

Kathmandu University School of Management

Balkumari, Lalitpur

Final Individual Project On
Loan Default Prediction

Submitted to

Mr. Sanjay Pudasaini

Assistant Professor (KUSOM)

Submitted by:

Ankit Bhandari Kshetri

21107

MBA Fall 2021

July 03, 2023

Acknowledgement

I would like to express my sincere gratitude and appreciation to all those who have contributed to the successful completion of this business analytics final project. This endeavor would not have been possible without the support, guidance, and encouragement of several individuals, whom I would like to acknowledge and thank.

First and foremost, I am deeply grateful to Mr. Sanjay Pudasaini, Assistant professor, KUSOM, whose expertise, knowledge, and mentorship were invaluable throughout this project. His insightful feedback, constructive criticism, and unwavering support played a crucial role in shaping the direction and quality of my work. I am truly indebted to him for his dedication and commitment to our learning.

I would also like to extend my heartfelt thanks to my classmates who actively participated in discussions, shared valuable insights, and provided assistance whenever needed. Their collaboration and camaraderie created a conducive learning environment, where we could collectively tackle challenges and explore different perspectives. Their contributions enriched my understanding of the subject matter and inspired me to push the boundaries of my own abilities.

I am grateful to everyone who played a part in this project, directly or indirectly. Your contributions have shaped my learning experience, and I am honored to have had the opportunity to work with such remarkable individuals. Thank you all for your support, guidance, and friendship.

Sincerely,

Ankit Bhandari Kshetri

Contents

Problem statement	1
Objectives	1
Hypotheses generation.....	2
Assumptions.....	2
Research Question	3
About the dataset	3
Importing the dependencies.....	4
Importing/reading the dataset	5
Chapter 1 Understanding the dataset	6
1.1 Gathering information about the dataset	6
1.2 Basic descriptive statistics.....	6
1.3 Assessing the shape of the dataset.....	7
1.4 Checking for missing values	7
Chapter 2	9
Exploratory data analysis (EDA)/Preprocessing	9
2.1 Filling missing values.....	9
2.2 Handling/Removing the outliers	9
2.2.1 Pre cleaning Univariate analysis (Before handling outliers).....	10
2.2.2 Checking the data distribution of the fields	11
2.2.3 Removing the outliers by PERCENTILE METHOD	13
2.2.4 Creating a clean new dataframe with no outliers	15
2.3 Post cleaning Univariate analysis (After handling outliers)	15
2.3.1 Checking the distribution of data	17
2.4 Bivariate and multivariate analysis	19
2.4.1 Other multivariate analysis	26
Chapter 3	30
Feature engineering	30
3.1 Feature encoding	30
3.1.1 Counting values in each feature	30
3.1.2 Encoding using label encoder	35
3.1.3 Counting categorical values after encoding	37
3.2 Feature selection.....	38
3.2.1 Correlation-based Feature Selection	38
3.2.2 Principle component analysis.....	39
3.2.3 Reasons for selecting Correlation based method over PCA.....	41
3.2.4 Segregating dependent and independent variables.....	41

3.3 Checking for multicollinearity between the features	42
Chapter 4	43
Preparation for Machine Learning modelling	43
4.1 Splitting the features and target into test and train data.....	43
Chapter 5 Predictive Analysis through ML models.....	44
5.1 “Model”: Logistic regression	44
5.1.1 Accuracy check on train and test data.....	45
5.1.2 Actual prediction.....	46
5.1.3 Evaluation and visualization of logistic prediction	47
5.2 “Model 1”: Naïve Bayes classification algorithm	48
5.2.1 Accuracy check on train and test data.....	49
5.2.2 Actual prediction.....	49
5.3 “Model2”: K Nearest neighbor.....	50
5.3.1 Accuracy check on train and test data.....	51
5.3.2 Actual prediction.....	52
5.4 “Model 3”: Decision tree.....	53
5.4.1 Accuracy check on train and test data.....	54
5.4.2 Actual prediction.....	55
5.4.3 Visualization of decision tree.....	55
5.5 “Model 4”: Support vector machine.....	57
5.5.1 Accuracy check on train and test data.....	58
5.5.2 Actual prediction.....	59
5.6 “Model 5”: Random forest classifier.....	59
5.6.1 Accuracy check on train and test data.....	60
5.6.2 Actual prediction.....	61
5.6.3 Visualization of Random forest	61
Chapter 6 Selection of best model for prediction	63
6.1 Stratified K folds cross validation	63
6.1.1 Importing required libraries	63
6.1.2 Cross validation score and standard deviation between the folds for each model	64
6.2 Visualization of std deviation of models across the folds	67
Chapter 7	70
Managerial implications and limitations of the study	70
7.1 Managerial implications.....	70
7.2 Limitations of the study.....	74

Problem statement

When loan recipient data is not properly analyzed in the banking sector, several problems can arise. Firstly, there is an increased risk of granting loans to individuals who have a higher likelihood of defaulting. Without proper analysis, crucial factors such as the borrower's credit history, financial stability, and repayment capacity may be overlooked, leading to poor loan decisions. **This can result in a higher number of defaults and financial losses** for the banking institution. Secondly, inaccurate or incomplete data analysis may lead to **biased lending practices**, potentially leading to discrimination or unfair treatment of certain individuals or groups. Inadequate analysis may overlook relevant socioeconomic factors, further exacerbating inequalities in access to credit.

Additionally, improper data analysis can **hinder effective risk management** and portfolio diversification, making it difficult for banks to assess and mitigate the overall risk associated with their loan portfolios. Overall, the lack of proper loan recipient data analysis in the banking sector can undermine financial stability, customer trust, and profitability.

The successful completion of this project will provide the banking institution with a reliable predictive model that can assist in identifying customers at a higher risk of loan default. By utilizing this model, the institution can make more informed decisions regarding loan approvals, risk assessment, and creditworthiness, ultimately improving their loan portfolio management and minimizing potential financial losses.

Objectives

The objective of this project is to develop and evaluate machine learning models to predict loan defaults in a banking dataset. The dataset contains various features related to the borrowers' financial information, credit history, and loan details. By analyzing this dataset and building accurate predictive models, the aim is to assist the banking institution in finding new patterns, **assessing the risk, opportunities and future strategies** associated with granting loans identifying customers who are more likely to default on their loan payments and new product development for the bank.

We will be using **six** classification models. Namely,

- “Model”: Logistic regression
- “Model 1”: Naïve bayes classifier
- “Model 2”: Kneighbors classifier
- “Model 3”: Decision tree
- “Model 4”: Support vector matrix
- “Model 5”: Random forest

After the models have been build and tested, we will CROSS VALIDATE the models using the K folds cross validation. The model with the lowest standard deviation across the cross validation folds will be selected and its prediction will be regarded a final one.

Hypotheses generation

In any project for data science or machine learning, hypotheses generation is crucial. It entails thoroughly comprehending the issue and outlining all the potential influences on the result in a brainstorming session. It is accomplished by fully comprehending the problem statement before looking at the facts.

Here are a few elements that can influence the target variable of our interest i.e. the loan default

Assumptions

- **Job:** Customers with blue collar job and entrepreneurial profession are more likely to default in comparison to other jobs because of income uncertainty
- **Marital:** Customers with married marital status are more likely to default than those who are single because they have more expenses.
- **Age:** Customers with young age are more likely to default of a higher risk taking appetite.
- **Balance:** Customers with lower bank balance are more likely to default because of absence of reserve when financial instability strikes.
- **Housing:** Customers who have taken a housing loan are more likely to default because housing market is the primary sector that shows the effect of economic downturn (For example, 2008/09 financial crisis started because of the housing bubble burst in USA because of SUBPRIME MORTGAGE).

On the basis of above features, taking the target variable loan default, the following hypothesis can be made:

- **Null hypothesis (H0):** The customer is predicted to be unlikely to default
- **Alternative hypothesis (H1):** The customer is predicted to be likely to default

Research Questions

- What is the median balance and age of the customers in the defaulter's category?
- What are the customer segments with highest number of customers?
- Based on certain circumstances, will the customer default or not?

About the dataset

The analyzed dataset is a BANK LOAN DEFAULT DATASET of a Portuguese banking institution sourced from 'kaggle'. We are trying to predict the loan default likelihood from the given inputs.

The inputs/ features/ independent variables in the dataset are as follows:

- ID: Client ID of the bank's customers(Numeric)
- marital: marital status (categorical: 'divorced', 'married', 'single'; note: 'divorced' means divorced or widowed)
- age: Age of the customers(Numeric)
- education: Educational qualification of the customers (Categorical: 'unknown', 'primary', 'secondary', 'tertiary')
- housing: has housing loan? (categorical: 'no', 'yes')
- loan: has personal loan? (categorical: 'no', 'yes')
- contact: contact communication type (categorical: 'cellular', 'telephone', 'unknown')
- month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- day: last contact day of the week (categorical: 'mon', 'tue', 'wed', 'thu', 'fri')
- duration: last contact duration, in seconds (numeric). # # other attributes:
- campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

- pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
- previous: number of contacts performed before this campaign and for this client (numeric)
- poutcome: outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')
- y: Did the customer subscribe to the plan? (Categorical: 'yes', 'no')

Similarly, target variable/ variable of interest or dependent variables in the dataset are as follows:

- default: has credit in default? (categorical: 'no', 'yes')

Importing the dependencies

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

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

The basic libraries for statistical analysis and data frame operation are imported for example Numpy and pandas, Matplotlib and Seaborn are the libraries used for visualization. Warnings have been ignored to make the codes appear cleaner. We are primarily using scikit learn ML library for the analysis. The basics such as test train split, accuracy score and label encoder have been imported to split data into test and train data, check the accuracy score of the data and encode the categorical variables to numerical variables.

Similarly, the libraries related to Logistic regression, Kneighbors classifier, Naïve bayes classifier, Decision tree, support vector matrix and random forest are imported. Also, other libraries such as K fold cross validation are imported whenever necessary.

Importing/reading the dataset

```
In [2]: df = pd.read_csv("outlier.csv")

In [3]: df.head(20)
```

Out[3]:

	id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	1001	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown
1	1002	44.0	technician	single	secondary	no	29.0	yes	no	unknown	5	may	151	1	-1	0	unknown
2	1003	33.0	entrepreneur	married	secondary	no	2.0	yes	yes	unknown	5	may	76	1	-1	0	unknown
3	1004	47.0	blue-collar	married	unknown	no	1506.0	yes	no	unknown	5	may	92	1	-1	0	unknown
4	1005	33.0	unknown	single	unknown	no	1.0	no	no	unknown	5	may	198	1	-1	0	unknown
5	1006	35.0	management	married	tertiary	no	231.0	yes	no	unknown	5	may	139	1	-1	0	unknown
6	1007	28.0	management	single	tertiary	no	447.0	yes	yes	unknown	5	may	217	1	-1	0	unknown
7	1008	NaN	entrepreneur	divorced	tertiary	yes	NaN	yes	no	unknown	5	may	360	1	-1	0	unknown
8	1009	58.0	retired	married	primary	no	NaN	yes	no	unknown	5	may	50	1	-1	0	unknown
9	1010	43.0	technician	single	secondary	no	NaN	yes	no	unknown	5	may	55	1	-1	0	unknown
10	1011	41.0	admin.	divorced	secondary	no	270.0	yes	no	unknown	5	may	222	1	-1	0	unknown

The csv file named outlier has been imported by the name “df”. Also, the first 20 rows are displayed using the head function.

Chapter 1

Understanding the dataset

1.1 Gathering information about the dataset

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

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 18 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Id           45211 non-null  int64
1   age          45202 non-null  float64
2   job          45211 non-null  object
3   marital      45211 non-null  object
4   education    45211 non-null  object
5   default      45211 non-null  object
6   balance      45208 non-null  float64
7   housing      45211 non-null  object
8   loan         45211 non-null  object
9   contact      45211 non-null  object
10  day          45211 non-null  int64
11  month        45211 non-null  object
12  duration     45211 non-null  int64
13  campaign     45211 non-null  int64
14  pdays        45211 non-null  int64
15  previous     45211 non-null  int64
16  poutcome     45211 non-null  object
17  y            45211 non-null  object
dtypes: float64(2), int64(6), object(10)
memory usage: 6.2+ MB
```

We have displayed the basic info such as columns, Non null count, index number and data types. It seems like all the variables of use are on the usable data type. The datatypes are on float, integer and object type. The size of the dataset is 6.2 mb.

1.2 Basic descriptive statistics

```
In [5]: df.describe()

Out[5]:
```

	Id	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45202.000000	45208.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	23606.000000	40.954714	1362.346620	15.806419	258.163080	2.763841	40.197828	0.580323
std	13051.435847	11.539144	3044.852387	8.322476	257.527812	3.098021	100.128746	2.303441
min	1001.000000	-1.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	12303.500000	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	23606.000000	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	34908.500000	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	46211.000000	999.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

Through the describe function we can depict the basic summary descriptive statistics of the dataset. For example, the average balance of the customers is 102127 and the maximum age of the customer is 999 which might be a mistake that we will fix while removing the outliers.

1.3 Assessing the shape of the dataset

```
In [6]: df.shape
```

```
Out[6]: (45211, 18)
```

```
In [7]: df.isna().sum()
```

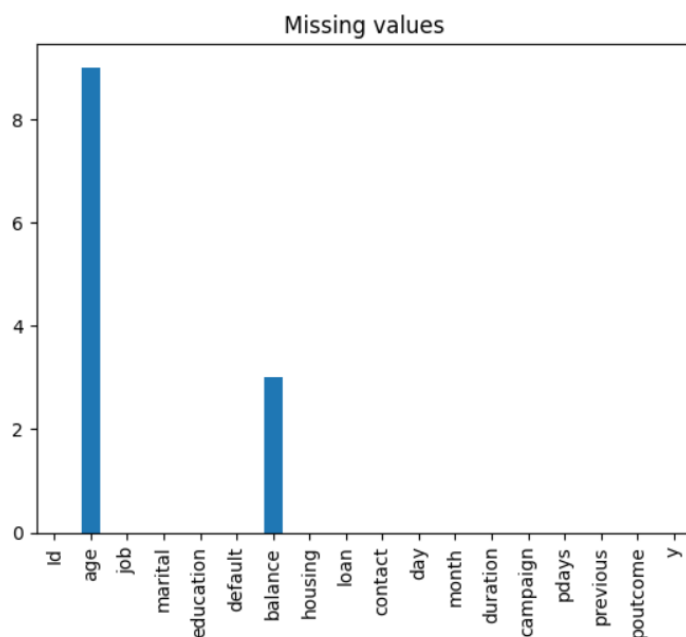
```
Out[7]: Id          0
age            9
job            0
marital        0
education      0
default        0
balance        3
housing        0
loan           0
contact        0
day            0
month          0
duration       0
campaign       0
pdays        0
previous      0
poutcome      0
y             0
dtype: int64
```

Here, we have observed the shape of the dataset. The data frame “df” has 45211 rows and 18 columns.

1.4 Checking for missing values

```
In [8]: df.isna().sum().plot(kind='bar', title='Missing values')
```

```
Out[8]: <Axes: title={'center': 'Missing values'}>
```



Also, we are checking for null values in our dataset. The column “age” has 9 missing values and the column “balance” has 3 missing values. The length of the missing values is way less than 5% of the length of the dataset which is the threshold to drop the missing values. **HOWEVER**, we are not dropping the dataset but rather filling it in order to obtain a **FULL DATA RETENTION**.

Chapter 2

Exploratory data analysis (EDA)/Preprocessing

2.1 Filling missing values

```
In [10]: age_mean= df['age'].mean()
         print(age_mean)
40.954714393168445

In [11]: df['age'].fillna(age_mean,inplace=True)

In [12]: balance_mean= df['balance'].mean()
         print(balance_mean)
1362.3466200672447

In [13]: df['balance'].fillna(balance_mean,inplace=True)
```

Since we have missing values in two columns of our dataset, we are filling the missing values with the mean of the column. The in place equals to true argument will replace the missing values on the original dataframe itself.

