Predicting Term Deposit Subscriptions Using ML

BANK MARKETING

GROUP NUMBER 8

BHAVAN'S VIVEKANANDA COLLEGE

R.SREE VARDHINI TULASI | PULIGADDA VENKATA KRISHNA KIREETI

PULIGADDA VENKATA SAI KIRAN | RAMIDI HARISHWAR REDDY

Abstract

The project aims to develop a predictive model for bank marketing, specifically targeting the likelihood of individuals subscribing to term deposits. This study investigates various machine learning algorithms, including Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Machines, Bagging, Boosting and ANN to enhance the accuracy of predictions regarding customer behavior.

Objective

Predict whether a specific customer will subscribe to a term deposit based on their characteristics for a bank marketing classification project.

Content

Introduction

The banking sector reflects economic health, playing a key role in supporting growth by managing financial resources and effectively engaging customers through targeted marketing.

Technological advancements: Automation and digital tools are transforming bank marketing, enabling more efficient customer targeting, personalized services, and improved campaign effectiveness.

Literature review

Literature Review - 1

“Prediction of Term Deposit in Bank: Using Logistic Model”

2022

Enjing Jiang
Zihao Wang
Jiaying Zhao

Telemarketing remains essential in direct marketing, urging accurate predictions. Machine learning enables banks to identify target customers, boosting time deposit growth by analyzing Portuguese bank data from Kaggle.

After data preprocessing and correlation analysis, logistic regression and decision tree comparisons showed logistic regression’s accuracy. AUC scores validated model effectiveness, aiding banks in targeted, accurate marketing efforts.

https://www.researchgate.net/publication/366302440_Prediction_of_Term_Deposit_in_Bank_Using_Logistic_Model

Literature Review - 2

“Machine Learning Performance on Predicting Banking Term Deposit”
2022

Sakya Tuan

Financial deposits are essential amidst global economic crises, yet banks struggle to identify ideal customers. Deep neural networks can analyze key customer attributes to enhance prediction accuracy.

Models like GRU, BiLSTM, and SimpleRNN are used to predict customer suitability for deposits, with GRU achieving 90.08% accuracy, aiding banks in confirming deposit likelihood.

https://www.researchgate.net/publication/379954229_Machine_Learning_Performance_on_Predicting_Banking_Term_Deposit

DATA PRE-PROCESSING

Data

Dataset: Our Dataset consists of 21 variables and 41189 records

Source: https://archive.ics.uci.edu/dataset/222/bank+marketing

Variables:

- Continuous variables-
  1. Age
  2. Duration
  3. Campaign
  4. pdays
  5. Previous
  6. Employment Variation Rate
  7. Consumer Price Index
  8. Consumer Confidence Index
  9. Euro Interbank Offered Rate - 3 Month
  10. Number of Employees

- Categorical variables -
  1. Job
  2. Martial
  3. Education
  4. Default
  5. Housing
  6. Loan
  7. Contact
  8. Month
  9. Day of week
  10. previous marketing campaign
  11. Term deposit(Y)

Age	Job	Martial	Education	Default	Housing	Loan	Contact	Month	Day_of_week	Duration	Campaign	pdays	Previous	poutcome	emp_var_rate	cons_price_idx
37	admin.	married	university.	no	yes	no	cellular	jul	wed	353	1	999	0	nonexister	1.4	93.918
45	technician	married	profession	no	yes	no	cellular	jul	wed	95	1	999	0	nonexister	1.4	93.918
34	blue-collar	married	basic.9y	no	yes	no	cellular	jul	wed	477	1	999	0	nonexister	1.4	93.918
38	blue-collar	married	unknown	no	yes	no	cellular	jul	wed	131	1	999	0	nonexister	1.4	93.918
33	admin.	married	high.schoc	no	yes	no	cellular	jul	wed	134	1	999	0	nonexister	1.4	93.918
34	blue-collar	married	basic.9y	no	yes	no	cellular	jul	wed	85	1	999	0	nonexister	1.4	93.918
33	admin.	married	high.schoc	no	no	no	cellular	jul	wed	254	1	999	0	nonexister	1.4	93.918
45	technician	married	profession	no	no	no	cellular	jul	wed	394	1	999	0	nonexister	1.4	93.918
38	blue-collar	married	unknown	no	no	no	cellular	jul	wed	618	1	999	0	nonexister	1.4	93.918
54	services	married	high.schoc	no	no	no	telephone	jul	wed	250	1	999	0	nonexister	1.4	93.918
54	services	married	high.schoc	no	no	no	cellular	jul	wed	82	1	999	0	nonexister	1.4	93.918
35	manageme	married	university.	no	no	no	cellular	jul	wed	182	1	999	0	nonexister	1.4	93.918
25	admin.	married	high.schoc	no	no	no	cellular	jul	wed	95	1	999	0	nonexister	1.4	93.918
31	services	married	basic.9y	no	yes	no	cellular	jul	wed	537	1	999	0	nonexister	1.4	93.918
57	blue-collar	married	basic.6y	no	yes	no	cellular	jul	wed	64	1	999	0	nonexister	1.4	93.918
31	services	married	basic.9y	no	no	no	cellular	jul	wed	679	1	999	0	nonexister	1.4	93.918
29	blue-collar	married	high.schoc	no	yes	no	telephone	jul	wed	42	1	999	0	nonexister	1.4	93.918

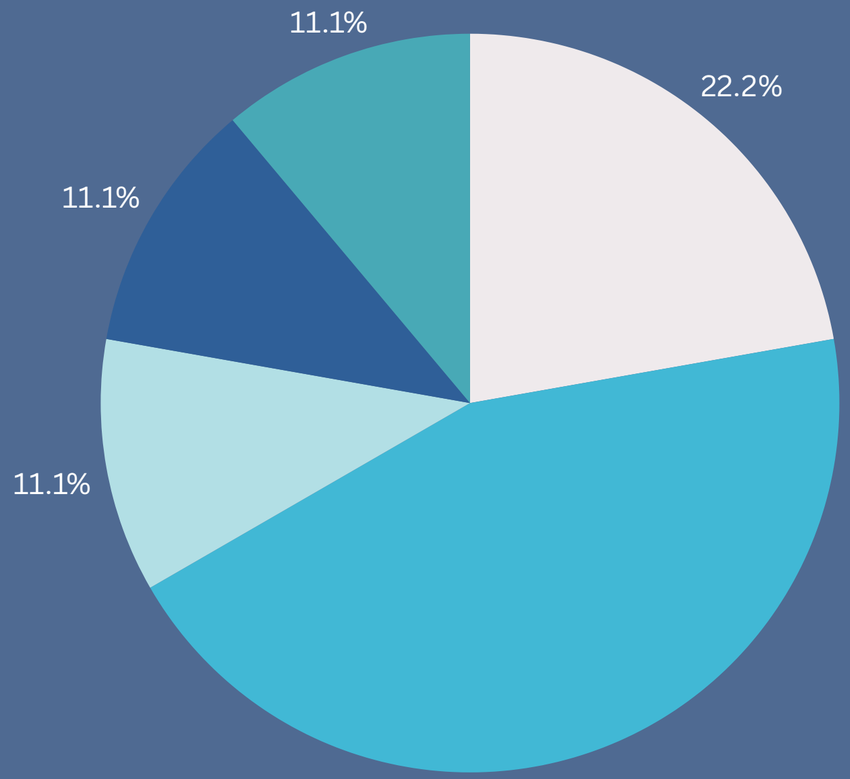
Data Cleaning :

- The "day of week" column has been removed from the dataset as it is unnecessary for further analysis.
- Examined the dataset for both missing and unique values, it was found to be complete with no missing entries.

- The values within the columns job, education and month have been consolidated based on their similarities, resulting in the formation of distinct categories.
- Then encoding and dummification of the categorical columns is done.

