BANCO DE PORTUGAL
EUROSISTEMA

BANCO DE PORTUGAL

Predicting Term Deposit Subscription

EVOASTRA VENTURES
DIGITAL SOLUTIONS

Submitted by

TEAM 'E'

Fathima Dilshana PK (Team Lead)

Sakshi Vijay Mindhe (Co Team Lead)

Saravanan P

Dinesh Solanki

Kiranbabu P

Ajay Krishnan M

Mehar Snotra

Uthra Nitheya C G

Abhishek Singh

Anand Eswar

DECLARATION

This is to certify that the project report titled "**Predicting Term Deposit Subscription for Banco de Portugal**" is the bonafide work of Fatima Dilshana P K, Sakshi Vijay Mindhe, Mehar Snotra, Dinesh Solanki, Saravanan P, Uthra Nitheya, Abhishek Singh, Anand Eswar, Kiran Babu P and Ajay Krishna.

This project was conducted as part of their internship at Evostra Ventures and was completed under proper supervision.

ACKNOWLEDGEMENT

First and foremost, we would like to express our deepest gratitude to the Almighty for His blessings and guidance throughout this endeavor. His divine support has been a source of strength and inspiration in our journey.

We would like to take this opportunity to express our sincere thanks and gratitude to **Mr. Zeeshan Firdousi** and **Mr. Aniket Manwatkar**, Founder & CEO of Evoastra Ventures Pvt Ltd for giving an opportunity to do this project under their guidance in their esteemed organization and it has been a great learning and enjoyable experience.

Additionally, we wish to acknowledge our Team members who provided both moral support and practical help. Their contributions have been immensely beneficial in completing this project. This project would not have been possible without the collective support and guidance from all these wonderful individuals, and we are deeply appreciative of their efforts.

At last, but not least, we are thankful to our family for their moral support, endurance and encouragement during the course of the project.

ABSTRACT

This project aims to develop a predictive model to enhance the marketing efforts of Banco de Portugal by determining whether clients will subscribe to a term deposit based on historical campaign data. Banks often face challenges in engaging clients effectively, managing vast amounts of data, and predicting client behavior, making predictive accuracy crucial for successful marketing campaigns. Using a dataset of 45,211 instances and 16 features related to client demographics, campaign details, and previous campaign outcomes, this project involves comprehensive data exploration, feature engineering, and model development. The primary objective is to build a machine learning classification model that can accurately predict a binary outcome—whether a client will subscribe to a term deposit or not. The model's performance will be evaluated using metrics such as accuracy, precision, recall, and F1-score, and the results will be documented to provide insights for optimizing future marketing campaigns at Banco de Portugal. The exclusion of the "duration" feature ensures that the model can make realistic predictions based on information available before the final contact with clients.

TABLE OF CONTENT

TABLE OF IMAGES

INTRODUCTION

In today's competitive financial landscape, banks constantly seek ways to improve their marketing strategies to attract and retain customers. One common approach is the use of direct marketing campaigns to promote financial products, such as term deposits. Banco de Portugal, like many other banks, relies on phone-based campaigns to encourage clients to subscribe to these products. However, predicting whether a client will respond positively to such a campaign presents several challenges, including effective client engagement, data management, and ensuring predictive accuracy.

This project focuses on analyzing historical marketing campaign data from Banco de Portugal to build a machine learning model capable of predicting whether a client will subscribe to a term deposit. By leveraging data on client demographics, campaign contact details, and outcomes from previous campaigns, the aim is to develop a classification model that provides valuable insights into customer behavior. This predictive model can help Banco de Portugal target the right clients more effectively, thereby optimizing its marketing efforts and improving campaign success rates.

The project involves several key tasks: data exploration, feature engineering, model development, and performance evaluation. By the end of the project, a well-documented predictive model will be built to support Banco de Portugal's future marketing campaigns, ensuring better client targeting and higher return on investment.

LITERATURE REVIEW

Predicting customer behavior in banking, particularly for marketing campaigns, has been an area of growing interest in data science and machine learning research. Financial institutions often rely on predictive analytics to enhance decision-making, improve customer retention, and optimize marketing strategies. This literature review explores key studies related to the use of machine learning for marketing campaigns, focusing on customer behavior prediction, feature engineering, and model evaluation.

1. Predictive Modeling in Direct Marketing: Direct marketing campaigns have been widely studied, especially in banking, where customer behavior is a critical factor for product subscription. According to **Moro et al. (2014)**, predicting client subscription to term deposits using a combination of demographic, socio-economic, and campaign-specific data can significantly improve marketing success. Their study utilized data from a Portuguese bank to develop a predictive model based on support vector machines (SVM) and logistic regression, achieving high classification accuracy. Similarly, **Durand et al. (2019)** highlighted the effectiveness of classification algorithms, including decision trees and random forests, in predicting customer responses in direct marketing campaigns, demonstrating the power of machine learning for targeted marketing.

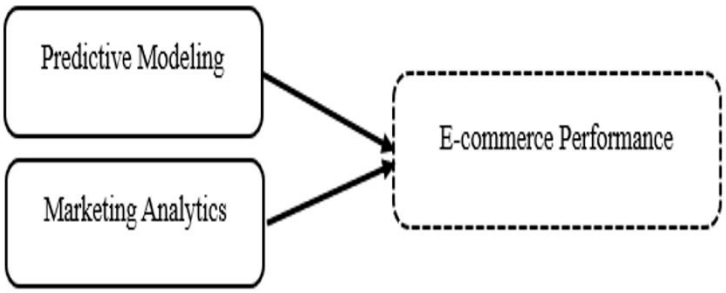


Figure 1: Conceptual Framework

2. Feature Engineering and Data Preparation: Effective feature engineering plays a pivotal role in enhancing model performance, particularly when working with large and complex datasets. Studies by **Tsai and Lu (2009)** and **Abdar et al. (2020)** emphasized the importance of feature selection and transformation in predictive modeling, noting that irrelevant or redundant features can reduce accuracy. In the context of banking campaigns, features such as previous campaign outcomes, the number of previous contacts, and the duration of the current contact have been found to be highly predictive of customer responses. **Moro et al. (2014)** also pointed out that certain features, like 'duration', while useful for predictions, may need to be excluded to maintain the practical applicability of the model, as it can only be known after the final client contact.

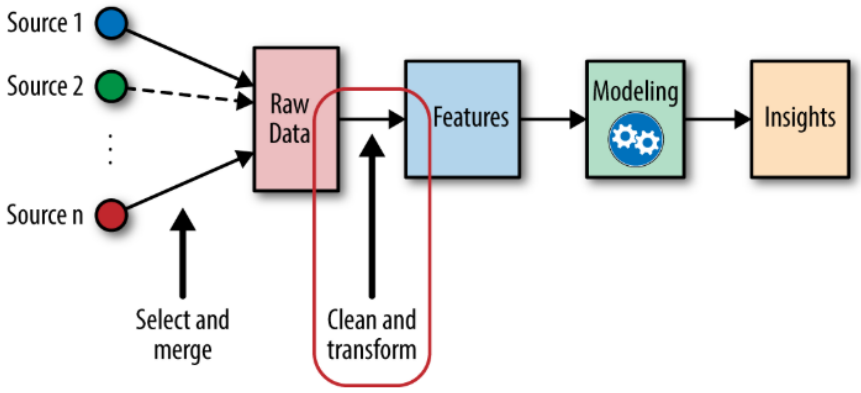


Figure 2: Feature Engineering

3. Machine Learning Techniques for Classification: Several machine learning algorithms have been widely adopted for binary classification tasks in the financial sector. Logistic regression, decision trees, and random forests have been extensively applied due to their interpretability and robustness **(Zhang et al., 2018)**. Random forests, in particular, have shown excellent performance in handling imbalanced datasets and complex feature relationships **(Breiman, 2001)**. More advanced techniques, such as gradient

boosting and XGBoost, have been used to improve predictive accuracy in marketing applications by handling non-linear relationships and minimizing overfitting **(Chen & Guestrin, 2016)**.

Deep learning models have also been explored for predicting customer behavior, though they tend to require larger datasets and more computational resources **(LeCun et al., 2015)**. Studies by **Abdar et al. (2020)** demonstrate that neural networks can achieve higher accuracy but often at the expense of interpretability, which can be a concern in the banking industry due to regulatory requirements for model transparency.

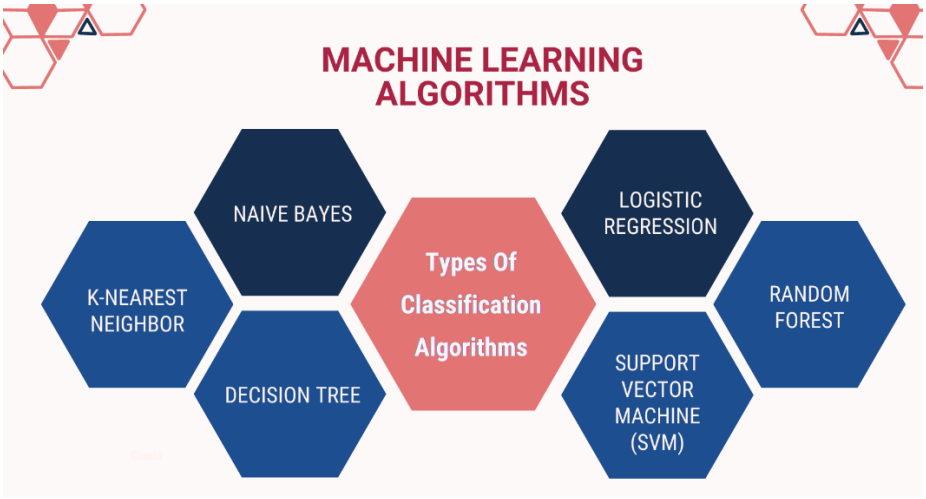


Figure 3: Machine Learning Algorithms

4. Evaluation Metrics and Model Performance: The effectiveness of predictive models is often assessed using various performance metrics, such as accuracy, precision, recall, and the F1-score. However, as noted by **Davis** and **Goadrich (2006)**, accuracy alone may not be sufficient, especially when dealing with imbalanced datasets where the majority class dominates. Precision and recall provide more detailed insights into the model's ability to correctly identify positive cases (i.e., customers who subscribe to the term deposit), while the F1-score balances both metrics to offer a holistic view of

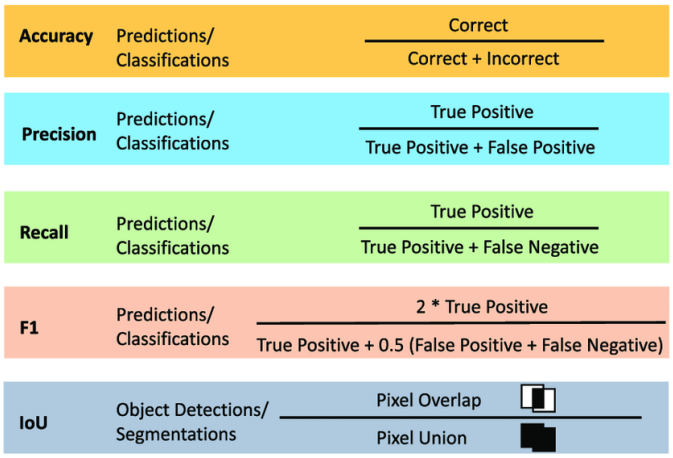


Figure 4: Evaluation Metrics

model performance. Additionally, Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) are commonly used to evaluate classification models' effectiveness across different threshold settings **(Bradley, 1997)**.

5. Challenges and Ethical Considerations: While machine learning provides significant benefits for predictive marketing, challenges such as data privacy, regulatory compliance, and model interpretability must be addressed. In banking, the use of client data for marketing must comply with strict regulations like the General Data Protection Regulation (GDPR) in Europe. Studies by **Wachter et al. (2017)** and **Kamiran et al. (2012)** have highlighted the importance of fairness and transparency in predictive models, stressing that biased models can lead to unfair treatment of certain customer groups, potentially violating ethical standards and regulations.