```
In [16]: df.isna().sum()
Out[16]: Id          0
         age         0
         job         0
         marital     0
         education   0
         default     0
         balance     0
         housing     0
         loan        0
         contact     0
         day         0
         month       0
         duration    0
         campaign    0
         pdays       0
         previous    0
         poutcome    0
         y           0
         dtype: int64
```

As we can see here, there are no missing values in our dataset. Now, we can move further with handling the outliers in our dataset.

2.2 Handling/Removing the outliers

Outliers in a dataset are data points that deviate significantly from the majority of the observations. They can be unusually high or low values that lie far away from the typical range of values in the dataset. Outliers can arise due to various reasons such as measurement errors, data entry mistakes,

or genuinely exceptional cases. Therefore, it is very important to remove them to obtain a standardized data distribution.

Before removing outliers, let us check for the outliers in our dataset using the boxplot. We will be checking and removing the outliers from the columns **Age, balance and duration**. The reasons for removing outliers from these columns only are as follows:

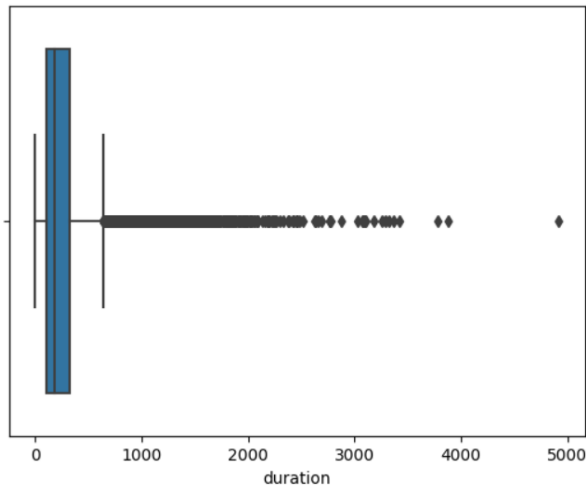
- These fields are expected to be directly influencing the target variable i.e loan default.
- Outliers can be removed only from the numeric data type variables. Other variables are categorical.
- Removal of outliers from a lot of columns will cost us a lot of data records which might hamper out accuracy score later.

2.2.1 Pre cleaning Univariate analysis (Before handling outliers)



```
sns.boxplot(df['duration'])
```

```
<Axes: xlabel='duration'>
```

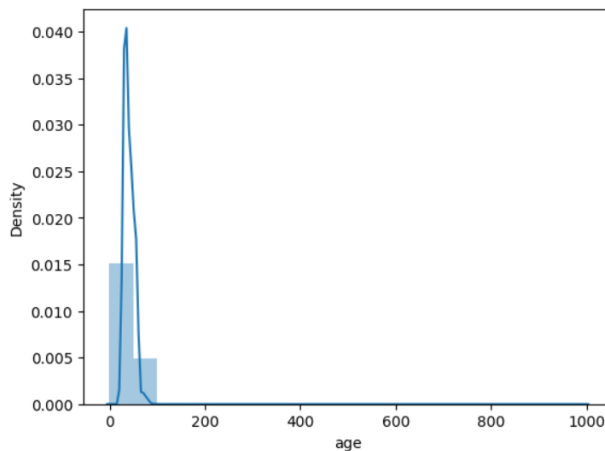


The black dots on the either side of the whiskers represent the anomaly values or outliers. Here, we can see that there are a lot of outliers on the columns balance and duration. The age column however has very few outliers on either side of the whiskers, which is probably a mistake. Overall, we can say that the dataset is **RIGHT TAIL HEAVY**. We have to remove the outliers in order to obtain a standardized dataset and to avoid the problem of **OVERFITTING**.

2.2.2 Checking the data distribution of the fields

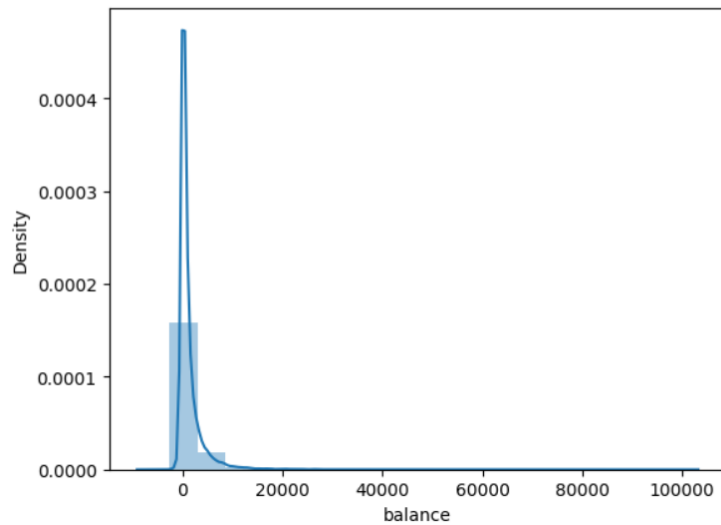
```
sns.distplot(df['age'],bins=20)
```

```
<Axes: xlabel='age', ylabel='Density'>
```



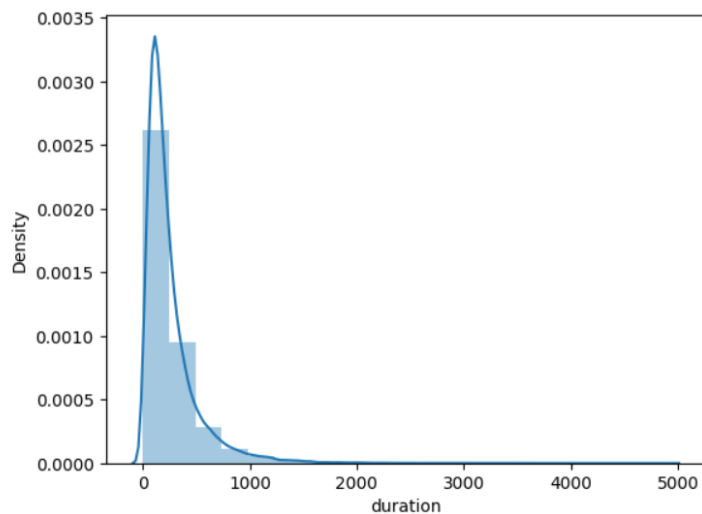
```
sns.distplot(df['balance'],bins=20)
```

```
<Axes: xlabel='balance', ylabel='Density'>
```



```
sns.distplot(df['duration'],bins=20)
```

```
<Axes: xlabel='duration', ylabel='Density'>
```



We have used the distribution plot for visualizing the data distribution of the selected fields. As we can see here, the data distribution far beyond the normal distribution. The distribution is highly skewed. The values seem to be collected in one single place. Therefore, it seems very necessary for us to remove the outliers to obtain a normal distributed dataset.

2.2.3 Removing the outliers by PERCENTILE METHOD

The percentile method of outlier removal is a technique used to identify and eliminate outliers from a dataset based on their position relative to the rest of the values. It involves defining a threshold based on percentiles and removing any data points that fall outside that threshold.

Defining the threshold

Threshold for age column

```
max_threshold_age = df['age'].quantile(0.95)
```

```
min_threshold_age = df['age'].quantile(0.05)
```

```
print(max_threshold_age,min_threshold_age)
```

```
59.0 27.0
```

Here, we have defined the maximum threshold for the age column at the 95th percentile i.e 59 and the minimum threshold at the 5th percentile i.e 27. The next step is to view the data points that are to be removed.

```
df[df['age']>max_threshold_age]
```

	id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	1001	999.0	management	married	tertiary	no	2143.0	yes	no	unknown	5	may	261	1	-1	0	unknown
18	1019	60.0	retired	married	primary	no	60.0	yes	no	unknown	5	may	219	1	-1	0	unknown
32	1033	60.0	admin.	married	secondary	no	39.0	yes	yes	unknown	5	may	208	1	-1	0	unknown
42	1043	60.0	blue-collar	married	unknown	no	104.0	yes	no	unknown	5	may	22	1	-1	0	unknown
66	1067	60.0	retired	married	tertiary	no	100.0	no	no	unknown	5	may	528	1	-1	0	unknown
...
45185	46186	60.0	services	married	tertiary	no	4256.0	yes	no	cellular	16	nov	200	1	92	4	success
45191	46192	75.0	retired	divorced	tertiary	no	3810.0	yes	no	cellular	16	nov	262	1	183	1	failure
45195	46196	68.0	retired	married	secondary	no	1146.0	no	no	cellular	16	nov	212	1	187	6	success
45207	46208	71.0	retired	divorced	primary	no	1729.0	no	no	cellular	17	nov	456	2	-1	0	unknown
45208	46209	72.0	retired	married	secondary	no	5715.0	no	no	cellular	17	nov	1127	5	184	3	success

1784 rows × 18 columns

```
df[df['age']<min_threshold_age]
```

36	1037	25.0	blue-collar	married	secondary	no	-7.0	yes	no	unknown	5	may	365	1	-1	0	unknown
135	1136	23.0	blue-collar	married	secondary	no	94.0	yes	no	unknown	5	may	193	1	-1	0	unknown
151	1152	26.0	student	single	secondary	no	0.0	yes	no	unknown	5	may	610	2	-1	0	unknown
165	1166	26.0	admin.	single	secondary	no	82.0	yes	no	unknown	5	may	228	1	-1	0	unknown
...
45189	46190	25.0	services	single	secondary	no	199.0	no	no	cellular	16	nov	173	1	92	5	failure
45196	46197	25.0	student	single	secondary	no	358.0	no	no	cellular	16	nov	330	1	-1	0	unknown

Here are the values for the age column that are above the 95th percentile and below the 5th percentile respectively. All of these values will be removed to obtain a standardized age column.

Thresholds For balance

```
max_threshold_bal= df['balance'].quantile(0.85)
```

```
min_threshold_bal= df['balance'].quantile(0.25)
```

```
print(min_threshold_bal,max_threshold_bal)
```

```
72.0 2539.0
```

Here, we have defined the maximum threshold for the balance column at the 85th percentile i.e 72 and the minimum threshold at the 25th percentile i.e 2539. The values for the balance column that are above the 85th percentile and below the 25th percentile respectively will be removed to obtain a standardized balance column.

Threshold for duration

```
: max_threshold_dur= df['duration'].quantile(0.90)  
  min_threshold_dur= df['duration'].quantile(0.15)
```

```
: print(max_threshold_dur,min_threshold_dur)
```

```
548.0 75.0
```

Here, we have defined the maximum threshold for the duration column at the 90th percentile i.e 548 and the minimum threshold at the 15th percentile i.e 75. The values for the balance column that are above the 90th percentile and below the 15th percentile respectively will be removed to obtain a standardized duration column.

2.2.4 Creating a clean new dataframe with no outliers

Creating a clean new dataframe with no outliers

```
: df1 = df[(df['age'] > min_threshold_age) & (df['age'] < max_threshold_age) &
          (df['balance'] > min_threshold_bal) & (df['balance'] < max_threshold_bal)&
          (df['duration']>min_threshold_dur) & (df['duration']< max_threshold_dur)]

: df1.head()
```

	Id	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcor
3	1004	47.000000	blue-collar	married	unknown	no	1506.00000	yes	no	unknown	5	may	92	1	-1	0	unkno
5	1006	35.000000	management	married	tertiary	no	231.00000	yes	no	unknown	5	may	139	1	-1	0	unkno
6	1007	28.000000	management	single	tertiary	no	447.00000	yes	yes	unknown	5	may	217	1	-1	0	unkno
7	1008	40.954714	entrepreneur	divorced	tertiary	yes	1362.34662	yes	no	unknown	5	may	380	1	-1	0	unkno
10	1011	41.000000	admin.	divorced	secondary	no	270.00000	yes	no	unknown	5	may	222	1	-1	0	unkno

After we made the thresholds for the columns, we created a new subset by the name of “df1” which would only consist of those records that lie under the created thresholds for all the three columns. We used the head function again to view the first few rows of the cleaned dataset.

```
: df1.shape
: (17698, 18)

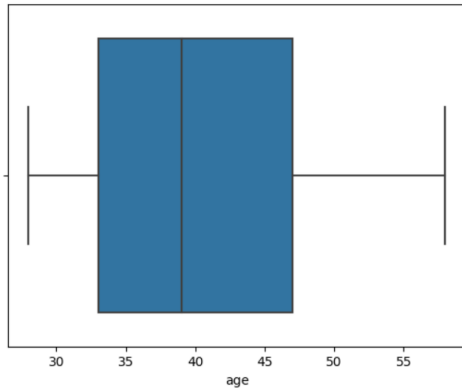
: df.shape
: (45211, 18)
```

If we compare the shape of the uncleaned dataframe to the dataframe that has been cleaned and contains no outliers, the number of rows in “df1” have decreased by more than half of the original dataframe “df”.

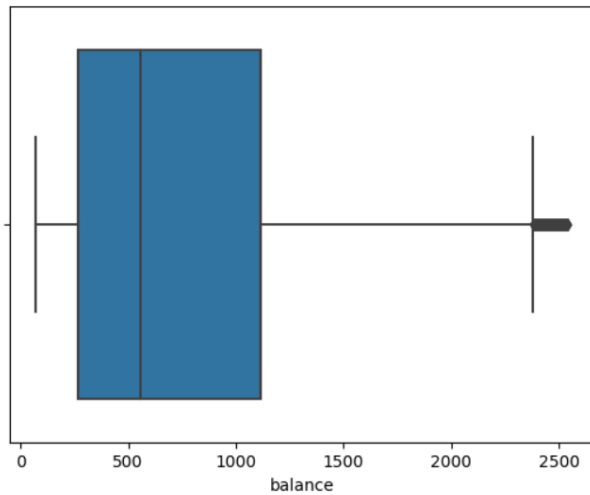
2.3 Post cleaning Univariate analysis (After handling outliers)

Let’s begin the univariate analysis with boxplot visualizations which will also help us to visualize the difference in distribution after we have removed the outliers.

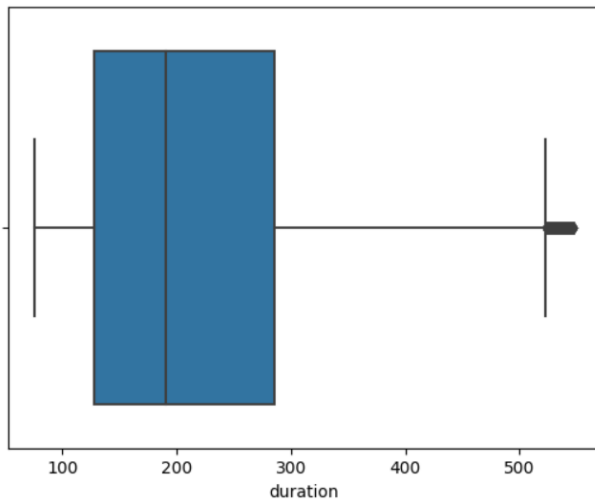
```
sns.boxplot(data=df1, x= 'age')  
<Axes: xlabel='age'>
```



```
sns.boxplot(data=df1, x= 'balance')  
<Axes: xlabel='balance'>
```



```
sns.boxplot(df1['duration'])  
<Axes: xlabel='duration'>
```

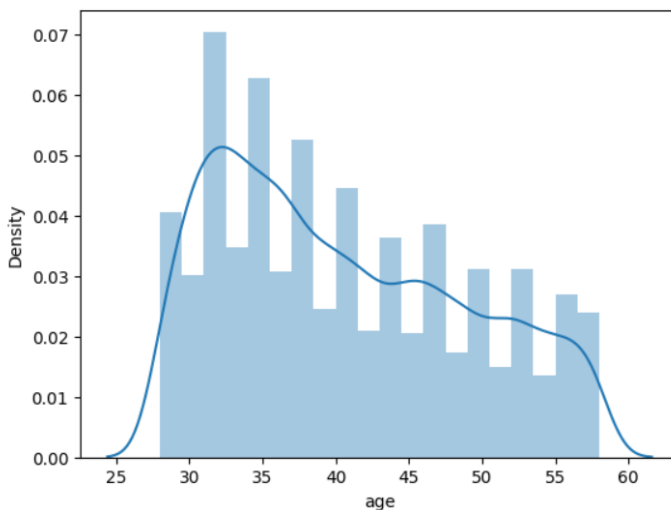


Here, we can clearly observe the reduction in outliers. There are very few outliers left in all the three columns. If we observe the boxplots statistically, the median age of the customers seems to be around 39. The median balance of the customers seems to be about 600. Similarly, the median duration seems to be around 195 days.

2.3.1 Checking the distribution of data

```
sns.distplot(df1['age'], bins=20)
```

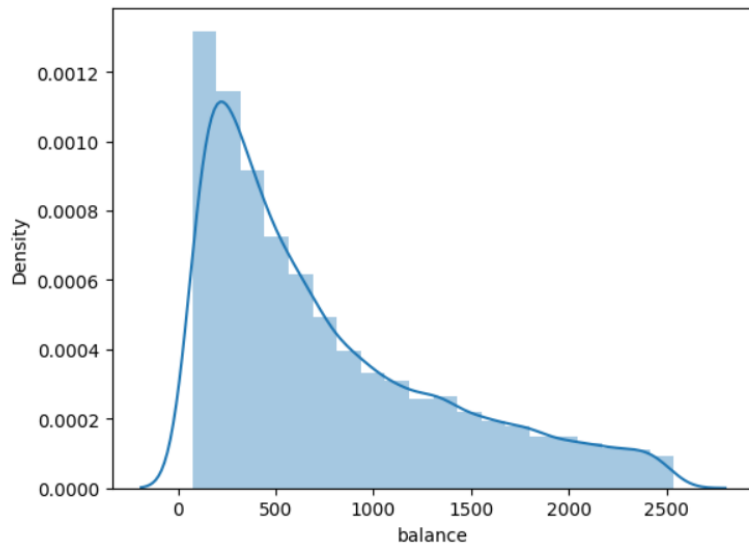
<Axes: xlabel='age', ylabel='Density'>



The distribution plot after the removal of outliers is depicting a normal distribution lookalike distribution for the age variable. If we observe the descriptive statistics here, majority of customers seem to be at the age of around 32. Least number of customers are in the age below 27 and above 58 respectively.

```
sns.distplot(df1['balance'],bins=20)
```

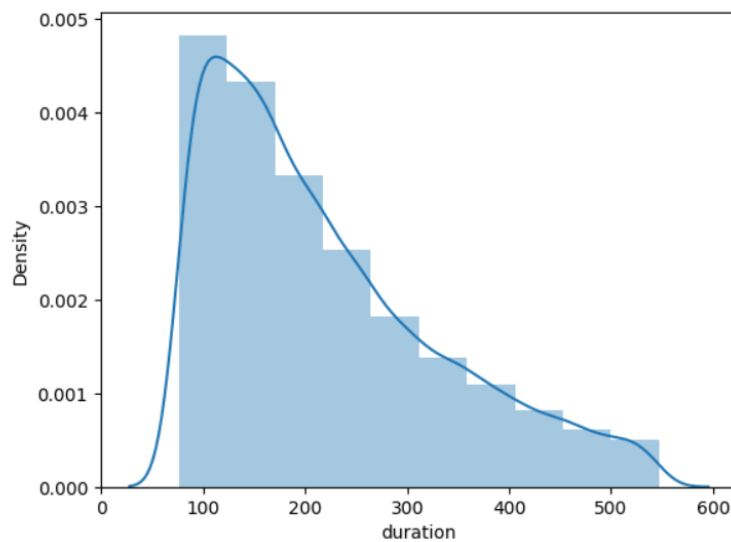
```
<Axes: xlabel='balance', ylabel='Density'>
```



The distribution plot after the removal of outliers is depicting a normal distribution lookalike distribution for the balance variable. Although not as normally distributed as that of age variable, this seems to be way standardized than Pre outlier removal's distribution. If we observe the descriptive statistics here, majority of customers seem to have a balance of around 4800. Least number of customers have the balance close to zero and above 2400 respectively.