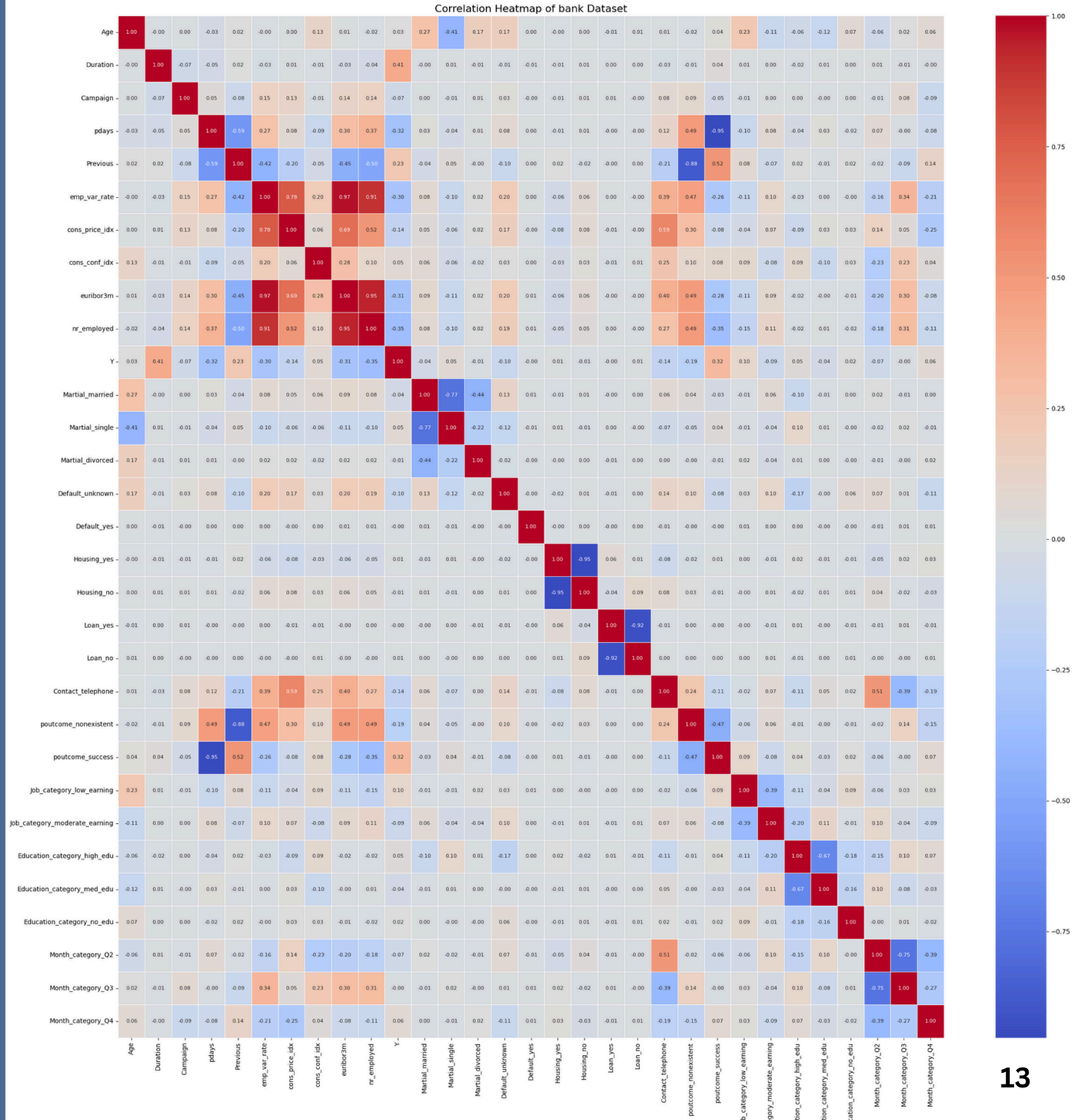
```
Age                                int64
Duration                           int64
Campaign                           int64
pdays                              int64
Previous                           int64
emp_var_rate                     float64
cons_price_idx                   float64
cons_conf_idx                    float64
euribor3m                        float64
nr_employed                      float64
Y                                  int64
Martial_married                    int64
Martial_single                     int64
Martial_divorced                   int64
Default_unknown                    int64
Default_yes                        int64
Housing_yes                        int64
Housing_no                         int64
Loan_yes                           int64
Loan_no                            int64
Contact_telephone                  int64
poutcome_nonexistent               int64
poutcome_success                   int64
Job_category_low_earning           int64
Job_category_moderate_earning      int64
Education_category_high_edu        int64
Education_category_med_edu         int64
Education_category_no_edu          int64
Month_category_Q2                  int64
Month_category_Q3                  int64
Month_category_Q4                  int64
dtype: object
```

EXPLORATORY DATA ANALYSIS



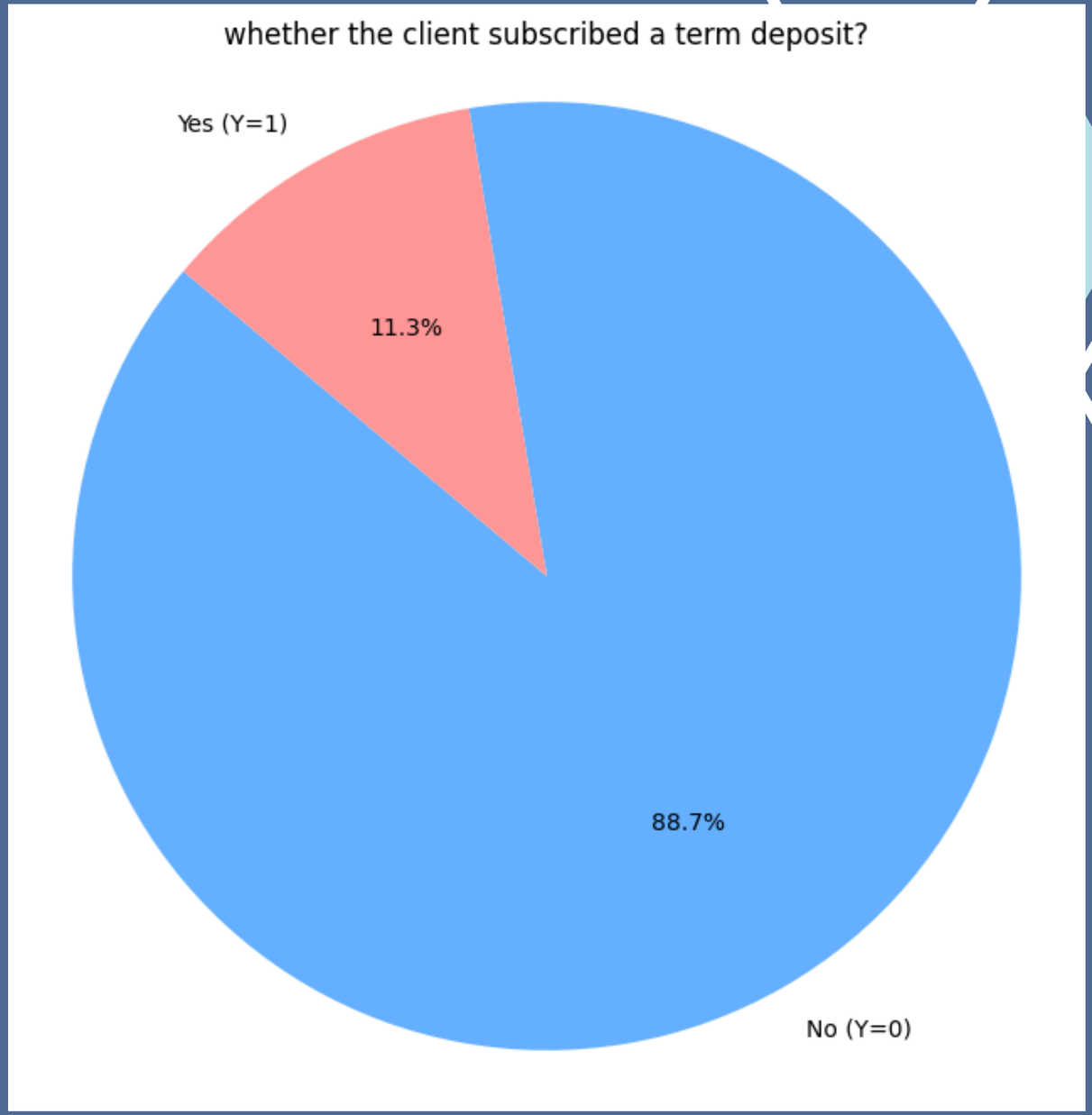
Correlation Matrix

Red indicates a strong positive correlation. Blue indicates a strong negative correlation. Lighter colors indicate weaker correlations. Diagonal Line: The diagonal from the top left to the bottom right shows perfect positive correlations (each variable compared to itself).