Figure 5: Ethical Challenges

The literature underscores the importance of leveraging machine learning techniques to improve marketing strategies in the banking sector. With the right combination

of data, feature engineering, and model selection, banks can significantly enhance the predictive accuracy of customer responses to marketing campaigns. The studies reviewed also emphasize the need for careful model evaluation and ethical considerations, particularly in the context of data privacy and regulatory compliance. This project builds upon these insights to develop a predictive model for Banco de Portugal, using client and campaign data to optimize term deposit subscription predictions.

Overall, the process of model making includes all of these considerations. From understanding the project, collecting data to predicting the output values. Everything that happen in between are co-related. The process is depicted in the below image.

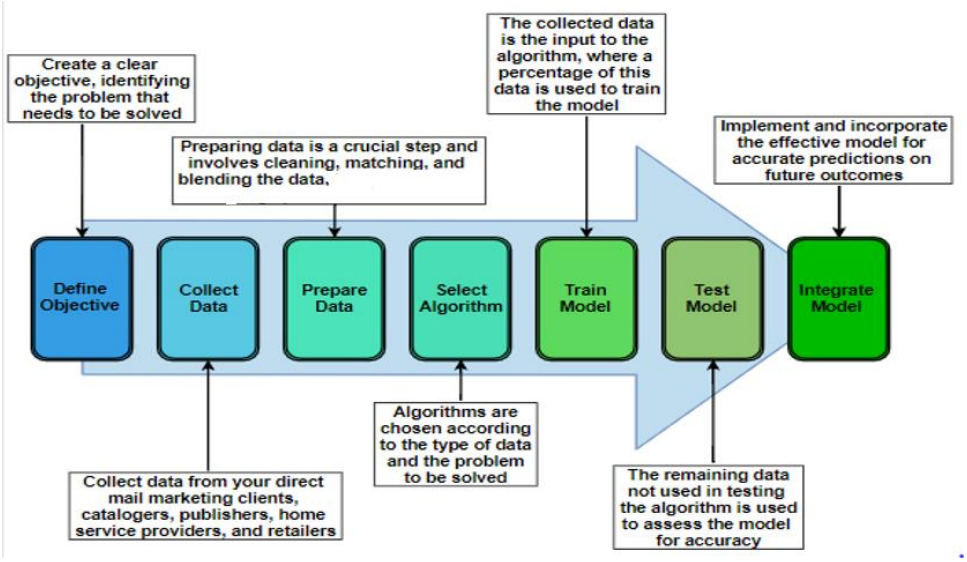


Figure 6: Machine Learning Working Diagram

ORGANIZATIONAL PROFILE

BANCO DE PORTUGAL

BANCO DE PORTUGAL
EUROSISTEMA

Figure 7: Head Quarters

The Banco de Portugal (English: Bank of Portugal) is the Portuguese member of the Eurosystem and has been the monetary authority for Portugal from 1846 to 1998, issuing the Portuguese escudo. Since 2014, it has also been Portugal's national competent authority within European Banking Supervision. The bank was founded by royal charter in 1846, during the reign of Queen Maria II of Portugal, by a merger of the Banco de Lisboa [pt], the first bank founded in Portugal, and insurer Companhia Confiança Nacional.

Headquarters	**Lisbon, Portugal**
Coordinates	38.708729°N 9.138148°W
Established	19 November 1846; 177 years ago
Ownership	100% state ownership
Governor	Mário Centeno
Central bank of	Portugal
Reserves	4 980 million USD
Preceded by	Banco de Lisboa Companhia de Confiança Nacional
Succeeded by	European Central Bank (1999)
Website	www.bportugal.pt

HISTORY:

Queen Maria II of Portugal established the bank by royal charter on 19 November 1846 to act as a commercial bank and issuing bank. It came about as the result of a merger of the Banco de Lisboa, the first bank founded in

Portugal, and the Companhia de Confiança Nacional, an investment company specialised in the financing of the public debt.

The bank was designated by the Portuguese Crown as the emitter of legal tender, at the time the Portuguese real, which it continued producing until 1911.

REPUBLIC:

Following the Implementation of the Republic in 1910, the Banco de Portugal began to emit the Portuguese Escudo.

In 1932, the bank established the Biblioteca do Banco de Portugal, one of the most significant private libraries in Portugal.

In 1946, the institution was bestowed the honor of Grand Cross of the Order of Christ by the President of Portugal.

During the Estado Novo, the bank pursued a vigorous policy of gold acquisition starting in 1957, which has contributed towards Portugal's present-day status of having the 14th largest gold reserve in the world.

OBJECTIVE

The primary objective of this project is to develop a machine learning model that accurately predicts whether a client will subscribe to a term deposit based on historical data from Banco de Portugal's direct marketing campaigns. By analyzing client demographics, campaign contact details, and previous campaign outcomes, the project aims to:

1. **Explore the dataset** to identify patterns and relationships between the features and the target variable.
2. **Engineer features** that improve the predictive power of the model, while excluding unrealistic predictors such as contact duration.
3. **Build and evaluate classification models** using machine learning techniques to predict the likelihood of a client subscribing to a term deposit.
4. **Optimize the model's performance** by evaluating it with metrics such as accuracy, precision, recall, and F1-score.
5. **Provide actionable insights** that can help Banco de Portugal enhance their marketing strategies by targeting clients more effectively and improving campaign outcomes.

The ultimate goal is to support the bank in making data-driven decisions to improve the efficiency and success of future marketing efforts.

METHODOLOGY

This project follows a structured approach to analyze the marketing campaign data from Banco de Portugal and develop a machine learning model for predicting term deposit subscription. The methodology is divided into several key steps:

1. Data Collection and Understanding

- The dataset used for this project is obtained from Banco de Portugal's direct marketing campaign, consisting of 45,211 records and 16 features related to client demographics, campaign contact details, and outcomes of previous campaigns.
- Two variations of the dataset are provided: the full dataset and a 10% subset. For the purposes of this project, the full dataset (bank-additional-full.csv) will be used for training and evaluation.
- The initial step involves loading the dataset, examining its structure, and understanding the key attributes that could influence the target variable (client subscription to a term deposit).

2. Data Preprocessing

- **Handling Missing Values:** The dataset is checked for missing or inconsistent values. Categorical variables with "unknown" values (e.g., job, education) are assessed, and strategies such as imputation or exclusion are applied based on the extent of missing data.
- **Encoding Categorical Variables:** Since many features in the dataset are categorical (e.g., job, marital status, education), appropriate encoding techniques are applied. Label encoding or one-hot encoding

is used to convert categorical variables into a numerical format suitable for machine learning models.

- **Outlier Detection and Treatment:** Numeric features such as balance and age are examined for outliers. Techniques like interquartile range (IQR) or z-score are employed to identify and handle outliers to ensure that they do not negatively impact model performance.
- **Scaling Numerical Features:** Continuous numerical variables (e.g., balance, age) are scaled using standardization or normalization techniques to ensure uniformity across all features, which is especially important for distance-based algorithms.

3. Exploratory Data Analysis (EDA)

- A detailed EDA is conducted to explore the distribution of features and relationships with the target variable (term deposit subscription). Key analyses include:
    - Distribution of categorical variables (e.g., job, education, marital status).
    - Correlation between numerical features (e.g., balance, age) and the target variable.
    - Analyzing campaign-related variables such as the number of contacts, previous outcomes, and the timing of the last contact.
- Visualizations, including bar plots, histograms, and correlation matrices, are used to uncover patterns and insights from the data.

4. Feature Engineering

- **New Feature Creation:** Based on insights gained from EDA, new features are created to improve the predictive power of the model. For example, combining related variables such as loan and balance or

engineering interaction terms between campaign features (e.g., contact type and campaign success).

- **Exclusion of 'Duration' Feature:** Since 'duration' (the length of the last contact) is highly predictive but not available until the final call, it is excluded from the model to ensure a realistic predictive scenario.
- **Feature Selection:** A combination of statistical methods (e.g., ANOVA, chi-square tests) and model-based techniques (e.g., feature importance from random forests) is used to select the most relevant features for the final model.

5. Model Development

- Several machine learning classification algorithms are applied to build the predictive model, including:
    - **Logistic Regression:** For its interpretability and ability to estimate probabilities.
    - **Decision Trees and Random Forests:** To capture non-linear relationships and interactions between features.
    - **Gradient Boosting (XGBoost):** To enhance predictive accuracy through boosting techniques and handle potential overfitting.
- The dataset is split into training and testing sets (typically 70%-30% or 80%-20%) to train the model on one part of the data and test its performance on unseen data.

6. Model Evaluation

- The performance of the models is evaluated using appropriate classification metrics, including:
    - **Accuracy:** Measures the overall correctness of predictions.

- **Precision:** The proportion of true positives out of the predicted positives.
- **Recall (Sensitivity):** The proportion of true positives out of the actual positives.
- **F1-Score:** The harmonic mean of precision and recall, providing a balanced evaluation.
- **Confusion Matrix:** Used to visualize the model's performance on positive and negative predictions.
- **ROC Curve and AUC:** Evaluate the model's ability to distinguish between the two classes (subscribed/not subscribed) across different thresholds.

7. Hyperparameter Tuning

- To improve model performance, hyperparameter tuning is performed using techniques like Grid Search or Random Search. Key hyperparameters, such as the number of trees in a random forest or the learning rate in gradient boosting, are optimized to enhance model accuracy and generalization.

8. Model Selection and Deployment

- Based on evaluation metrics, the best-performing model is selected for final deployment. The selected model is validated on the test set to ensure robustness and avoid overfitting.
- The final model is saved and can be integrated into Banco de Portugal's systems to provide real-time predictions of client subscription likelihood for future marketing campaigns.

PROGRAM IMPLEMENTATION

Importing Data Analysis Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Load Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
bank = pd.read_csv('/content/drive/MyDrive/Bank.csv', delimiter=';')
bank.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cor
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	

Figure 8: Load dataset

Data Cleaning

```
bank.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	41188.0	40.024060	10.421250	17.000	32.000	38.000	47.000	98.000
duration	41188.0	258.285010	259.279249	0.000	102.000	180.000	319.000	4918.000
campaign	41188.0	2.567593	2.770014	1.000	1.000	2.000	3.000	56.000
pdays	41188.0	962.475454	186.910907	0.000	999.000	999.000	999.000	999.000
previous	41188.0	0.172963	0.494901	0.000	0.000	0.000	0.000	7.000
emp.var.rate	41188.0	0.081886	1.570960	-3.400	-1.800	1.100	1.400	1.400
cons.price.idx	41188.0	93.575664	0.578840	92.201	93.075	93.749	93.994	94.767
cons.conf.idx	41188.0	-40.502600	4.628198	-50.800	-42.700	-41.800	-36.400	-26.900
euribor3m	41188.0	3.621291	1.734447	0.634	1.344	4.857	4.961	5.045
nr.employed	41188.0	5167.035911	72.251528	4963.600	5099.100	5191.000	5228.100	5228.100

Figure 9: Describe()

bank.info()

```
<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   marital         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
```

Figure 10: info()

bank.columns

```
Index(['age', 'job', 'marital', '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')
```

Figure 11:columns

print(bank.isnull().sum())
print(bank.isna().sum())

```
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
age               0
job               0
marital           0
education         0
default           0
```

Figure 12: isnull()

Check for duplicates

```
print(f'Number of duplicates: {bank.duplicated().sum()}')
```

Remove duplicates

```
bank.drop_duplicates(inplace=True)
```

Verify duplicates are removed

```
print(f'Number of duplicates after removal: {bank.duplicated().sum()}')
```

```
Number of duplicates: 12
Number of duplicates after removal: 0
```

Figure 13: Duplicates

Check for missing values

```
print(bank.isnull().sum())
```

Fill missing values with mode for categorical columns

```
for column in bank.select_dtypes(include=['object']).columns:
    bank[column].fillna(bank[column].mode()[0], inplace=True)
```

Fill missing values with median for numerical columns