```
sns.distplot(df1['duration'], bins=10)
```

```
<Axes: xlabel='duration', ylabel='Density'>
```



The distribution plot after the removal of outliers is depicting a normal distribution lookalike distribution for the duration variable. Although not as normally distributed as that of age variable, this seems to be way standardized than Pre outlier removal's distribution. If we observe the descriptive statistics here, majority of customers seem to have a duration of around 110. Least number of customers have the balance close to zero and above 500 respectively.

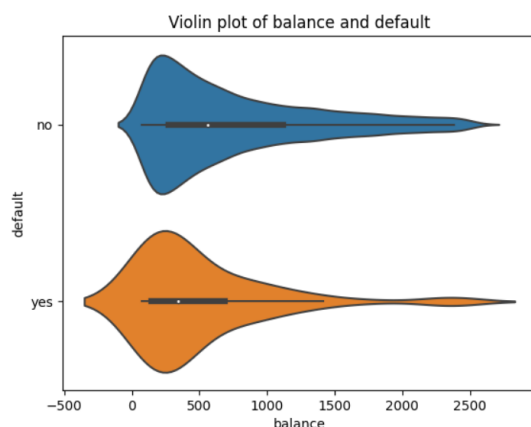
2.4 Bivariate and multivariate analysis

Here, we will be analyzing more than two variables unlike the univariate analysis. Also, we will be relating these analyses with the **ASSUMPTIONS** that we have made while formulating the hypotheses.

- **Balance**

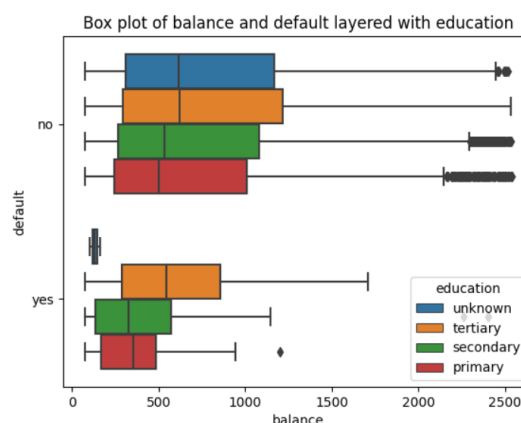
```
sns.violinplot(data=df1, x='balance', y='default')
plt.title('Violin plot of balance and default')
```

Text(0.5, 1.0, 'Violin plot of balance and default')



```
sns.boxplot(data=df1, x='balance', y='default', hue='education')
plt.title('Box plot of balance and default layered with education')
```

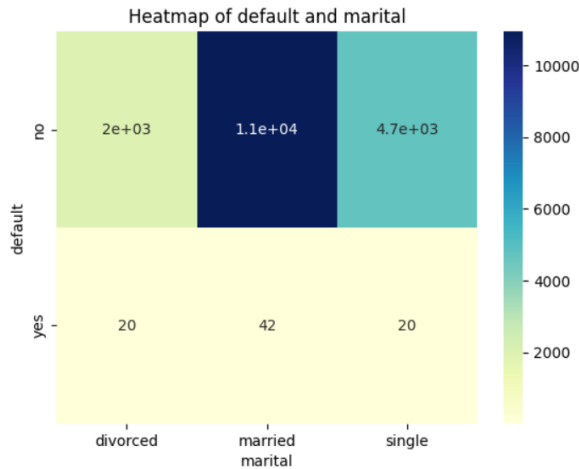
Text(0.5, 1.0, 'Box plot of balance and default layered with education')



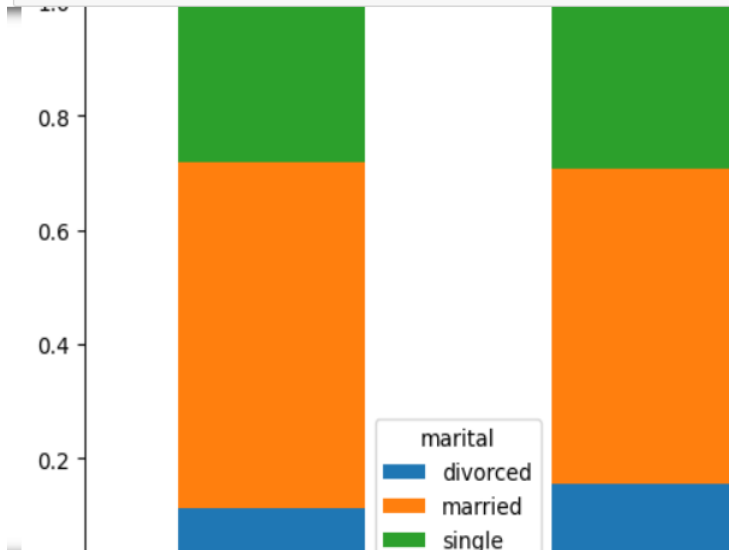
Earlier, we had made an assumption that Customers with lower bank balance are more likely to default because of absence of reserve when financial instability strikes. Well, it seems to be true. In the adjoining violin plot, we can observe that the median balance in case of NON DEFAULTERS is around 500 and that of DEFAULTERS is way below 500. If we combine the findings with a layer of education, we can observe that Among NON DEFAULTERS, primary level education holders have the least amount of balance whereas among DEFAULTERS, secondary level education holders have the least amount of bank balance.

- Marital

```
cross_tab = pd.crosstab(df1['default'], df['marital'])
sns.heatmap(cross_tab, annot=True, cmap='YlGnBu')
plt.title('Heatmap of default and marital')
Text(0.5, 1.0, 'Heatmap of default and marital')
```



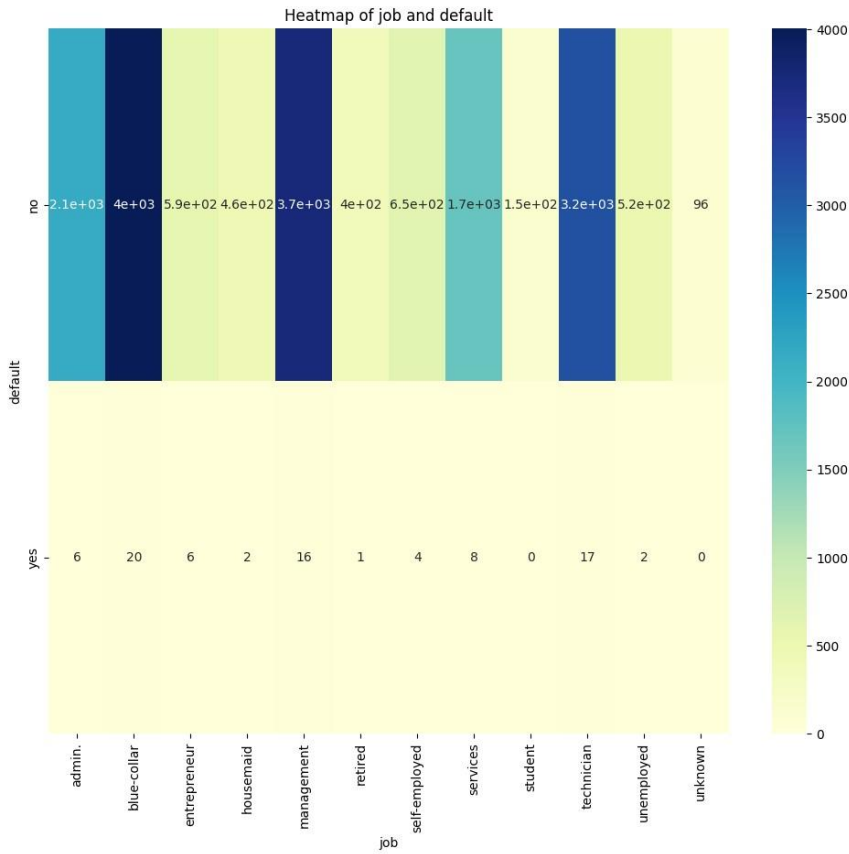
```
cross_tab_stack = pd.crosstab(df['default'], df['marital'])
stacked_data = cross_tab_stack.apply(lambda x: x / x.sum(), axis=1)
plt.figure(figsize=(12,10))
stacked_data.plot(kind='bar', stacked=True)
```



Earlier, we had assumed that Customers with married marital status are more likely to default than those who are single because they have more expenses. Well, that assumption seems to be true because if we look at the number of defaulters through the adjoining heatmap and stacked bar chart, married customers are the highest number of loan defaulters.

- **Job**

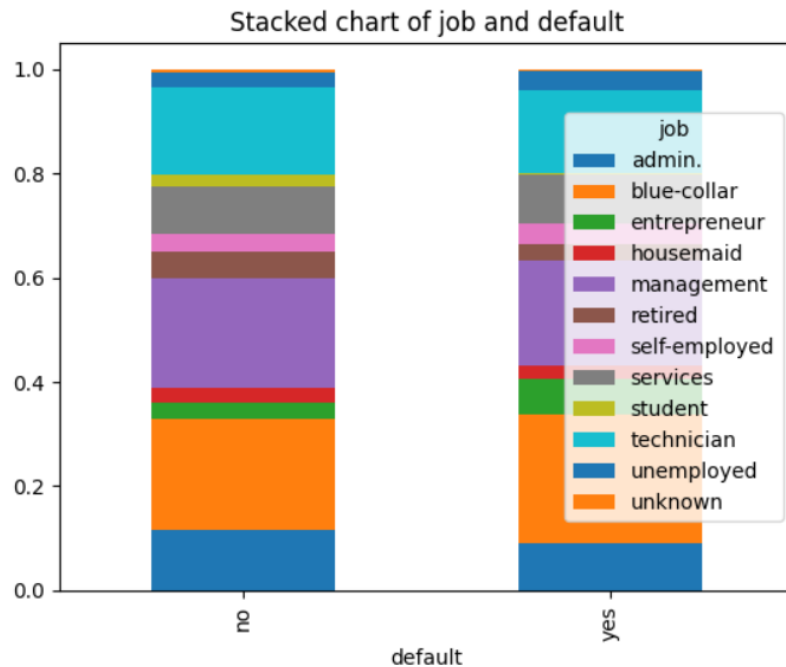
```
cross_tab_job = pd.crosstab(df1['default'], df1['job'])
plt.figure(figsize=(12,10))
sns.heatmap(cross_tab_job, annot=True, cmap='YlGnBu')
plt.title('Heatmap of job and default')
```



```
cross_tab_stacked_job = pd.crosstab(df['default'], df['job'])
stacked_data = cross_tab_stacked_job.apply(lambda x: x / x.sum(), axis=
plt.figure(figsize=(15,12))
stacked_data.plot(kind='bar', stacked=True)
plt.title('Stacked chart of job and default')
```

```
Text(0.5, 1.0, 'Stacked chart of job and default')
```

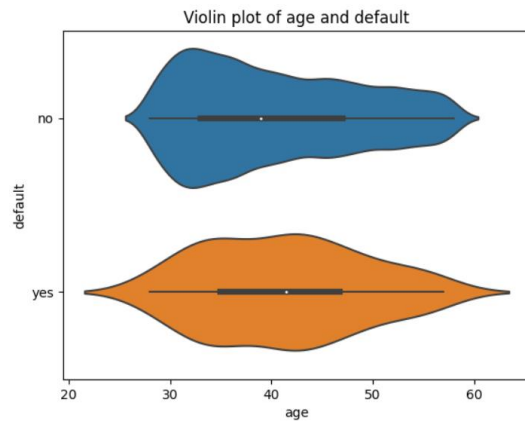
```
<Figure size 1500x1200 with 0 Axes>
```



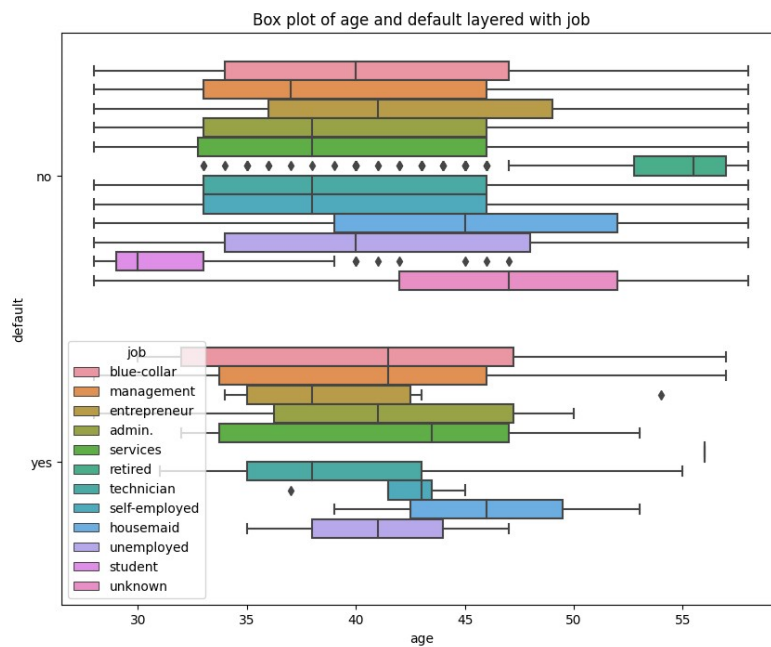
Earlier, we had assumed that customers with blue collar job and entrepreneurial profession are more likely to default in comparison to other jobs because of income uncertainty. However, one of the assumptions came true. Blue collar workers are the highest number of defaulters in the default section as seen on both of our heatmap and stacked bar chart.

- Age

```
: sns.violinplot(data=df1, x= 'age', y='default')
plt.title('Violin plot of age and default')
: Text(0.5, 1.0, 'Violin plot of age and default')
```

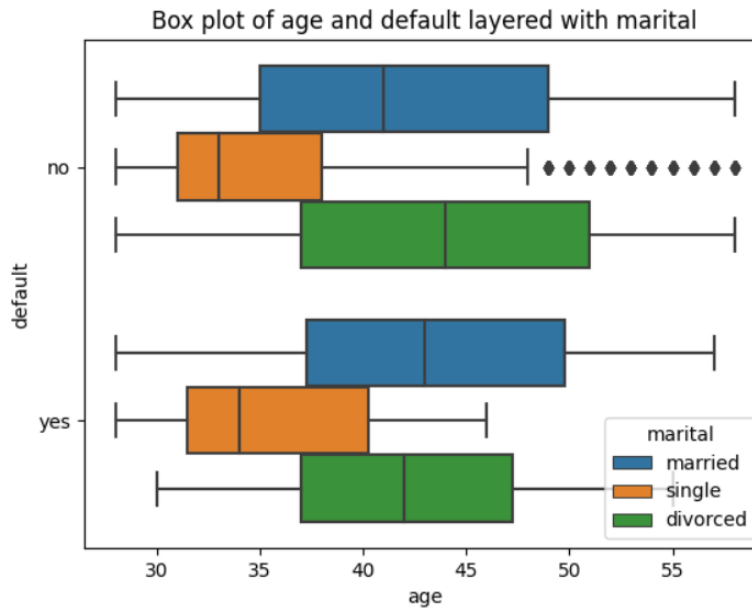


```
plt.figure(figsize=(10,8))
sns.boxplot(data=df1, x= 'age', y='default',hue= 'job')
plt.title('Box plot of age and default layered with job')
```



```
sns.boxplot(data=df1, x= 'age', y='default',hue= 'marital')
plt.title('Box plot of age and default layered with marital')
```

```
Text(0.5, 1.0, 'Box plot of age and default layered with marital')
```

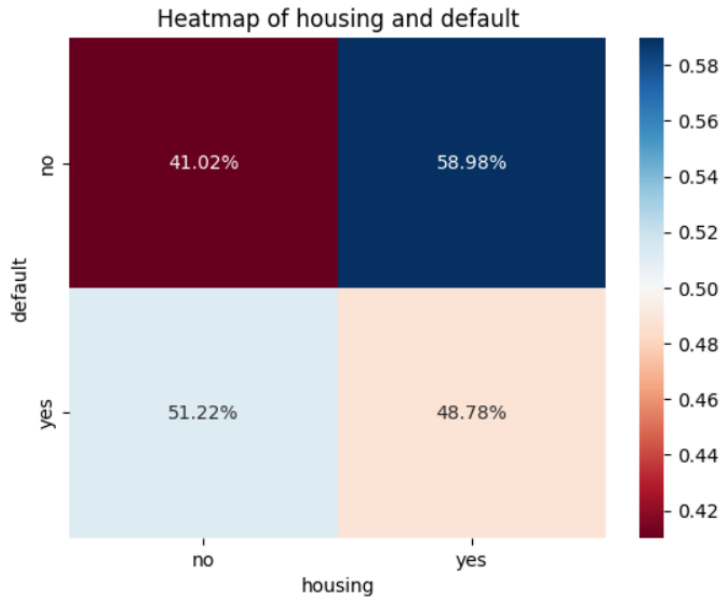


Earlier, we had assumed that Customers with younger age are more likely to default of a higher risk taking appetite. However, through the actual multivariate analysis it is discovered that the median age of defaulters is higher than that of non-defaulters. Therefore, it is the elder customers who are defaulting more than that of younger customers. Again, if we layer this prediction to the job of the customers, technician is the job category among defaulters who are the youngest and housemaid is the jib category among defaulters who are the oldest. Again, if we layer the findings with marital status of the customers, it is very obvious that single people are the youngest among the defaulters.

- **Housing**

```
cross_tab_housing = pd.crosstab(df['default'], df['housing'], normalize='index')
sns.heatmap(cross_tab_housing, annot=True, cmap='RdBu', fmt='.2%')
plt.title('Heatmap of housing and default')
```

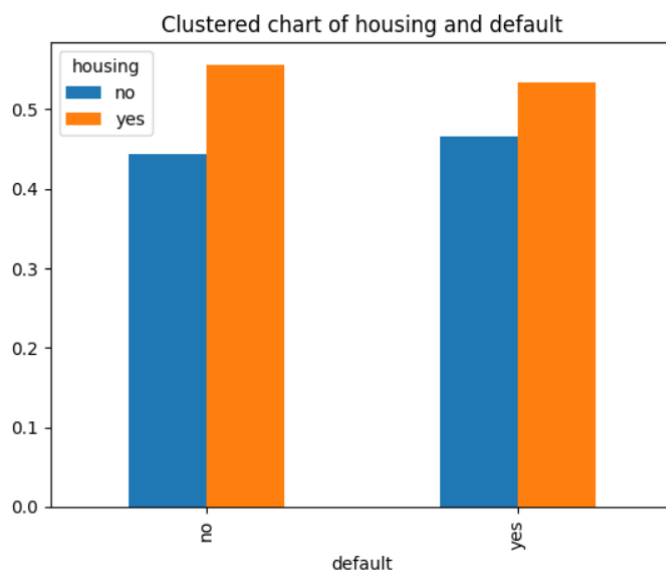
Text(0.5, 1.0, 'Heatmap of housing and default')