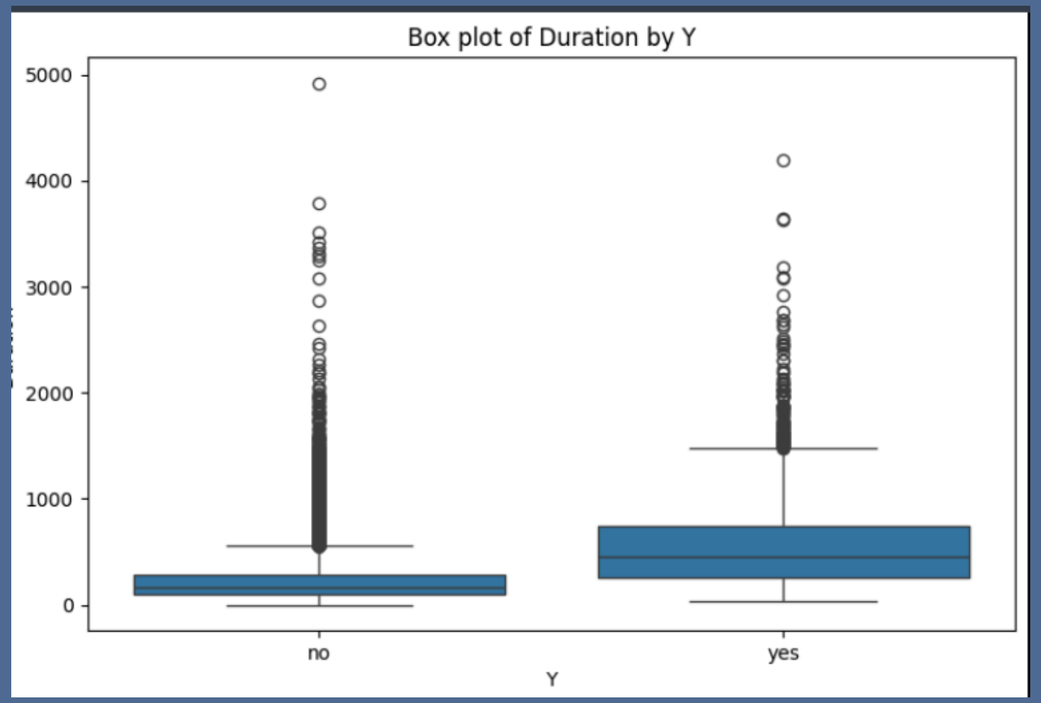


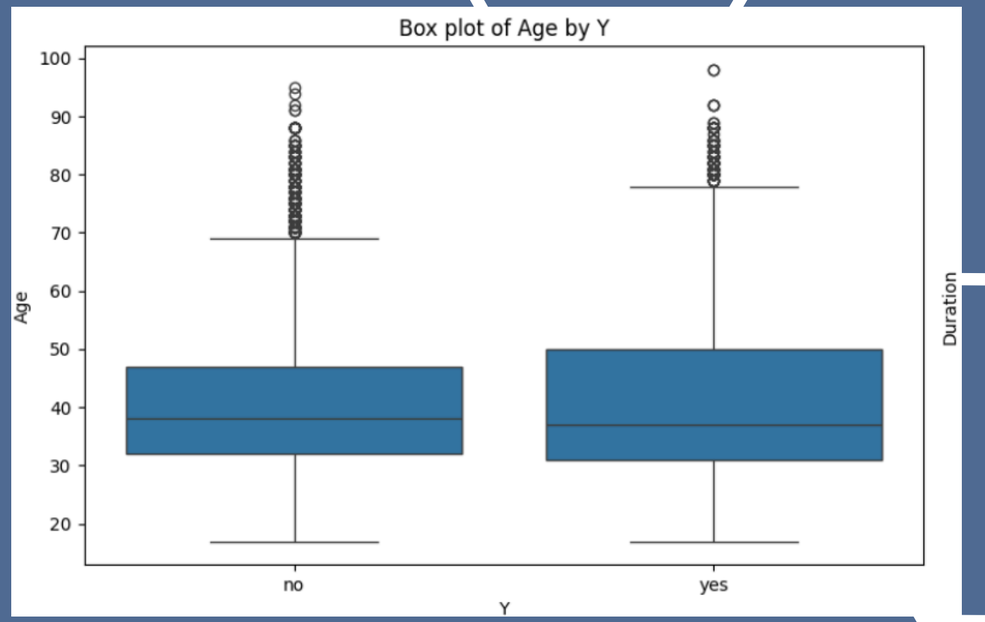
Pie Chart

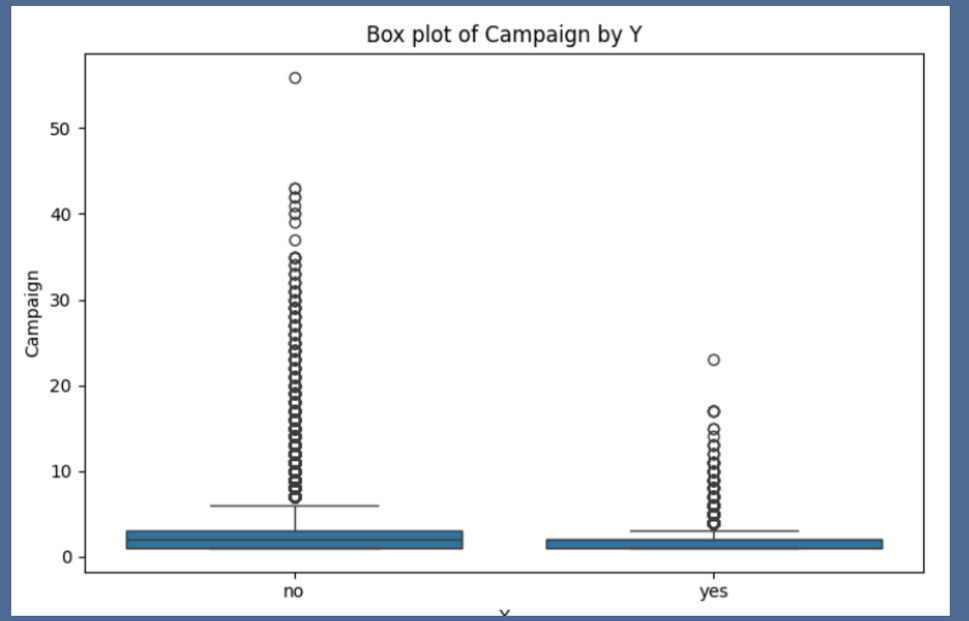
No (Y=0): This section is colored blue and represents the proportion of instances where the target variable (Y) is 0.
Yes (Y=1): This section is colored red the proportion of instances where the target variable (Y) is 1.



Box Plot

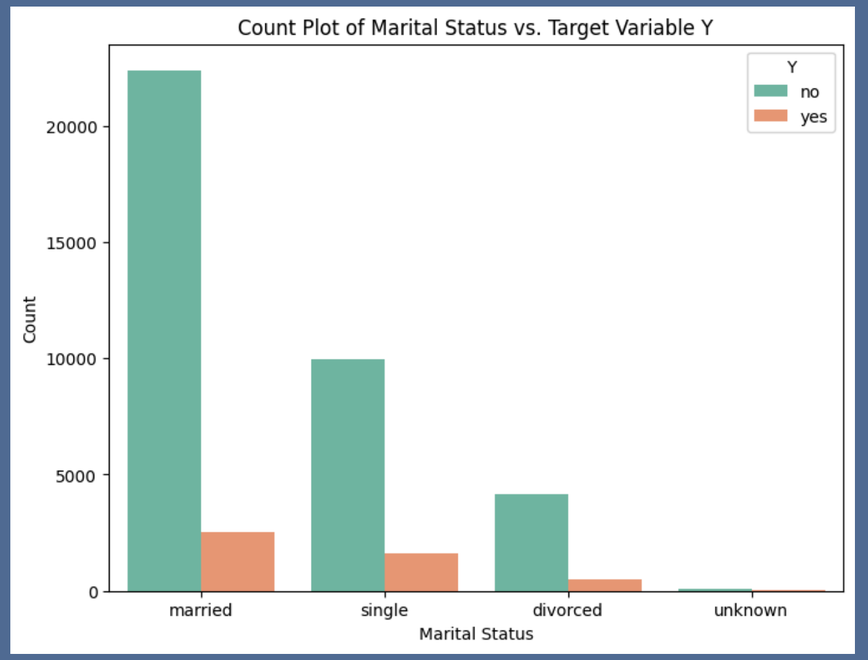


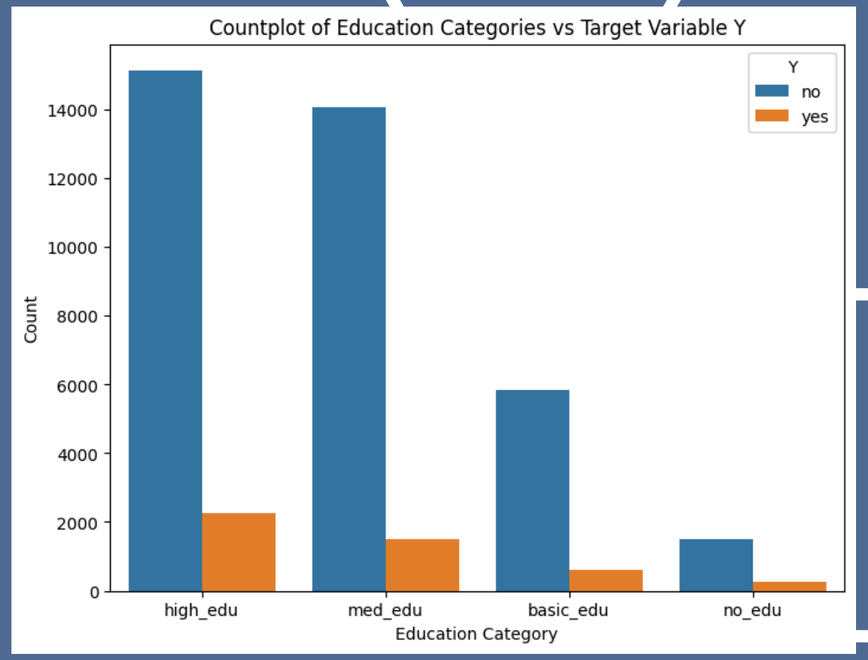




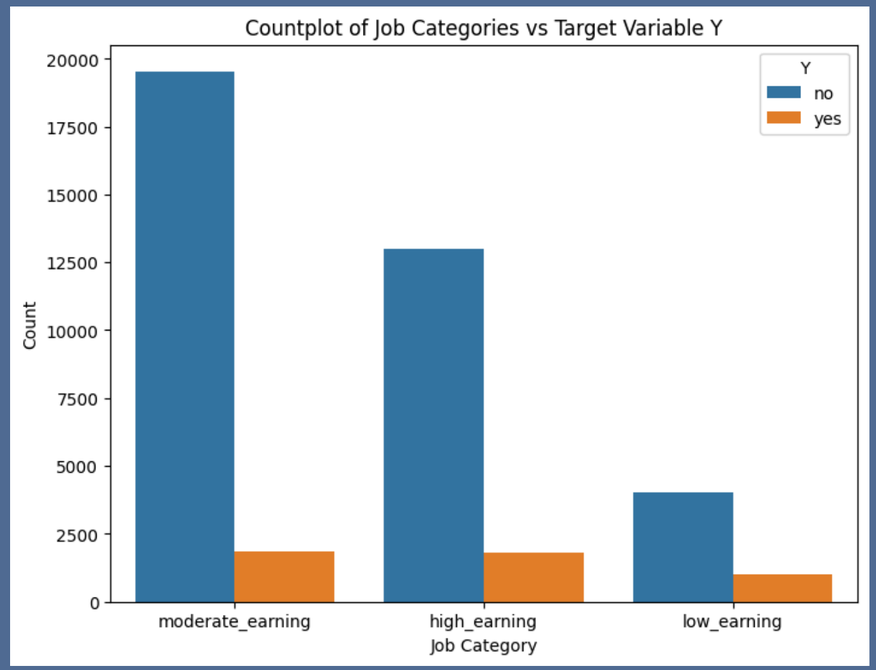
These are box plots of 'Duration', 'Age', and 'Campaign' segmented by the target variable 'Y'.