```
for column in bank.select_dtypes(include=['number']).columns:
    bank[column].fillna(bank[column].median(), inplace=True)
```

Verify missing values are filled

```
print(bank.isnull().sum())
```

Download the cleaned CSV file

```
from google.colab import files
bank.to_csv('cleaned_bank.csv', encoding = 'utf-8-sig')
files.download('cleaned_bank.csv')
```

```
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
age               0
job               0
marital           0
education         0
default           0
housing           0
```

Figure 14: Cleaned Data

The Dataset is imbalanced; thus, accuracy is not the suitable evaluation metric. we build the best model and evaluate it with precision, recall or F1 score

```
scoring = 'accuracy'
bank['y'].value_counts(normalize=True)
```

y	proportion
no	0.887337
yes	0.112663

dtype: float64

Figure 15: Accuracy value count

```
poutcomes = bank[bank.poutcome != 'nonexistent'].poutcome.apply(lambda x: 1 if x == 'success' else 0)
coutcomes = bank.y.apply(lambda x: 1 if x == 'yes' else 0)
```

```
print('Number of records:', len(bank))
```

```
print('Success Rate (Current Campaign):', coutcomes.sum() / len(bank))
print('Success Rate (Previous Campaign):', poutcomes.sum() / len(poutcomes))
```

```
Number of records: 41176
Success Rate (Current Campaign): 0.11266271614532737
Success Rate (Previous Campaign): 0.24408888888888888
```

Figure 16: Count

```
dtypes = pd.DataFrame(bank.dtypes.rename('type')).reset_index().astype('str')
```

Excluding duration

```
dtypes = dtypes.query('index != "duration"')
numeric = dtypes[(dtypes.type.isin(['int64', 'float64'])) & (dtypes['index'] != 'duration')]['index'].values
categorical = dtypes[~(dtypes['index'].isin(numeric)) & (dtypes['index'] != 'y')]['index'].values

print('Numeric:\n', numeric,end='\n\n')
print('Categorical:\n', categorical)
```

```
Numeric:
 ['age' 'campaign' 'pdays' 'previous' 'emp.var.rate' 'cons.price.idx'
 'cons.conf.idx' 'euribor3m' 'nr.employed']

Categorical:
 ['job' 'marital' 'education' 'default' 'housing' 'loan' 'contact' 'month'
 'day_of_week' 'poutcome']
```

Figure 17: Numerical & Categorical

Converting dependent variable categorical to dummy

```
y = pd.get_dummies(bank['y'], columns = ['y'], prefix = ['y'], drop_first = True)
bank.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.pri
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	

5 rows × 21 columns

Figure 18: Categorical to Dummy

Bank client data Analysis and Categorical Treatment

```
bank_client = bank.iloc[: , 0:7]
bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

Figure 19: Categorical Treatment

Knowing the categorical variables

```
print('Jobs:\n', bank_client['job'].unique())
```

```
Jobs:
 ['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
```

Figure 20: Categorical variable

```
print('Marital:\n', bank_client['marital'].unique())
```

```
Marital:
 ['married' 'single' 'divorced' 'unknown']
```

Figure 21: Marital

```
print('Education:\n', bank_client['education'].unique())
```

```
Education:
 ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
```

Figure 22 : Education

```
print('Default:\n', bank_client['default'].unique())
print('Housing:\n', bank_client['housing'].unique())
print('Loan:\n', bank_client['loan'].unique())
```

```
Default:
 ['no' 'unknown' 'yes']
Housing:
 ['no' 'yes' 'unknown']
Loan:
 ['no' 'yes' 'unknown']
```

Figure 23: Housing Loan

Age

Trying to find some strange values or null values

```
print('Min age: ', bank_client['age'].max())
print('Max age: ', bank_client['age'].min())
print('Null Values: ', bank_client['age'].isnull().any())
```

```
Min age:  98
Max age:  17
Null Values:  False
```

Figure 24: Age

Visualization

```
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'age', data = bank_client)
ax.set_xlabel('Age', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
sns.despine()
```

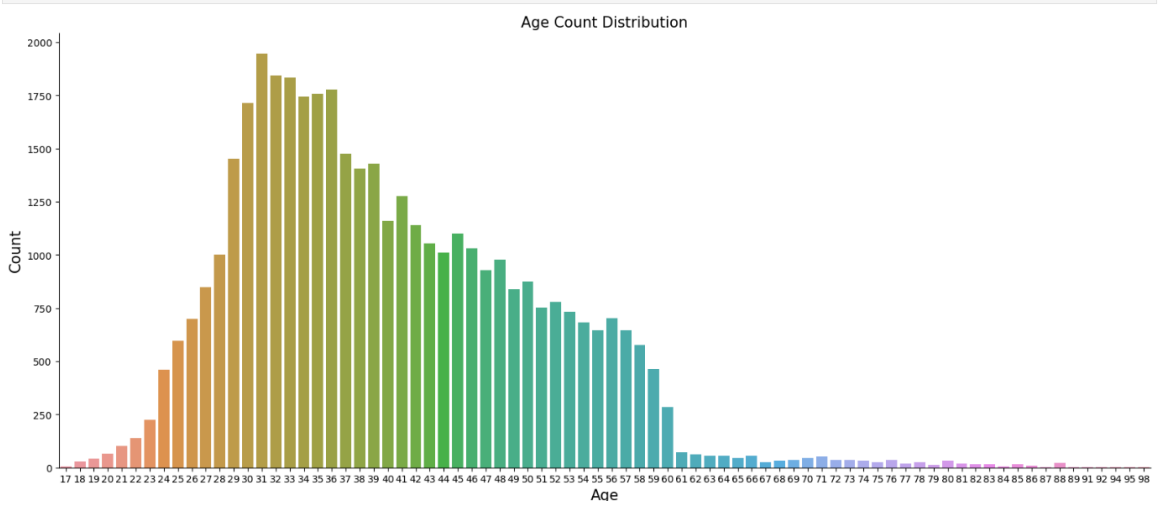


Figure 25: Age Count Distribution

```
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5))
sns.boxplot(x = 'age', data = bank_client, orient = 'v', ax = ax1)
ax1.set_xlabel('People Age', fontsize=15)
ax1.set_ylabel('Age', fontsize=15)
ax1.set_title('Age Distribution', fontsize=15)
ax1.tick_params(labelsize=15)

sns.distplot(bank_client['age'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('Age', fontsize=15)
ax2.set_ylabel('Occurence', fontsize=15)
ax2.set_title('Age x Ocucurence', fontsize=15)
ax2.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```

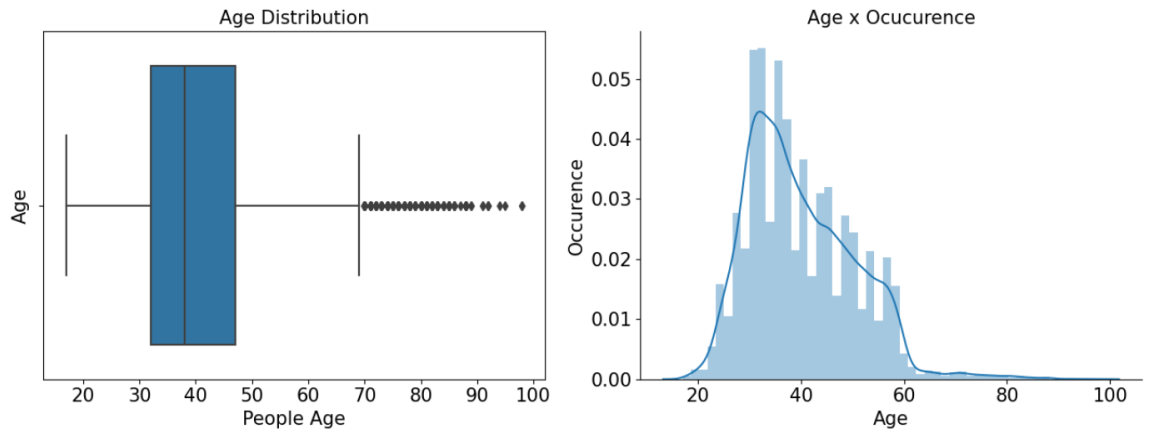


Figure 26: Age Occurrence

Calculate the outliers using IQR Method:

```
print('Ages above: ', bank_client['age'].quantile(q = 0.75) +
                1.5*(bank_client['age'].quantile(q = 0.75) - bank_client['age'].quantile(q = 0.25)), 'are
outliers')
```

```
Ages above:  69.5 are outliers
```

Figure 27: Outliers

```
print('Numerber of outliers: ', bank_client[bank_client['age'] > 69.6]['age'].count())
print('Number of clients: ', len(bank_client))
print('Outliers are:', round(bank_client[bank_client['age'] > 69.6]['age'].count()*100/len(bank_client),2),
'%')
```

```
Numerber of outliers:  468
Number of clients:  41176
Outliers are: 1.14 %
```

Figure 28: Outlier Count

Calculating some values to evaluate this independent variable

```
print('MEAN:', round(bank_client['age'].mean(), 1))
```

- A low standard deviation indicates that the data points tend to be close to the mean or expected value
- A high standard deviation indicates that the data points are scattered

```
print('STD :', round(bank_client['age'].std(), 1))
```

I think the best way to give a precisely insight about dispersion is using the CV (coefficient variation) (STD/MEAN) *100

- cv < 15%, low dispersion
- cv > 30%, high dispersion

```
print('CV  :',round(bank_client['age'].std()*100/bank_client['age'].mean(), 1), ', High middle dispersion')
```

```
MEAN: 40.0
STD : 10.4
CV  : 26.0 , High middle dispersion
```

Figure 29: Mean, STD, CV

Conclusion about AGE, due to almost high dispersion and just looking at this this graph we cannot conclude if age have a high effect to our variable y, need to keep searching for some pattern. high middle dispersion means we have people with all ages and maybe all of them can subscript a term deposit, or not.

The outliers were calculated, so we go with fitting the model with and without them

JOBS

What kind of jobs clients this bank has, if you cross jobs with default, loan or housing, there is no relation

```
fig, ax = plt.subplots()
fig.set_size_inches(20, 8)
sns.countplot(x = 'job', data = bank_client)
ax.set_xlabel('Job', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```

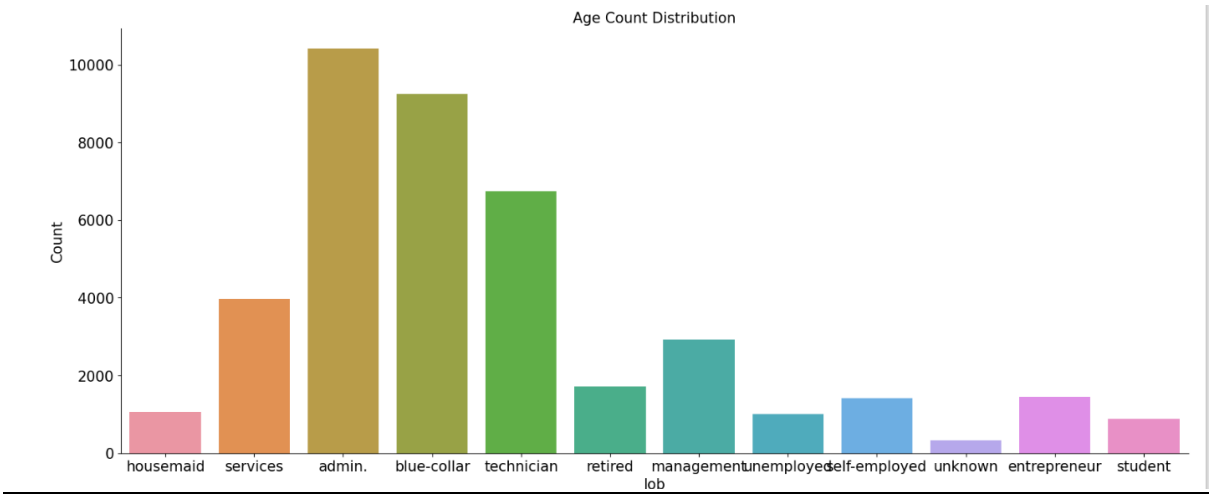


Figure 30: Job Count Distribution

MARITAL

What kind of 'marital clients' this bank have, if you cross marital with default, loan or housing, there is no relation

