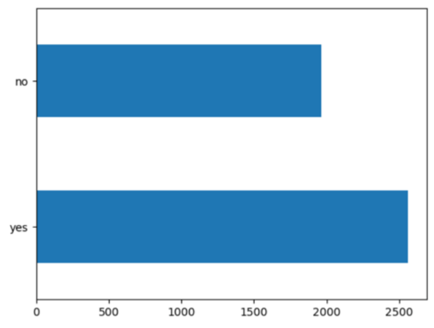


Fig. 12. Visualization of customers based on Housing Loan.

F) Users with Personal Loan

Generate count of users on whether they have a personal loan.

```
In [13]:  Personal_loan_count=df.loan.value_counts()
          Personal_loan_count
          Personal_loan_count.plot(kind="barh")
Out[13]:  <Axes: >
```

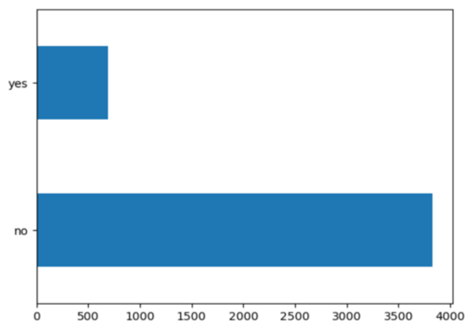
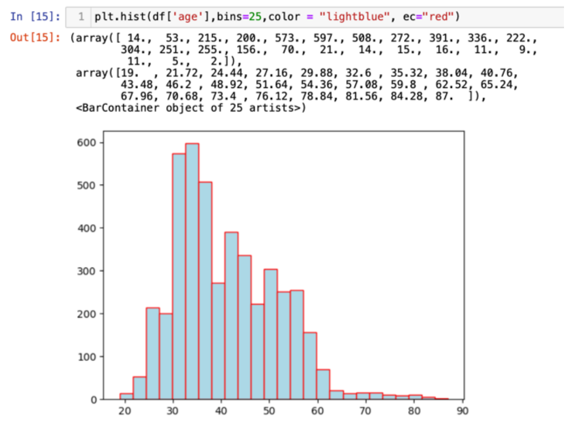


Fig. 13. Visualization of customers based on Personal Loan.

Generate histograms on some of the metrics to understand them better.

G) *User by Age*

Generate a count of users by their age.

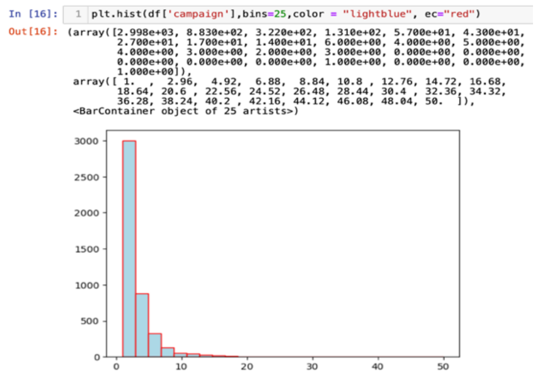


```
In [15]: plt.hist(df['age'],bins=25,color = "lightblue", ec="red")
Out[15]: (array([ 14.,  53., 215., 200., 573., 597., 508., 272., 391., 336., 222.,
         304., 251., 255., 156.,  70.,  21.,  14.,  15.,  16.,  11.,   9.,
          11.,   5.,   2.]),
 array([19.  , 21.72, 24.44, 27.16, 29.88, 32.6 , 35.32, 38.04, 40.76,
        43.48, 46.2 , 48.92, 51.64, 54.36, 57.08, 59.8 , 62.52, 65.24,
        67.96, 70.68, 73.4 , 76.12, 78.84, 81.56, 84.28, 87.  ]),
 <BarContainer object of 25 artists>)
```

Fig. 14. Visualization of customers based on Age.

H) *Users by Campaign Type*

Count of users by the different Type of Campaigns

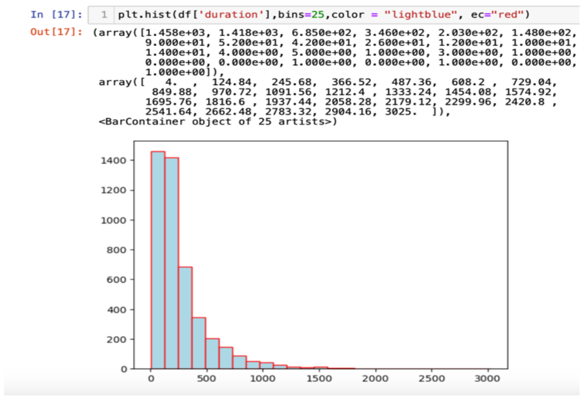


```
In [16]: plt.hist(df['campaign'],bins=25,color = "lightblue", ec="red")
Out[16]: (array([2.998e+03, 8.830e+02, 3.220e+02, 1.310e+02, 5.700e+01, 4.300e+01,
         2.700e+01, 1.700e+01, 1.400e+01, 6.000e+00, 4.000e+00, 5.000e+00,
         4.000e+00, 3.000e+00, 2.000e+00, 3.000e+00, 0.000e+00, 0.000e+00,
         0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00,
         1.000e+00]),
 array([ 1.  ,  2.96,  4.92,  6.88,  8.84, 10.8 , 12.76, 14.72, 16.68,
        18.64, 20.6 , 22.56, 24.52, 26.48, 28.44, 30.4 , 32.36, 34.32,
        36.28, 38.24, 40.2 , 42.16, 44.12, 46.08, 48.04, 50.  ]),
 <BarContainer object of 25 artists>)
```