```
cross_tab_stacked_housing = pd.crosstab(df['default'], df['housing'])
stacked_data = cross_tab_stacked_housing.apply(lambda x: x / x.sum(), axis=1)
plt.figure(figsize=(15,12))
stacked_data.plot(kind='bar', stacked=False)
plt.title('Clustered chart of housing and default')
```

Text(0.5, 1.0, 'Clustered chart of housing and default')

<Figure size 1500x1200 with 0 Axes>



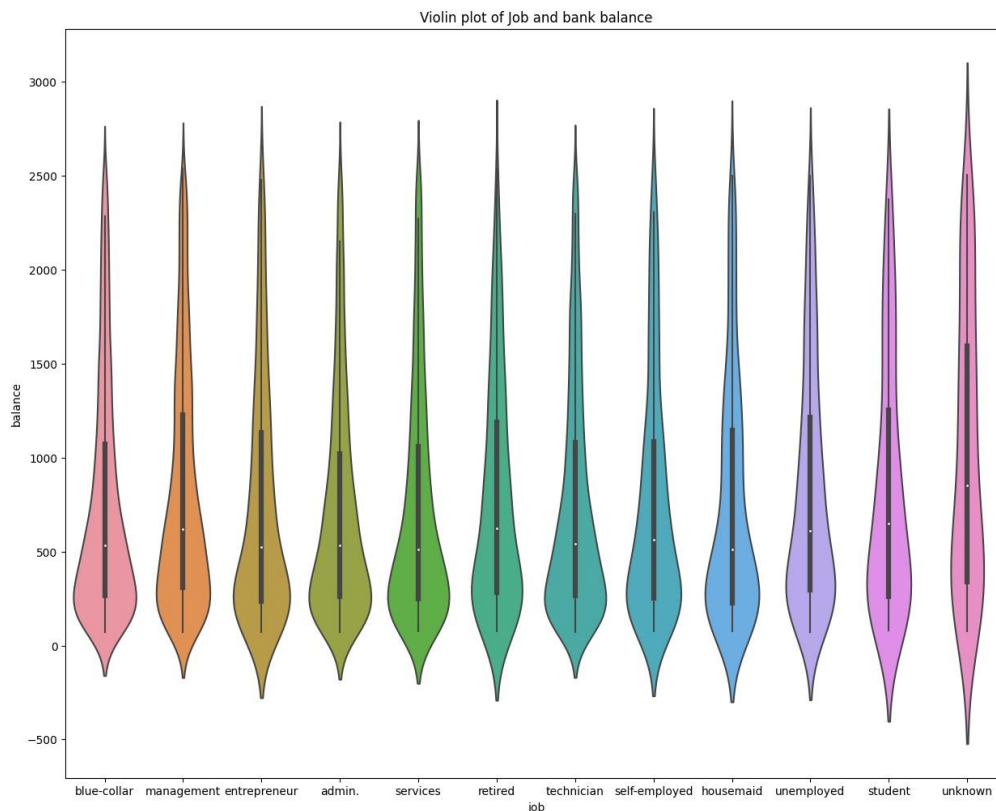
Earlier, we had assumed that Customers who have taken a housing loan are more likely to default because housing market is the primary sector that shows the effect of economic downturn(For

example, 2008/09 financial crisis started because of the housing bubble burst in USA because of (SUBPRIME MORTGAGE). Well, it seems that assumption has been held true. In the adjoining Heatmap and Column chart for default and housing loan type, we can clearly observe that housing loan takers cover a larger portion i.e. 51% among the DEFAULTERS. We can also infer that a large number of customers take housing loan than another types of loans.

2.4.1 Other multivariate analysis

- **Visualization of bank balance by job categories**

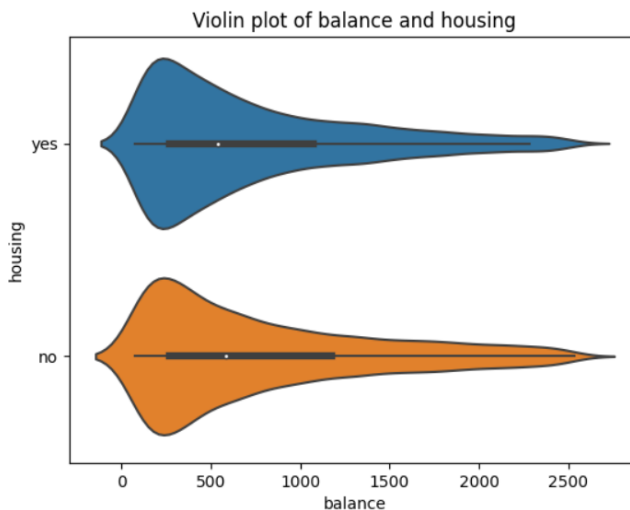
```
plt.figure(figsize=(15,12))
sns.violinplot(data=df1, x= 'job', y='balance')
plt.title('Violin plot of Job and bank balance')
```



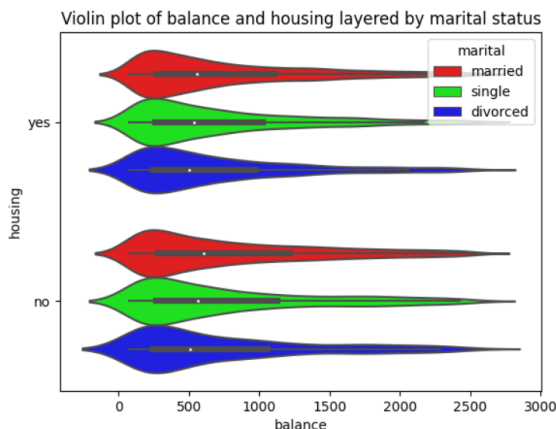
Diving further deep into the EDA, we have visualized the bank balance of the customers according to their profession type. Unknown category actually has the highest median bank balance. Following that, retired job category is the second highest balance holding category which makes sense because they save up their whole life.

- **Visualization of relationship between bank balance and housing loan type**

```
sns.violinplot(data=df1, x= 'balance', y='housing')
plt.title('Violin plot of balance and housing')
Text(0.5, 1.0, 'Violin plot of balance and housing')
```



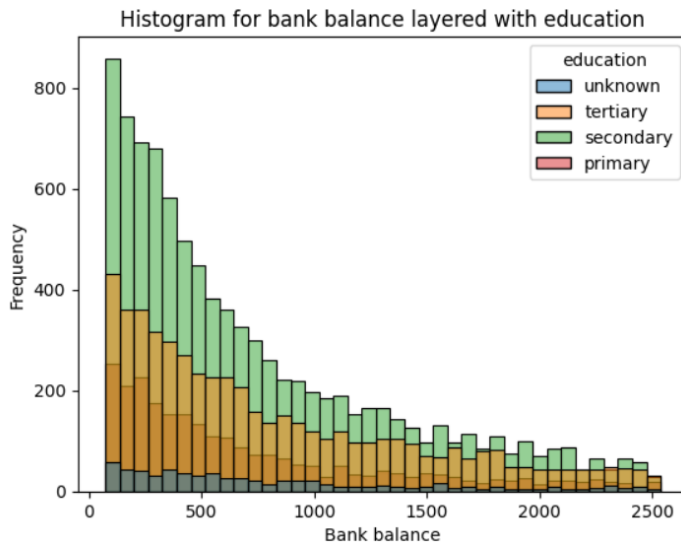
```
colors = ["#FF0000", "#00FF00", "#0000FF"]
sns.violinplot(data=df1, x= 'balance', y='housing', hue='marital', palette=colors)
plt.title('Violin plot of balance and housing layered by marital status')
Text(0.5, 1.0, 'Violin plot of balance and housing layered by marital status')
```



Here, we have visualized the relationship between bank balance and housing loan type using the violin plot. In the first graph we can see that the median balance of those customers who did not take the housing loan is higher than that of the customers who took housing loans. This, to an extent supports our earlier observation of more defaulters taking the housing loans. In the second chart where the relationship between housing and balance is layered by marital status, we can clearly see that among the housing loan takers, married customers have the highest bank balance. The same trend can be seen on the non-housing loan takers as well.

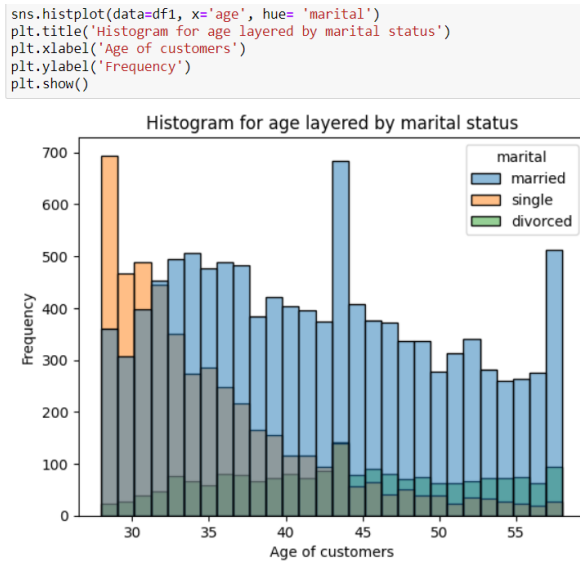
- **Visualization of bank balance according to education**

```
sns.histplot(data=df1, x='balance', hue= 'education')  
plt.title('Histogram for bank balance layered with education')  
plt.xlabel('Bank balance')  
plt.ylabel('Frequency')  
plt.show()
```



Here, we have tried to visualize the bank balance of the customers according to the education level. We can observe that secondary level education holders have the most amount of bank balance among all the customers. The maximum number of customers have the bank balance between 0 and 500 and among them secondary level followed by tertiary level are on the top of the rank. As the amount of bank balance is rising, the number of account holders is decreasing. Therefore, a NEGATIVE relationship can be observed between bank balance and increment in number of customers.

- **Visualization of age of customers according to marital status**



Here, we have tried to visualize the age of customers according to the marital status to get an idea about the DEMOGRAPHICS OF THE CUSTOMER BASE. We can observe that Customers aged 28 and 43 are present in the highest number in customer base. It seems that the highest number of customers in the customer base are married with just a small portion of customers being single. Therefore, a large majority of customer base is married and below 45. However, a prominent section of customers is present around the age group of 60s. This might be the retired group of customers.

Chapter 3

Feature engineering

Feature engineering is the process of transforming raw data into a more meaningful format for machine learning models. It involves techniques such as feature extraction, where relevant information is selected or derived from the data, feature transformation, which applies mathematical transformations to ensure data compatibility, and feature encoding, which converts categorical variables into numerical representations. Under feature engineering, we will be performing following exercises:

3.1 Feature encoding

Feature encoding is the process of converting categorical variables into numerical representations that can be effectively used by machine learning algorithms. In this dataset, we have five categorical variables which are encoded using the label encoder function in following ways:

3.1.1 Counting values in each feature

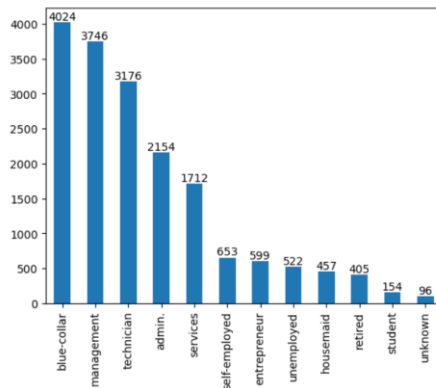
First of all, we will have to count the values present under each category of features before encoding. The counting is as follows:

- **For job**

```
df1.job.value_counts()
blue-collar      4024
management      3746
technician       3176
admin.           2154
services         1712
self-employed    653
entrepreneur     599
unemployed       522
housemaid        457
retired          405
student          154
unknown          96
Name: job, dtype: int64
```

```
job_bar = df1.job.value_counts().plot(kind='bar')
job_bar.bar_label(job_bar.containers[0])
```

```
[Text(0, 0, '4024'),
Text(0, 0, '3746'),
Text(0, 0, '3176'),
Text(0, 0, '2154'),
Text(0, 0, '1712'),
Text(0, 0, '653'),
Text(0, 0, '599'),
Text(0, 0, '522'),
Text(0, 0, '457'),
Text(0, 0, '405'),
Text(0, 0, '154'),
Text(0, 0, '96')]
```



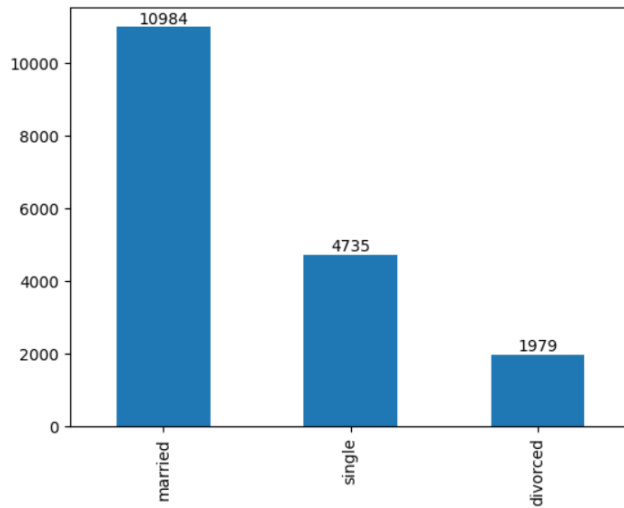
Here, we analyzed the 'job' column in the DataFrame df1. By using the value_counts () method, we determined the frequency of each unique job category. Next, we created a bar plot of the job categories using the plot () function with the argument 'bar' to represent the data in a visually appealing manner. To display the count labels on top of each bar, we utilized the bar_label () function, which allows us to access the containers of the bar plot and attach the count values. Here, there are 4024 rows of blue collar counts, 3746 rows of management counts and so on.

- **For marital**

```
df1.marital.value_counts()
```

```
married      10984
single       4735
divorced      1979
Name: marital, dtype: int64
```

```
marital_bar= df1.marital.value_counts().plot(kind= 'bar')
marital_bar.bar_label(marital_bar.containers[0])
[Text(0, 0, '10984'), Text(0, 0, '4735'), Text(0, 0, '1979')]
```



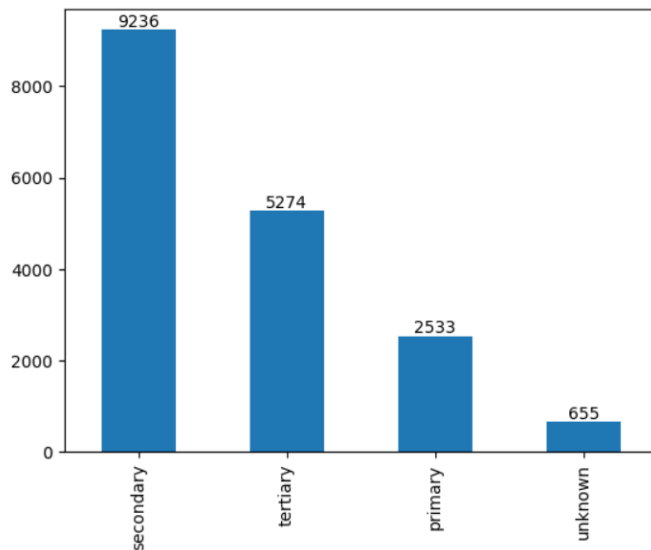
Here, we analyzed the 'marital' column in the DataFrame df1 to understand the distribution of marital status categories. By using the value_counts() method, we obtained the frequency of each unique category. We then created a bar plot of the marital status categories using the plot() function with the argument 'bar', visually representing the data. To display the count labels on top of each bar, we used the bar_label() function, which allowed us to access the containers of the bar plot and attach the count values. Finally we have obtained the value count of rows for married to be 10984 and single to be 4735 and so on.

- **For education**

```
df1.education.value_counts()
secondary    9236
tertiary     5274
primary      2533
unknown       655
Name: education, dtype: int64
```

```
edu_bar= df1.education.value_counts().plot(kind= 'bar')
edu_bar.bar_label(edu_bar.containers[0])

[Text(0, 0, '9236'), Text(0, 0, '5274'), Text(0, 0, '2533'), Text(0, 0, '655')]
```



Here, we analyzed the 'education' column in the DataFrame df1 to examine the distribution of education levels. By applying the value_counts() method, we obtained the frequency of each unique education category. Subsequently, we generated a bar plot representing the education levels using the plot() function with the argument 'bar'. To display the count labels on top of each bar, we utilized the bar_label() function, accessing the containers of the bar plot. By doing so, we attached the count values to their respective bars. We can see that the value count for secondary is 9236, tertiary is 5274 and so on.

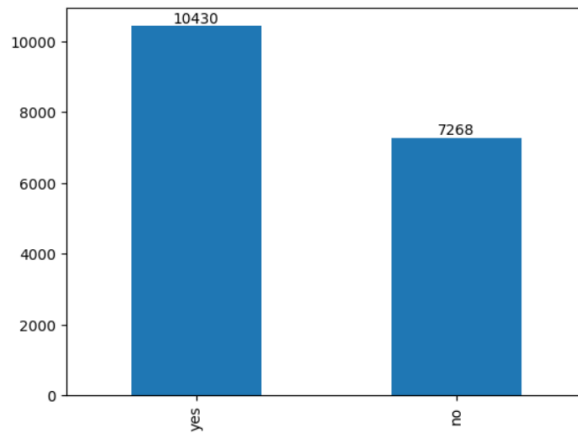
- **For housing**

```
df1.housing.value_counts()
```

```
yes    10430  
no      7268  
Name: housing, dtype: int64
```

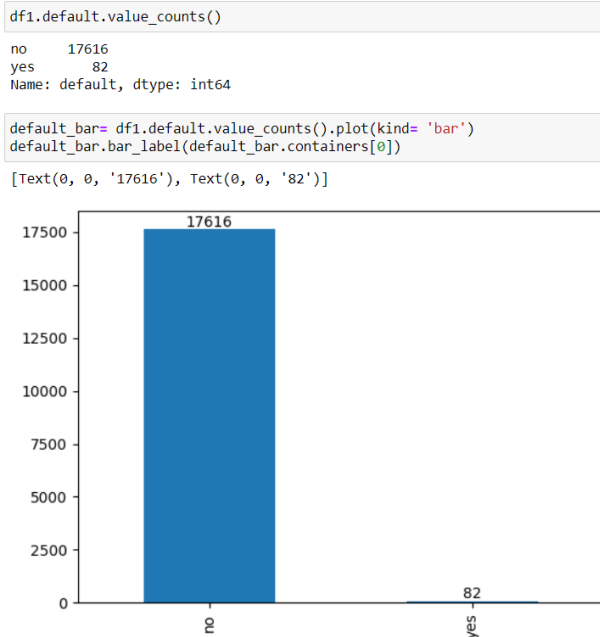
```
housing_bar = df1.housing.value_counts().plot(kind='bar')  
housing_bar.bar_label(housing_bar.containers[0])
```