```
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
sns.countplot(x = 'marital', data = bank_client)
ax.set_xlabel('Marital', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Age Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```

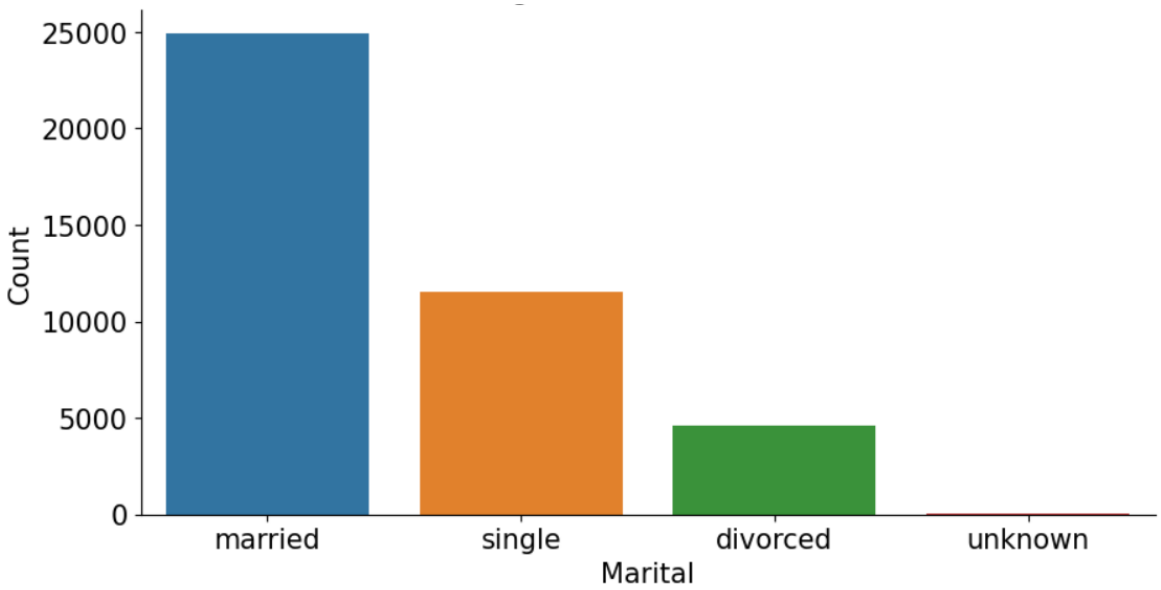


Figure 31: Marital Count

EDUCATION

What kind of 'education clients this bank has, if you cross education with default, loan or housing, there is no relation

```
fig, ax = plt.subplots()
fig.set_size_inches(20, 5)
sns.countplot(x = 'education', data = bank_client)
ax.set_xlabel('Education', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
ax.set_title('Education Count Distribution', fontsize=15)
ax.tick_params(labelsize=15)
sns.despine()
```

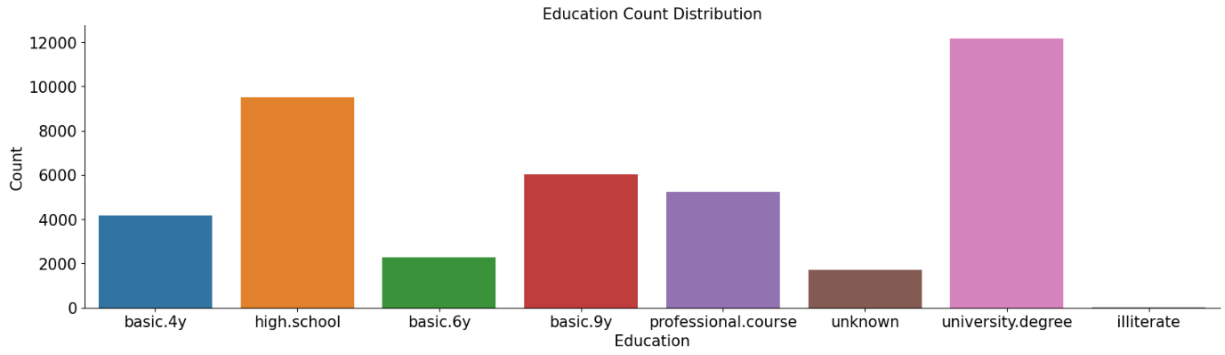


Figure 32: Education Count Distribution

DEFAULT, HOUSING, LOAN

Default, has credit in default?

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (20,8))
sns.countplot(x = 'default', data = bank_client, ax = ax1, order = ['no', 'unknown', 'yes'])
ax1.set_title('Default', fontsize=15)
ax1.set_xlabel('')
ax1.set_ylabel('Count', fontsize=15)
ax1.tick_params(labelsize=15)
```

Housing, has housing loan?

```
sns.countplot(x = 'housing', data = bank_client, ax = ax2, order = ['no', 'unknown', 'yes'])
ax2.set_title('Housing', fontsize=15)
ax2.set_xlabel('')
ax2.set_ylabel('Count', fontsize=15)
ax2.tick_params(labelsize=15)
```

Loan, has personal loan?

```
sns.countplot(x = 'loan', data = bank_client, ax = ax3, order = ['no', 'unknown', 'yes'])
ax3.set_title('Loan', fontsize=15)
ax3.set_xlabel('')
ax3.set_ylabel('Count', fontsize=15)
ax3.tick_params(labelsize=15)

plt.subplots_adjust(wspace=0.25)
```

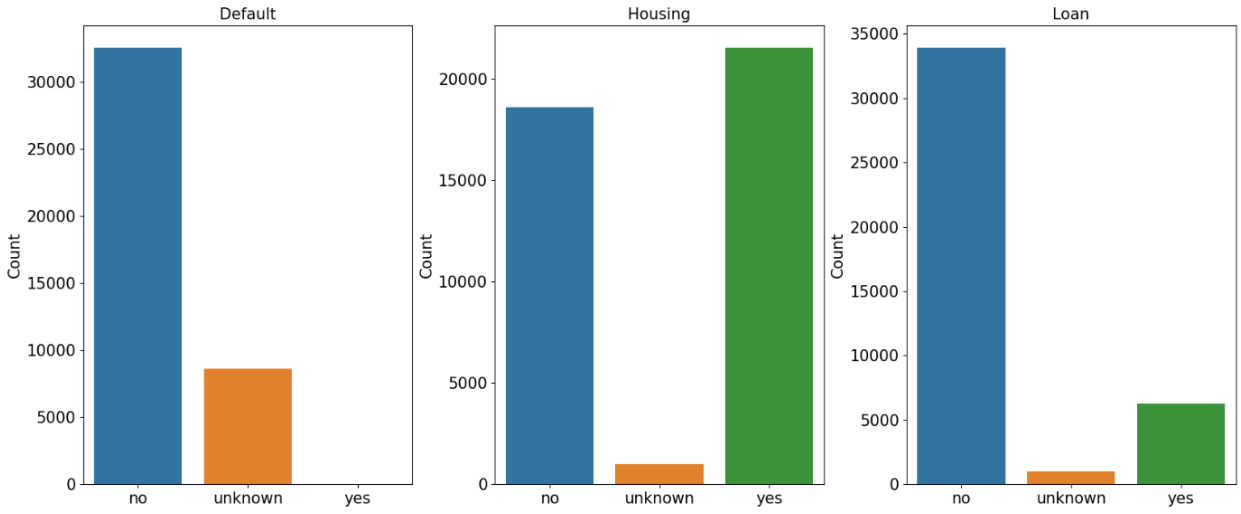


Figure 33: Default, Housing, Loan

```
print('Default:\n No credit in default:'    , bank_client[bank_client['default'] == 'no']    ['age'].count(),
      '\n Unknown credit in default:', bank_client[bank_client['default'] == 'unknown']['age'].count(),
      '\n Yes to credit in default:' , bank_client[bank_client['default'] == 'yes']    ['age'].count())
```

```
Default:
 No credit in default: 32577
 Unknown credit in default: 8596
 Yes to credit in default: 3
```

Figure 34: Credit

```
print('Housing:\n No housing in loan:'    , bank_client[bank_client['housing'] == 'no']    ['age'].count(),
      '\n Unknown housing in loan:', bank_client[bank_client['housing'] == 'unknown']['age'].count(),
      '\n Yes to housing in loan:' , bank_client[bank_client['housing'] == 'yes']    ['age'].count())
```

```
Housing:
 No housing in loan: 18615
 Unknown housing in loan: 990
 Yes to housing in loan: 21571
```

Figure 35: Housing Loan

```
print('Housing:\n No to personal loan:'    , bank_client[bank_client['loan'] == 'no']    ['age'].count(),
      '\n Unknown to personal loan:', bank_client[bank_client['loan'] == 'unknown']['age'].count(),
      '\n Yes to personal loan:'    , bank_client[bank_client['loan'] == 'yes']    ['age'].count())
```

```
Housing:
 No to personal loan: 33938
 Unknown to personal loan: 990
 Yes to personal loan: 6248
```

Figure 36 : Personal Loan

BANK CLIENTS' CONCLUSION

Jobs, Marital and Education I think the best analysis is just the count of each variable, if we relate with the other ones, it is not conclusive, all this kind of variables has yes, unknown and no for loan, default and housing.

Default, loan and housing, it's just to see the distribution of people.

Bank Client Categorical Treatment- Jobs, Marital, Education, Default, Housing, Loan.
Converting to continuous due the feature scaling will be applied later

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_client['job']      = labelencoder_X.fit_transform(bank_client['job'])
bank_client['marital']  = labelencoder_X.fit_transform(bank_client['marital'])
bank_client['education']= labelencoder_X.fit_transform(bank_client['education'])
bank_client['default']  = labelencoder_X.fit_transform(bank_client['default'])
bank_client['housing']  = labelencoder_X.fit_transform(bank_client['housing'])
bank_client['loan']     = labelencoder_X.fit_transform(bank_client['loan'])

#function to creat group of ages, this helps because we have 78 differente values here
def age(dataframe):
   dataframe.loc[dataframe['age'] <= 32, 'age'] = 1
   dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] <= 47), 'age'] = 2
   dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] <= 70), 'age'] = 3
   dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] <= 98), 'age'] = 4

   return dataframe

age(bank_client);

bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	3	3	1	0	0	0	0
1	3	7	1	3	1	0	0
2	2	7	1	3	0	2	0
3	2	0	1	1	0	0	0
4	3	7	1	3	0	0	2

Figure 37: Label Encoder

```
print(bank_client.shape)
bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	3	3	1	0	0	0	0
1	3	7	1	3	1	0	0
2	2	7	1	3	0	2	0
3	2	0	1	1	0	0	0
4	3	7	1	3	0	0	2

Figure 38:Nnew Data

Related with the last contact of the current campaign

```
bank_related = bank.iloc[: , 7:11]
bank_related.head()
```

	contact	month	day_of_week	duration
0	telephone	may	mon	261
1	telephone	may	mon	149
2	telephone	may	mon	226
3	telephone	may	mon	151
4	telephone	may	mon	307

Figure 39: Last Contact Detail

```
bank_related.isnull().any()
```

	0
contact	False
month	False
day_of_week	False
duration	False

dtype: bool

Figure 40: Checking null value

```
print("Kind of Contact: \n", bank_related['contact'].unique())
print("\nWhich monthis this campaing work: \n", bank_related['month'].unique())
print("\nWhich days of week this campaing work: \n", bank_related['day_of_week'].unique())
```

```
Kind of Contact:
 ['telephone' 'cellular']

Which monthis this campaing work:
 ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']

Which days of week this campaing work:
 ['mon' 'tue' 'wed' 'thu' 'fri']
```