Count Plot



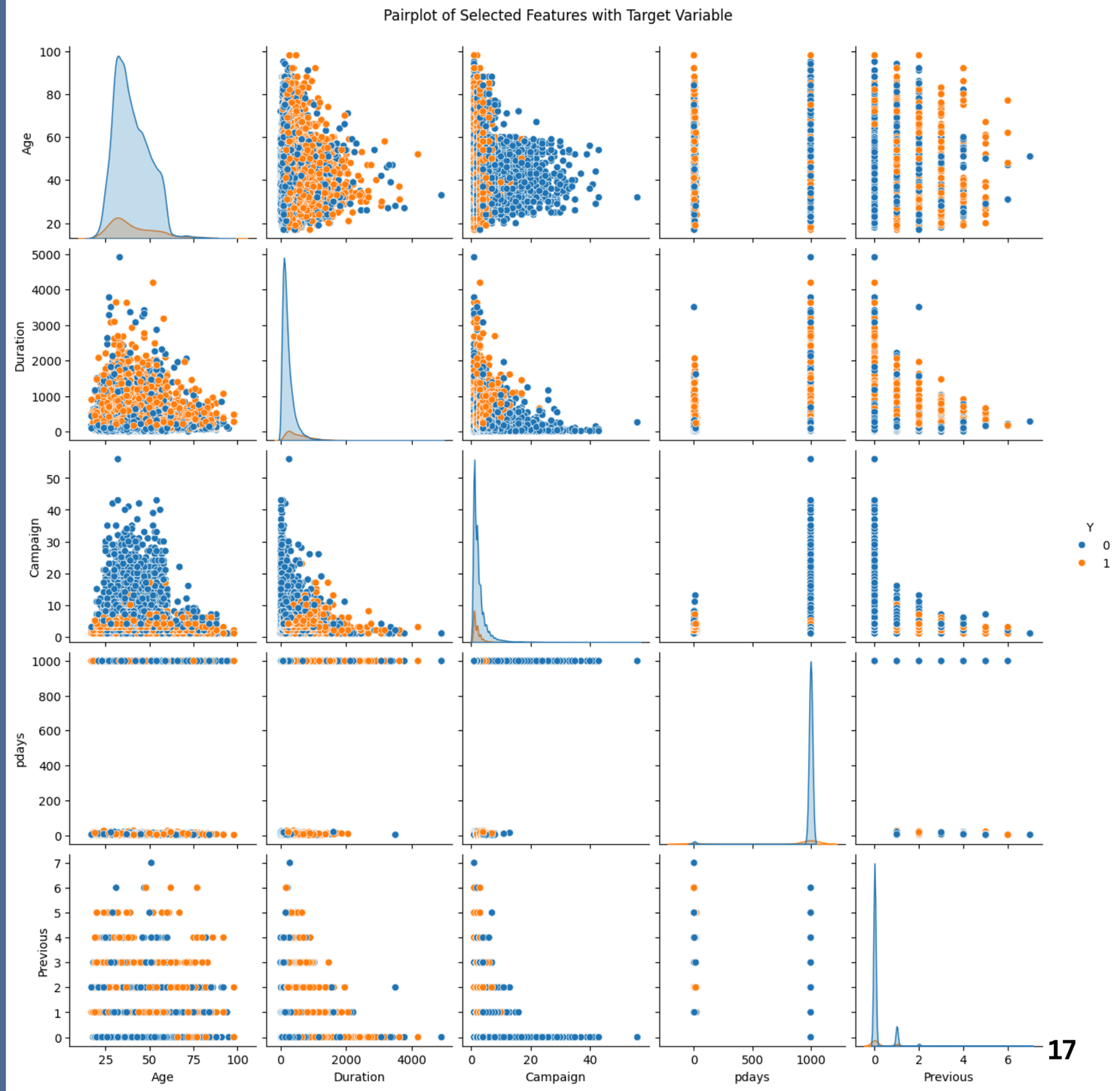


The three bar plots for education, marital status, and job reveal that most individuals in each category did not respond positively to the campaign.



Pairplots

The pair plot displays the relationships between Age, Duration, Campaign, pdays, and Previous with the target variable (Y). It helps visualize how these features correlate with each other and with the target.



Multicollinearity Check

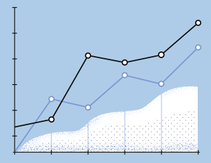
After removing the highest VIF(>3) columns

	variables	VIF
0	Age	21.7
1	Duration	2.0
2	Campaign	1.9
3	pdays	346.2
4	Previous	6.7
5	emp_var_rate	61.4
6	cons_price_idx	39204.5
7	cons_conf_idx	166.1
8	euribor3m	485.4
9	nr_employed	46880.8
10	Martial_married	322.9
11	Martial_single	151.2
12	Martial_divorced	59.9
13	Default_unknown	1.4
14	Default_yes	1.0
15	Housing_yes	inf
16	Housing_no	inf
17	Loan_yes	inf
18	Loan_no	inf
19	Contact_telephone	5.1
20	poutcome_nonexistent	37.7
21	poutcome_success	11.7
22	Job_category_low_earning	1.5
23	Job_category_moderate_earning	2.7
24	Education_category_high_edu	4.5
25	Education_category_med_edu	3.7
26	Education_category_no_edu	1.3
27	Month_category_Q2	42.8
28	Month_category_Q3	29.1
29	Month_category_Q4	12.0

	variables	VIF
0	Duration	1.9
1	Campaign	1.8
2	Previous	1.8
3	emp_var_rate	2.0
4	Martial_single	1.5
5	Martial_divorced	1.2
6	Default_unknown	1.3
7	Default_yes	1.0
8	Housing_no	1.8
9	Loan_yes	1.2
10	Contact_telephone	2.8
11	poutcome_success	1.5
12	Job_category_low_earning	1.4
13	Job_category_moderate_earning	2.2
14	Education_category_high_edu	2.8
15	Education_category_med_edu	2.5
16	Education_category_no_edu	1.2
17	Month_category_Q3	2.7
18	Month_category_Q4	1.3

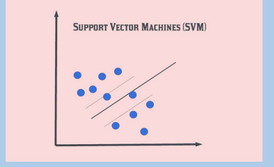
MACHINE LEARNING ALGORITHMS

ML Algorithms

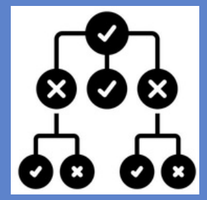


01.
Logistic Regression

02.
K-Nearest Neighbors (KNN)

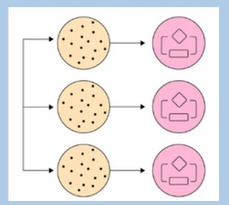


03.
Support Vector Machine (SVM)

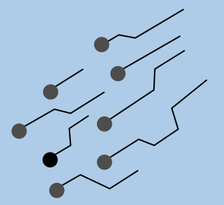


04.
Decision Tree

06.
Boosting



05.
Random Forest



06.
ANN

Model 1: With all features | Model 2: After removing multi-collinear variables

70:30 Train-Test split

Algorithms	Model - 1 (Accuracy)	Model - 2 (Accuracy)
Logistic Regression	0.906	0.903
KNN	0.909	0.893
SVM	0.894	0.893
Decision Tree	0.910	0.907
Random Forest	0.911	0.896
XGBoost	0.912	0.906
AdaBoost	0.906	0.903
ANN	0.886	0.899

80:20 Train-Test split

Algorithms	Model - 1 (Accuracy)	Model - 2 (Accuracy)
Logistic Regression	0.910	0.903
KNN	0.912	0.896
SVM	0.901	0.889
Decision Tree	0.915	0.909
Random Forest	0.914	0.901
XGBoost	0.914	0.909
AdaBoost	0.907	0.901
ANN	0.885	0.908

60:40 Train-Test split

Algorithms	Model - 1 (Accuracy)	Model - 2 (Accuracy)
Logistic Regression	0.909	0.911
KNN	0.911	0.896
SVM	0.899	0.898
Decision Tree	0.913	0.908
Random Forest	0.913	0.901
XGBoost	0.913	0.909
AdaBoost	0.907	0.907
ANN	0.889	0.907

75:25 Train-Test split

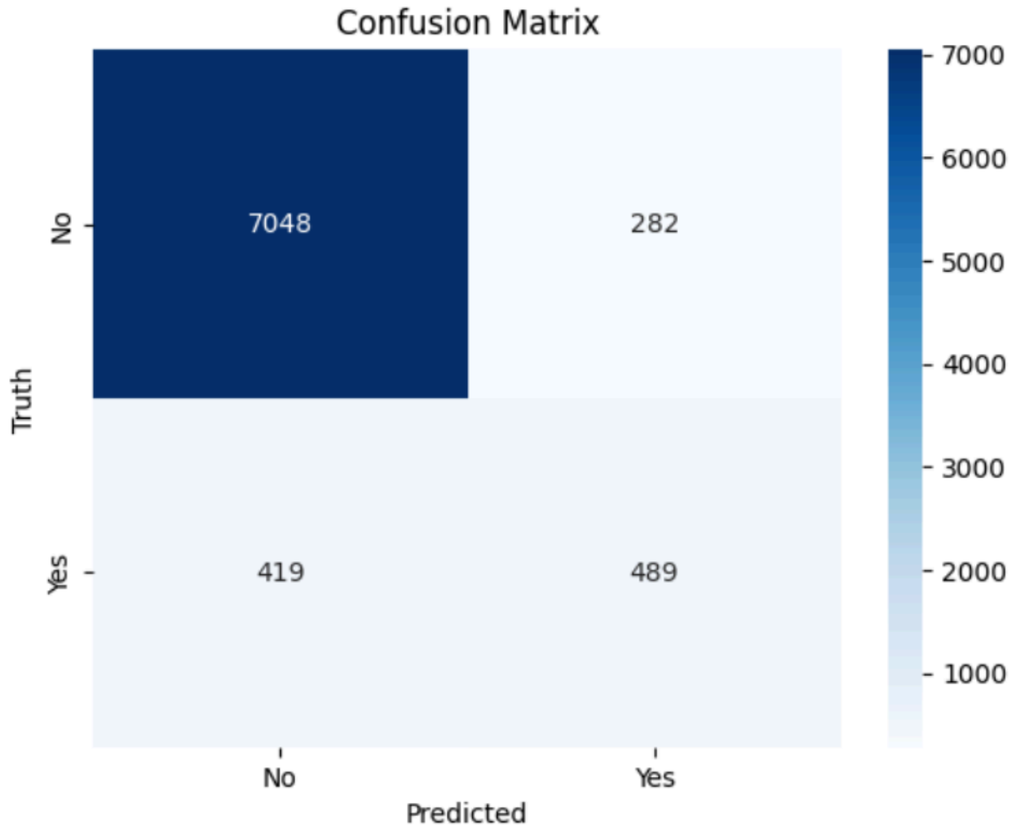
Algorithms	Model - 1 (Accuracy)	Model - 2 (Accuracy)
Logistic Regression	0.910	0.908
KNN	0.910	0.900
SVM	0.897	0.896
Decision Tree	0.913	0.908
Random Forest	0.890	0.904
XGBoost	0.913	0.911
AdaBoost	0.909	0.907
ANN	0.887	0.909