Fig. 15. Visualization of customers based on Campaign

I) *Users by Campaign Duration*

Count of users by the campaign duration



```
In [17]: plt.hist(df['duration'],bins=25,color = "lightblue", ec="red")
Out[17]: (array([1.458e+03, 1.418e+03, 6.850e+02, 3.460e+02, 2.030e+02, 1.480e+02,
         9.000e+01, 5.200e+01, 4.200e+01, 2.600e+01, 1.200e+01, 1.000e+01,
         1.400e+01, 4.000e+00, 5.000e+00, 1.000e+00, 3.000e+00, 1.000e+00,
         0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 1.000e+00, 0.000e+00,
         1.000e+00]),
 array([   4.  ,  124.84,  245.68,  366.52,  487.36,  608.2 ,  729.04,
         849.88,  970.72, 1091.56, 1212.4 , 1333.24, 1454.08, 1574.92,
        1695.76, 1816.6 , 1937.44, 2058.28, 2179.12, 2299.96, 2420.8 ,
        2541.64, 2662.48, 2783.32, 2904.16, 3025.  ]),
 <BarContainer object of 25 artists>)
```

Fig. 16. Visualization of customers based on Campaign Duration.

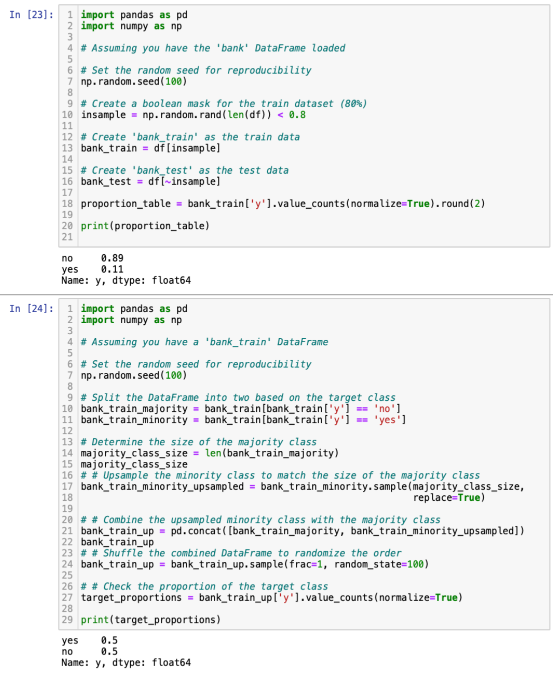
VIII. FEATURE ENGINEERING

Feature engineering is an important step in Machine Learning. Feature engineering helps us modify the data and its features in such a way that there is a very balanced dataset for training the model. Feature Engineering includes the following techniques.

A) Data Upsampling
B) One-Hot Encoding
C) Label Encoding

A) *Data Upsampling*

Data Upsampling is a technique to handle imbalanced data in the dataset. In data Upsampling first find out the majority and minority samples. After which increase the number of minority samples by synthetically generating new samples or replicating the existing samples. Thus, boosting the minority samples helps in the ML models work more efficiently and provide accurate predictions.



```
In [23]:
import pandas as pd
import numpy as np

# Assuming you have the 'bank' DataFrame loaded

# Set the random seed for reproducibility
np.random.seed(100)

# Create a boolean mask for the train dataset (80%)
insample = np.random.rand(len(df)) < 0.8

# Create 'bank_train' as the train data
bank_train = df[insample]

# Create 'bank_test' as the test data
bank_test = df[~insample]

proportion_table = bank_train['y'].value_counts(normalize=True).round(2)

print(proportion_table)

no     0.89
yes    0.11
Name: y, dtype: float64
```

```
In [24]:
import pandas as pd
import numpy as np

# Assuming you have a 'bank_train' DataFrame

# Set the random seed for reproducibility
np.random.seed(100)

# Split the DataFrame into two based on the target class
bank_train_majority = bank_train[bank_train['y'] == 'no']
bank_train_minority = bank_train[bank_train['y'] == 'yes']

# Determine the size of the majority class
majority_class_size = len(bank_train_majority)
majority_class_size
# # Upsample the minority class to match the size of the majority class
bank_train_minority_upsampled = bank_train_minority.sample(majority_class_size,
                                                           replace=True)

# # Combine the upsampled minority class with the majority class
bank_train_up = pd.concat([bank_train_majority, bank_train_minority_upsampled])
bank_train_up
# # Shuffle the combined DataFrame to randomize the order
bank_train_up = bank_train_up.sample(frac=1, random_state=100)

# # Check the proportion of the target class
target_proportions = bank_train_up['y'].value_counts(normalize=True)

print(target_proportions)

yes    0.5
no     0.5
Name: y, dtype: float64
```