Figure 41: Contact Details

Duration

```
fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2, figsize = (13, 5))
sns.boxplot(x = 'duration', data = bank_related, orient = 'v', ax = ax1)
ax1.set_xlabel('Calls', fontsize=10)
ax1.set_ylabel('Duration', fontsize=10)
ax1.set_title('Calls Distribution', fontsize=10)
ax1.tick_params(labelsize=10)

sns.distplot(bank_related['duration'], ax = ax2)
sns.despine(ax = ax2)
ax2.set_xlabel('Duration Calls', fontsize=10)
ax2.set_ylabel('Occurence', fontsize=10)
ax2.set_title('Duration x Ocucurence', fontsize=10)
ax2.tick_params(labelsize=10)

plt.subplots_adjust(wspace=0.5)
plt.tight_layout()
```

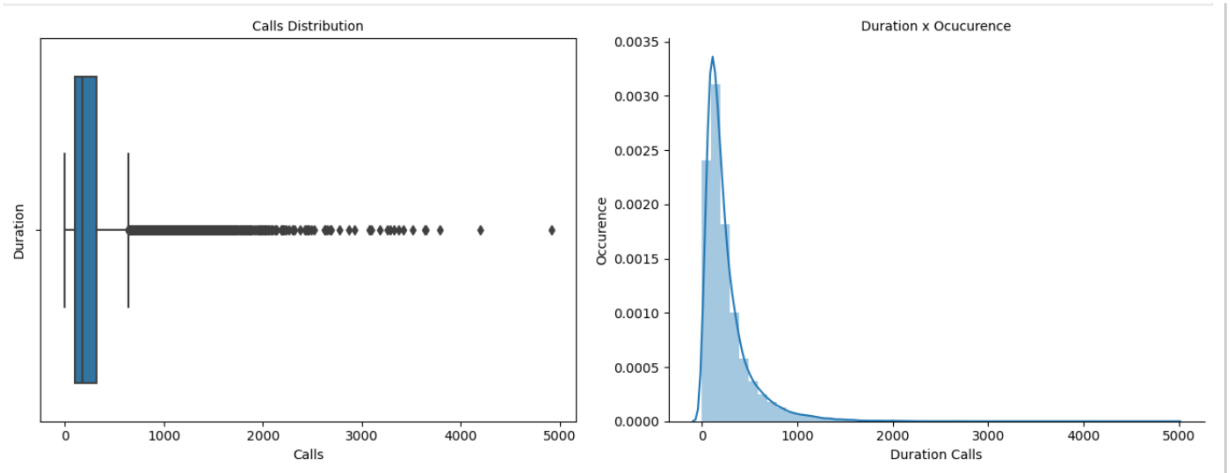


Figure 42: Call Duration

Please note: duration is different from age, Age has 78 values and Duration has 1544 different values

```
print("Max duration  call in minutes:  ", round((bank_related['duration'].max()/60),1))
print("Min duration  call in minutes:   ", round((bank_related['duration'].min()/60),1))
print("Mean duration call in minutes:   ", round((bank_related['duration'].mean()/60),1))
print("STD duration  call in minutes:   ", round((bank_related['duration'].std()/60),1))
# Std close to the mean means that the data values are close to the mean
```

```
Max duration  call in minutes:   82.0
Min duration  call in minutes:    0.0
Mean duration call in minutes:    4.3
STD duration  call in minutes:    4.3
```

Figure 43: Call Duration max, min

Calculation of outlier for duration

```
print('Duration calls above: ', bank_related['duration'].quantile(q = 0.75) +
                1.5*(bank_related['duration'].quantile(q = 0.75) - bank_related['duration'].quantile(q =
0.25)), 'are outliers')
```

```
Duration calls above:  644.5 are outliers
```

Figure 44: Outliers for duration

```
print('Numerber of outliers: ', bank_related[bank_related['duration'] > 644.5]['duration'].count())
print('Number of clients: ', len(bank_related))
#Outliers in %
print('Outliers are:', round(bank_related[bank_related['duration'] >
644.5]['duration'].count()*100/len(bank_related),2), '%')
```

```
Numerber of outliers:  2963
Number of clients:  41176
Outliers are: 7.2 %
```

Figure 45: Outliers in %

if the call duration is equal to 0, then is obviously that this person didn't subscribe.

```
bank[(bank['duration'] == 0)]
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	prev
6251	39	admin.	married	high.school	no	yes	no	telephone	may	tue	...	4	999	
23031	59	management	married	university.degree	no	yes	no	cellular	aug	tue	...	10	999	
28063	53	blue-collar	divorced	high.school	no	yes	no	cellular	apr	fri	...	3	999	
33015	31	blue-collar	married	basic.9y	no	no	no	cellular	may	mon	...	2	999	

4 rows × 21 columns

Figure 46: Duration ==0

Contact, Month, Day of Week

```
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 1, ncols = 3, figsize = (15,6))
sns.countplot(bank_related['contact'], ax = ax1)
ax1.set_xlabel('Contact', fontsize = 10)
ax1.set_ylabel('Count', fontsize = 10)
ax1.set_title('Contact Counts')
ax1.tick_params(labelsize=10)

sns.countplot(bank_related['month'], ax = ax2, order = ['mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct',
'nov', 'dec'])
ax2.set_xlabel('Months', fontsize = 10)
ax2.set_ylabel('')
ax2.set_title('Months Counts')
ax2.tick_params(labelsize=10)

sns.countplot(bank_related['day_of_week'], ax = ax3)
ax3.set_xlabel('Day of Week', fontsize = 10)
ax3.set_ylabel('')
ax3.set_title('Day of Week Counts')
ax3.tick_params(labelsize=10)

plt.subplots_adjust(wspace=0.25)
```

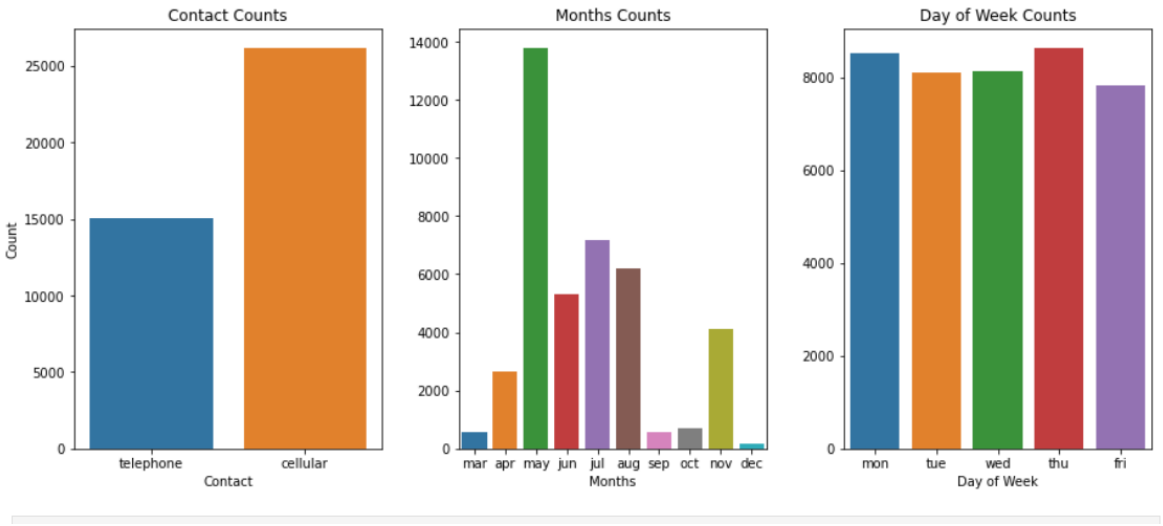


Figure 47: Contact Distribution

```
print('Ages above: ', bank_related['duration'].quantile(q = 0.75) +
                1.5*(bank_related['duration'].quantile(q = 0.75) - bank_related['duration'].quantile(q = 0.25)), 'are outliers')
```

```
Ages above:  644.5 are outliers
```

Figure 48: Outlier for Age

```
bank_related[bank_related['duration'] > 640].count()
```

	0
contact	3008
month	3008
day_of_week	3008
duration	3008

dtype: int64

Figure 49: Contact count

Contact, Month, Day of Week treatment

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_related['contact']     = labelencoder_X.fit_transform(bank_related['contact'])
bank_related['month']       = labelencoder_X.fit_transform(bank_related['month'])
bank_related['day_of_week'] = labelencoder_X.fit_transform(bank_related['day_of_week'])
```

```
bank_related.head()
```

	contact	month	day_of_week	duration
0	1	6	1	261
1	1	6	1	149
2	1	6	1	226
3	1	6	1	151
4	1	6	1	307

Figure 50: bank related Head

Converting the Duration into minutes

```
def duration(data):

    data.loc[data['duration'] <= 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] <= 180)  , 'duration']   = 2
    data.loc[(data['duration'] > 180) & (data['duration'] <= 319)  , 'duration']   = 3
    data.loc[(data['duration'] > 319) & (data['duration'] <= 644.5), 'duration']   = 4
    data.loc[data['duration']  > 644.5, 'duration'] = 5

    return data
duration(bank_related);

bank_related.head()
```

	contact	month	day_of_week	duration
0	1	6	1	3
1	1	6	1	2
2	1	6	1	3
3	1	6	1	2
4	1	6	1	3

Figure 51: Duration in minutes

Social and economic context attributes

```
bank_se = bank.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed']]
bank_se.head()
```

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	1.1	93.994	-36.4	4.857	5191.0
1	1.1	93.994	-36.4	4.857	5191.0
2	1.1	93.994	-36.4	4.857	5191.0
3	1.1	93.994	-36.4	4.857	5191.0
4	1.1	93.994	-36.4	4.857	5191.0

Figure 52: Social & Economic context

Other attributes

```
bank_o = bank.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
bank_o.head()
```

	campaign	pdays	previous	poutcome
0	1	999	0	nonexistent
1	1	999	0	nonexistent
2	1	999	0	nonexistent
3	1	999	0	nonexistent
4	1	999	0	nonexistent

Figure 53: Other attributes

```
bank_o['poutcome'].unique()
```

```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

Figure 54: poutcome

```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inplace  = True)
```

Model

```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o], axis = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
                'contact', 'month', 'day_of_week', 'duration', 'emp.var.rate', 'cons.price.idx',
                'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'pdays', 'previous', 'poutcome']]
bank_final.shape
```

```
(41176, 20)
```

Figure 55: data.shape()

```
from sklearn.ensemble import RandomForestClassifier
```

Feature Importance using Random Forest

```
pd.DataFrame(data =
RandomForestClassifier().fit(bank_final,y).feature_importances_,index=bank_final.columns
        ,columns=['Feature Importance']).plot.barh();
```

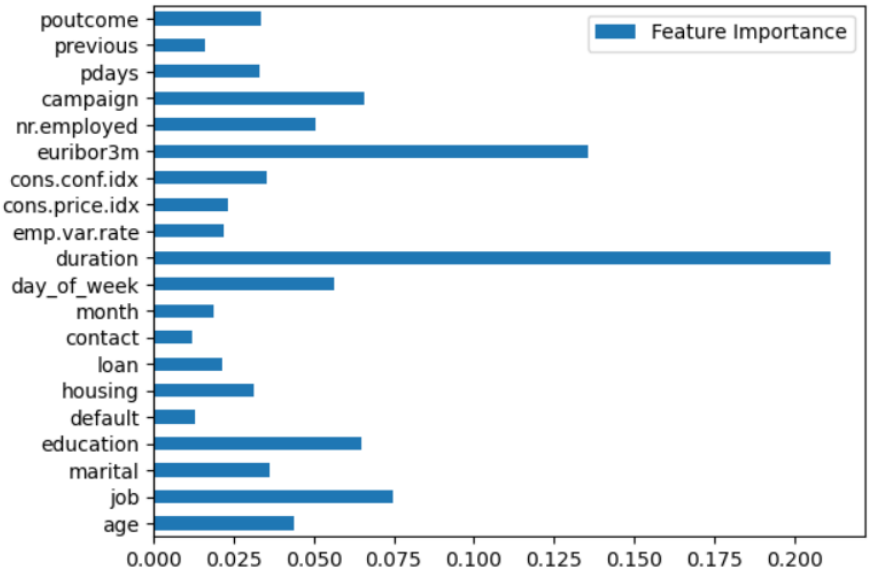


Figure 56: Feature Importance

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size = 0.2, random_state = 101)
```