Algorithm Comparison

Algorithms	Model 1 (Accuracy)
Logistic Regression	0.910
KNN	0.912
SVM	0.901
Decision Tree	0.915
Random Forest	0.914
XGBoost	0.914
AdaBoost	0.907
ANN	0.855

Confusion matrix for the decision tree of 80:20 ratio

```
array([[7048,  282],
       [ 419,  489]])
```

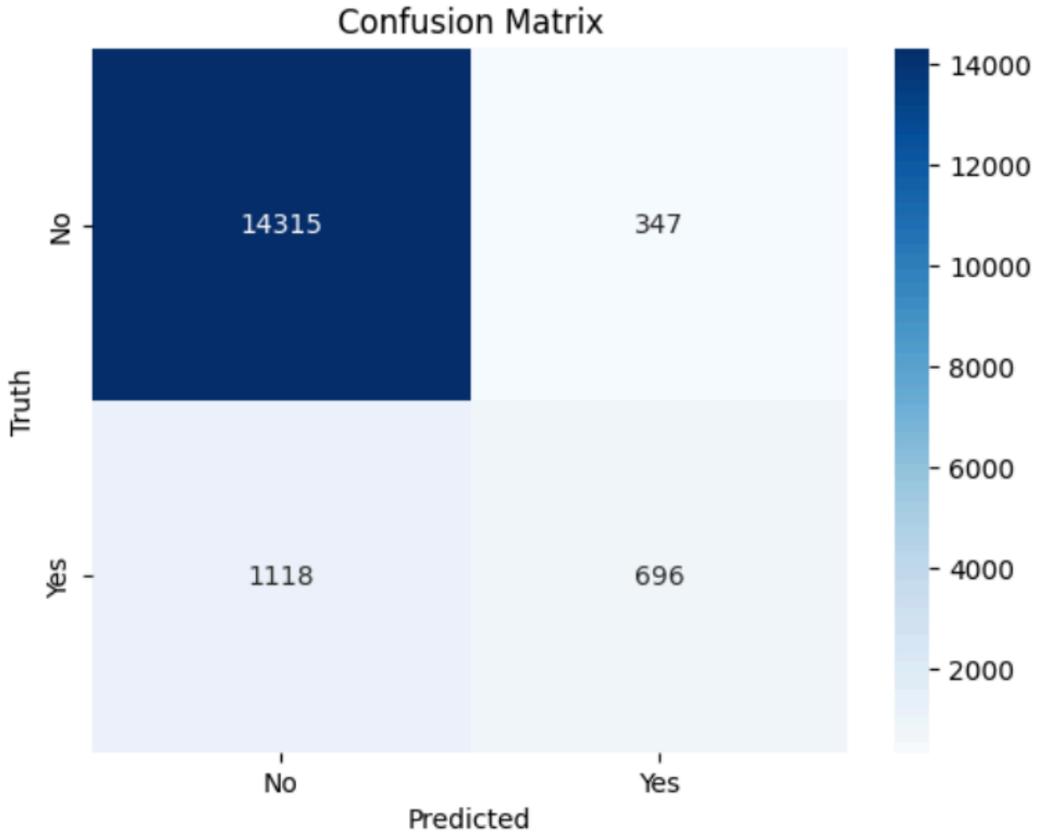


Algorithm Comparison

Algorithms	Model 2 (Accuracy)
Logistic Regression	0.911
KNN	0.896
SVM	0.898
Decision Tree	0.908
Random Forest	0.909
XGBoost	0.911
AdaBoost	0.907
ANN	0.907

Confusion matrix for the Logistic regression for 60:40 ratio

```
array([[14315,   347],
       [ 1118,   696]])
```



SUMMARY

SUMMARY

The primary objective of this research is to identify the most effective machine learning techniques for predicting term deposit subscriptions in the banking sector.

Among the classification models evaluated, the Decision tree model stands out by achieving a notable accuracy of 91.5%, surpassing the performance of other models considered in this study for predicting term deposit subscriptions in the banking sector.

In the model-2 the Logistic Regression model emerged as the best performer, achieving an accuracy of 91.1%. Although slightly lower than Decision tree's accuracy in Model-1

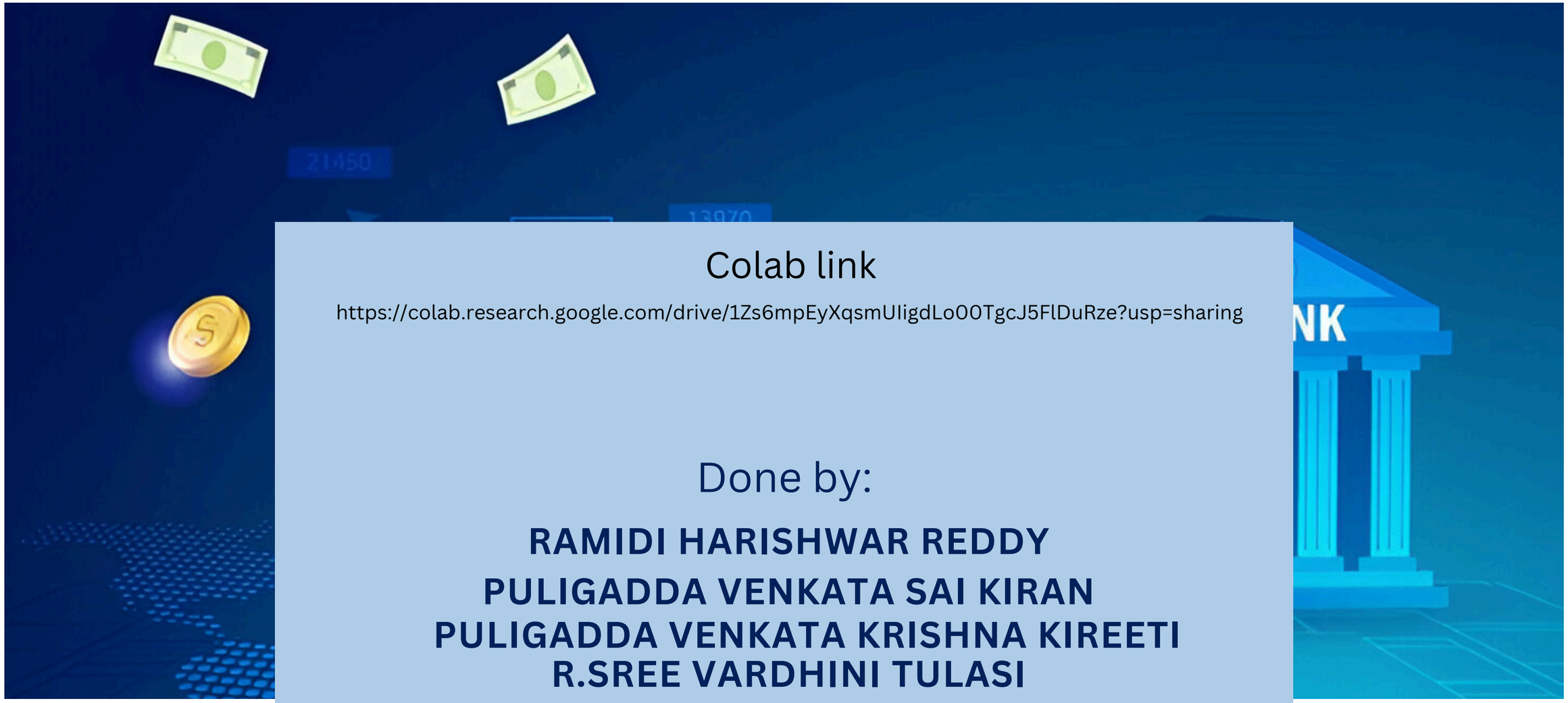
FUTURE SCOPE

In the context of bank marketing, data preprocessing plays a crucial role in preparing raw financial data—such as loans, deposits, and savings—for analysis and decision-making. A comprehensive analysis based on these factors can lead to insights that help banks enhance their customer targeting strategies, improve marketing campaigns, and optimize product offerings.

WORK DISTRIBUTION

PULIGADDA VENKATA SAI KIRAN	Collect information about Bank Marketing term deposits & Literature Review
PULIGADDA VENKATA KRISHNA KIREETI	Data Preprocessing
RAMIDI HARISHWAR REDDY	Exploratory Data Analysis
R.SREE VARDHINI TULASI	Implement Machine Learning Algorithms

THANK YOU



Colab link

https://colab.research.google.com/drive/1Zs6mpEyXqsmUIigdLo00TgcJ5FlDuRze?usp=sharing

Done by:

RAMIDI HARISHWAR REDDY

PULIGADDA VENKATA SAI KIRAN

PULIGADDA VENKATA KRISHNA KIREETI

R.SREE VARDHINI TULASI

APPENDIX

CONTENT

	Age	Job	Martial	Education	Default	Housing	Loan	\
0	37	admin.	married	university.degree	no	yes	no	
1	45	technician	married	professional.course	no	yes	no	
2	34	blue-collar	married	basic.9y	no	yes	no	
3	38	blue-collar	married	unknown	no	yes	no	
4	33	admin.	married	high.school	no	yes	no	