```
[Text(0, 0, '10430'), Text(0, 0, '7268')]
```



Here, we analyzed the 'housing' column in the DataFrame `df1` to examine the distribution of housing types. By utilizing the `value_counts()` method, we obtained the frequency of each unique housing category. We then created a bar plot representing the housing types using the `plot()` function with the argument 'bar'. To display the count labels on top of each bar, we used the `bar_label()` function, accessing the containers of the bar plot. By doing so, we attached the count values to their respective bars. We can see that the value count for housing loan takers is 10430 and non-housing loan takers is 7268.

- **For default**



Here, we analyzed the 'default' column in the DataFrame df1 to examine the distribution of default status. By utilizing the value_counts() method, we obtained the frequency of each unique default category. We then created a bar plot representing the default status using the plot() function with the argument 'bar'. To display the count labels on top of each bar, we used the bar_label() function, accessing the containers of the bar plot. By doing so, we attached the count values to their respective bars. By applying this code snippet, we visually depicted the distribution of default status and gained insights into the dataset. Here, we can see the value for non defaulters is 17616 and defaulter customers is 82.

3.1.2 Encoding using label encoder

```
encoder= LabelEncoder()

df1['job']= encoder.fit_transform(df1['job'])
df1['marital']= encoder.fit_transform(df1['marital'])
df1['education']= encoder.fit_transform(df1['education'])
df1['housing']= encoder.fit_transform(df1['housing'])
df1['default']= encoder.fit_transform(df1['default'])
```

Here, we utilized the `LabelEncoder` class from the `scikit-learn` library to encode categorical variables in a `DataFrame` called `df1`. The purpose of encoding categorical variables is to convert them into numerical values that machine learning algorithms can effectively process.

We applied the label encoding process to several columns in `df1`, including 'job', 'marital', 'education', 'housing', and 'default'. Each column represents a different categorical feature in the dataset. By using the `fit_transform` method of the `LabelEncoder` instance, we fit the encoder on the corresponding column and transformed the categorical values into numerical labels.

For example, the 'job' column contains categories such as 'Blue collar', 'management', 'technician', 'student' etc. After applying label encoding, these categories would be replaced with numerical labels like 0, 1, 2, and 3, respectively. The same process is repeated for the other columns mentioned.

3.1.3 Counting categorical values after encoding

```
df1.job.value_counts()
```

```
1      4024
4      3746
9      3176
0      2154
7      1712
6       653
2       599
10      522
3       457
5       405
8       154
11       96
Name: job, dtype: int64
```

```
df1.marital.value_counts()
```

```
1      10984
2       4735
0       1979
Name: marital, dtype: int64
```

```
df1.education.value_counts()
```

```
1      9236
2      5274
0      2533
3       655
Name: education, dtype: int64
```

```
df1.housing.value_counts()
```

```
1      10430
0       7268
Name: housing, dtype: int64
```

```
df1.default.value_counts()
```

```
0      17616
1         82
Name: default, dtype: int64
```

Here, we used the `value_counts()` function on the 'job' column of the DataFrame `df1`. This function provides a count of unique values in a column, along with their respective frequencies. By calling

`df1.job.value_counts()`, we obtained a series that displays the count of each unique value in the 'job' column. The output shows the job categories as the index and the corresponding frequency counts as the values. We can observe how categorical columns are changed into numerical values. For example, Blue collar job category has been encoded as value 1.

For example, the output indicates that the category with label '1' appears 4024 times in the 'job' column, '4' appears 3746 times, '9' appears 3176 times, and so on. The same goes for all the other columns that are encoded above. This information is helpful in understanding the distribution and frequency of different job categories present in the dataset. It allows us to gain insights into the prevalence or representation of each job category within the dataset, which can be useful for further analysis or modeling tasks.

3.2 Feature selection

Feature selection in data science refers to the process of identifying and selecting the most relevant and informative features or variables from a dataset that contribute the most towards predicting or explaining the target variable. By eliminating irrelevant or redundant features, feature selection reduces the dimensionality of the data, improves model performance, and enhances interpretability.

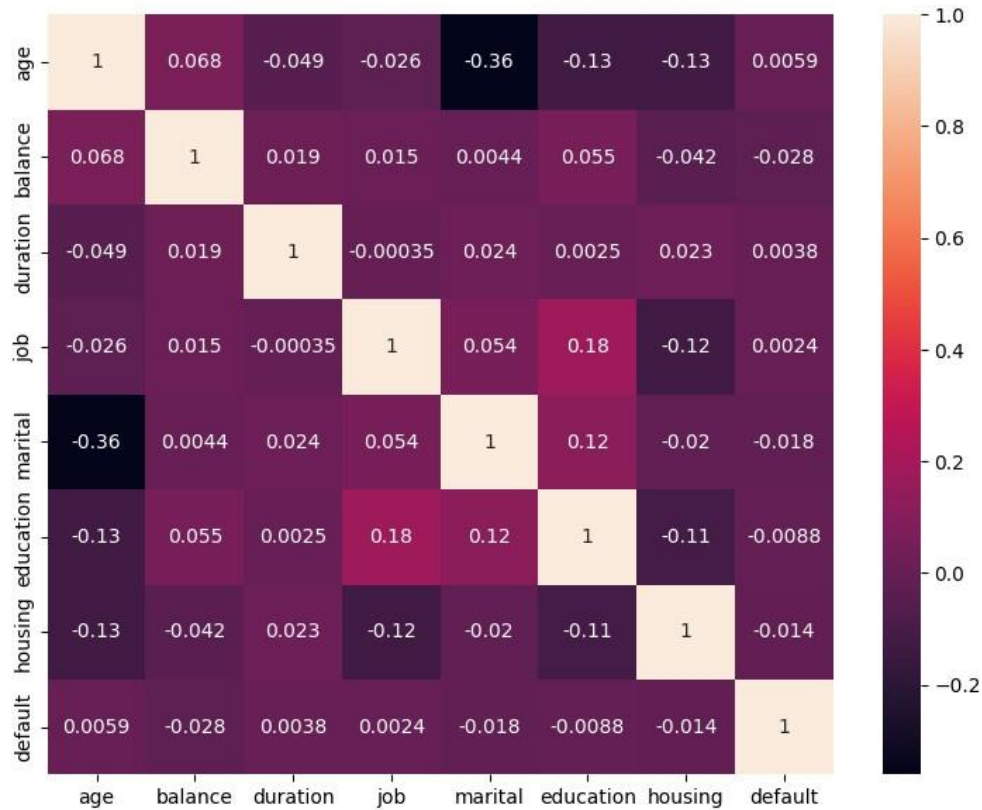
We have analyzed the features by using two main methods of feature selection. **We will choose one of the methods that will give the best result.** The two methods of feature selection are explained below:

3.2.1 Correlation-based Feature Selection

The correlation method of feature selection involves selecting features that exhibit the highest correlation with the target variable. This technique measures the strength and direction of the linear relationship between each feature and the target, typically using correlation coefficients such as Pearson's correlation. By focusing on features with high correlation, we can prioritize those variables that have the strongest association with the target, thereby increasing the likelihood of capturing important predictive patterns and improving the accuracy of the model's predictions. We have done the following analyses under correlation method for the feature selection for this dataset.

Checking the relationship between features and target variable

```
plt.figure(figsize=(9, 7))
sns.heatmap(df1[['age', 'balance', 'duration', 'job', 'marital', 'education', 'housing', 'default']].corr(), annot=True)
```



If we observe this correlation matrix, the features balance, marital, and housing have the most significant NEGATIVE relationship with the target variable default. Their correlation scores are -0.028, -0.018 and -0.014 respectively. Since other features do not have a significant relationship with the target variable, we have taken these 3 variables as our FEATURES.

3.2.2 Principle component analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space by identifying a set of orthogonal axes called principal components. These components capture the maximum variance in the data, enabling data scientists to represent the data with fewer dimensions while retaining the most important information and patterns.

Principal component analysis

```
from sklearn.decomposition import PCA

pca= PCA(0.95)

X_pca= pca.fit_transform(X) X_pca.shape

pca.explained_variance_ratio_

pca.n_components_

X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size= 0.2, random_state=2)

pca_test_model= LogisticRegression()

pca_test_model.fit(X_train,Y_train)

pca_test_model.score(X_train,Y_train)

pca_test_model.score(X_test,Y_test)
```

The code initializes a PCA object, specifying a desired explained variance ratio of 0.95. This means that the resulting transformed data will retain 95% of the total variance in the original dataset. The `fit_transform()` method is then used to fit the PCA model to the input data (X) and transform it into the lower-dimensional space (X_pca). The `explained_variance_ratio_` attribute provides the variance explained by each principal component, indicating the proportion of variance accounted for by each component. The `n_components_` attribute tells us the number of principal components selected to achieve the desired explained variance ratio.

Next, the code splits the transformed data (X_pca) and the target variable (Y) into training and testing sets using the `train_test_split()` function. This is a common practice to evaluate the performance of machine learning models on unseen data. A logistic regression model (pca_test_model) is then created and trained using the training data (X_train and Y_train) using the `fit()` method. The `score()` method is used to assess the model's accuracy on both the training and testing sets, providing a measure of how well the model generalizes to new, unseen data.

3.2.3 Reasons for selecting Correlation based method over PCA

PCA analysis has provided us with only one feature for further analysis i.e housing. However, it was decided to move further with the features that we selected from Correlation method (balance, marital and housing) of feature selection because of the following two reasons:

- **PCA analysis reduced the dimension of features to one which limited our scope of analysis** to use the other classification techniques such as Decision tree and random forest analysis which require more number of features.
- The PCA ANALYSIS here provided the **same test and train score** which correlation method does i.e. 99.5 on the logistic regression. It is plausible for us to move forward with correlation based selection which provides more dimensions at the same score since the test and train score are also unaffected.

3.2.4 Segregating dependent and independent variables

```
X= df1[['balance','marital','housing']]
Y= df1['default']
```

```
print(X,Y)
```

```
      balance  marital  housing
3    1506.00000      1        1
5     231.00000      1        1
6     447.00000      2        1
7    1362.34662      0        1
10    270.00000      0        1
...         ...      ...      ...
45197  1511.00000      2        1
45198  1428.00000      1        0
45201   583.00000      1        0
45202   557.00000      2        0
45209   668.00000      1        0
```

```
[17698 rows x 3 columns] 3      0
5          0
6          0
7          1
10         0
..
45197      0
45198      0
45201      0
45202      0
45209      0
```

```
Name: default, Length: 17698, dtype: int32
```

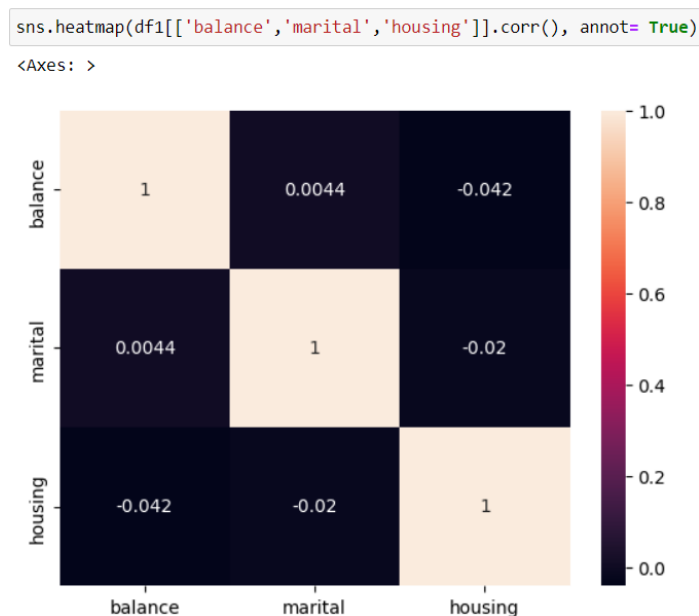
Here, DataFrame df1 is being used to select specific columns and assign them to the variable X. The columns selected are 'balance', 'marital', and 'housing'. These columns are typically features

or independent variables that are considered relevant for predicting or explaining the target variable. Similarly, the target variable is assigned to the variable Y by selecting the column 'default' from the DataFrame df1. This column typically represents the dependent variable that we want to predict or classify.

By assigning the selected features to X and the target variable to Y, we are preparing the data for further analysis or modeling tasks, such as training a machine learning algorithm to predict the default status based on the given features.

3.3 Checking for multicollinearity between the features

Checking for multicollinearity is crucial because high correlation between features can lead to problems in statistical modeling. Multicollinearity inflates the standard errors of the coefficients, making them less reliable and potentially leading to incorrect interpretations. It can also affect the stability of the model and make it challenging to isolate the individual effects of the correlated variables.



Here, we can observe a slight negative correlation between balance and housing and marital and housing respectively. The correlation between balance and marital is too insignificant to consider. The other two correlations are also not very significant which might not affect our models. If we observe a deviation in the test train score later, we will again use the dimensionality reduction technique to rip off some features.

Chapter 4

Preparation for Machine Learning modelling

4.1 Splitting the features and target into test and train data

```
: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.2, random_state=2)
: print(X.shape, X_train.shape, X_test.shape)
(17698, 3) (14158, 3) (3540, 3)
```

Here, the `train_test_split()` function from the `scikit-learn` library is being used to split the feature matrix `X` and the target variable `Y` into separate training and testing sets. The function takes several arguments: `X` represents the features or independent variables, `Y` represents the target variable or dependent variable, `test_size` determines the proportion of the data that should be allocated for testing (in this case, 20% or 0.2), and `random_state` is used to ensure reproducibility of the results.

After calling `train_test_split()`, the data is divided into `X_train` and `X_test`, representing the feature matrices for the training and testing sets, respectively. Similarly, `Y_train` and `Y_test` correspond to the target variables for the training and testing sets, respectively. The `print()` statement is used to display the shapes of the original feature matrix `X`, as well as the training set (`X_train`) and testing set (`X_test`). This allows us to verify the sizes and dimensions of the data splits.

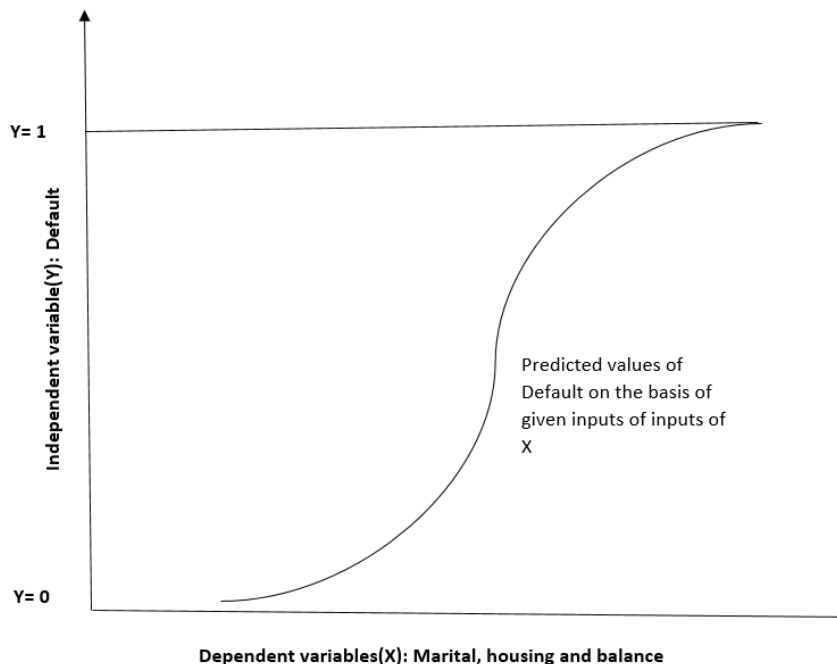
Chapter 5

Predictive Analysis through ML models

5.1 “Model”: Logistic regression

Logistic regression is a machine learning model used for binary classification, where it predicts the probability of an instance belonging to a particular class. It applies a sigmoid function to the linear combination of input features, transforming the output into a probability value. By applying a threshold, instances are classified into one of the classes. The model is trained by estimating optimal coefficients that minimize a specific loss function.

Let us visualize this model in the setting of loan default prediction by using our own features and target variable.