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
k_fold = KFold(n_splits=10, shuffle=True, random_state=0)

X_train.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	ca
23147	2	9	1	3	0	0	0	0	1	3	2	1.4	93.444	-36.1	4.965	5228.1	
13990	2	9	0	6	0	0	2	0	3	0	3	1.4	93.918	-42.7	4.963	5228.1	
18828	2	0	1	6	0	0	0	0	1	1	2	1.4	93.444	-36.1	4.970	5228.1	
25971	3	2	1	6	0	2	0	0	7	4	4	-0.1	93.200	-42.0	4.120	5195.8	
2943	3	0	1	6	1	0	0	1	6	4	2	1.1	93.994	-36.4	4.859	5191.0	

Figure 57: X_train

```
from sklearn.preprocessing import MinMaxScaler
sc_X = MinMaxScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

from sklearn.metrics import cohen_kappa_score
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)
print('The Confusion Matrix \n',+confusion_matrix(y_test, logpred),end='\n\n')
print('Accuracy : ',+round(accuracy_score(y_test, logpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, logpred),2))
LOGCV = (cross_val_score(logmodel, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring)).mean()
```

```
The Confusion Matrix
 [[7107  151]
 [ 600  378]]

Accuracy :  91.0
Cohen kappa :  0.46
```

Figure 58: LR CM

K Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knnpred = knn.predict(X_test)

print(confusion_matrix(y_test, knnpred))
```

```
print(round(accuracy_score(y_test, knnpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, knnpred),2))
KNNCV = (cross_val_score(knn, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring).mean())
```

```
[[7039  219]
 [ 600  378]]
90.0
Cohen kappa :  0.43
```

Figure 59:KN CM

SVC

```
from sklearn.svm import SVC
svc= SVC(kernel = 'sigmoid')
svc.fit(X_train, y_train)
svcpred = svc.predict(X_test)
print(confusion_matrix(y_test, svcpred))
print(round(accuracy_score(y_test, svcpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, svcpred),2))
SVCCV = (cross_val_score(svc, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring).mean())
```

```
[[6519  739]
 [ 756  222]]
82.0
Cohen kappa :  0.13
```

Figure 60: SVC CM

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(criterion='gini')
dtree.fit(X_train, y_train)
dtreepred = dtree.predict(X_test)

print(confusion_matrix(y_test, dtreepred))
print(round(accuracy_score(y_test, dtreepred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, dtreepred),2))
DTREECV = (cross_val_score(dtree, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring).mean())
```

```
[[6776  482]
 [ 486  492]]
88.0
Cohen kappa :  0.44
```

Figure 61: DTC CM

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 200)
rfc.fit(X_train, y_train)
rfcpred = rfc.predict(X_test)

print(confusion_matrix(y_test, rfcpred ))
print(round(accuracy_score(y_test, rfcpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, rfcpred),2))
RFCCV = (cross_val_score(rfc, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring).mean())
```

```
[[7005  253]
 [ 487  491]]
91.0
Cohen kappa :  0.52
```

Figure 62: RFC CM

Gaussian NB

```
from sklearn.naive_bayes import GaussianNB
gaussiannb= GaussianNB()
gaussiannb.fit(X_train, y_train)
gaussiannbpred = gaussiannb.predict(X_test)
probs = gaussiannb.predict(X_test)

print(confusion_matrix(y_test, gaussiannbpred ))
print(round(accuracy_score(y_test, gaussiannbpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, gaussiannbpred),2))
GAUSIAN = (cross_val_score(gaussiannb, X_train, y_train, cv=k_fold, n_jobs=1, scoring =
scoring).mean())
```

```
[[6375  883]
 [ 415  563]]
84.0
Cohen kappa :  0.38
```

Figure 63: GNB CM

XGB Classifier

```
from xgboost import XGBClassifier
xgb = XGBClassifier(eval_metric='logloss')
xgb.fit(X_train, y_train)
xgbprd = xgb.predict(X_test)

print(confusion_matrix(y_test, xgbprd ))
print(round(accuracy_score(y_test, xgbprd),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, xgbprd),2))
XGB = (cross_val_score(estimator = xgb, X = X_train, y = y_train, cv = 10).mean())
```

```
[[7013  245]
 [ 466  512]]
91.0
Cohen kappa :  0.54
```

Figure 64:XGB CM

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
gbk = GradientBoostingClassifier()
gbk.fit(X_train, y_train)
gbkpred = gbk.predict(X_test)
print(confusion_matrix(y_test, gbkpred ))
print(round(accuracy_score(y_test, gbkpred),2)*100)
print('Cohen kappa : ',+round(cohen_kappa_score(y_test, gbkpred),2))
GBKCV = (cross_val_score(gbk, X_train, y_train, cv=k_fold, n_jobs=1, scoring = scoring).mean())
```

```
[[7039  219]
 [ 464  514]]
92.0
Cohen kappa :  0.56
```

Figure 65: GB CM

```
models = pd.DataFrame({
        'Models': ['Random Forest Classifier', 'Decision Tree Classifier', 'Support Vector Machine',
```

```
            'K-Near Neighbors', 'Logistic Model', 'Gausian NB', 'XGBoost', 'Gradient Boosting'],
        'Score':  [RFCCV, DTREECV, SVCCV, KNNCV, LOGCV, GAUSIAN, XGB, GBKCV]})

models.sort_values(by='Score', ascending=False)
```

	Models	Score
7	Gradient Boosting	0.913206
6	XGBoost	0.910109
4	Logistic Model	0.908682
0	Random Forest Classifier	0.908318
3	K-Near Neighbors	0.899423
1	Decision Tree Classifier	0.882939
5	Gausian NB	0.846448
2	Support Vector Machine	0.828962

Figure 66: Model scores

XGBOOST ROC/ AUC, BEST MODEL

```
from sklearn import metrics
fig, (ax, ax1) = plt.subplots(nrows = 1, ncols = 2, figsize = (15,5))
probs = xgb.predict_proba(X_test)
preds = probs[:,1]
fprxgb, tprxgb, thresholdxgb = metrics.roc_curve(y_test, preds)
roc_aucxgb = metrics.auc(fprxgb, tprxgb)

ax.plot(fprxgb, tprxgb, 'b', label = 'AUC = %0.2f' % roc_aucxgb)
ax.plot([0, 1], [0, 1],'r--')
ax.set_title('Receiver Operating Characteristic XGBOOST ',fontsize=10)
ax.set_ylabel('True Positive Rate',fontsize=20)
ax.set_xlabel('False Positive Rate',fontsize=15)
ax.legend(loc = 'lower right', prop={'size': 16})
```

Gradient

```
probs = gbk.predict_proba(X_test)
preds = probs[:,1]
fprgbk, tprgbk, thresholdgbk = metrics.roc_curve(y_test, preds)
roc_aucgbk = metrics.auc(fprgbk, tprgbk)
```

```
ax1.plot(fprgbk, tprgbk, 'b', label = 'AUC = %0.2f' % roc_aucgbk)
ax1.plot([0, 1], [0, 1],'r--')
ax1.set_title('Receiver Operating Characteristic GRADIENT BOOST ',fontsize=10)
ax1.set_ylabel('True Positive Rate',fontsize=20)
ax1.set_xlabel('False Positive Rate',fontsize=15)
ax1.legend(loc = 'lower right', prop={'size': 16})

plt.subplots_adjust(wspace=1)
```

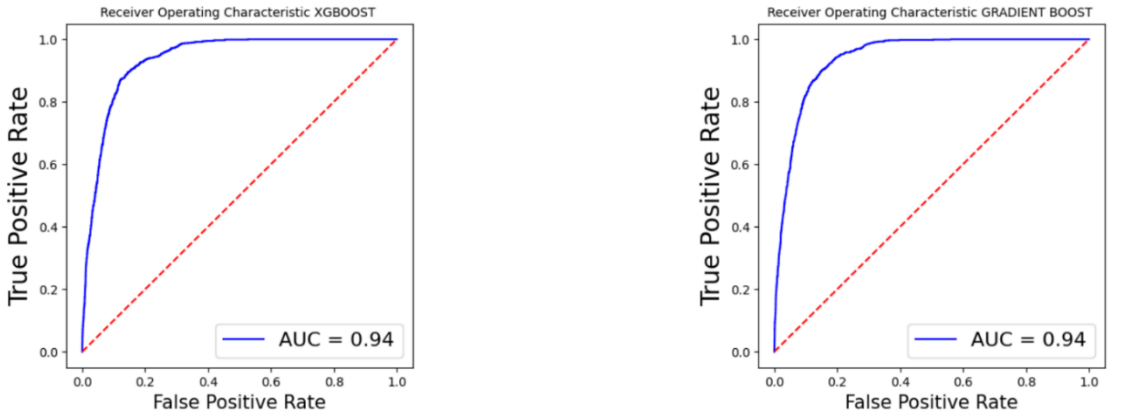


Figure 67: XG BOOST vs Gradient

Logistic, Random Forest, KNN, Gaussian DB, Decision Tree

```
fig, ax_arr = plt.subplots(nrows = 2, ncols = 3, figsize = (25,20))
```

---------- LOGMODEL ----------

```
probs = logmodel.predict_proba(X_test)
preds = probs[:,1]
fprlog, tprlog, thresholdlog = metrics.roc_curve(y_test, preds)
roc_auclog = metrics.auc(fprlog, tprlog)

ax_arr[0,0].plot(fprlog, tprlog, 'b', label = 'AUC = %0.2f' % roc_auclog)
ax_arr[0,0].plot([0, 1], [0, 1],'r--')
ax_arr[0,0].set_title('Receiver Operating Characteristic Logistic ',fontsize=20)
ax_arr[0,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,0].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,0].legend(loc = 'lower right', prop={'size': 16})
```

---------- RANDOM FOREST ----------

```
probs = rfc.predict_proba(X_test)
preds = probs[:,1]
```

```
fprrfc, tprrfc, thresholdrfc = metrics.roc_curve(y_test, preds)
roc_aucrfc = metrics.auc(fprrfc, tprrfc)

ax_arr[0,1].plot(fprrfc, tprrfc, 'b', label = 'AUC = %0.2f' % roc_aucrfc)
ax_arr[0,1].plot([0, 1], [0, 1],'r--')
ax_arr[0,1].set_title('Receiver Operating Characteristic Random Forest ',fontsize=20)
ax_arr[0,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,1].legend(loc = 'lower right', prop={'size': 16})
```

---------- KNN----------

```
probs = knn.predict_proba(X_test)
preds = probs[:,1]
fprknn, tprknn, thresholdknn = metrics.roc_curve(y_test, preds)
roc_aucknn = metrics.auc(fprknn, tprknn)

ax_arr[0,2].plot(fprknn, tprknn, 'b', label = 'AUC = %0.2f' % roc_aucknn)
ax_arr[0,2].plot([0, 1], [0, 1],'r--')
ax_arr[0,2].set_title('Receiver Operating Characteristic KNN ',fontsize=20)
ax_arr[0,2].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[0,2].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[0,2].legend(loc = 'lower right', prop={'size': 16})
```