	Contact	Month	Day_of_week	...	Campaign	pdays	Previous	poutcome	\
0	cellular	jul	wed	...	1	999	0	nonexistent	
1	cellular	jul	wed	...	1	999	0	nonexistent	
2	cellular	jul	wed	...	1	999	0	nonexistent	
3	cellular	jul	wed	...	1	999	0	nonexistent	
4	cellular	jul	wed	...	1	999	0	nonexistent	

	emp_var_rate	cons_price_idx	cons_conf_idx	euribor3m	nr_employed	Y
0	1.4	93.918	-42.7	4.962	5228.1	no
1	1.4	93.918	-42.7	4.962	5228.1	no
2	1.4	93.918	-42.7	4.962	5228.1	no
3	1.4	93.918	-42.7	4.962	5228.1	no
4	1.4	93.918	-42.7	4.962	5228.1	no

[5 rows x 21 columns]

NO.OF COLUMNS

```
data.columns #no. of columns
```

```
Index(['Age', 'Job', 'Martial', 'Education', 'Default', 'Housing', 'Loan',
       'Contact', 'Month', 'Day_of_week', 'Duration', 'Campaign', 'pdays',
       'Previous', 'poutcome', 'emp_var_rate', 'cons_price_idx',
       'cons_conf_idx', 'euribor3m', 'nr_employed', 'Y'],
      dtype='object')
```

DATA FRAME STRUCTURE AND DATA TYPES

```
data.info() #summary of DataFrame structure and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             41188 non-null  int64  
 1   Job             41188 non-null  object 
 2   Martial         41188 non-null  object 
 3   Education       41188 non-null  object 
 4   Default         41188 non-null  object 
 5   Housing         41188 non-null  object 
 6   Loan            41188 non-null  object 
 7   Contact         41188 non-null  object 
 8   Month           41188 non-null  object 
 9   Day_of_week     41188 non-null  object 
 10  Duration        41188 non-null  int64  
 11  Campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  Previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp_var_rate    41188 non-null  float64
 16  cons_price_idx  41188 non-null  float64
 17  cons_conf_idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr_employed     41188 non-null  float64
 20  Y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

NULL VALUES

```
data.isna().sum() #no. of null values
```

	0
Age	0
Job	0
Martial	0
Education	0
Default	0
Housing	0
Loan	0
Contact	0
Month	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

CATEGORIZATION

```
# Define a function to categorize the job types
def categorize_job(job):
    if job in ['management', 'entrepreneur', 'admin.']:
        return 'high_earning'
    elif job in ['technician','blue-collar', 'services','self-emp
        return 'moderate_earning'
    elif job in [ 'housemaid', 'retired', 'unemployed', 'student'
        return 'low_earning'


# Apply the categorization to the 'Job' column
data['Job_category'] = data['Job'].apply(categorize_job)

# Display the first few rows with the new 'Job_category' column
data[['Job', 'Job_category']].head()
```

	Job	Job_category
0	admin.	high_earning
1	technician	moderate_earning
2	blue-collar	moderate_earning
3	blue-collar	moderate_earning
4	admin.	high_earning

```
# Define a function to categorize the education types
def categorize_education(education):
    if education in ['unknown', 'illiterate']:
        return 'no_edu'
    elif education in ['basic.4y', 'basic.6y']:
        return 'basic_edu'
    elif education in ['high.school', 'basic.9y']:
        return 'med_edu'
    elif education in ['professional.course', 'university.degree
        return 'high_edu'


# Apply the categorization to the 'Education' column
data['Education_category'] = data['Education'].apply(categorize_

# Display the first few rows with the new 'Education_category' c
data[['Education', 'Education_category']].head()
```

	Education	Education_category
0	university.degree	high_edu
1	professional.course	high_edu
2	basic.9y	med_edu
3	unknown	no_edu
4	high.school	med_edu

```
# Define a function to categorize the months
def categorize_month(month):
    if month in ['jan', 'feb', 'mar']:
        return 'Q1'
    elif month in ['apr', 'may', 'jun']:
        return 'Q2'
    elif month in ['jul', 'aug', 'sep']:
        return 'Q3'
    elif month in ['oct', 'nov', 'dec']:
        return 'Q4'


# Apply the categorization to the 'Month' column
data['Month_category'] = data['Month'].apply(categorize_month)

# Display the first few rows with the new 'Month_category' colum
data[['Month', 'Month_category']].head()
```

	Month	Month_category
0	jul	Q3
1	jul	Q3
2	jul	Q3
3	jul	Q3
4	jul	Q3

DATA ENCODING

```
data_org.columns
```

```
Index(['Age', 'Martial', 'Default', 'Housing', 'Loan', 'Contact', 'Duration',
       'Campaign', 'pdays', 'Previous', 'poutcome', 'emp_var_rate',
       'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'Y',
       'Job_category', 'Education_category', 'Month_category'],
      dtype='object')
```

```
data_encoded.columns
```

```
Index(['Age', 'Duration', 'Campaign', 'pdays', 'Previous', 'emp_var_rate',
       'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'Y',
       'Martial_married', 'Martial_single', 'Martial_divorced',
       'Default_unknown', 'Default_yes', 'Housing_yes', 'Housing_no',
       'Loan_yes', 'Loan_no', 'Contact_telephone', 'poutcome_nonexistent',
       'poutcome_success', 'Job_category_low_earning',
       'Job_category_moderate_earning', 'Education_category_high_edu',
       'Education_category_med_edu', 'Education_category_no_edu',
       'Month_category_Q2', 'Month_category_Q3', 'Month_category_Q4'],
      dtype='object')
```

```
# Reorder the categorical columns to ensure 'unknown' is first
data_org['Loan'] = pd.Categorical(data_org['Loan'], categories=['unknown', 'yes', 'no'], ordered=True)
data_org['Martial'] = pd.Categorical(data_org['Martial'], categories=['unknown', 'married', 'single', 'divorced'], ordered=True)
data_org['Housing'] = pd.Categorical(data_org['Housing'], categories=['unknown', 'yes', 'no'], ordered=True)

# List of categorical columns including your custom categories
categorical_columns = [
    'Martial', 'Default', 'Housing',
    'Loan', 'Contact', 'poutcome', 'Job_category', 'Education_category', 'Month_category'
]

# Apply dummy variable encoding (one-hot encoding with drop_first=True)
data_encoded = pd.get_dummies(data_org, columns=categorical_columns, drop_first=True)

# Convert 'yes' and 'no' to 1 and 0 respectively for remaining columns
data_encoded = data_encoded.replace({'yes': 1, 'no': 0})


# Display the first few rows of the dummy variable encoded dataset
print(data_encoded.head())
```

```
# Identify boolean columns
boolean_columns = data_encoded.select_dtypes(include='bool').columns

# Convert only boolean columns to 0/1 (int)
data_encoded[boolean_columns] = data_encoded[boolean_columns].astype(int)

# Display the first few rows and check the dtypes to confirm only boolean columns are convert
print(data_encoded.dtypes)
print(data_encoded.head())
```

EDA

```
# Calculate the correlation matrix
corr_matrix = data_encoded.corr()

# Set a larger plot size
plt.figure(figsize=(25, 25))

# Create the heatmap with smaller annotation font size
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5,
            annot_kws={"size": 8})
#heatmap - coorelation by using an heatmap
#annot=True: Annotates each cell with the correlation coefficient.
#cmap='coolwarm': Uses the 'coolwarm' colormap to color the heatmap.
#fmt='.2f': Formats the annotation text to two decimal places.
#linewidths=0.5: Sets the width of the lines that divide the cells.
#annot_kws={"size": 8}: Sets the font size of the annotations to 8


# Add a title with increased font size for better visibility
plt.title('Correlation Heatmap of bank Dataset', fontsize=16)
```

```
# Set the figure size
plt.figure(figsize=(15, 10))

# Create box plots for 'Age', 'Duration', and 'Campaign' with respect to 'Y'
numeric_columns = ['Age', 'Duration', 'Campaign']
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(2, 2, i)  # Create a subplot for each variable
    sns.boxplot(x='Y', y=col, data=data)
    plt.title(f'Box plot of {col} by Y')

# Adjust layout
plt.tight_layout()

# Show the plot
plt.show()
```

```
# Count the values of 'Y'
y_counts = data_encoded['Y'].value_counts()

# Create labels and sizes for the pie chart
labels = ['No (Y=0)', 'Yes (Y=1)']
sizes = y_counts.values

# Create the pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=['#66b3ff', '#ff9999'])
#'%1.1f%%' specifies that the labels should be formatted as percentages with one decimal place.
#Setting startangle=140 rotates the start of the pie chart by 140 degrees counterclockwise.

# Customize the plot
plt.title('whether the client subscribed a term deposit?')
plt.axis('equal')  # Equal aspect ratio ensures that the pie is drawn as a circle.

# Show the plot
plt.show()
```

```
# Create a countplot showing the distribution of Education_category against Y
plt.figure(figsize=(8,6))
sns.countplot(x='Education_category', hue='Y', data=data, order=['high_edu', 'med_edu',

# Add title and labels
plt.title('Countplot of Education Categories vs Target Variable Y')
plt.xlabel('Education Category')
plt.ylabel('Count')

# Show the plot
plt.show()
```

```
# Create the count plot
plt.figure(figsize=(8, 6))
sns.countplot(x='Martial', hue='Y', data=data, palette='Set2')

# Customize the plot
plt.title('Count Plot of Marital Status vs. Target Variable Y')
plt.xlabel('Marital Status')
plt.ylabel('Count')
plt.legend(title='Y')

# Show the plot
plt.show()
```

SPLITTING THE DATASET

Splitting in different ratios before feature selection

```
from sklearn.model_selection import train_test_split
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.30, train_size=0.70)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.20, train_size=0.80)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.40, train_size=0.60)
X_train4, X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.25, train_size=0.75)
```


Splitting the dataset after the feature selection

```
from sklearn.model_selection import train_test_split
X_train5, X_test5, y_train5, y_test5 = train_test_split(X_new, y, test_size=0.30, train_size=0.70)
X_train6, X_test6, y_train6, y_test6 = train_test_split(X_new, y, test_size=0.20, train_size=0.80)
X_train7, X_test7, y_train7, y_test7 = train_test_split(X_new, y, test_size=0.40, train_size=0.60)
X_train8, X_test8, y_train8, y_test8 = train_test_split(X_new, y, test_size=0.25, train_size=0.75)
```

ALGORIHMS USED:

Algorithms
Logistic Regression
KNN
SVM
Decision Tree
Random Forest
XGBoost
AdaBoost

Logistic regression

LOGISTIC REGRESSION