```
model= LogisticRegression()
```

```
model.fit(X_train,Y_train)
```

```
LogisticRegression()
```

Here, a logistic regression model is instantiated using `LogisticRegression()` and assigned to `model`. The `fit()` method is then applied to train the model on the training data (`X_train` and `Y_train`). This

process estimates the optimal coefficients that minimize the difference between predicted and actual target values. By fitting the model to the training data, it learns the underlying patterns and relationships.

5.1.1 Accuracy check on train and test data

Accuracy check on train data

```
X_train_pred= model.predict(X_train)

train_accuracy= accuracy_score(Y_train, X_train_pred)

print("Accuracy of training data is :", train_accuracy)
Accuracy of training data is : 0.9956914818477186
```

Here, the predict() method is used to make predictions on the training data (X_train) using the trained logistic regression model (model). The predicted values are assigned to X_train_pred. Next, the accuracy_score() function is applied to compare the predicted values (X_train_pred) with the actual target values (Y_train). This calculates the accuracy of the model's predictions on the training data. The obtained accuracy value is then printed as a statement indicating the accuracy of the training data, which in this case is reported to be **99.5% which is way above the threshold of 70%. Ideally, this is considered a perfect score for a model because a model cannot be 100% correct.**

Accuracy check on test data

```
X_test_pred= model.predict(X_test)

test_accuracy= accuracy_score(Y_test,X_test_pred)

print("Accuracy of test data is :", test_accuracy)
Accuracy of test data is : 0.9940677966101695
```

Here, the trained logistic regression model (model) is used to make predictions on the testing data (X_test) using the predict() method. The predicted values are assigned to X_test_pred. Subsequently, the accuracy_score() function is utilized to evaluate the accuracy of the model's predictions on the testing data by comparing the predicted values (X_test_pred) with the actual target values (Y_test). The obtained accuracy score is then printed as a statement indicating the

accuracy of the model on the test data which came out to be 99.4%. Since there is very negligible difference between the test and train score, there is **NO PROBLEM OF OVERFITTING**.

5.1.2 Actual prediction

```
default= {  
    'balance':500,  
    'marital':1,  
    'housing': 1  
}
```

```
default= pd.DataFrame([default])
```

```
logistic_pred= model.predict(default)
```

```
if logistic_pred == 0:  
    print("Null hypothesis is accepted i.e The customer is unlikely to default")  
else:  
    print("Alternative hypothesis is accepted i.e The customer is likely to default")
```

```
Null hypothesis is accepted i.e The customer is unlikely to default
```

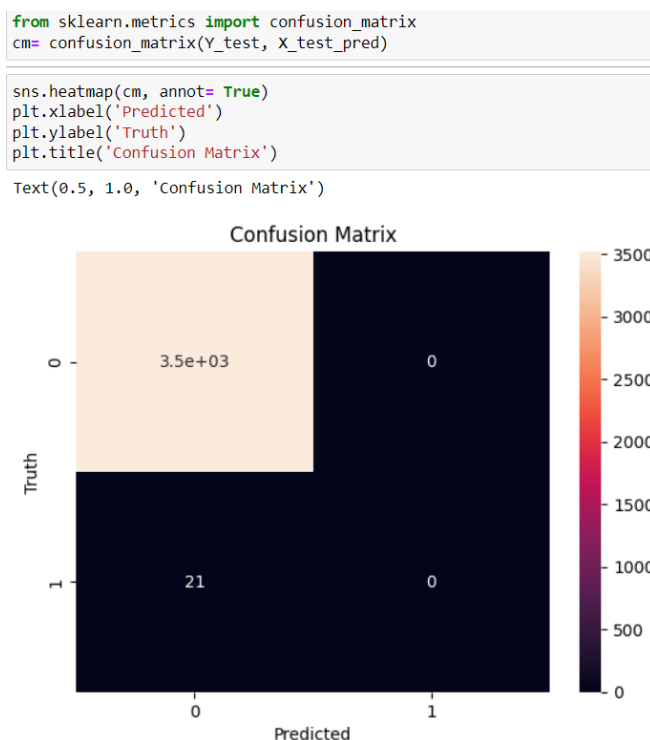
Here, a dictionary default is defined with specific values for the features 'balance', 'marital', and 'housing'. This dictionary is then used to create a DataFrame named default using pd.DataFrame([default]), representing a single instance of data. The trained logistic regression model (model) is then used to predict the target variable for this instance using the predict() method, and the predicted value is assigned to logistic_pred.

To interpret the prediction, an if statement is used to check if logistic_pred is equal to 0. If it is, the statement "Null hypothesis is accepted, i.e., the customer is unlikely to default" is printed. Otherwise, if logistic_pred is not equal to 0, the statement "Alternative hypothesis is accepted, i.e., the customer is likely to default" is printed. This code segment demonstrates using the trained logistic regression model to make a prediction for a new instance (default) and interpreting the prediction based on the predicted target variable value.

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.1.3 Evaluation and visualization of logistic prediction

We are using a confusion matrix to visualize the prediction from logistic regression. A confusion matrix is a square matrix that summarizes the performance of a classification model by displaying the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. It allows for the evaluation of the model's accuracy, precision, recall, and other performance metrics by comparing the predicted and actual classes. The rows of the matrix represent the actual classes, while the columns represent the predicted classes, providing a concise representation of the model's classification performance.



Here, the `confusion_matrix()` function from scikit-learn's metrics module is imported. The confusion matrix is then computed by passing the true target values (`Y_test`) and the predicted values (`X_test_pred`) to the `confusion_matrix()` function, and the resulting matrix is assigned to `cm`. To visualize the confusion matrix, `sns.heatmap()` from the seaborn library is used. The `cm` matrix is passed as the data to be visualized, and the `annot=True` parameter is set to display the numerical values within the heatmap. Additional plot formatting is performed using matplotlib: `plt.xlabel('Predicted')` sets the x-axis label as "Predicted", `plt.ylabel('Truth')` sets the y-axis label as "Truth", and `plt.title('Confusion Matrix')` sets the title of the plot as "Confusion Matrix".

The confusion matrix here can be interpreted in a simple way. For 3500 times, The logistic regression model here predicted the output to be Non default(0) and the truth was Non default. Similarly, 21 times, the truth was default and the model predicted to be Non default. We can clearly see a very low error rate. **Now, if we calculate (3500-21)/3500 we will get our test data score i.e 99.4%.** This is how our test score gets derived.

5.2 “Model 1”: Naïve Bayes classification algorithm

Bayes' theorem is a fundamental concept in probability theory and statistics. It provides a way to update the probability of an event based on new evidence or information.

$$P(A|B) = (P(B|A) * P(A)) / P(B)$$

Since we are applying Naive bayes to Classify defaulters from non defaulters, lets us understand the theorem in our context (P(Default)/Bank balance, Marital status, Housing loan). Also, we will make some

NAIVE ASSUMPTION:

- Features Bank balance, Marital status, Housing loan are independent of each other (No multicollinearity)

```
model1 = GaussianNB()
```

```
model1.fit(X_train, Y_train)
```

```
GaussianNB()
```

Here, a Gaussian Naive Bayes model is created and assigned to the variable model1. This model is then trained using the training data (X_train and Y_train) by calling the fit() method on model1.

The fit() method calculates the necessary statistical parameters from the training data to build the Naive Bayes model. In the case of Gaussian Naive Bayes, it estimates the mean and variance for each feature in each class based on the provided training instances. By executing this code, the Gaussian Naive Bayes model (model1) learns the underlying patterns and relationships in the training data, enabling it to make predictions based on the input features.

5.2.1 Accuracy check on train and test data

Accuracy on train data

```
model1.score(X_train, Y_train)

0.9956914818477186
```

The code `model1.score(X_train, Y_train)` calculates the accuracy of the trained Gaussian Naive Bayes model (`model1`) on the training data. The `score()` method evaluates the model's performance by comparing the predicted labels for the input features (`X_train`) with the true labels (`Y_train`) from the training data. It returns the accuracy of the model, which represents the percentage of correctly predicted instances in the training set which stood at 99.5%.

Accuracy on test data

```
model1.score(X_test, Y_test)

0.9940677966101695
```

The code `model1.score(X_test, Y_test)` calculates the accuracy of the trained Gaussian Naive Bayes model (`model1`) on the test data.

Similar to the explanation provided earlier, the `score()` method evaluates the model's performance by comparing the predicted labels for the input features (`X_test`) with the true labels (`Y_test`) from the test data. It returns the accuracy of the model, representing the percentage of correctly predicted instances in the test set. The obtained accuracy score is then printed as a statement indicating the accuracy of the model on the test data which came out to be 99.4%. Since there is very negligible difference between the test and train score, there is **NO PROBLEM OF OVERFITTING**.

5.2.2 Actual prediction

```
naive_pred= model1.predict(default)

if naive_pred == 0:
    print("Null hypothesis is accepted i.e The customer is unlikely to default")
else:
    print("Alternative hypothesis is accepted i.e The customer is likely to default")

Null hypothesis is accepted i.e The customer is unlikely to default
```

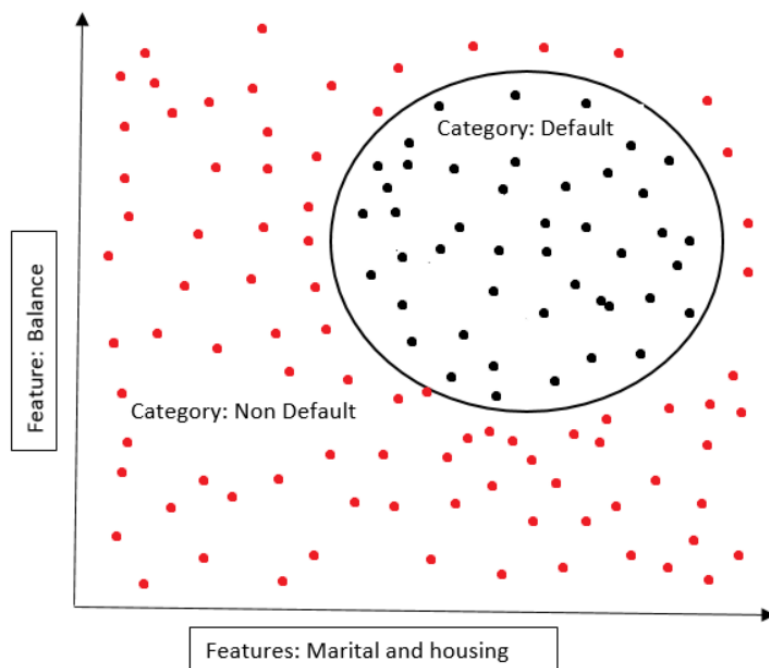
The trained Naïve bayes model (model 1) is then used to predict the target variable for this instance using the predict() method, and the predicted value is assigned to naive_pred. we asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan.

To interpret the prediction, an if statement is used to check if naive_pred is equal to 0. If it is, the statement "Null hypothesis is accepted, i.e., the customer is unlikely to default" is printed. Otherwise, if naive_pred is not equal to 0, the statement "Alternative hypothesis is accepted, i.e., the customer is likely to default" is printed. This code segment demonstrates using the trained naïve bayes model to make a prediction for a new instance (default) and interpreting the prediction based on the predicted target variable value.

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.3 “Model2”: K Nearest neighbor

K-Nearest Neighbors (KNN) is a machine learning algorithm that predicts the class or value of a data point based on the classes or values of its neighboring data points. It uses a distance metric to find the K nearest neighbors in the training data and assigns the majority class or average value of those neighbors as the prediction for the new data point. KNN is a versatile and intuitive algorithm, applicable to both classification and regression tasks, and is useful when decision boundaries are not well-defined or when there is limited training data.



```
model2= KNeighborsClassifier(n_neighbors=3)
```

```
model2.fit(X_train, Y_train)
```

```
KNeighborsClassifier(n_neighbors=3)
```

Here, a K-Nearest Neighbors (KNN) model is created and assigned to the variable model2. The model is initialized with n_neighbors=3, which sets the number of neighbors considered for prediction to 3. The KNN model is then trained using the training data (X_train and Y_train) by calling the fit() method on model2. During the training process, the model learns the relationships between the input features in X_train and their corresponding target labels in Y_train, enabling it to make predictions based on the nearest neighbors in the feature space.

5.3.1 Accuracy check on train and test data

Accuracy on train data

```
model2.score(X_train, Y_train)
```

```
0.9959740076281961
```

The code `model2.score(X_train, Y_train)` calculates the accuracy of the trained KNN model (`model2`) on the training data. The `score()` method evaluates the model's performance by comparing the predicted labels for the input features (`X_train`) with the true labels (`Y_train`) from the training data. It returns the accuracy of the model, which represents the percentage of correctly predicted instances in the training set which stood at 99.5%.

Accuracy on test data

```
model2.score(X_test, Y_test)
0.9937853107344633
```

The code `model2.score(X_test, Y_test)` calculates the accuracy of the trained KNN model (`model2`) on the test data.

Similar to the explanation provided earlier, the `score()` method evaluates the model's performance by comparing the predicted labels for the input features (`X_test`) with the true labels (`Y_test`) from the test data. It returns the accuracy of the model, representing the percentage of correctly predicted instances in the test set. The obtained accuracy score is then printed as a statement indicating the accuracy of the model on the test data which came out to be 99.3%. Since there is very negligible difference between the test and train score, there is **NO PROBLEM OF OVERFITTING**.

5.3.2 Actual prediction

```
knn_pred = model2.predict(default)

if knn_pred == 0:
    print("Null hypothesis is accepted i.e The customer is unlikely to default")
else:
    print("Alternative hypothesis is accepted i.e The customer is likely to default")

Null hypothesis is accepted i.e The customer is unlikely to default
```

Here, the trained K-Nearest Neighbors (KNN) model (`model2`) is used to make predictions on a dataframe (`default`). The `predict()` method is applied to the default dataframe, which returns the predicted class label based on the learned patterns from the training data.

The code then checks the predicted label (`knn_pred`) and compares it to 0. If the predicted label is 0, it implies that the null hypothesis is accepted, meaning that the customer is unlikely to default.

Otherwise, if the predicted label is not 0, the alternative hypothesis is accepted, indicating that the customer is likely to default.

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.4 “Model 3”: Decision tree

A decision tree algorithm is a machine learning algorithm that builds a predictive model in the form of a tree structure. It is a popular algorithm for both classification and regression tasks. The decision tree algorithm partitions the input data into subsets based on various attributes and makes decisions at each node of the tree based on these attributes. Each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents the final decision or the predicted outcome.

```
model3= tree.DecisionTreeClassifier(max_depth=3)
```

```
model3.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(max_depth=3)
```

Here, the variable model3 is assigned to an instance of the DecisionTreeClassifier class from the tree module. This class is part of the scikit-learn library, which is a popular machine learning library in Python. The DecisionTreeClassifier is a specific implementation of the decision tree algorithm for classification tasks. It creates a decision tree model that can be used to classify new instances based on their features.

The parameter max_depth=3 is passed to the DecisionTreeClassifier constructor. This parameter sets the maximum depth of the decision tree. The depth of a tree refers to the length of the longest path from the root node to any leaf node. By setting max_depth=3, we are limiting the depth of the tree to 3 levels. This means that the decision tree will have a maximum of 3 decision nodes, which helps to control the complexity and overfitting of the model.

The `fit()` method is used to train the `model3` decision tree classifier on the labeled training dataset (`X_train` for features and `Y_train` for labels). This process involves analyzing the patterns and relationships between the features and labels in the training data to construct an internal representation of the decision tree model.

5.4.1 Accuracy check on train and test data

```
model3.score(X_train, Y_train)
```

```
0.9956914818477186
```

```
model3.score(X_test, Y_test)
```

```
0.9940677966101695
```

`model3.score(X_train, Y_train)`, the `score()` method is used to evaluate the accuracy or performance of the `model3` decision tree classifier on the training data. The method compares the predictions made by the trained model (`model3`) with the actual labels (`Y_train`) from the training dataset, and returns the accuracy score.

Similarly, in the code snippet `model3.score(X_test, Y_test)`, the `score()` method is used to evaluate the accuracy or performance of the `model3` decision tree classifier on the test data. The method compares the predictions made by the trained model (`model3`) with the actual labels (`Y_test`) from the test dataset, and returns the accuracy score.

The accuracy score represents the proportion of correct predictions made by the model on the given dataset. A score of 99.5% on the train data and 99.4% on the test data indicates that the decision tree model (`model3`) is performing well and is able to predict the class labels with high accuracy on both the training and test datasets. This suggests that the model has learned the underlying patterns and relationships in the data effectively and is generalizing well to unseen instances.

5.4.2 Actual prediction

```
: Tree_pred= model3.predict(default)

: if Tree_pred == 0:
    print("Null hypothesis is accepted i.e The customer is unlikely to default")
else:
    print("Alternative hypothesis is accepted i.e The customer is likely to default")

Null hypothesis is accepted i.e The customer is unlikely to default
```

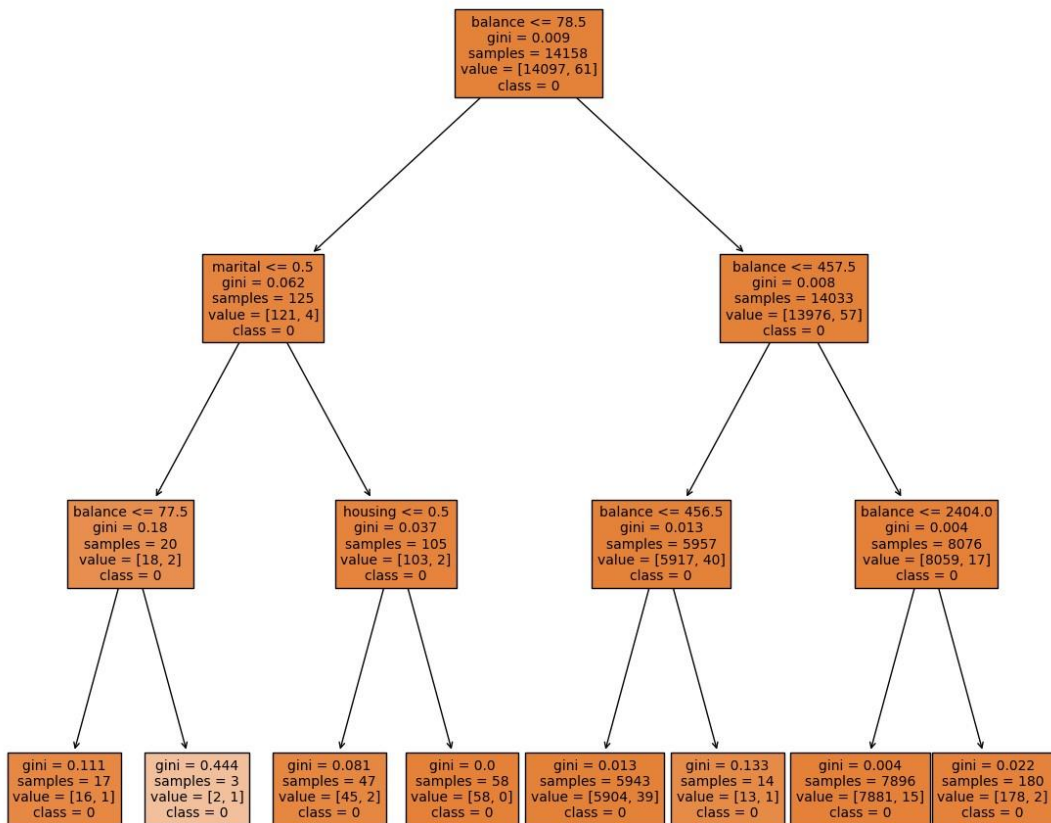
Here, the trained model3 decision tree classifier is used to predict whether a customer is likely to default or not, based on the input represented by the variable default. The predict() method is used to obtain the prediction result (Tree_pred). The code then uses an if-else statement to check the predicted label: if it is 0, the code prints that the null hypothesis is accepted, indicating that the customer is unlikely to default; otherwise, it prints that the alternative hypothesis is accepted, suggesting that the customer is likely to default. This code snippet helps classify new instances based on the learned decision tree model's predictions.

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.4.3 Visualization of decision tree