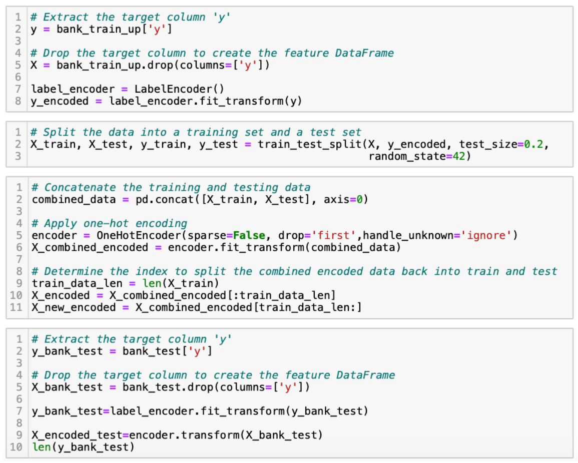
Fig. 17. Data Upsampling

B) *One-hot encoding*

One hot encoding is a technique to handle categorical data. There are certain algorithms that don't work well with categorical data and perform well with numerical data. To handle such a scenario, one-hot encoding is used. One-hot encoding transforms the categorical data records in the non-numerical columns. This allows the ML algorithms to be trained efficiently and provide accurate results.

C) Label encoding

Label encoding is a technique where the label values are encoded. It is like one-hot encoding, except that Label encoding is applied to the label field. If the label or the target variable is a categorical value, label encoding is performed to transform it to a numerical value.



```
# Extract the target column 'y'
y = bank_train_up['y']

# Drop the target column to create the feature DataFrame
X = bank_train_up.drop(columns=['y'])

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,
                                                     random_state=42)
```

```
# Concatenate the training and testing data
combined_data = pd.concat([X_train, X_test], axis=0)

# Apply one-hot encoding
encoder = OneHotEncoder(sparse=False, drop='first',handle_unknown='ignore')
X_combined_encoded = encoder.fit_transform(combined_data)

# Determine the index to split the combined encoded data back into train and test
train_data_len = len(X_train)
X_encoded = X_combined_encoded[:train_data_len]
X_new_encoded = X_combined_encoded[train_data_len:]
```

```
# Extract the target column 'y'
y_bank_test = bank_test['y']

# Drop the target column to create the feature DataFrame
X_bank_test = bank_test.drop(columns=['y'])

y_bank_test=label_encoder.fit_transform(y_bank_test)

X_encoded_test=encoder.transform(X_bank_test)
len(y_bank_test)
```

Fig. 18. One-hot encoding and Label Encoding

The Fig.18 shows the splitting of data into train and test in 80:20 ratio, that is 80 percent of data for training and 20 percent of data for testing. One-hot encoding is performed on features of train and test data. Label encoding is performed on labels of train and test data.

IX. MACHINE LEARNING MODELS

Once the pre-processing of the data and feature engineering is done, the focus shifts towards training the data with the various Machine Learning models. In this project three major Machine Learning models are being used:

A) Gaussian Naïve Bayes
B) Decision Tree
C) XGBoost

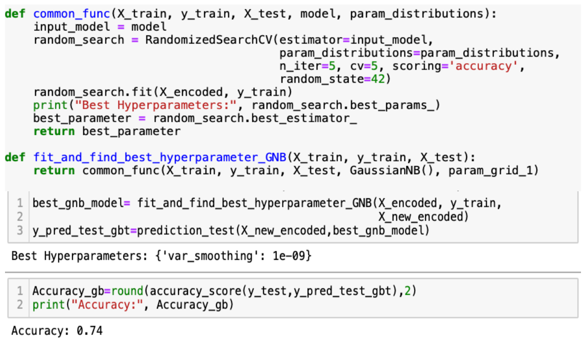
A) Gaussian Naïve Bayes

Gaussian Naïve Bayes classifier is a very powerful algorithm in Machine Learning. It is an algorithm based on probabilistic approach and Gaussian reasons. It works on the Bayes theorem principle. Naïve Bayes algorithm assumes that features are independent of each other. Modifying value of one feature does not directly impact other features.

According to Bayes' Theorem, the likelihood of the second event given the first event multiplied by the probability of the first event is the conditional probability of an event, contingent on the occurrence of another event.

$$P(Y \mid X) = P(X \mid Y) * P(Y) / P(X)$$

This project implements the Naïve Bayes classifier based on this Bayes theorem.



```
def common_func(X_train, y_train, X_test, model, param_distributions):
    input_model = model
    random_search = RandomizedSearchCV(estimator=input_model,
                                       param_distributions=param_distributions,
                                       n_iter=5, cv=5, scoring='accuracy',
                                       random_state=42)
    random_search.fit(X_encoded, y_train)
    print("Best Hyperparameters:", random_search.best_params_)
    best_parameter = random_search.best_estimator_
    return best_parameter

def fit_and_find_best_hyperparameter_GNB(X_train, y_train, X_test):
    return common_func(X_train, y_train, X_test, GaussianNB(), param_grid_1)
```

```
best_gnb_model= fit_and_find_best_hyperparameter_GNB(X_encoded, y_train,
                                                      X_new_encoded)
y_pred_test_gbt=prediction_test(X_new_encoded,best_gnb_model)
```