```
#70 - 30

# fit a logistic regression model and store the class predictions
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9)

logreg.fit(X_train5, y_train5)
predictions5 = logreg.predict(X_test5)
print(predictions5)
```

```
[0 0 0 ... 0 0 0]
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
```

```
from sklearn.metrics import confusion_matrix
```

```
z=confusion_matrix(y_test5, predictions5)
z
```

```
array([[10698,   267],
       [  929,   463]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test5,predictions5))
```

	precision	recall	f1-score	support
0	0.92	0.98	0.95	10965
1	0.63	0.33	0.44	1392
accuracy			0.90	12357
macro avg	0.78	0.65	0.69	12357
weighted avg	0.89	0.90	0.89	12357

70 - 30 ratio

```
[ ] from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] model=KNeighborsClassifier(n_neighbors=25)
```

```
[ ] model.fit(X_train5,y_train5)
```

```
KNeighborsClassifier(n_neighbors=25)
```

```
[ ] y_pred5 = model.predict(X_test5)
    y_pred5
```

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

```
[ ] knn = pd.DataFrame({'Predicted':y_pred5,'Actual':y_test5})
    knn
```

	Predicted	Actual
2065	0	0
6387	0	0
11658	0	0
3611	0	0
14337	0	0
...	...	...
8197	0	0
19137	1	0
18302	0	0
25830	0	0
6914	0	0

12357 rows × 2 columns

```
[ ] #Evaluation Metric
    from sklearn.metrics import accuracy_score
    accuracy_score(y_test5,y_pred5)
```

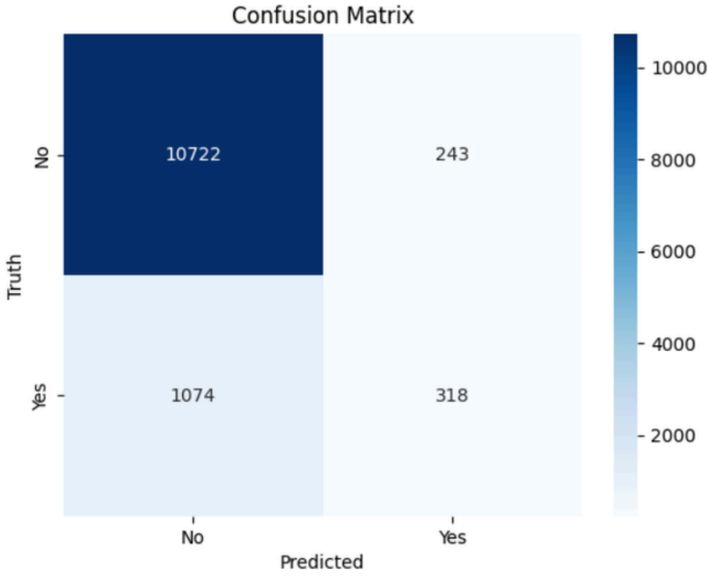
```
0.8934207331876669
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test5,y_pred5)
    cm
```

```
array([[10722,   243],
       [ 1074,   318]])
```

```
[ ] sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
    plt.title('Confusion Matrix')
    plt.xlabel("Predicted")
    plt.ylabel("Truth")
    plt.show()
```

KNN




```
from sklearn.metrics import classification_report
classification_rep = classification_report(y_test5, y_pred5)
print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94     10965
           1       0.57      0.23      0.33      1392

    accuracy                           0.89     12357
   macro avg       0.74      0.60      0.63     12357
weighted avg       0.87      0.89      0.87     12357
```

SVM

70 - 30 ratio

```
[ ] from sklearn.svm import SVC
```

```
[ ] model = SVC(kernel='rbf')
```

```
[ ] model.fit(X_train5, y_train5)
```

```
SVC()
```

```
[ ] y_pred5 = model.predict(X_test5)
    y_pred5
```

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

```
[ ] svm = pd.DataFrame({'Predicted':y_pred5,'Actual':y_test5})
    svm
```

	Predicted	Actual
2065	0	0
6387	0	0
11658	0	0
3611	0	0
14337	0	0
...	...	...
8197	0	0
19137	1	0
18302	0	0
25830	0	0
6914	0	0

12357 rows × 2 columns

```
[ ] #evaluation metrics
    from sklearn.metrics import accuracy_score
    accuracy_score(y_test5,y_pred5)
```

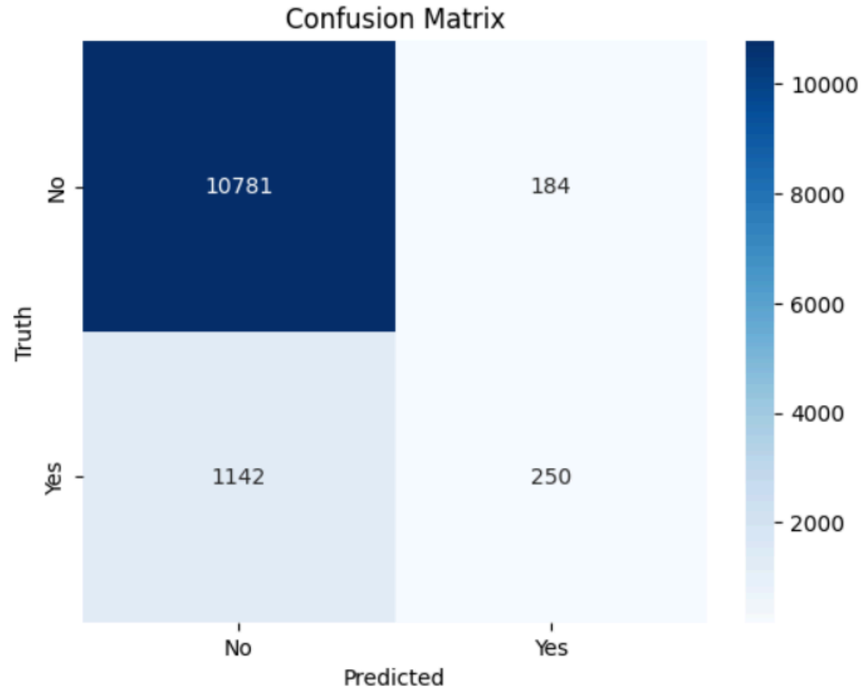
```
0.8926924010682205
```

```
[ ] from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test5,y_pred5)
    cm
```

```
array([[10781,   184],
       [ 1142,   250]])
```

```
[ ] sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
    plt.title('Confusion Matrix')
    plt.xlabel("Predicted")
    plt.ylabel("Truth")
    plt.show()
```

SVM




```
[ ] from sklearn.metrics import classification_report
    classification_rep = classification_report(y_test5, y_pred5)
    print(classification_rep)
```

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94     10965
           1       0.58      0.18      0.27      1392

    accuracy                           0.89     12357
   macro avg       0.74      0.58      0.61     12357
weighted avg       0.87      0.89      0.87     12357
```

Decision tree

70 - 30 ratio

```
[ ] from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
    from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
```

```
[ ] clf = DecisionTreeClassifier()

    # Train Decision Tree Classifer
    clf = clf.fit(X_train5,y_train5)
```

```
[ ] y_pred5 = clf.predict(X_test5)
    y_pred5
```

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

```
[ ] print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.8732702112163147
```

```
[ ] clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

    # Train Decision Tree Classifer
    clf = clf.fit(X_train5,y_train5)

    #Predict the response for test dataset
    y_pred5 = clf.predict(X_test5)
```

```
Accuracy: 0.8975479485311969
```

```
clf = DecisionTreeClassifier(criterion="gini", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train5,y_train5)

#Predict the response for test dataset
y_pred5 = clf.predict(X_test5)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.9027271991583717
```

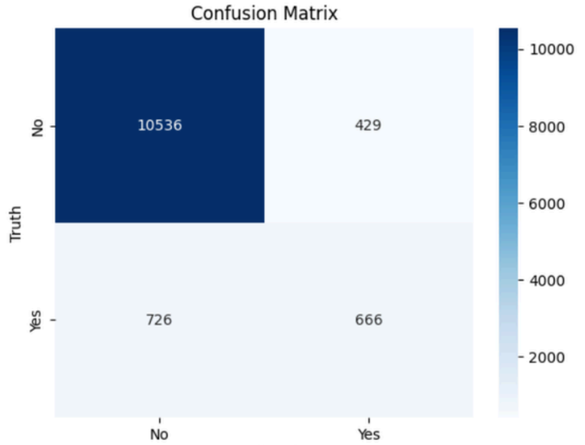
```
clf = DecisionTreeClassifier(criterion="gini", max_depth=6)

# Train Decision Tree Classifer
clf = clf.fit(X_train5,y_train5)

#Predict the response for test dataset
y_pred5= clf.predict(X_test5)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.9065307113377034
```

DECISION TREE

```
array([[10536,   429],
       [  726,   666]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```




```
clf = DecisionTreeClassifier(criterion="gini", max_depth=24)

# Train Decision Tree Classifer
clf = clf.fit(X_train5,y_train5)

#Predict the response for test dataset
y_pred5 = clf.predict(X_test5)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.8748887270373068
```

```
#Predict the response for train dataset
y_pred_train5 = clf.predict(X_train5)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_train5, y_pred_train5))
```

```
Accuracy: 0.9935486108702438
```

Bagging - Random forest

70 - 30 ratio

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report, confusion_matrix
```

```
[ ] rf = RandomForestClassifier()
```

```
[ ] rf.fit(X_train5, y_train5)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
[ ] y_pred5 = rf.predict(X_test5)
    y_pred5
```

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

```
[ ] print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.8955248037549567
```

```
[ ] print(classification_report(y_test5, y_pred5))
```