```
from sklearn.tree import plot_tree
```

The above code imports the plot_tree function from the tree module of the scikit-learn library. This function allows us to visualize and plot a decision tree model. By importing this function, we gain access to a useful tool that helps in understanding the structure and decision-making process of a decision tree by providing a graphical representation of the tree's nodes, branches, and leaf nodes.



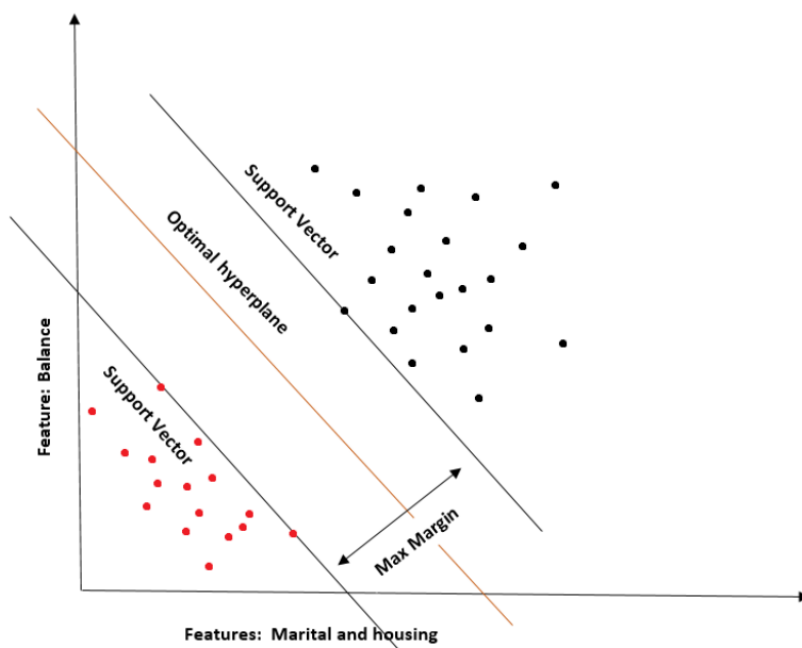
Here, the provided code snippet, a figure with the name "Decision Tree" is created using `plt.figure()`, specifying the figure size. The unique class labels (`Y.unique()`) are converted to strings (`astype(str)`) and assigned to the variable `class_names`. The `plot_tree` function is then called to plot and visualize the model3 decision tree. It sets the font size to 10, fills the tree nodes with color (`filled=True`), and labels the features using the column names of the input matrix `X` (`feature_names=X.columns`). The class labels are assigned as class names for the tree nodes (`class_names=class_names`). Finally, `plt.show()` is used to display the plotted decision tree figure. This code snippet visualizes the decision tree model, providing a graphical representation of its structure and key decision points.

In the above figure, we can observe the boxes with lighter colors signifying a higher gini coefficient. It simply means that there is higher IMPURITY there. In the first box, if we have to interpret it, if the balance is smaller than or equals to with gini coefficient 0.0009 in 14158

samples, the predicted class of the default will be 0 i.e The customer is unlikely to default. In the same way, all the boxes in the tree can be interpreted.

5.5 “Model 4”: Support vector machine

The support vector machine (SVM) algorithm is a powerful supervised machine learning technique used for classification and regression tasks. SVM aims to find an optimal hyperplane in a high-dimensional feature space that best separates different classes of data points. It achieves this by constructing a decision boundary that maximizes the margin, i.e., the distance between the decision boundary and the closest data points from each class. SVM can handle both linearly separable and non-linearly separable datasets by using various kernel functions that transform the original input space into a higher-dimensional space where linear separation is possible. During training, SVM identifies a subset of data points called support vectors that play a crucial role in defining the decision boundary. In the case of classification, new instances are classified based on which side of the decision boundary they fall on. Let’s try to envision the support vector machine to fit in our dataset and variable of interest.



```
model4= SVC()
```

```
model4.fit(X_train, Y_train)
```

```
SVC()
```

Here, model4 is assigned to an instance of the SVC class, which stands for Support Vector Classifier. The SVC class is part of the scikit-learn library and represents the implementation of the Support Vector Machine (SVM) algorithm for classification tasks. By creating model4 as an instance of SVC(), we are initializing a SVM classifier without specifying any hyperparameters. This allows the classifier to use default settings for parameters like the kernel function, regularization parameter, and so on.

The code then proceeds to train the model4 SVM classifier using the fit() method by providing the labeled training data (X_train for features and Y_train for labels). This process involves finding the optimal hyperplane that maximizes the margin between different classes of data points, using the SVM algorithm's optimization principles

5.5.1 Accuracy check on train and test data

```
: model4.score(X_train, Y_train)
```

```
: 0.9956914818477186
```

```
: model4.score(X_test, Y_test)
```

```
: 0.9940677966101695
```

Here, the score() method is used to evaluate the accuracy or performance of the model4 Support Vector Classifier (SVC) on both the training and test datasets. By calling model4.score(X_train, Y_train), the accuracy score of the model4 SVC is computed on the training data (X_train for features and Y_train for labels). This score represents the proportion of correctly predicted labels on the training dataset. In this case, the accuracy score is reported as 99.5%, indicating that the model achieves a high level of accuracy in predicting the labels on the training data.

Similarly, by calling model4.score(X_test, Y_test), the accuracy score of the model4 SVC is computed on the test data (X_test for features and Y_test for labels). This score represents the

proportion of correctly predicted labels on the test dataset. The reported accuracy score is 99.4%, suggesting that the model generalizes well and performs with high accuracy on unseen data.

5.5.2 Actual prediction

```
svc_pred= model4.predict(default)
```

```
if svc_pred == 0:  
    print("Null hypothesis is accepted i.e The customer is unlikely to default")  
else:  
    print("Alternative hypothesis is accepted i.e The customer is likely to default")
```

```
The customer is unlikely to default
```

Here, the provided code snippet, the trained model4 Support Vector Classifier (SVC) is used to predict whether a customer is likely to default or not based on a new instance represented by the variable default. By calling model4.predict(default), the code obtains the predicted label (svc_pred) for the input instance. The subsequent if-else statement checks the predicted label and prints the corresponding hypothesis: if svc_pred is 0, the null hypothesis is accepted, indicating that the customer is unlikely to default, while any other value of svc_pred leads to accepting the alternative hypothesis, suggesting that the customer is likely to default.

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.6 “Model 5”: Random forest classifier

The random forest classifier is a versatile and powerful supervised machine learning algorithm used for classification tasks. It combines the principles of ensemble learning and decision trees to make predictions. In a random forest, multiple decision trees are constructed using random subsets of the training data and random subsets of the input features. Each decision tree independently makes predictions, and the final prediction is determined through a voting mechanism, where the class with the most votes is chosen. This ensemble approach helps to reduce overfitting, improve generalization, and increase accuracy. Random forests are robust to noise, handle high-dimensional data well, and can capture complex relationships between variables.

```
model5 = RandomForestClassifier(n_estimators=5, random_state=42)
```

```
model5.fit(X_train, Y_train)
```

```
RandomForestClassifier(n_estimators=5, random_state=42)
```

Here, we used the Random Forest classifier from scikit-learn library and created an instance of it called "model5". We specified that the random forest should consist of 5 decision trees by setting the parameter "n_estimators" to 5. Additionally, we set the random seed to 42 using the "random_state" parameter for reproducibility purposes. Next, we trained the model on the training data, where "X_train" represents the input features and "Y_train" represents the corresponding target labels. By calling the "fit" method on the model and passing the training data, we allowed the model to learn patterns and relationships between the input features and their corresponding labels.

5.6.1 Accuracy check on train and test data

```
model5.score(X_train, Y_train)
```

```
0.9964684277440317
```

```
model5.score(X_test, Y_test)
```

```
0.9929378531073446
```

Here, The line "model5.score(X_train, Y_train)" computes the accuracy of the model on the training data, represented by "X_train" (input features) and "Y_train" (corresponding target labels). The score of 99.6 indicates that the model predicted the correct labels for 99.6% of the samples in the training set.

Likewise, the line "model5.score(X_test, Y_test)" evaluates the accuracy of the model on the testing data. Here, "X_test" represents the input features of the testing set, and "Y_test" represents the true labels. The score of 99.2 implies that the model correctly predicted the labels for 99.2% of the samples in the testing set. These scores suggest that the model performs very well on both the training and testing data, exhibiting high accuracy in its predictions.

5.6.2 Actual prediction

```
: random_pred= model5.predict(default)

: if random_pred == 0:
    print("Null hypothesis is accepted i.e The customer is unlikely to default")
else:
    print("Alternative hypothesis is accepted i.e The customer is likely to default")

The customer is unlikely to default
```

Here, we make predictions using the trained "model5" on a new data instance called "default" using the predict method. The result is stored in the variable "random_pred". Next, we use a conditional statement to check the predicted value. If the predicted value, "random_pred", is equal to 0, it means that the model predicts the customer is unlikely to default. In this case, the code prints the message "Null hypothesis is accepted i.e The customer is unlikely to default". On the other hand, if the predicted value is not equal to 0, it implies that the model predicts the customer is likely to default. In this scenario, the code prints the message "Alternative hypothesis is accepted i.e The customer is likely to default".

We asked the model to predict whether or not the customers will default if the customer has a balance of 500, a marital status of “married” and has taken a “housing” loan. Here, since the output gave a 0 i.e no prediction, Null hypothesis is accepted since the customers is predicted to be unlikely to default.

5.6.3 Visualization of Random forest

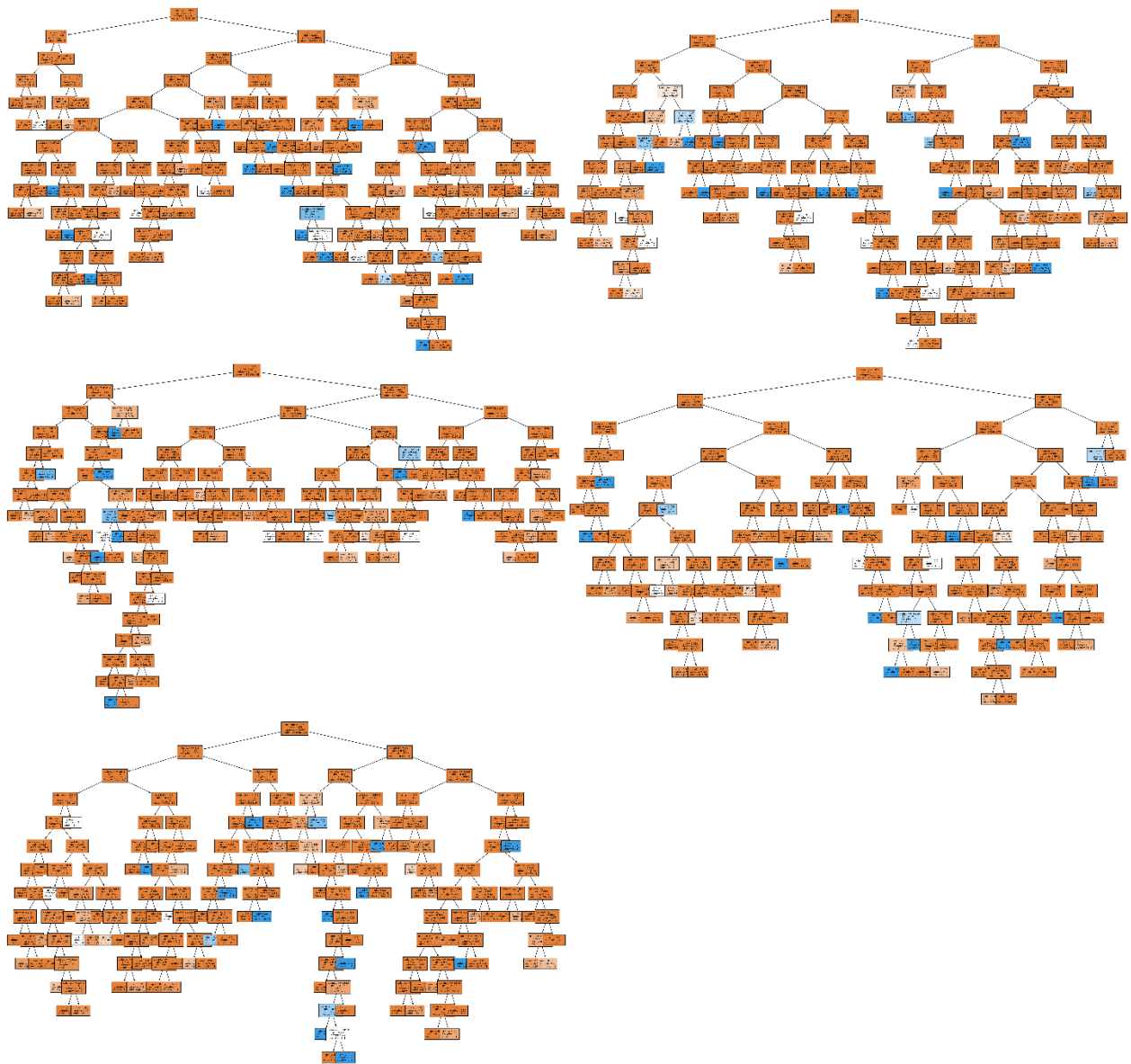
```
import math

estimators = model5.estimators_
num_estimators = len(estimators)
num_rows = math.ceil(math.sqrt(num_estimators))
num_cols = math.ceil(num_estimators / num_rows)

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(40, 38), dpi=300)

for i, ax in enumerate(axes.flatten()):
    if i < num_estimators:
        ax.set_title('Tree {}'.format(i + 1))
        tree.plot_tree(estimators[i], ax=ax, fontsize=6, filled=True, feature_names=X.columns)
    else:
        fig.delaxes(ax)

plt.tight_layout()
plt.show()
```



In the above figure, Lets observe the first decision tree in the forest, we can observe the boxes with lighter colors signifying a higher gini coefficient. It simply means that there is higher IMPURITY there. In the first box, if we have to interpret it, if the balance is smaller than or equals to with gini coefficient 0.0009 in 14158 samples, the predicted class of the default will be 0 i.e The customer is unlikely to default. In the same way, all the trees in the forest can be interpreted.

Chapter 6

Selection of best model for prediction

6.1 Stratified K folds cross validation

Cross-validation is a valuable technique used in data science to select the best model among multiple candidate models. It involves dividing the dataset into several subsets or folds, where each fold is used as a validation set while the remaining folds are used for training. The models are then trained and evaluated on each fold, and their performance metrics, such as accuracy or mean squared error, are recorded. By averaging the performance across all folds, cross-validation provides an objective measure of how well each model generalizes to unseen data. This enables data scientists to compare and select the model with the highest average performance, helping to identify the most effective model for a given task.

We will be selecting the model with the lowest standard deviation in the cross validation scores. The standard deviation is a measure of the dispersion or spread of a dataset. In this context, the standard deviation of the cross-validation scores indicates the degree of variation in the model's performance across the different folds. A lower standard deviation suggests more consistent performance across the folds, while a higher standard deviation implies greater variability in the model's performance. In this case, the relatively low standard deviation value suggests that the model's performance is consistent and stable across the cross-validation folds.

6.1.1 Importing required libraries

```
from sklearn.model_selection import StratifiedKFold  
folds= StratifiedKFold(n_splits=5)
```

```
from sklearn.model_selection import cross_val_score
```

Here, the StratifiedKFold class from the sklearn.model_selection module. StratifiedKFold is a cross-validation method that splits the dataset into multiple folds while preserving the distribution of class labels. In this case, it is used to create an instance of StratifiedKFold with 5 splits (folds).

Next, the code snippet imports the cross_val_score function from the same module. This function is used to perform cross-validation by splitting the dataset into folds and evaluating a model's performance on each fold. It takes as input a machine learning model, the dataset, and a cross-

validation strategy (in this case, the StratifiedKFold object) to perform the evaluation. The output of cross_val_score is an array of scores, where each score represents the performance of the model on a particular fold.

6.1.2 Cross validation score and standard deviation between the folds for each model

For "Model": Logistic regression

```
cross_val_score(model, X, Y)
array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])

std_model = np.array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])
std_dev = np.std(std_model)

print(std_dev)

0.00013797401688719847
```

Here, the cross_val_score function is used to perform cross-validation on a machine learning model (model) using features X and target values Y. The function returns an array of scores representing the performance of the model on each fold of the cross-validation procedure. The scores obtained from cross_val_score are: 0.99548023, 0.99519774, 0.99519774, 0.99547895, and 0.99547895.

To calculate the standard deviation of these scores, the std function from the NumPy library is applied to the array std_model, which contains the scores as a NumPy array. The calculated standard deviation is then printed, resulting in the value 0.00013797401688719847.

For "Model 1": Naive Bayesian regression

```
cross_val_score(model1, X, Y)
array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])

std_model1 = np.array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])
std_dev1 = np.std(std_model1)

print(std_dev1)

0.00013797401688719847
```

The cross_val_score function is used to perform cross-validation on the machine learning model model1 using the features X and the target values Y. This function returns an array of scores representing the performance of the model on each fold of the cross-validation procedure. The std_model1 array is created by assigning the scores obtained from cross_val_score to a NumPy

array. The scores are: 0.99548023, 0.99519774, 0.99519774, 0.99547895, and 0.99547895. Next, the std function from the NumPy library is used to calculate the standard deviation of the scores stored in the std_model1 array. The calculated standard deviation is then printed, resulting in the value 0.00013797401688719847.

For "Model 2": K Nearest neighbor

```
cross_val_score(model2, X, Y)
array([0.99491525, 0.99491525, 0.99519774, 0.99547895, 0.99491382])

std_model2 = np.array([0.99491525, 0.99491525, 0.99519774, 0.99547895, 0.99491382])
std_dev2 = np.std(std_model2)

print(std_dev2)

0.00022575936015146265
```

Here, the provided code performs cross-validation on a machine learning model (model2). The cross_val_score function is used to perform cross-validation on model2 using the features X and the target values Y. It returns an array of scores representing the performance of the model on each fold of the cross-validation procedure. The std_model2 array is created by assigning the scores obtained from cross_val_score to a NumPy array. The scores in this case are: 0.99491525, 0.99491525, 0.99519774, 0.99547895, and 0.99491382. Using the std function from the NumPy library, the standard deviation of the scores stored in the std_model2 array is calculated. The calculated standard deviation is then printed, resulting in the value 0.00022575936015146265.

For “Model 3”: Decision tree