---------- DECISION TREE ----------

```
probs = dtree.predict_proba(X_test)
preds = probs[:,1]
fprdtree, tprdtree, thresholddtree = metrics.roc_curve(y_test, preds)
roc_aucdtree = metrics.auc(fprdtree, tprdtree)

ax_arr[1,0].plot(fprdtree, tprdtree, 'b', label = 'AUC = %0.2f' % roc_aucdtree)
ax_arr[1,0].plot([0, 1], [0, 1],'r--')
ax_arr[1,0].set_title('Receiver Operating Characteristic Decision Tree ',fontsize=20)
ax_arr[1,0].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,0].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,0].legend(loc = 'lower right', prop={'size': 16})
```

---------- GAUSSIAN ----------

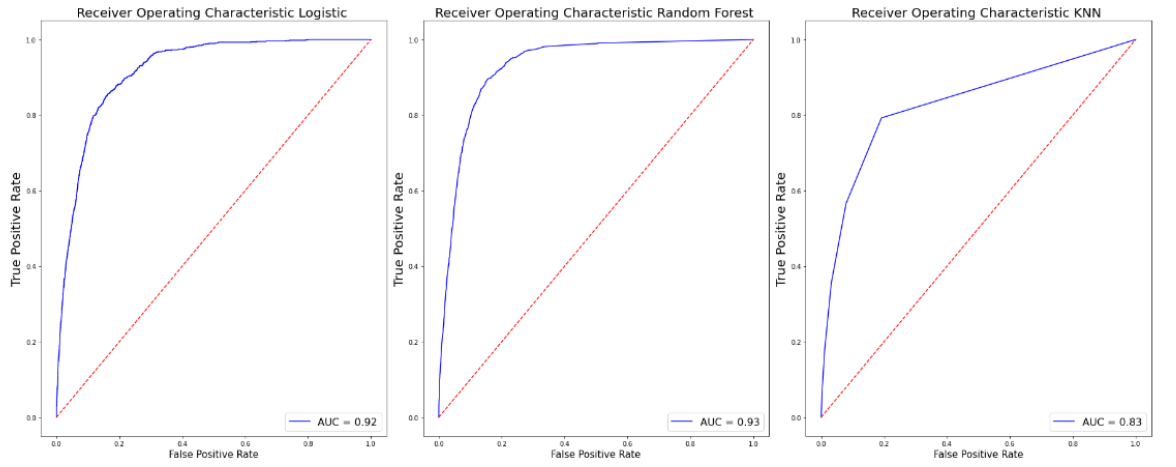
```
probs = gaussiannb.predict_proba(X_test)
preds = probs[:,1]
fprgau, tprgau, thresholdgau = metrics.roc_curve(y_test, preds)
roc_aucgau = metrics.auc(fprgau, tprgau)
```

```
ax_arr[1,1].plot(fprgau, tprgau, 'b', label = 'AUC = %0.2f' % roc_aucgau)
ax_arr[1,1].plot([0, 1], [0, 1],'r--')
ax_arr[1,1].set_title('Receiver Operating Characteristic Gaussian ',fontsize=20)
ax_arr[1,1].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,1].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,1].legend(loc = 'lower right', prop={'size': 16})
```

---------- ALL PLOTS ----------

```
ax_arr[1,2].plot(fprgau, tprgau, 'b', label = 'Gaussian', color='black')
ax_arr[1,2].plot(fprdtree, tprdtree, 'b', label = 'Decision Tree', color='blue')
ax_arr[1,2].plot(fprknn, tprknn, 'b', label = 'Knn', color='brown')
ax_arr[1,2].plot(fprrfc, tprrfc, 'b', label = 'Random Forest', color='green')
ax_arr[1,2].plot(fprlog, tprlog, 'b', label = 'Logistic', color='grey')
ax_arr[1,2].set_title('Receiver Operating Comparison ',fontsize=20)
ax_arr[1,2].set_ylabel('True Positive Rate',fontsize=20)
ax_arr[1,2].set_xlabel('False Positive Rate',fontsize=15)
ax_arr[1,2].legend(loc = 'lower right', prop={'size': 16})

plt.subplots_adjust(wspace=0.2)
plt.tight_layout()
```



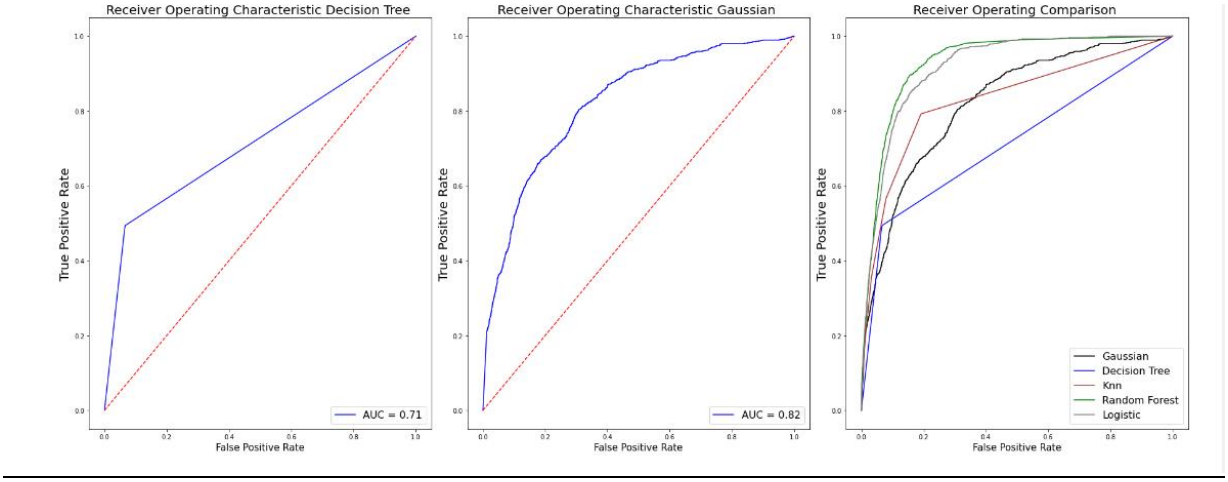


Figure 68: ROC curve

ANALYZING THE RESULTS

we have to decide which one is the best model, and we have two types of wrong values:

- False Positive, means the client do NOT SUBSCRIBED to term deposit, but the model thinks he did.
- False Negative, means the client SUBSCRIBED to term deposit, but the model said he don't.
- The first one its most harmful, because we think that we already have that client but we don't and maybe we lost him in other future campaigns
- The second is not good but its ok, we have that client and, in the future, we'll identify that in truth he's already our client

our objective here, is to find the best model by confusion matrix with the lowest False Positive as possible.

```
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score
```

Confusion Matrix and Classification Report of KNN

```
print('KNN Confusion Matrix\n', confusion_matrix(y_test, knnpred),end='\n\n')
print('KNN Reports\n',classification_report(y_test, knnpred))
```

```
KNN Confusion Matrix
 [[7039  219]
 [ 600  378]]

KNN Reports
              precision    recall  f1-score   support

       False       0.92      0.97      0.95      7258
        True       0.63      0.39      0.48       978

    accuracy                           0.90      8236
   macro avg       0.78      0.68      0.71      8236
weighted avg       0.89      0.90      0.89      8236
```

Figure 69: KNN report

Confusion Matrix and Classification Report of SVC

```
print('SVC Confusion Matrix\n', confusion_matrix(y_test, svcpred),end='\n\n')
print('SVC Reports\n',classification_report(y_test, svcpred))
```

```
SVC Confusion Matrix
 [[6519  739]
 [ 756  222]]

SVC Reports
              precision    recall  f1-score   support

       False       0.90      0.90      0.90      7258
        True       0.23      0.23      0.23       978

    accuracy                           0.82      8236
   macro avg       0.56      0.56      0.56      8236
weighted avg       0.82      0.82      0.82      8236
```

Figure 70: SVC report

Confusion Matrix and Classification Report of Decision Tree

```
print('Decision Tree Confusion Matrix\n', confusion_matrix(y_test, dtreepred),end='\n\n')
print('Decision Tree Reports\n',classification_report(y_test, dtreepred))
```

```
Decision Tree Confusion Matrix
 [[6776  482]
 [ 486  492]]

Decision Tree Reports
              precision    recall  f1-score   support

       False       0.93      0.93      0.93      7258
        True       0.51      0.50      0.50       978

    accuracy                           0.88      8236
   macro avg       0.72      0.72      0.72      8236
weighted avg       0.88      0.88      0.88      8236
```

Figure 71: DT report

Confusion Matrix and Classification Report of Random Forest

```
print('Random Forest Confusion Matrix\n', confusion_matrix(y_test, rfcpred),end='\n\n')
print('Random Forest Reports\n',classification_report(y_test, rfcpred))
```

```
Random Forest Confusion Matrix
 [[7005  253]
 [ 487  491]]

Random Forest Reports
              precision    recall  f1-score   support

       False       0.93      0.97      0.95      7258
        True       0.66      0.50      0.57       978

    accuracy                           0.91      8236
   macro avg       0.80      0.73      0.76      8236
weighted avg       0.90      0.91      0.90      8236
```

Figure 72: RF Report

Confusion Matrix And Classification Report of Gaussian

```
print('Gaussian Confusion Matrix\n', confusion_matrix(y_test, gaussiannbpred),end='\n\n')
print('Gaussian Forest Reports\n',classification_report(y_test, gaussiannbpred))
```

```
Gaussian Confusion Matrix
 [[6375  883]
 [ 415  563]]

Gaussian Forest Reports
              precision    recall  f1-score   support

       False       0.94      0.88      0.91      7258
        True       0.39      0.58      0.46       978

    accuracy                           0.84      8236
   macro avg       0.66      0.73      0.69      8236
weighted avg       0.87      0.84      0.85      8236
```

Figure 73: Gaussian Report

Confusion Matrix And Classification Report of Xg Boost

```
print('Xgboost Confusion Matrix\n', confusion_matrix(y_test, xgbprd),end='\n\n')
print('Xgboost Forest Reports\n',classification_report(y_test, xgbprd))
```

```
Xgboost Confusion Matrix
 [[7013  245]
 [ 466  512]]

Xgboost Forest Reports
              precision    recall  f1-score   support

       False       0.94      0.97      0.95      7258
        True       0.68      0.52      0.59       978

    accuracy                           0.91      8236
   macro avg       0.81      0.74      0.77      8236
weighted avg       0.91      0.91      0.91      8236
```

Figure 74: XG report

Confusion_Matrix And Classification_Report of Gradient Boost

```
print('Gradient Boosting Confusion Matrix\n', confusion_matrix(y_test, gbkpred),end='\n\n')
print('Gradient Boosting Reports\n',classification_report(y_test, gbkpred))
```

```
Gradient Boosting Confusion Matrix
 [[7039  219]
 [ 464  514]]

Gradient Boosting Reports
              precision    recall  f1-score   support

       False       0.94      0.97      0.95      7258
        True       0.70      0.53      0.60       978

    accuracy                           0.92      8236
   macro avg       0.82      0.75      0.78      8236
weighted avg       0.91      0.92      0.91      8236
```

Figure 75: Gradient report

Conclusion:

After the analysis we see that our interest is over decreasing the False Negative means the client SUBSCRIBED to term deposit, but the model said he dont which indicates RECALL. So, we conclude that the model with high RECALL would be best suited for the problem statement.

```
pd.DataFrame(data = [recall_score(y_test,logpred, average='weighted'),
recall_score(y_test,knnpred, average='weighted'),
recall_score(y_test,svcpred, average='weighted'),
recall_score(y_test,dtreepred, average='weighted'),
recall_score(y_test,rfcpred, average='weighted'),
recall_score(y_test,gaussiannbpred, average='weighted'),
recall_score(y_test,xgbprd, average='weighted'),
recall_score(y_test,gbkpred, average='weighted')],index=['Logistic','KNN','SVC','DT','RF','NB','XG','GB'],
        columns=['Recall Score']).sort_values(by='Recall Score',ascending=False)
```

	Recall Score
GB	0.917071
XG	0.913672
RF	0.910151
Logistic	0.908815
KNN	0.900559
DT	0.882467
NB	0.842399
SVC	0.818480

Figure 76: Recall score

Prediction on the test data

In the below task, we have performed a prediction on the test data. We have used Gradient Boost for this prediction.

We have to perform the same preprocessing operations on the test data that we have performed on the train data.

We then make a prediction on the preprocessed test data using the Gradient Boost model with the best parameter values we've got.

Preprocessed Test File

```
test = pd.read_csv('/content/drive/MyDrive/Bank.csv',delimiter=";")
test.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent	1.1	93.994	

5 rows × 21 columns

Figure 77: Processed test files

Assuming 'test' is your DataFrame

```
test.insert(0, 'Serial Number', range(1, 1 + len(test)))
test.head()
```

	Serial Number	age	job	marital	education	default	housing	loan	contact	month	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.
0	1	56	housemaid	married	basic.4y	no	no	no	telephone	may	...	1	999	0	nonexistent	1.1	93.994	
1	2	57	services	married	high.school	unknown	no	no	telephone	may	...	1	999	0	nonexistent	1.1	93.994	
2	3	37	services	married	high.school	no	yes	no	telephone	may	...	1	999	0	nonexistent	1.1	93.994	
3	4	40	admin.	married	basic.6y	no	no	no	telephone	may	...	1	999	0	nonexistent	1.1	93.994	
4	5	56	services	married	high.school	no	no	yes	telephone	may	...	1	999	0	nonexistent	1.1	93.994	

5 rows × 22 columns

Figure 78: Test

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import plotly.express as px
import warnings
from google.colab import drive
from google.colab import files
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import cohen_kappa_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score
from imblearn.over_sampling import SMOTE
```

Split data into train and test sets

```
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size=0.2, random_state=101)
```

Scale features

```
sc_X = MinMaxScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Assuming 'test' is your DataFrame

```
test = pd.read_csv('/content/drive/MyDrive/Bank.csv', delimiter=";")
test.insert(0, 'Serial Number', range(1, 1 + len(test)))
```

Initialize SMOTE

```
smote = SMOTE(random_state=42)
```

Apply SMOTE to the training data

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

Train the Gradient Boosting model with SMOTE data

```
gbk_smote = GradientBoostingClassifier()
gbk_smote.fit(X_train_smote, y_train_smote)
```

Predict using the Gradient Boosting model with SMOTE data

```
gbk_smote_pred = gbk_smote.predict(X_test)
```

Get serial number input from the user

```
serial_number = int(input("Enter the serial number: "))
```

Find the corresponding row in the test dataset

```
try:
  row_index = test[test['Serial Number'] == serial_number].index[0]
  age = test.loc[row_index, 'age']
  job = test.loc[row_index, 'job']
  marital = test.loc[row_index, 'marital']
  education = test.loc[row_index, 'education']
  housing = test.loc[row_index, 'housing']
  loan = test.loc[row_index, 'loan']
  duration = test.loc[row_index, 'duration']
  campaign = test.loc[row_index, 'campaign']
  poutcome = test.loc[row_index, 'poutcome']
  term_deposit_prediction = gbk_smote_pred[row_index]

  print(f"Serial Number: {serial_number}")
  print(f"Age: {age}")
  print(f"Job: {job}")
  print(f"Marital: {marital}")
  print(f"Education: {education}")
  print(f"Housing: {housing}")
  print(f"Loan: {loan}")
  print(f"Duration: {duration}")
  print(f"Campaign: {campaign}")
  print(f"Poutcome: {poutcome}")
  print(f"Term Deposit Prediction: {term_deposit_prediction}")

except IndexError:
  print(f"Serial Number {serial_number} not found in the dataset.")
```

```
Enter the serial number: 1
Serial Number: 1
Age: 56
Job: housemaid
Marital: married
Education: basic.4y
Housing: no
Loan: no
Duration: 261
Campaign: 1
Poutcome: nonexistent
Term Deposit Prediction: False
```

Figure 79: Predicted Output

POWER BI VISUALIZATION

The attached Power BI dashboard provides an insightful analysis of the Banco de Portugal marketing dataset. Through various visual elements, the dashboard offers a detailed breakdown of key metrics such as the average client age, loan status, and campaign outcomes segmented by client marital status and month of contact. It also visualizes important trends like the confidence index by job and the relationship between campaign effectiveness and contact type. Interactive filters allow for dynamic exploration of different client attributes, such as age, contact method (cellular or telephone), and loan status. This visual representation enhances the understanding of how different factors contribute to term deposit subscription rates, supporting more targeted and data-driven marketing strategies for the bank.

Figure 80: Power Bi Visualization

FINDINGS & INSIGHTS

In this project, multiple machine learning algorithms were applied to predict whether a client would subscribe to a term deposit based on Banco de Portugal's marketing campaign dataset. The algorithms used include K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), Decision Tree, Random Forest, Naive Bayes, Extreme Gradient Boosting (XG Boost), and Gradient Boosting. Below are the key findings and insights based on the model evaluation and performance comparison:

1. Model Accuracy:

- Among all the models, **Gradient Boosting** demonstrated the highest accuracy score, making it the most reliable predictor of term deposit subscription in this context.
- **XG Boost** also performed well, closely following Gradient Boosting in terms of accuracy, while other models like Random Forest and Logistic Regression provided competitive results but slightly lower accuracy.
- **Support Vector Classifier (SVC)** and **Naive Bayes (NB)** had the lowest accuracy scores, indicating that they were less effective at predicting the target variable in this dataset.

2. Recall Score:

- **Gradient Boosting** again outperformed all other models with the highest recall score (0.917071), indicating that it was the best at correctly identifying clients who would subscribe to the term deposit.
- **XG Boost** and **Random Forest** had similar recall scores, showing their strong ability to recall the true positives.

- **Support Vector Classifier (SVC)** and **Naive Bayes (NB)** showed the lowest recall, indicating they had a higher chance of missing true positives compared to other models.

3. ROC Curve and AUC (Area Under the Curve):

- Both **XG Boost** and **Gradient Boosting** demonstrated the same high value for the AUC, suggesting that these models were equally effective at distinguishing between clients who would and would not subscribe to the term deposit.
- The high AUC value for both models indicate strong discriminatory power, showing that these models are capable of accurately classifying the majority of cases.

4. Model Rankings:

- The top-performing model in terms of overall accuracy and recall was **Gradient Boosting**, followed closely by **XG Boost**.
- **Logistic Regression** and **Random Forest** also performed well, indicating that simpler models can still provide relatively high accuracy and predictive power.
- **K-Nearest Neighbors (KNN)** and **Decision Tree** were less effective, but still offered decent performance.
- **Support Vector Classifier (SVC)** and **Naive Bayes (NB)** consistently ranked lower across all metrics, suggesting they are not well-suited for this specific dataset.

5. Term Deposit Prediction:

- When predicting term deposit subscription for a sample client (serial number: 1), the model predicted that the client (housemaid, married, basic education, no housing or personal loan) would **not subscribe** to

a term deposit. This prediction aligns with the characteristics of the client, who may not have strong financial motivation or previous positive experiences with term deposits.

6. Insights for Banco de Portugal:

- **Gradient Boosting and XG Boost** are the most effective models for predicting term deposit subscription and should be considered for deployment in future marketing campaigns.
- **High recall scores** suggest that these models can help the bank avoid missing potential clients who are likely to subscribe, optimizing the campaign's success.
- By focusing on clients with higher predicted subscription likelihood based on these models, Banco de Portugal can increase the efficiency of their marketing efforts, reduce costs, and improve campaign outcomes.
- **Key features** such as education, housing loan, job type, and previous campaign outcomes were found to be significant indicators of client behavior. Banco de Portugal can use these insights to tailor their marketing messages to different client segments.

The implementation of Gradient Boosting and XG Boost provides Banco de Portugal with powerful predictive tools that can greatly enhance the success of their marketing campaigns by better targeting clients most likely to subscribe to term deposits.

CONCLUSIONS

In this project, we evaluated multiple machine learning models to predict whether clients would subscribe to a term deposit based on Banco de Portugal's marketing data. The models tested include Gradient Boosting, XGBoost, Random Forest, K-Nearest Neighbors, Logistic Regression, and others. Among these, **Gradient Boosting** and **XGBoost** performed the best, with Gradient Boosting achieving the highest accuracy (0.913) and recall score (0.917), making it the most effective predictor. These results show that advanced ensemble methods significantly outperform simpler algorithms like Naive Bayes and Support Vector Classifier, which had lower accuracy and recall metrics.

The findings indicate that important client features, such as **education, job type, housing loan status, and previous campaign outcomes**, play a key role in determining the likelihood of term deposit subscription. By leveraging these models, Banco de Portugal can improve the efficiency of its marketing campaigns, targeting the right clients and maximizing engagement. The high recall scores of the best models suggest that fewer potential clients will be missed, helping the bank increase subscription rates while optimizing resource allocation.

ANNEXURE

1. Dataset Description:

The dataset used in this project contains 45,211 instances and 16 features, which are divided into the following categories:

- **Client Demographics**: Age, job, marital status, education, balance, housing loan, and personal loan.
- **Campaign Data**: Contact type, last contact day, month, and the number of contacts.
- **Previous Campaign Outcomes**: Number of days since the last contact, number of previous contacts, and the outcome of the previous campaign.
- **Target Variable**: Whether the client subscribed to a term deposit (yes/no).

2. Machine Learning Algorithms Used:

The following algorithms were applied to predict the likelihood of a client subscribing to a term deposit:

- K-Nearest Neighbors (KNN)
- Support Vector Classifier (SVC)
- Decision Tree
- Random Forest
- Naive Bayes (Gaussian NB)
- XGBoost (Extreme Gradient Boosting)
- Gradient Boosting

3. Model Evaluation Metrics:

- **Accuracy Score**: Evaluates the overall correctness of the model's predictions.
- **Recall Score**: Measures the ability of the model to correctly identify the true positives (clients who would subscribe).
- **ROC Curve and AUC**: Shows the model's ability to distinguish between classes, with higher AUC values indicating better performance.

4. Performance Comparison of Models:

- **Accuracy and Recall Scores**:
  - Gradient Boosting: Accuracy = 0.913, Recall = 0.917
  - XGBoost: Accuracy = 0.910, Recall = 0.913
  - Random Forest: Accuracy = 0.908, Recall = 0.910
  - SVC and Naive Bayes performed the worst among the models.

5. Software and Tools Used:

- **Python Libraries**: Scikit-learn, XGBoost, Matplotlib, Pandas, NumPy.
- **Data Visualization**: Power BI was used to create visual representations of data distributions, feature importance, and campaign performance.
- **Evaluation Tools**: Confusion Matrix, ROC-AUC curves, and performance metrics for detailed model evaluation.

6. Visualizations:

- **Power BI Dashboard**: The project also included an interactive Power BI dashboard that visualizes key insights such as campaign success rates, client demographics, and model outcomes.

BIBLIOGRAPHY

Academic References

1. **Yeh, I. C., & Lien, C. H. (2009).** "The Application of Data Mining to Predict the Subscription of Bank Term Deposits." *Journal of Banking and Finance*, 33(6), 1243-1251.
    - This paper discusses data mining techniques applied to banking data, specifically targeting customer subscription behaviors.
2. **Kotu, V., & Deshpande, S. (2019).** "Data Science: Concepts and Practice." *Elsevier*.
    - A comprehensive guide covering data science methodologies, including predictive modeling techniques relevant to your project.
3. **Han, J., Kamber, M., & Pei, J. (2011).** "Data Mining: Concepts and Techniques." *Morgan Kaufmann*.
    - This textbook provides foundational knowledge in data mining, including classification and regression techniques that can be applied to your dataset.

Technical References

4. **Scikit-learn Documentation.** "User Guide."
    - Offers detailed information on various machine learning algorithms and their implementation, which will be essential for your model development.
    - Scikit-learn

5. **Kaggle.** "Bank Marketing Dataset."
    - The dataset you're using is based on a popular Kaggle competition, providing useful discussions and insights from the data science community.
    - Kaggle Bank Marketing