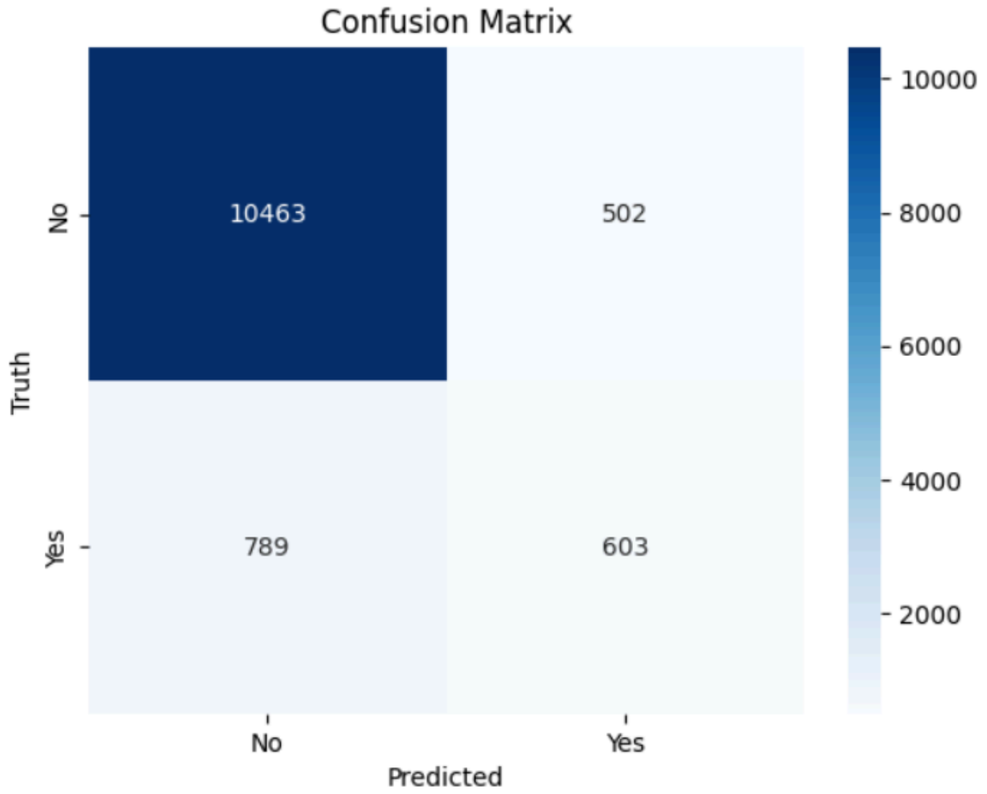
```
              precision    recall  f1-score   support

           0       0.93      0.95      0.94     10965
           1       0.55      0.43      0.48      1392

    accuracy                           0.90     12357
   macro avg       0.74      0.69      0.71     12357
weighted avg       0.89      0.90      0.89     12357
```

```
cm = confusion_matrix(y_test5, y_pred5)
cm
```

```
array([[10463,   502],
       [  789,   603]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

BAGGING



Boosting

XGboost

70 - 30 ratio

```
[ ] import xgboost as xgb
```

```
[ ] model = xgb.XGBClassifier()

    model = model.fit(X_train5, y_train5)
```

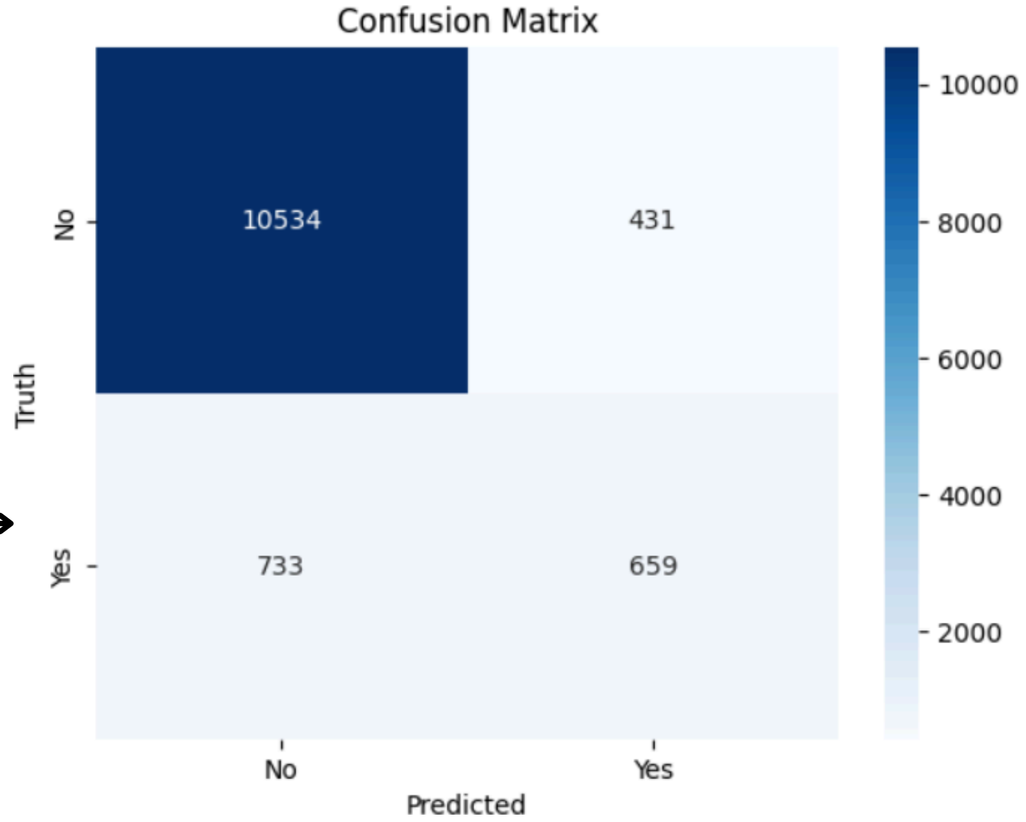
```
[ ] from sklearn import metrics

    y_pred5 = model.predict(X_test5)
    y_pred5
```

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

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(y_test5, y_pred5))
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95     10965
           1       0.60      0.47      0.53      1392

    accuracy                           0.91     12357
   macro avg       0.77      0.72      0.74     12357
weighted avg       0.90      0.91      0.90     12357
```

XG-BOOST




```
from sklearn.metrics import accuracy_score
print("Accuracy:",metrics.accuracy_score(y_test5, y_pred5))
```

```
Accuracy: 0.9058023792182569
```

```
cm = confusion_matrix(y_test5, y_pred5)
cm
```

```
array([[10534,   431],
       [  733,   659]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

ADAboost

70 - 30 ratio

ADA-BOOST

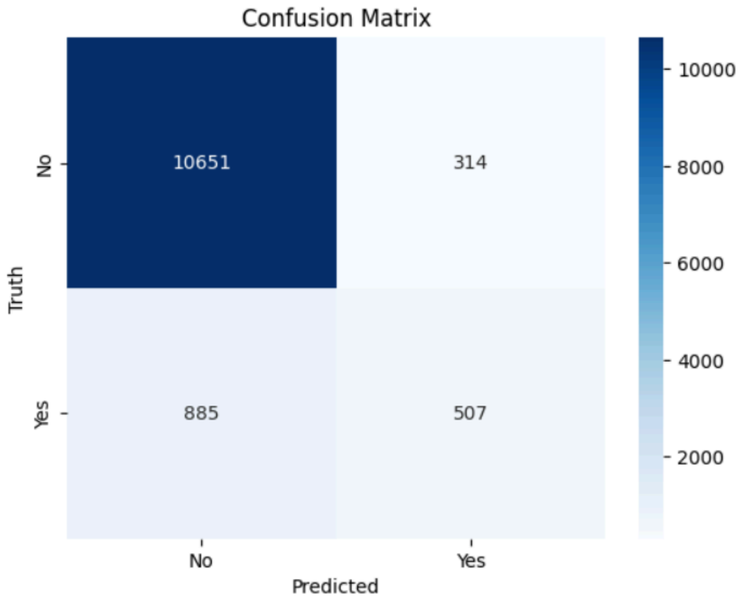
```
[ ]  from sklearn.ensemble import AdaBoostClassifier
     from sklearn.metrics import accuracy_score

     # Initialize the AdaBoost classifier
     model = AdaBoostClassifier(n_estimators=50, random_state=42)

     # Fit the model on the training data
     model.fit(X_train5, y_train5)

     # Predict on the test data
     y_pred5 = model.predict(X_test5)

     # Calculate the accuracy
     accuracy = accuracy_score(y_test5, y_pred5)
     print(f"Accuracy: {accuracy:.3f}")
     print(classification_report(y_test5, y_pred5))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_we
  warnings.warn(
Accuracy: 0.903
              precision    recall  f1-score   support

           0       0.92      0.97      0.95     10965
           1       0.62      0.36      0.46      1392

    accuracy                           0.90     12357
   macro avg       0.77      0.67      0.70     12357
weighted avg       0.89      0.90      0.89     12357
```

```
cm = confusion_matrix(y_test5, y_pred5)
cm
```

```
array([[10651,   314],
       [  885,   507]])
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No", "Yes"], yticklabels=["No", "Yes"])
plt.title('Confusion Matrix')
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```



```
[ ] import tensorflow as tf
    import pandas as pd
    import matplotlib.pyplot as plt
```

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(10, activation='relu'),
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(learning_rate=0.01),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='a=recall')
              ]
              )

# STEP1: Fit the model

history= model.fit(X_train1, y_train1, epochs= 100)
```

ANN

```
[ ] model.evaluate(X_test1, y_test1)
```

```
387/387 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - a=recall: 0.0000e+00 - accuracy: 0.8862 - loss: 0.3543 - precision: 0.0000e+00
[0.35968470573425293, 0.8836287260055542, 0.0, 0.0]
```

```
[ ] pd.DataFrame(history.history).plot()
```

```
<Axes: >
```