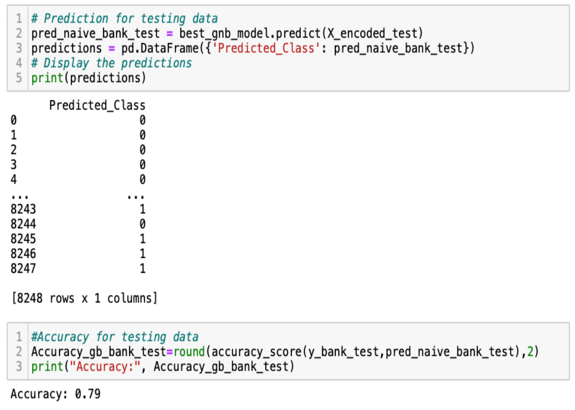
```
Best Hyperparameters: {'var_smoothing': 1e-09}
```

```
Accuracy_gb=round(accuracy_score(y_test,y_pred_test_gbt),2)
print("Accuracy:", Accuracy_gb)
```

```
Accuracy: 0.74
```

Fig. 19. Implementation of Gaussian Naïve Bayes

As can be seen from the above figure the model is trained on the Gaussian Naïve Bayes Classifier. The Naïve Bayes classifier imported from the scikit-learn library is fit on the data to train the model.



```
# Prediction for testing data
pred_naive_bank_test = best_gnb_model.predict(X_encoded_test)
predictions = pd.DataFrame({'Predicted_Class': pred_naive_bank_test})
# Display the predictions
print(predictions)
```

```
      Predicted_Class
0                   0
1                   0
2                   0
3                   0
4                   0
...               ...
8243                1
8244                0
8245                1
8246                1
8247                1

[8248 rows x 1 columns]
```

```
#Accuracy for testing data
Accuracy_gb_bank_test=round(accuracy_score(y_bank_test,pred_naive_bank_test),2)
print("Accuracy:", Accuracy_gb_bank_test)
```

```
Accuracy: 0.79
```

Fig. 20. Gaussian Naïve Bayes Predictions & Accuracy

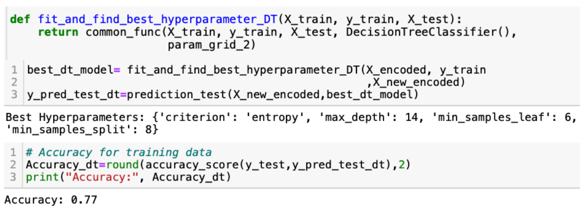
Once the fit is performed the model is run on test data. predictions are made on the test data and then the accuracy of the model is calculated.

B) Decision Tree

Decision Tree is one of the more popular Supervised Machine learning algorithms. The Decision Tree can be used as both regression and classification algorithm. In this project, Decision Tree is used as a Classification Algorithm. Decision Trees resemble a flowchart where a decision is made at each branch of the tree. Some of the key terms for Decision Tree are as follows:

Root Node: It forms the base of the Decision Tree.

Splitting: It is the process of dividing nodes into further sub nodes. Decision Node: When a sub node is further split into additional nodes.
Leaf Node: When a sub node does not split into additional nodes. This is node where decisions are made.
Pruning: It is the process of removing sub nodes from the Tree.
Branch: It is the sub-section of the Decision Tree consisting of multiple nodes.
A decision tree looks like a tree. The root node is the base of the tree. A sequence of decision nodes that represent decisions that need to be made flow from the root node. Leaf nodes, which show the decisions' effects, branch off the decision nodes. A question or split point is represented by each decision node, and potential responses are represented by the leaf nodes that branch off a decision node. Just as a leaf sprout on a tree limb, decision nodes give rise to leaf nodes. For this reason, every portion of a decision tree is referred to as a branch.



```
def fit_and_find_best_hyperparameter_DT(X_train, y_train, X_test):
    return common_func(X_train, y_train, X_test, DecisionTreeClassifier(),
                       param_grid_2)
```

```
best_dt_model= fit_and_find_best_hyperparameter_DT(X_encoded, y_train
                                                   ,X_new_encoded)
y_pred_test_dt=prediction_test(X_new_encoded,best_dt_model)
```

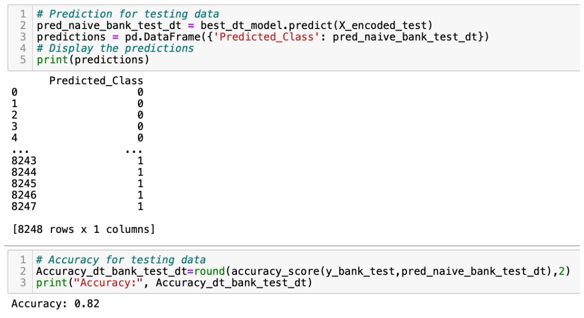
```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 14, 'min_samples_leaf': 6,
'min_samples_split': 8}
```

```
# Accuracy for training data
Accuracy_dt=round(accuracy_score(y_test,y_pred_test_dt),2)
print("Accuracy:", Accuracy_dt)
```

```
Accuracy: 0.77
```

Fig. 21. Implementation of Decision Tree.

As can be seen from the above figure the model is trained on the Decision Tree Classifier. The Decision tree classifier imported from the scikit-learn library is fit on the data to train the model.



```
# Prediction for testing data
pred_naive_bank_test_dt = best_dt_model.predict(X_encoded_test)
predictions = pd.DataFrame({'Predicted_Class': pred_naive_bank_test_dt})
# Display the predictions
print(predictions)
```

```
      Predicted_Class
0                   0
1                   0
2                   0
3                   0
4                   0
...               ...
8243                1
8244                1
8245                1
8246                1
8247                1

[8248 rows x 1 columns]
```

```
# Accuracy for testing data
Accuracy_dt_bank_test_dt=round(accuracy_score(y_bank_test,pred_naive_bank_test_dt),2)
print("Accuracy:", Accuracy_dt_bank_test_dt)
```

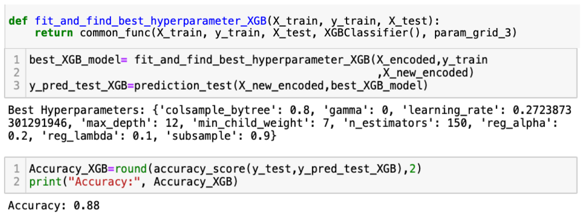
```
Accuracy: 0.82
```

Fig. 22. Decision Tree Predictions & Accuracy

Once the fit is performed the model is run on test data. predictions are made on the test data and then the accuracy of the model is calculated.

C) XGBoost

XGBoost is a robust Machine Learning Algorithm that is used to help understand the data better and make more informed decision. It is a hugely popular algorithm that works on the implementation of gradient boosting decision trees. It finds significant implementation in terms of model optimization and performance improvement of the Machine Learning Algorithms. It is an algorithm known for its high speed and performance boosting capabilities. It provides regularization which allows to prevent overfitting of the model. It is also able to handle sparse datasets and works well with large datasets as well. Its ability to scale is also one of its key features and is what that makes it so popular.



```
def fit_and_find_best_hyperparameter_XGB(X_train, y_train, X_test):
    return common_func(X_train, y_train, X_test, XGBClassifier(), param_grid_3)
```

```
best_XGB_model= fit_and_find_best_hyperparameter_XGB(X_encoded,y_train
                                                     ,X_new_encoded)
y_pred_test_XGB=prediction_test(X_new_encoded,best_XGB_model)
```

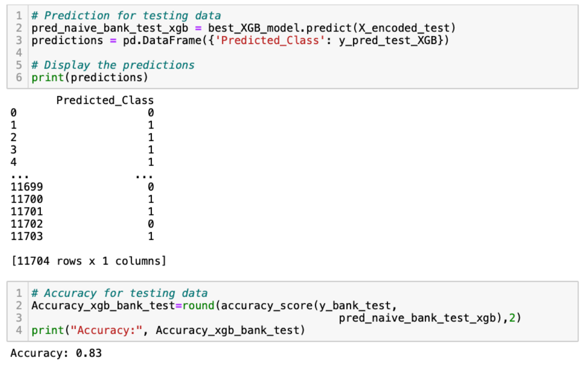
```
Best Hyperparameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.2723873
301291946, 'max_depth': 12, 'min_child_weight': 7, 'n_estimators': 150, 'reg_alpha':
0.2, 'reg_lambda': 0.1, 'subsample': 0.9}
```

```
Accuracy_XGB=round(accuracy_score(y_test,y_pred_test_XGB),2)
print("Accuracy:", Accuracy_XGB)
```

```
Accuracy: 0.88
```

Fig. 23. Implementation of XGBoost.

As can be seen from the above figure the model is trained on the XGBoost Algorithm. The XGBoost algorithm imported from the scikit-learn library is fit on the data to train the model.



```
# Prediction for testing data
pred_naive_bank_test_xgb = best_XGB_model.predict(X_encoded_test)
predictions = pd.DataFrame({'Predicted_Class': y_pred_test_XGB})

# Display the predictions
print(predictions)
```

```
       Predicted_Class
0                    0
1                    1
2                    1
3                    1
4                    1
...                ...
11699                0
11700                1
11701                1
11702                0
11703                1

[11704 rows x 1 columns]
```

```
# Accuracy for testing data
Accuracy_xgb_bank_test=round(accuracy_score(y_bank_test,
                                            pred_naive_bank_test_xgb),2)
print("Accuracy:", Accuracy_xgb_bank_test)
```

```
Accuracy: 0.83
```

Fig. 24. XGBoost Predictions & Accuracy

Once the fit is performed the model is run on test data. predictions are made on the test data and then the accuracy of the model is calculated.

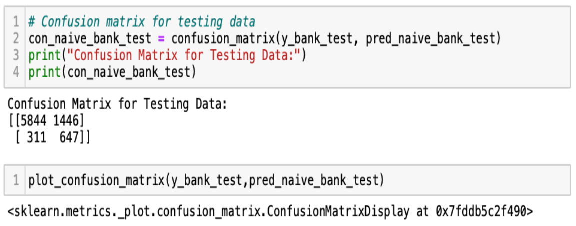
X. MODEL EVALUATION

Model evaluation is one of the key aspects in Machine learning. Model evaluation is the process of finding the right model for the data that generates the most accurate predictions on the data by looking at various metrics as part of the evaluation. There are few different metrics that are used for model evaluation. This project uses the following techniques:

A) Confusion Matrix
B) Classification Report
C) ROC-AUC Curve

A) Confusion Matrix

Confusion Matrix provides a representation of the model predictions in the form of a Matrix. It provides the total number of correct and incorrect predictions in a class.



```
# Confusion matrix for testing data
con_naive_bank_test = confusion_matrix(y_bank_test, pred_naive_bank_test)
print("Confusion Matrix for Testing Data:")
print(con_naive_bank_test)
```

```
Confusion Matrix for Testing Data:
[[5844 1446]
 [ 311  647]]
```

```
plot_confusion_matrix(y_bank_test,pred_naive_bank_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fddb5c2f490>
```

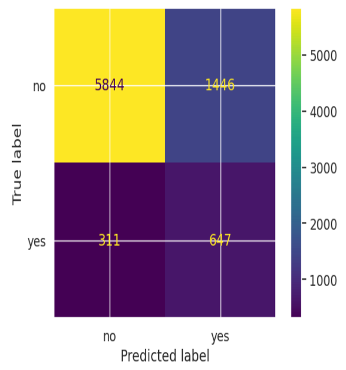
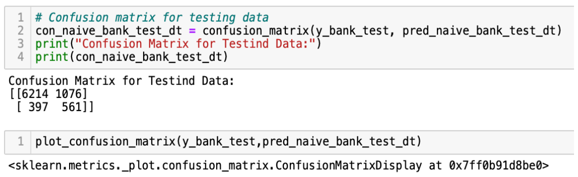


Fig. 25. Gaussian Naïve Bayes Confusion Matrix

The above figure shows the confusion matrix for the predictions made by the Gaussian Naïve Bayes Classifier.



```
# Confusion matrix for testing data
con_naive_bank_test_dt = confusion_matrix(y_bank_test, pred_naive_bank_test_dt)
print("Confusion Matrix for Testind Data:")
print(con_naive_bank_test_dt)
```

```
Confusion Matrix for Testind Data:
[[6214 1076]
 [ 397  561]]
```

```
plot_confusion_matrix(y_bank_test,pred_naive_bank_test_dt)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff0b91d8be0>
```

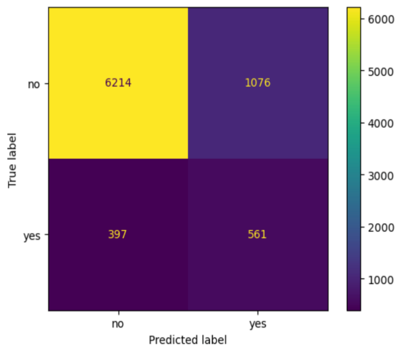
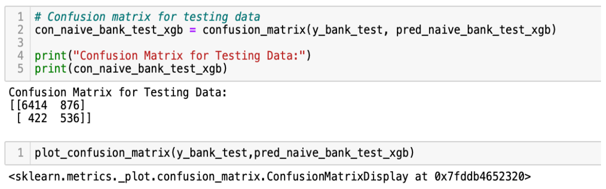


Fig. 26. Decision Tree Confusion Matrix

Fig. 26. shows the confusion matrix for the predictions made by the Decision Tree Algorithm.



```
# Confusion matrix for testing data
con_naive_bank_test_xgb = confusion_matrix(y_bank_test, pred_naive_bank_test_xgb)

print("Confusion Matrix for Testing Data:")
print(con_naive_bank_test_xgb)
```

```
Confusion Matrix for Testing Data:
[[6414  876]
 [ 422  536]]
```

```
plot_confusion_matrix(y_bank_test,pred_naive_bank_test_xgb)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fddb4652320>
```

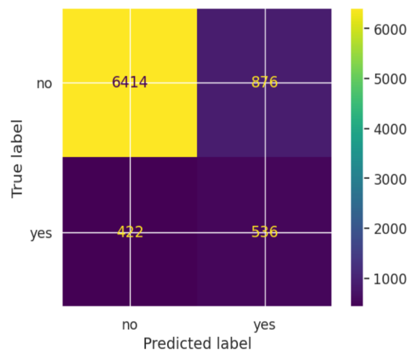


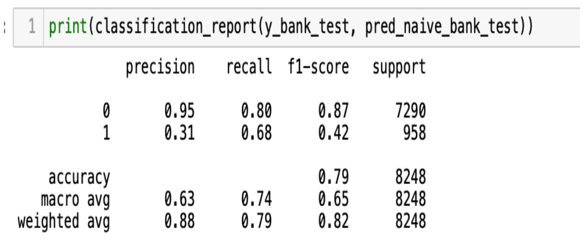
Fig. 27. XGBoost Confusion Matrix

The above figure shows the confusion matrix for the predictions made by the XGBoost Algorithm.

Comparing the confusion matrix of three models, XGBoost algorithm gives highest number of correctly predicted observations compared to Decision tree and Gaussian Naïve Bayes algorithms.

B) Classification Report

Classification Report is one of the key indicators in terms of model selection and comparison. Classification Report provides various key metrics about the model such as precision, recall, F1 and support score for each model.

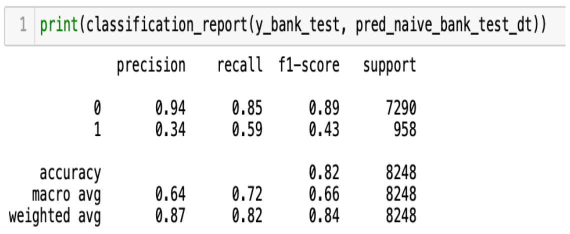


```
print(classification_report(y_bank_test, pred_naive_bank_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.80      0.87      7290
           1       0.31      0.68      0.42       958

    accuracy                           0.79      8248
   macro avg       0.63      0.74      0.65      8248
weighted avg       0.88      0.79      0.82      8248
```

Fig. 28. Gaussian Naïve Bayes Classification Report

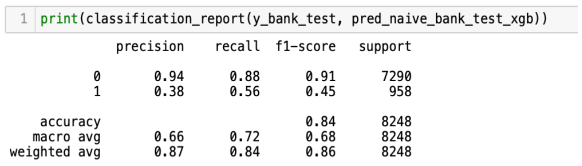
The above figure shows the classification report for the Gaussian Naïve Bayes algorithm from which we can understand the accuracy is 0.79.



```
print(classification_report(y_bank_test, pred_naive_bank_test_dt))
              precision    recall  f1-score   support

           0       0.94      0.85      0.89      7290
           1       0.34      0.59      0.43       958

    accuracy                           0.82      8248
   macro avg       0.64      0.72      0.66      8248
weighted avg       0.87      0.82      0.84      8248
```

Fig. 29. Decision Tree Classification Report

The above figure shows the classification report for Decision Tree algorithm from which we can understand the accuracy is 0.82.



```
print(classification_report(y_bank_test, pred_naive_bank_test_xgb))
              precision    recall  f1-score   support

           0       0.94      0.88      0.91      7290
           1       0.38      0.56      0.45       958

    accuracy                           0.84      8248
   macro avg       0.66      0.72      0.68      8248
weighted avg       0.87      0.84      0.86      8248
```

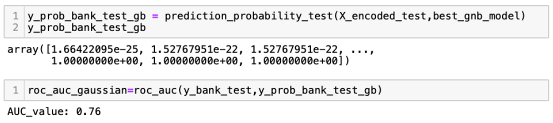
Fig. 30. XGBoost Classification Report

The above figure shows the classification report for the XGBoost algorithm from which we can understand the accuracy is 0.84.

Comparing the Classification report of three models, XGBoost algorithm has highest accuracy compared to Decision tree and Gaussian Naïve Bayes algorithms.

C) ROC-AUC Curve

ROC-AUC curve is another important indicator for model selection. ROC stands for receiver operating characteristic curve and AUC stands for Area under the curve. ROC is a probability curve and AUC represents the measure of separability. AUC measures the 2-dimensional Area under the ROC curve.



```
y_prob_bank_test_gb = prediction_probability_test(X_encoded_test,best_gnb_model)
y_prob_bank_test_gb
array([1.66422095e-25, 1.52767951e-22, 1.52767951e-22, ...,
       1.00000000e+00, 1.00000000e+00, 1.00000000e+00])
roc_auc_gaussian=roc_auc(y_bank_test,y_prob_bank_test_gb)
AUC_value: 0.76
```

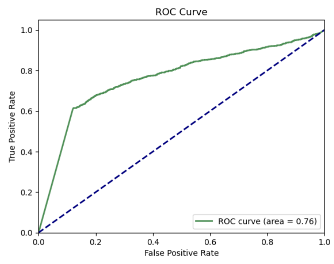
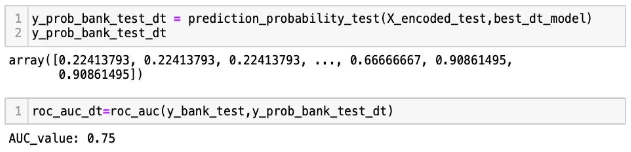


Fig. 31. Gaussian Naïve Bayes ROC-AUC Curve

The Fig.31 above shows the ROC-AUC curve for the Gaussian Naïve Bayes Classifier which shows the Area under the curve is 0.76.



```
y_prob_bank_test_dt = prediction_probability_test(X_encoded_test,best_dt_model)
y_prob_bank_test_dt
array([0.22413793, 0.22413793, 0.22413793, ..., 0.66666667, 0.90861495,
       0.90861495])
roc_auc_dt=roc_auc(y_bank_test,y_prob_bank_test_dt)
AUC_value: 0.75
```

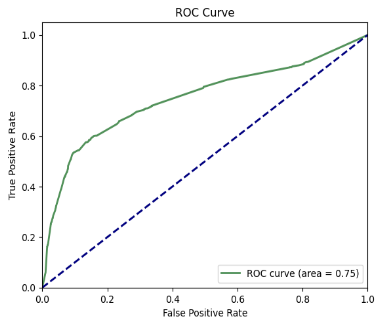
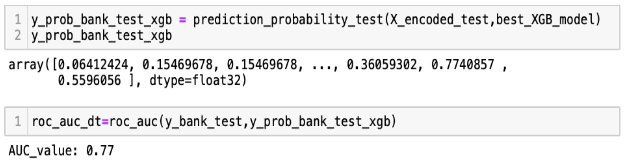


Fig. 32. Decision Tree ROC-AUC Curve

The Fig.32 shows the ROC-AUC curve for the Decision Tree algorithm which shows the Area under the curve is 0.75.



```
y_prob_bank_test_xgb = prediction_probability_test(X_encoded_test,best_XGB_model)
y_prob_bank_test_xgb
array([0.06412424, 0.15469678, 0.15469678, ..., 0.36059302, 0.7740857 ,
       0.5596056 ], dtype=float32)
roc_auc_dt=roc_auc(y_bank_test,y_prob_bank_test_xgb)
AUC_value: 0.77
```

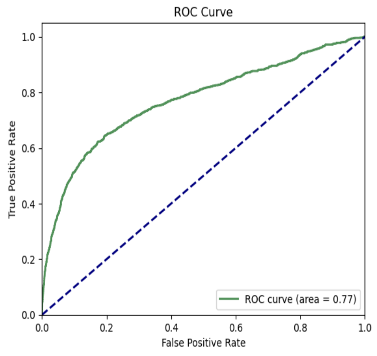


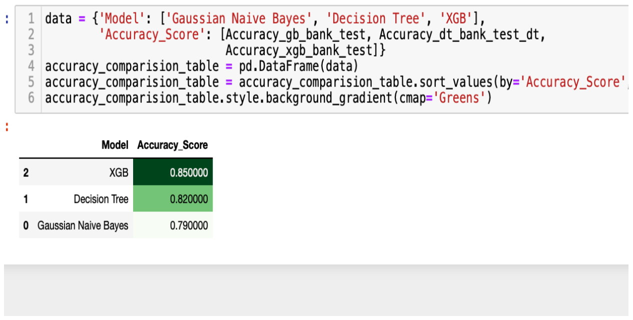
Fig. 33. XGBoost ROC-AUC Curve

The Fig.33 shows the ROC-AUC curve for the XGBoost algorithm which shows the Area under the curve is 0.77.

Comparing the ROC-Curves of three models, XGBoost algorithm has highest Area Under the Curve (AUC) value compared to Decision tree and Gaussian Naïve Bayes algorithms.

XI. ACCURACY COMPARISON

Accuracy comparison of all the three ML models is performed to determine the best model for this dataset.



```
data = {'Model': ['Gaussian Naive Bayes', 'Decision Tree', 'XGB'],
        'Accuracy_Score': [Accuracy_gb_bank_test, Accuracy_dt_bank_test_dt,
                           Accuracy_xgb_bank_test]}
accuracy_comparision_table = pd.DataFrame(data)
accuracy_comparision_table = accuracy_comparision_table.sort_values(by='Accuracy_Score'
accuracy_comparision_table.style.background_gradient(cmap='Greens')
```

	Model	Accuracy_Score
2	XGB	0.850000
1	Decision Tree	0.820000
0	Gaussian Naive Bayes	0.790000

Fig. 34. Accuracy Comparisons of ML Models

The Fig.34 shows the accuracy comparison of the various models used in this project. Looking at the figure the model prediction accuracy is as follows.

A) Gaussian Naïve Bayes - 0.79
B) Decision Trees – 0.82
C) XGB – 0.89

From the results obtained, it can be observed that XGB accuracy is higher compared to Decision tree and Gaussian Naïve Bayes models for this dataset. Thus, XGBoost is the best-fit model for this dataset.

XII. CONCLUSION

Machine Learning has become highly influential in the field of Banking. As From this project, the following conclusions can be made:

- This project effectively shows the application of Machine Learning Algorithms in the Banking Sector.
- It is able to demonstrate with high accuracy, the successful prediction of the likelihood of a specific customer signing up for a campaign.
- Banks can use this project to efficiently target the Banking Users that have a higher probability of signing up for the term deposit campaign thus having a more effective, efficient, and focus-driven process.

In terms of future work, this project can be utilized as a reference to process, analyze and predict other larger datasets in Banking. This project can also be extended to handle similar datasets in the field of Banking and payments.

XIII. REFERENCES

[1] J. Han, M. Kamber, and J. Pei, Data Mining Concepts and Techniques. Elsevier/Morgan Kaufmann, 2012.
[2] M. S. S. R. and M. Gurmendez, Mastering Machine Learning on AWS: Advanced Machine Learning in Python Using SageMaker, Apache Spark, and Tensorflow. Birmingham: Packt, 2019.
[3] S. Theodoridis, Machine Learning: A Bayesian and Optimization Perspective. London, UK: Academic Press, 2020.
[4] C. C.Aggarwal, Linear Algebra and Optimization for Machine Learning: A Textbook. S.l.: Springer Nature, 2021.
[5] A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol (CA): O'Reilly Media, 2023.
[6] "Bank marketing," UCI Machine Learning Repository, https://archive.ics.uci.edu/dataset/222/bank+marketing# (accessed Dec. 1, 2023).