```
cross_val_score(model3, X, Y)
array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])

std_model3 = np.array([0.99265537, 0.99378531, 0.9920904 , 0.99463125, 0.99463125])
std_dev3 = np.std(std_model3)

print(std_dev3)

0.0010318904083980714
```

Here, the cross_val_score function is used to perform cross-validation on machine learning model (model3). The cross_val_score function is applied to model3 using the features X and the target

values Y. It returns an array of scores representing the performance of the model on each fold of the cross-validation procedure.

The `std_model3` array is created by assigning the scores obtained from `cross_val_score` to a NumPy array. The scores in this case are: 0.99265537, 0.99378531, 0.9920904, 0.99463125, and 0.99463125. The `std` function from the NumPy library is used to calculate the standard deviation of the scores stored in the `std_model3` array. The calculated standard deviation is then printed, resulting in the value 0.0010318904083980714.

For “Model 4”: Support vector machine

```
cross_val_score(model4, X, Y)
array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])

std_model4 = np.array([0.99548023, 0.99519774, 0.99519774, 0.99547895, 0.99547895])
std_dev4 = np.std(std_model4)

print(std_dev4)

0.00013797401688719847
```

Here, the `cross_val_score` function is used to perform cross-validation on machine learning model (model4). The `cross_val_score` function is applied to model4 using the features X and the target values Y. It returns an array of scores representing the performance of the model on each fold of the cross-validation procedure. The `std_model4` array is created by assigning the scores obtained from `cross_val_score` to a NumPy array. The scores in this case are: 0.99548023, 0.99519774, 0.99519774, 0.99547895, and 0.99547895. The `std` function from the NumPy library is used to calculate the standard deviation of the scores stored in the `std_model4` array. The calculated standard deviation is then printed, resulting in the value 0.00013797401688719847.

For “Model 5”: Random forest classifier

```
cross_val_score(model5, X, Y)
array([0.99378531, 0.99293785, 0.99011299, 0.99293586, 0.99180559])

std_model5 = np.array([0.9940678 , 0.99350282, 0.99180791, 0.99434869, 0.99406612])
std_dev5 = np.std(std_model5)

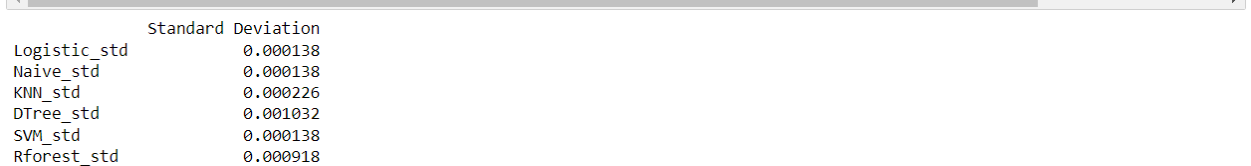
print(std_dev5)

0.0009175107912041146
```

Here, the `cross_val_score` function is used to perform cross-validation on machine learning model (model5). The `cross_val_score` function is applied to model5 using the features X and the target values Y. It returns an array of scores representing the performance of the model on each fold of the cross-validation procedure. The `std_model5` array is created by assigning the scores obtained from `cross_val_score` to a NumPy array. The scores in this case are: 0.9940678, 0.99350282, 0.99180791, 0.99434869, and 0.99406612. The `std` function from the NumPy library is used to calculate the standard deviation of the scores stored in the `std_model5` array. The calculated standard deviation is then printed, resulting in the value 0.0009175107912041146.

6.2 Visualization of std deviation of models across the folds

```
std_scores = [0.00013797401688719847,0.00013797401688719847 ,0.00022575936015146265,0.0010318904083980714,0.00013797401688719847,
score_names = ['Logistic_std', 'Naive_std','KNN_std', 'DTree_std', 'SVM_std', 'Rforest_std']
best_model = pd.DataFrame({'Standard Deviation': std_scores}, index=score_names)
print(best_model)
```

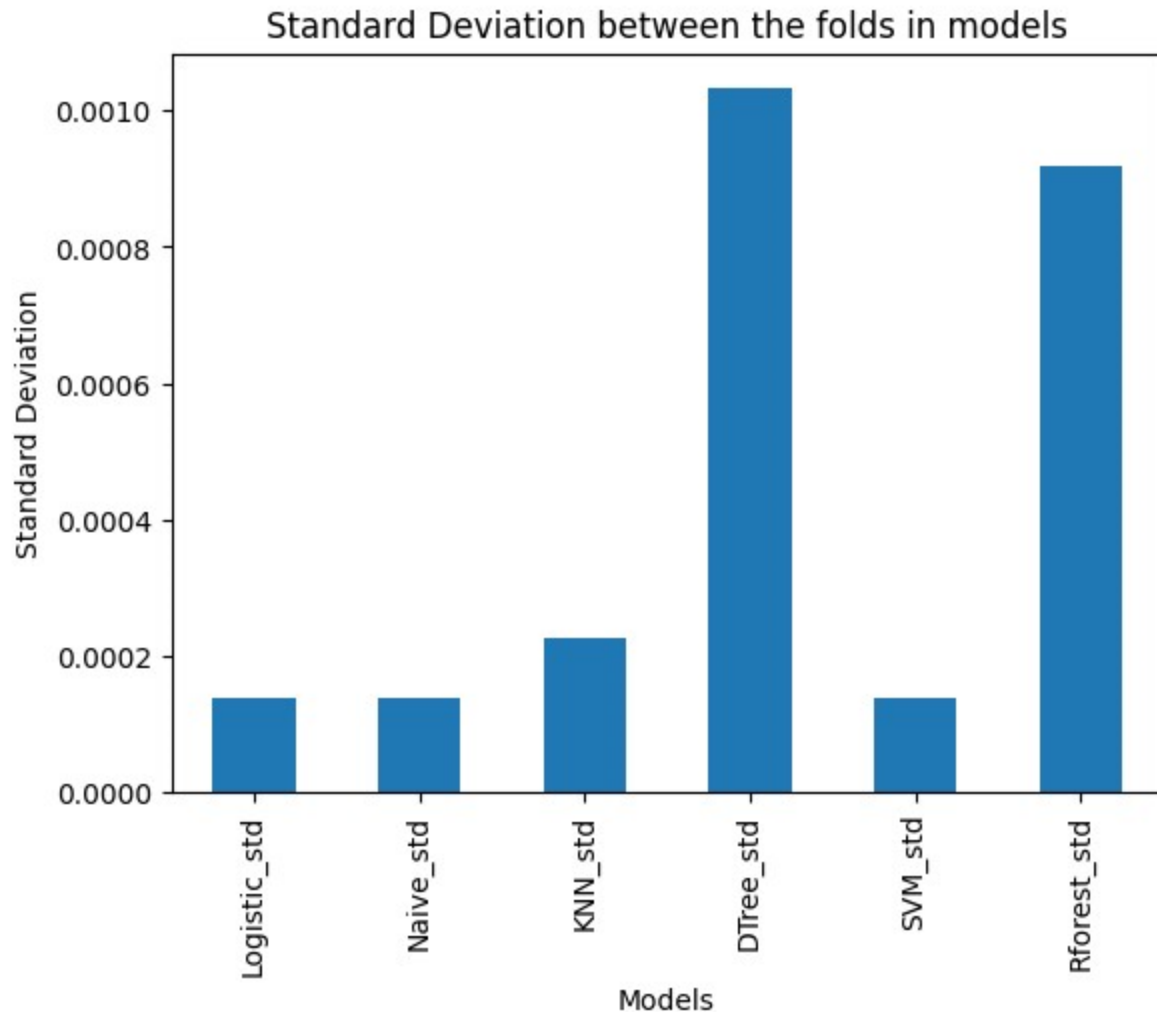


	Standard Deviation
Logistic_std	0.000138
Naive_std	0.000138
KNN_std	0.000226
DTree_std	0.001032
SVM_std	0.000138
Rforest_std	0.000918

Here, we created a DataFrame named `best_model` using pandas. The DataFrame consists of a single column labeled "Standard Deviation" that contains the values from the `std_scores` list. Each value corresponds to the standard deviation of a specific model's performance across different folds. The index of the DataFrame is set using the `score_names` list, which contains the names of the respective models.

```
best_model.plot(kind='bar', legend=None)
plt.xlabel('Models')
plt.ylabel('Standard Deviation')
plt.title('Standard Deviation between the folds in models')
plt.show()
```

Next, we created a bar chart using the `plot` function of the `best_model` DataFrame. The `kind` parameter is set to 'bar', indicating that we want to create a bar chart. Additional labels and title are added to the plot using the `xlabel`, `ylabel`, and `title` functions from `plt`.



Finally, the chart is displayed using `plt.show()`. The resulting bar chart visually represents the standard deviations of the different models' performance across the folds. Each bar represents a model, and the height of the bar corresponds to its standard deviation.

Here, this visualization allows for a quick comparison of the models' performance stability, **with lower bar heights indicating more consistent performance and higher bar heights indicating more variability**. Three models Logistic Regression, Naive Bayes regression and Support Vector Machines has the same LOWEST standard deviation of 0.00013797401688719847 across the folds. Any model out of these three can be chosen. **However, we will be choosing the Logistic regression model because of following reasons:**

- Limited number of features

Logistic regression is suitable in cases of limited features like what we have 3 features namely, marital balance and housing because it is interpretable and provides clear insights into the impact of each feature on the outcome. With a small number of predictors, logistic regression can effectively model the relationship between the predictors and the log-odds of the target variable

- Well-suited for binary classification: Logistic regression is specifically designed for binary classification problems, where the target variable has two classes like in our case of default or no default. It models the probability of the positive class, which can be useful when the goal is to estimate the likelihood of a particular outcome.
- SVM assumes no multicollinearity at all

The support vector matrix assumes no multicollinearity at all between the variables. However, we have observed a slight multicollinearity between the features in our multicollinearity check section. Therefore, it is plausible for us to not choose the SVM model but rather choose a logistic regression model.

Chapter 7

Managerial implications and limitations of the study

7.1 Managerial implications

The dataset here is related to a Portuguese banking institution. Based on the BIVARIATE AND MULTIVARIATE DATA ANALYSIS, the following managerial implications have been decided as a part of evidence based decision making.

i. Increasing the minimum bank balance threshold to above 500 Euros for loan approval

On the Bivariate analysis of Bank balance and default, we can observe that the median balance in case of NON DEFAULTERS is around 500 and that of DEFAULTERS is way below 500. Raising the minimum bank balance threshold to above 500 Euros for loan approval can be a potential strategy to mitigate default risk and improve loan repayment capacity. By ensuring that borrowers have a certain level of financial stability and reserve, the bank can reduce the likelihood of default.

ii. Targeted risk assessment and monitoring strategies for married customers

If we look at the number of defaulters through the adjoining heatmap and stacked bar chart in the bivariate data analysis section, married customers are the highest number of loan defaulters.

For this, one managerial implication could be to implement targeted risk assessment and monitoring strategies for married customers who have a higher likelihood of defaulting on loans. By focusing on this customer segment, the bank can proactively identify potential red flags and early warning signs of financial distress. This may include conducting regular credit checks, monitoring repayment patterns, and offering personalized financial counseling and support to married customers. Additionally, the bank can consider designing customized loan products or repayment plans that align with the specific needs and financial circumstances of married customers, helping them manage their expenses more effectively and reduce the risk of default. Tailored plans for maybe a longer maturity loan because the bank cannot be stringent on providing loans to them because they account for the largest section of customers.

iii. Increased scrutiny and constant monitoring for housemaids above the median age

Through the multivariate analysis, it is discovered that the median age of defaulters is higher than that of non-defaulters. Therefore, it is the elder customers who are defaulting more than that of younger customers. Again, if we layer this prediction to the job of the customers, housemaid is the job category among defaulters who are the oldest.

Based on the finding that elder customers have a higher default rate and among defaulters, housemaids are the oldest job category, it is reasonable to consider putting special scrutiny and constant monitoring for housemaids who are above the median age. These customers may require closer attention and monitoring due to their higher default risk. Implementing enhanced risk assessment measures, regular monitoring of their financial activities, and offering tailored financial support and guidance can help mitigate the risk of default and ensure better financial outcomes for this specific customer segment.

iv. Issue less SUBPRIME loans on the housing loan portfolio

Based on the observation that housing loan takers comprise a larger portion (51%) among defaulters, and considering the potential risks associated with subprime loans as evidenced by past financial crises, issuing fewer subprime loans in the housing loan portfolio can be a good idea.

By reducing exposure to subprime loans, the bank can lower the overall risk of default and potential negative impacts from housing market fluctuations or economic downturns. Instead, focusing on prime loans with stricter eligibility criteria and thorough risk assessment can help maintain a healthier loan portfolio, improve loan repayment rates, and mitigate the risks associated with subprime lending. It is important to strike a balance between providing access to housing loans and managing potential risks to ensure the long-term financial stability of the bank.

v. New product development; senior citizen fixed deposit scheme

Based on the finding that the Retired job category has the second-highest median bank balance and considering their tendency to save up throughout their lives, the bank can implement targeted strategies to attract deposits from retired customers through a senior citizen fixed deposit scheme.

Senior citizen fixed deposits typically offer higher interest rates and flexible tenure options, which can be attractive to retired customers looking for secure and reliable investment opportunities. It provides a safe and stable avenue for retirees to grow their savings while ensuring regular income

through interest payments. This will help to gather a large amount of deposits from the retired individuals and senior citizens.

vi. Targeted Marketing strategies for 3 major identified segments

We conducted a layered histogram on the multivariate analysis to observe the DEMOGRAPHICS OF THE CUSTOMER BASE. It was identified that Customers aged 28 and 43 are present in the highest number in customer base. It seems that the highest number of customers in the customer base are married with just a small portion of customers being single. Therefore, a large majority of customer base is married and below 45. Also, a prominent section of customers is present around the age group of 60s which is the retired group of customers.

Targeting the major customer segments, such as individuals aged between 28 and 43, married customers, and retired people in their 60s, is crucial for the bank's marketing efforts as they represent significant portions of the customer base. Here are some marketing strategies for each segment:

- **Age between 28 and 43:**

This age group represents individuals in their prime earning and spending years. They are likely to have financial goals such as homeownership, starting a family, or investing in their careers.

Marketing strategies:

The bank can focus on offering products and services that cater to their financial needs, such as mortgage loans, education loans, investment opportunities, and personalized financial planning. Implementing digital marketing strategies, social media campaigns, and engaging mobile banking applications can effectively reach this tech-savvy demographic.

- **Married customers:**

Married customers often have shared financial responsibilities, such as household expenses, mortgage payments, and family planning. They may require joint accounts, home loans, and insurance coverage for their families.

Marketing strategies:

The bank can promote joint account offerings, family-oriented financial planning services, and customized insurance packages for married customers. Targeted marketing through **digital marketing mediums** such as direct mail, local community events, and partnerships with wedding planners or family-oriented businesses can help attract and retain this segment.

- **Retired people in their 60s**

Retired individuals typically have accumulated savings and seek financial stability and income generation. They may require retirement planning, investment options, and insurance coverage tailored to their specific needs.

Marketing strategies:

The bank can provide retirement planning seminars, investment advisory services, and specialized savings accounts for seniors. Collaborating with retirement communities, senior centers, and organizations catering to the needs of the elderly can help reach and engage this segment. Offering attractive senior citizen fixed deposit schemes and providing personalized attention through dedicated relationship managers can also be effective strategies.

The managerial implications derived from our data analysis suggest several strategies for the bank. Firstly, implementing a more rigorous loan approval process by raising the minimum bank balance threshold above 500 Euros can help mitigate default risks. Secondly, housemaids above the median age should receive special scrutiny and monitoring due to their higher likelihood of default. Targeting retired customers with attractive schemes like senior citizen fixed deposits and personalized retirement planning services can attract their deposits. For married customers, offering joint account options, family-oriented financial planning, and customized insurance packages can cater to their specific needs. Finally, targeting individuals aged between 28 and 43 through digital marketing strategies and promoting products like mortgage loans and investment opportunities can capture their attention.

7.2 Limitations of the study

While conducting a business analytics project, it is important to acknowledge the limitations that may affect the reliability and generalizability of the findings. Some limitations of the banking loan default project may include:

- **Probable Sample Bias**

The dataset used for analysis may not represent the entire population accurately. It is possible that certain demographics or customer segments are underrepresented or overrepresented, leading to biased results.

In the context of a banking loan default dataset, sample bias could arise if the dataset primarily consists of a specific demographic group or customer segment. For example, if the dataset predominantly includes customers from a particular region, age group, or income bracket, it may not accurately represent the diversity of the overall population. This could lead to biased results, as the findings derived from the dataset may not be applicable or representative of the entire customer base or target population of the bank.

- **Data Quality**

The accuracy and completeness of the data could impact the validity of analysis. There may be missing values, inconsistencies, or errors in the dataset that could introduce biases or affect the reliability of the findings.

In the context of a banking loan default dataset, data quality is crucial for ensuring the validity of the analysis. Missing values, inconsistencies, or errors in the dataset can introduce biases and affect the reliability of the findings. For example, the important variables such as customer bank balance had missing values, although it has been removed, it impacts the accuracy of assessing default risk.

- **Exogenous Factors**

The analysis may not have considered external factors that could influence the observed patterns. Economic conditions, regulatory changes, or other market dynamics could impact customer behavior and default rates, which may not be captured in the dataset.

When analyzing a banking loan default dataset, it is important to consider external factors that can influence the observed patterns. Economic conditions, regulatory changes, or other market dynamics can significantly impact customer behavior and default rates. For instance, during an economic recession, default rates may increase due to widespread financial instability. These external factors may not be captured in the dataset used for analysis, leading to an incomplete understanding of the drivers of loan default.

References

Codebasics. (2017). Machine Learning Tutorial Python.

https://www.youtube.com/watch?v=FB5EdxAGxQg&ab_channel=codebasics.

Pandian, S. (2017, February 17). *Analytics Vidya* . Retrieved from

<https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/>

YAMAHATA, H. (2017). *Kaggle* . Retrieved from Kaggle.com :

<https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing>