FACULTY OF ENGINEERING, ENVIRONMENT & COMPUTING

Dataset Analysis and Visualization Using Big Data Programs

BANK MARKETING ANALYTICS

USING PYSPARK

(CLASSIFICATION OF CLIENT'S TERM DEPOSIT BEHAVIOUR)

Module-7153CEM – M138CEM

Module leader-Dr Marwan Fuad

Student Id-12637924

Email-ashokana2@uni.coventry.ac.uk

ABSTRACT

Bank marketing is the process by which companies create strong relationship with customers and harness it for developing business. With Marketing analysis and its improvement, the future of any Business will be a success. Especially in case of Bank, its profit can be improved only by inviting more Term deposit. Term deposit is a kind of deposit account in which the sum of money deposited will be locked up in bank for a short-term maturity period at a fixed interest. It adds value to Bank's Business. Here In this project, I aim to use a bank marketing campaign dataset to use Bigdata analytics and infer trends in client's behaviour on opening a Term deposit based based on campaign results. Exploratory Data Analysis is done using Tableau. Spark SQL used for data analysis and pre-processing. Classification of clients who would say Yes or no to bank product (term deposit) is done through Spark ML machine learning library. I have implemented Supervised learning through Classification using different methods (Logistic Regression, Naïve Bayes and Decision Tree Classifier)for comparing the accuracy of results.

Keywords: Bank marketing, Term deposit, Exploratory Data Analysis, PySpark, Spark ML, Supervised learning Classification.

1. INTRODUCTION

This projects aims to implement Big data analytics on the bank marketing dataset available in Kaggle public library to infer insights from the data on the behaviour of clients based on a marketing campaign results.

Dataset: Bank Marketing
https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing

Figure 1
Dataset Description

Attribute Information: -

Input variables:

Bank client data:

1.age (numeric)

2.job: type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3.marital: marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4.education(categorical:'basic.4y','basic.6y','basic.9y','high.school','illiterate',' professional.course','university.degree','unknown')

5.default: has credit in default? (Categorical: 'no','yes','unknown')

6.housing: has housing loan? (Categorical: 'no','yes','unknown')

7.loan: has personal loan? (Categorical: 'no','yes','unknown')

8.contact: contact communication type (categorical: 'cellular','telephone')

9.month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10.day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

.

11.duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

12. other attributes:12-campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13.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)

14.previous: number of contacts performed before this campaign and for this client (numeric)

15.poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Social and economic context attributes

16.emp.var.rate: employment variation rate - quarterly indicator (numeric)

17.cons.price.idx: consumer price index - monthly indicator (numeric)

18.cons.conf.idx: consumer confidence index - monthly indicator (numeric)

19.euribor3m: euribor 3 month rate - daily indicator (numeric)

20.nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

21.deposit- has the client subscribed a term deposit? (Binary: 'yes','no')

From Bank Marketing Campaign (2019)

Software:-The project involves application of Apache Spark version 3.0.0 integrated on top of Hadoop 3.2, specifically PySpark .OS used is Windows.
Visualization Software:-Tableau

Data Analysis tasks :-

1. PYSPARK INSTALLATION AND SETUP
2. CHECKING PYSPARK SETUP
3. LOADING DATATASET AND DERIVE INFORMATION (DROPPING IRRELEVANT FEATURES)
4. PRE-PROCESSING OF DATASET
    a. IDENTIFYING DUPLICATES
    b. IDENTIFYING NULL VALUES
    c. REMOVAL OF "UNKNOWN" VALUES
    d. RENAMING LABEL COLUMN ("deposit" to "Subscribed")
    e. INDEXING AND ENCODING CATEGORICAL VARIABLES
    f. NORMALISATION OF ENCODED COLUMNS
5. EXPLORATORY DATA ANALYSIS -DATA VISUALIZATION
6. SPLITTING DATA INTO TRAINING AND TESTING SETS
7. MACHINE LEARNING MODEL BUILDING & EVALUATION
    1. LOGISTIC REGRESSION
    2. DECISION TREE CLASSIFIER
    3. NAIVE BAYES CLASSIFIER
8. MODEL EVALUATION

2. RELATED WORK

Apache Spark

Spark SQL and Spark ML packages are used in this project.
Unique features of Apache Spark

- Speed-Run workloads 100 times faster
- Easy to use-Compatible with several languages Java,Scala,Python,R
- Generality- Combine SQL, streaming, and complex analytics.
- Runs everywhere-Runs on Hadoop, Kubernetes, standalone or in cloud. (Dushanthi Manthushika,2021)

Tableau

Tableau helps an organization to understand its data and learn trends and recommend solutions. It’s an intelligence tool which resolves the headache of data analysis and visualisation without any coding. It’s a replacement for all the visualization toolkits. In addition to graphs, charts and diagrams, tableau provides tools for data exploration and modelling.

Automatic Data cleaning: It can explain data using built in intelligence and perform data pre-processing (remove outliers, null values and duplicates) That is without any unnecessary coding ,It import dataset and pre-process.
Filters: In my dataset, the issue of” unknown” was recovered using Filters in Tableau .

Figure 2

Filter” Unknown “in Poutcome

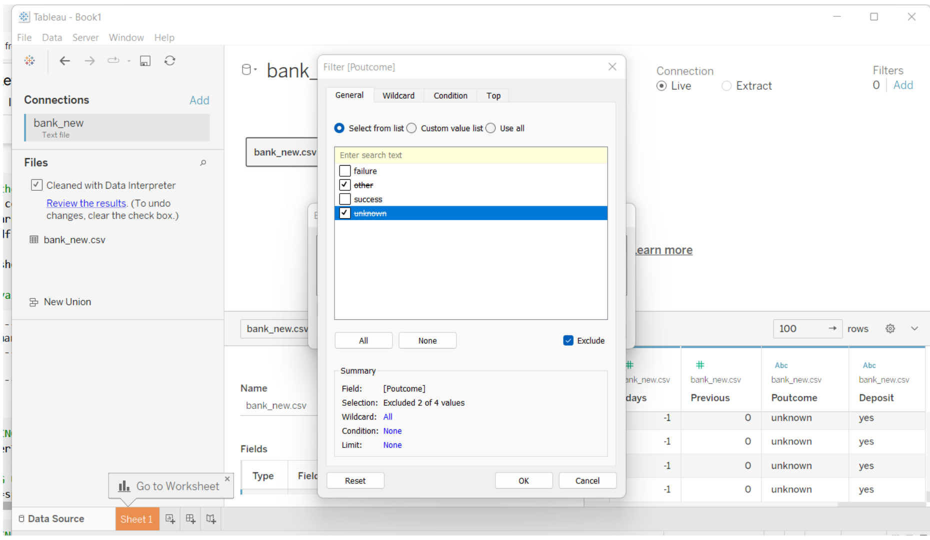


Figure 3

Filter” Unknown “ in Job

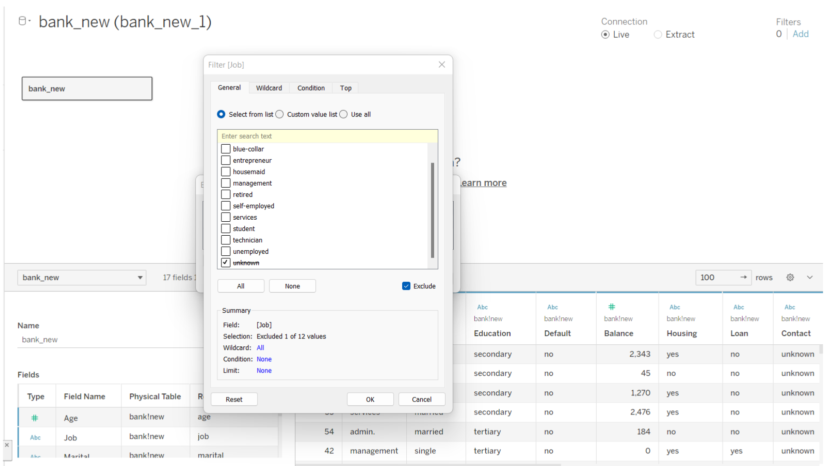
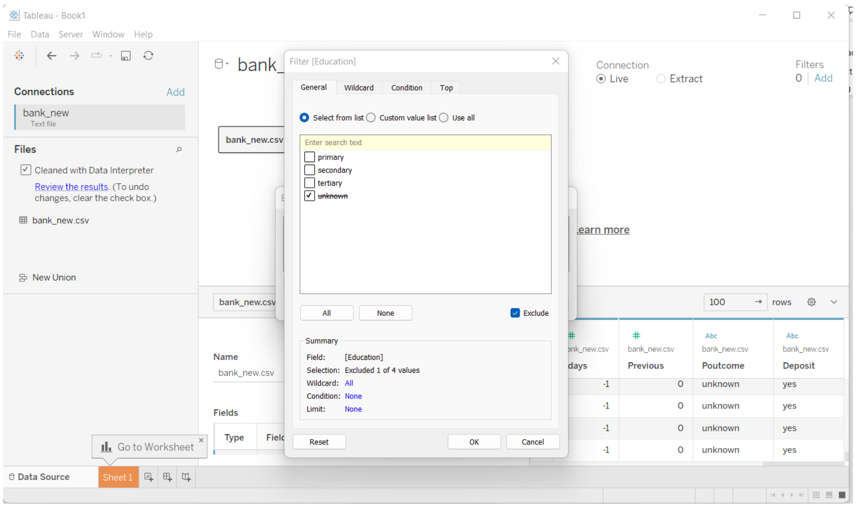


Figure 4

Filter" Unknown "in Education



Calculated Fields : Conditional logic is applied to certain string fields to make the tableau know meaning of data.

Figure 5

Calculated field

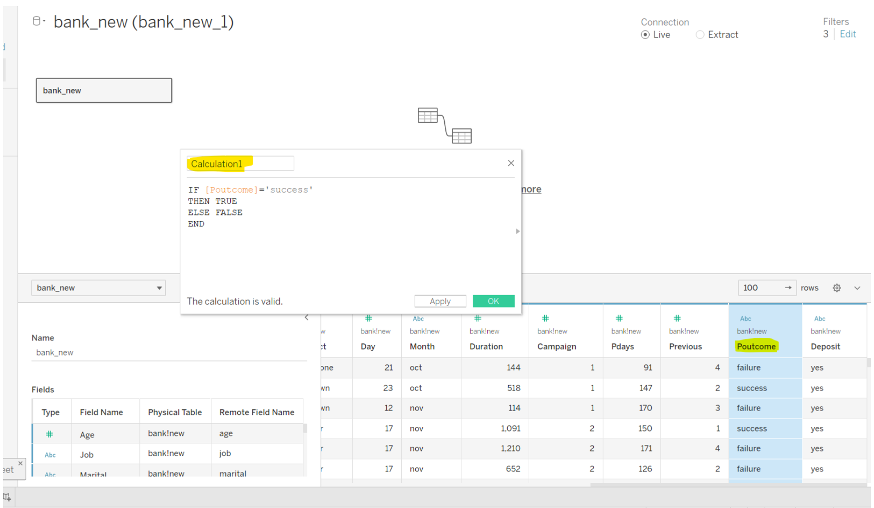
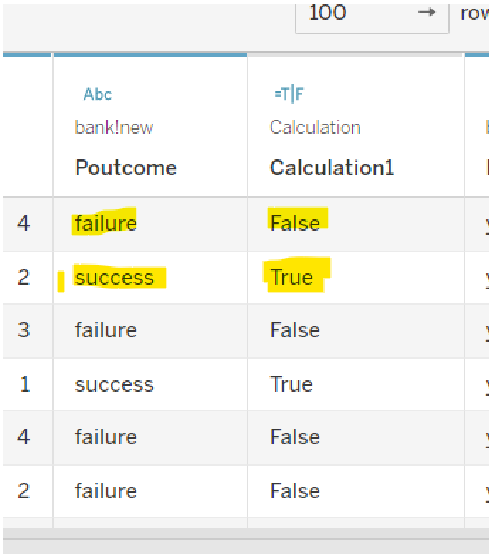


Figure 6

Results



	Abc bank!new Poutcome	=T|F Calculation Calculation1
4	failure	False
2	success	True
3	failure	False
1	success	True
4	failure	False
2	failure	False

3. IMPLEMENTATION

3.1 PYSPARK INSTALLATION AND SETUP

Apache spark is a high-speed computing framework used for data processing on large scale datasets. Colab is a Jupyter notebook service hosted by Google that requires no setup to use. When it comes to using Apache Spark on local machines, it is far easier to setup in google colab.(Walker Rowe,2019)

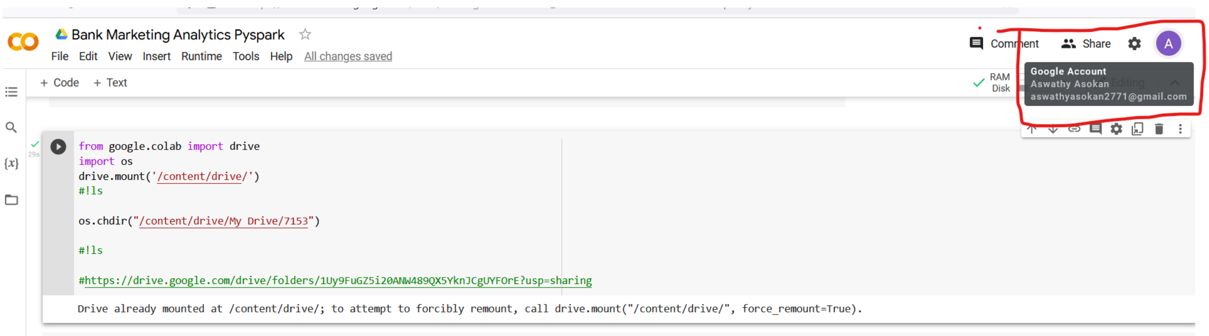
1.Connecting Drive to Colab

I have initially mounted the google drive so that it helps to access the files inside Gdrive directly in cola notebook. I have saved the data files and pyspark notebook in 7153 folder in MyDrive folder .

The evidence of my userid is given in the below screenshot and in the following steps that I installed everything on my own and ran my own code.

Figure 7

Mounting google drive



2.Installing wget

wget download files across browsers using input URL .

Figure 8

installing wget

Bank marketing Analytics Pyspark
File Edit View Insert Runtime Tools Help All changes saved

```
[1] !pip install wget

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting wget
  Downloading wget-3.2.zip (10 kB)
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9675 sha256=ff97b0807454caced128cc6c042618664099bfc7820b15baf90407db92a4a574
  Stored in directory: /root/.cache/pip/wheels/a1/b6/7c/0e63e34eb06634181c63adacca38b79ff8f35c37e3c13e3c02
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

3.Installing Java and Checking the installation

Figure 9

installing java

```
[8] !apt-get install openjdk-8-jdk-headless -qq

Selecting previously unselected package openjdk-8-jre-headless:amd64.
(Reading database ... 123934 files and directories currently installed.)
Preparing to unpack .../openjdk-8-jre-headless_8u342-b07-0ubuntu1~18.04_amd64.deb ...
Unpacking openjdk-8-jre-headless:amd64 (8u342-b07-0ubuntu1~18.04) ...
Selecting previously unselected package openjdk-8-jdk-headless:amd64.
Preparing to unpack .../openjdk-8-jdk-headless_8u342-b07-0ubuntu1~18.04_amd64.deb ...
Unpacking openjdk-8-jdk-headless:amd64 (8u342-b07-0ubuntu1~18.04) ...
Setting up openjdk-8-jre-headless:amd64 (8u342-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/orbd to provide /usr/bin/orbd (orbd) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/servertool to provide /usr/bin/servertool (servertool) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/tnameserv to provide /usr/bin/tnameserv (tnameserv) in auto mode
Setting up openjdk-8-jdk-headless:amd64 (8u342-b07-0ubuntu1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/idlj to provide /usr/bin/idlj (idlj) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsimport to provide /usr/bin/wsimport (wsimport) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jsadebugd to provide /usr/bin/jsadebugd (jsadebugd) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/native2ascii to provide /usr/bin/native2ascii (native2ascii) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/javah to provide /usr/bin/javah (javah) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/clhsdb to provide /usr/bin/clhsdb (clhsdb) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/xjc to provide /usr/bin/xjc (xjc) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/hsdb to provide /usr/bin/hsdb (hsdb) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/schemagen to provide /usr/bin/schemagen (schemagen) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/extcheck to provide /usr/bin/extcheck (extcheck) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jhat to provide /usr/bin/jhat (jhat) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsgen to provide /usr/bin/wsgen (wsgen) in auto mode
```

Figure 10

checking java version

```
#checking the existing installed java version
!java -version

openjdk version "11.0.16" 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu118.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu118.04, mixed mode, sharing)
```

4.Install apache spark version 3.0.0 on top of hadoop 3.2using browser URL using wget command

Figure 11

Install ApacheSpark on Hadoop

```
#installing Apache spark 3.0.0 on top of Hadoop 3.2
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.0-bin-hadoop3.2.tgz
```

5.The zip folder of apache spark will be in the downloads folder.It is moved to 7153 folder of my Gdrive .To decompress the file following tar command is used as below.

Figure 12

Unzip the spark folder

```
#decompressing the zipped file in current directory in gdrive
!tar xf spark-3.0.0-bin-hadoop3.2.tgz
```

6.The current working directory.

Figure 13

Current folder

```
[ ] ls

bank-additional-full.csv                       master.csv
bank.csv                                       spark-3.0.0-bin-hadoop3.2/
bank-full.csv                                  spark-3.0.0-bin-hadoop3.2.tgz
'Bank Marketing Analytics Pyspark.ipynb'       spark-3.0.0-bin-hadoop3.2.tgz.1
bank_new.csv                                   spark-3.0.0-bin-hadoop3.2.tgz.2
```

7.Setting the environment variable path for system variables

Figure 14

setting environment variable path

```
#setting the environment variables for spark and java
import os
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"]="/content/spark-3.0.0/spark-3.0.0-bin-hadoop3.2"
```

8.Initialising Pyspark

Installation of findspark to address the issue of Pyspark setup

PySpark being not available in sys.path by default, It should add sys.path at runtime using findspark . (Soumya Goyal,2022)

Figure 15

Install findspark

```
[12] #installing findspark
     !pip install -q findspark
```

If I call the findpark.init() now.it will throw not found error Because it is not imported yet.
So pyspark is installed first.

9.Installing pyspark matching with the version of spark for avoiding compatibility issues.

Figure 16

Installing pyspark

```
#installing the matching pyspark version
!pip install pyspark==3.0.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark==3.0.0
  Downloading pyspark-3.0.0.tar.gz (204.7 MB)
     |████████████████████████████████| 204.7 MB 20 kB/s
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 19.4 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.0.0-py2.py3-none-any.whl size=205044182 sha256=2c564a68cedd4d062621aac549a8f16353dca5e70ad82f1da3c0be71edfa3c4a
  Stored in directory: /root/.cache/pip/wheels/4e/c5/36/aef1bb711963a619063119cc032176106827a129c0be20e301
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.0.0
```

10.Import pyspark

To avoid the "PYSPARK NOT DEFINED"error while checking the version ,import pyspark.

Figure 17

pyspark import

```
import pyspark
```

Figure 18

version check

```
#checking pyspark version
print(pyspark.__version__)
```

```
3.0.0
```

11.Initialising pyspark after setup

Finspark.init() using the SPARK_HOME environment variable parameter .

Figure 19

Initialising pyspark

```
import findspark
findspark.init("spark-3.0.0-bin-hadoop3.2")#adding pyspark to sys.path at run time
```

12.Initialise pyspark session

Initializing spark session before coding. Spark session is an entry point to underlying pyspark functionality to programmatically create pyspark RDD and Dataframe.(Walker Rowe,2019). Spark session is to interact with Spark's numerous features in spark shell. Here in pyspark, spark session needs to be created to programmatically call spark features.

13.Testing pyspark session by creating a dummy dataframe

Testing the pyspark installation, initially Spark session and Spark Conf,Spark Context are imported and a session object spark is created. And using that spark object, a data frame is created .

Figure 20

Create dataframe

Testing Pyspark Setup and installation

```
[41] from pyspark.sql import SparkSession
     from pyspark import SparkConf,SparkContext

[42] spark = SparkSession.builder.master("local").appName("Search").config(conf=SparkConf()).getOrCreate()

#creating a dataframe with columnames and value
df=spark.createDataFrame([{"language,usercount" :("java,2000")}])

/usr/local/lib/python3.7/dist-packages/pyspark/sql/session.py:378: UserWarning: inferring schema from dict is deprecated,please use pyspark.sql.Row instead
  warnings.warn("inferring schema from dict is deprecated,"

df.show(1)

+------------------+
|language,usercount|
+------------------+
|         java,2000|
+------------------+
```

3.2 LOADING DATASET AND DERIVE INFORMATION

The dataset has 20 attributes which includes client information and Bank's previous campaign results. The Importing CSV file and it is stored as a spark dataframe .

Figure 21

Import Dataset

Loading Datset and deriving information

```
[ ] spark = SparkSession.builder.appName('Bank Marketing Analytics').getOrCreate()

[ ] #IMPORTING  DATASET AS A SPARK DATA FRAME
    df = spark.read.csv('bank_new.csv', header = True, inferSchema = True)

[ ] type(df)

pyspark.sql.dataframe.DataFrame
```

The data set **initially had 20 attributes** .I have omitted the Social and economic context attributes being irrelevant and taken for data analysis. Hence now dataset have following **17 attributes**

Figure 22

Dataframe

```
[ ] df.show()

+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
|age|       job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|deposit|
+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
| 59|    admin.| married|secondary|     no|   2343|    yes|  no|unknown|  5|  may|    1042|       1|   -1|       0| unknown|    yes|
| 56|    admin.| married|secondary|     no|     45|     no|  no|unknown|  5|  may|    1467|       1|   -1|       0| unknown|    yes|
| 41|technician| married|secondary|     no|   1270|    yes|  no|unknown|  5|  may|    1389|       1|   -1|       0| unknown|    yes|
| 55|  services| married|secondary|     no|   2476|    yes|  no|unknown|  5|  may|     579|       1|   -1|       0| unknown|    yes|
| 54|    admin.| married| tertiary|     no|    184|     no|  no|unknown|  5|  may|     673|       2|   -1|       0| unknown|    yes|
| 42|management|  single| tertiary|     no|      0|    yes| yes|unknown|  5|  may|     562|       2|   -1|       0| unknown|    yes|
| 56|management| married| tertiary|     no|    830|    yes| yes|unknown|  6|  may|    1201|       1|   -1|       0| unknown|    yes|
| 60|   retired|divorced|secondary|     no|    545|    yes|  no|unknown|  6|  may|    1030|       1|   -1|       0| unknown|    yes|
| 37|technician| married|secondary|     no|      1|    yes|  no|unknown|  6|  may|     608|       1|   -1|       0| unknown|    yes|
| 28|  services|  single|secondary|     no|   5090|    yes|  no|unknown|  6|  may|    1297|       3|   -1|       0| unknown|    yes|
| 38|    admin.|  single|secondary|     no|    100|    yes|  no|unknown|  7|  may|     786|       1|   -1|       0| unknown|    yes|
| 30|blue-collar| married|secondary|     no|    309|    yes|  no|unknown|  7|  may|    1574|       2|   -1|       0| unknown|    yes|
| 29|management| married| tertiary|     no|    199|    yes| yes|unknown|  7|  may|    1689|       4|   -1|       0| unknown|    yes|
| 46|blue-collar|  single| tertiary|     no|    460|    yes|  no|unknown|  7|  may|    1102|       2|   -1|       0| unknown|    yes|
| 31|technician|  single| tertiary|     no|    703|    yes|  no|unknown|  8|  may|     943|       2|   -1|       0| unknown|    yes|
| 35|management|divorced| tertiary|     no|   3837|    yes|  no|unknown|  8|  may|    1084|       1|   -1|       0| unknown|    yes|
| 32|blue-collar|  single|  primary|     no|    611|    yes|  no|unknown|  8|  may|     541|       3|   -1|       0| unknown|    yes|
| 49|  services| married|secondary|     no|     -8|    yes|  no|unknown|  8|  may|    1119|       1|   -1|       0| unknown|    yes|
| 41|    admin.| married|secondary|     no|     55|    yes|  no|unknown|  8|  may|    1120|       2|   -1|       0| unknown|    yes|
| 49|    admin.|divorced|secondary|     no|    168|    yes| yes|unknown|  8|  may|     513|       1|   -1|       0| unknown|    yes|
+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
only showing top 20 rows
```

The **datatypes** included are **Integer,string and categories**

Figure 23

Structure of Dataframe

```
[ ] df.printSchema()

root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

Figure 24

Categorical columns.

```
[ ] #SELECTING CATEGORICAL COLUMNS ONLY
    categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'poutcome']
```

The dataset composed two kinds of attributes. Numerical and nominal .
Numerical :-age,balance,duration,day,campaign,pdays,previous
Categorical:-job,marital,education,contact,month ,poutcome
Binary :- default,housing,loan,Subscribed.

It has 11162 rows and 17 columns .

Figure 25

Shape of Dataset

```
[ ]  print(f"dimension of dataframe is {(row,column)}")
     print(f"number of rows are {row}")
     print(f"number of columns are {column}")

     dimension of dataframe is (11162, 17)
     number of rows are 11162
     number of columns are 17
```

3.3 PRE-PROCESSING OF DATASET

a) Identifying duplicates

There are **no duplicates** in the dataset

Figure 26

Duplicate count of Dataset

```
[79]  #1.Finding the duplicates if any .Here returns same no of records so  no duplicates in dataframe
      #df.dropDuplicates().count()
      df.distinct().count()

      11162

      df.count()

      11162
```

b) Identifying null values

There are **no null or nan values** in the dataset

Figure 27

null count of Dataset

```
#2.finding the occurence of null values in all columns in dataframe
df_col=df.columns
from pyspark.sql.functions import col,isnan,when,count
null_col=df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_col]
   )
null_col.show()

#no null values occured

+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
|age|job|marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|deposit|
+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
|  0|  0|      0|        0|      0|      0|      0|   0|      0|  0|    0|       0|       0|    0|       0|       0|      0|
+---+---+-------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
```

c) Removing Unknown column values.

Unknown values wrongly interpret the data here.Unknown value are majority category for the categorical columns in data summary .

Figure 28

Wrong data summary

```
#SUMMARISING EACH COLUMN VALUES
df.summary().show()
```

summary	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign
count	11162	11162	11162	11162	11162	11162	11162	11162	11162	11162	11162	11162	11162
mean	41.231947679627304	null	null	null	null	1528.5385235620856	null	null	null	15.658036194230425	null	371.99381831213043	2.508421429851281
stddev	11.913369192215518	null	null	null	null	3225.413325946149	null	null	null	8.420739541006462	null	347.12838571630687	2.7220771816614824
min	18	admin.	divorced	primary	no	-6847	no	no	cellular	1	apr	2	1
25%	32	null	null	null	null	122	null	null	null	8	null	138	1
50%	39	null	null	null	null	550	null	null	null	15	null	255	2
75%	49	null	null	null	null	1708	null	null	null	22	null	496	3
max	95	unknown	single	unknown	yes	81204	yes	yes	unknown	31	sep	3881	63

campaign	pdays	previous	poutcome	deposit
11162	11162	11162	11162	11162
2.508421429851281	51.33040673714388	0.8325568894463358	null	null
2.7220771816614824	108.75828197197717	2.292007218670508	null	null
1	-1	0	failure	no
1	-1	0	null	null
2	-1	0	null	null
3	20	1	null	null
63	854	58	unknown	yes

The "job","Education" and "poutcome "column has unknown values . It has to be removed.

Figure 29

Unknown values

```
df.select('job').distinct().collect()
```

```
[Row(job='management'),
 Row(job='retired'),
 Row(job='unknown'),
 Row(job='self-employed'),
 Row(job='student'),
 Row(job='blue-collar'),
 Row(job='entrepreneur'),
 Row(job='admin.'),
 Row(job='technician'),
 Row(job='services'),
 Row(job='housemaid'),
 Row(job='unemployed')]
```

```
df.select('education').distinct().collect()
```

```
[Row(education='unknown'),
 Row(education='tertiary'),
 Row(education='secondary'),
 Row(education='primary')]
```

The “poutcome “column has “other “value as well. It is also irrelevant and affects data analysis. Hence it also should be removed.

```
df.select('poutcome').distinct().collect()
```

```
[Row(poutcome='success'),
 Row(poutcome='unknown'),
 Row(poutcome='other'),
 Row(poutcome='failure')]
```

Although “contact “ has unknown value.It doesn’t have any impact on data analysis .Hence it is kept as such.

Figure 30

Unknown values

```
df.select('contact').distinct().collect()
```

```
[Row(contact='unknown'), Row(contact='cellular'), Row(contact='telephone')]
```

unknown values are removed using spark sql “AND “ “OR” logic functions integrated in sql queries.

Figure 31

Displaying df2 dataframe after removal of unknown values

```
[25] #Creating a temporary Table named "bank" for filtering the columns
     df.registerTempTable("bank")

     #Filtering unknown values from all the columns using  "AND" "OR" in sparksql
     #with this query ,The "other"attrinbute in poutcome also gets removed as it is not valid for data analysis
     sqlfilter=spark.sql("SELECT * FROM bank WHERE job!='unknown' AND education!='unknown' AND marital!='unknown' AND loan!='unknown' AND (poutcome =='failure' OR poutcome == 'success')"
```

```
[26] #Storing in new variable
     df2=sqlfilter
```

```
[ ] #Displaying new dataframe
    df2.show()
```

```
+---+------------+--------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+-------+
|age|         job| marital|education|default|balance|housing|loan|  contact|day|month|duration|campaign|pdays|previous|poutcome|deposit|
+---+------------+--------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+-------+
| 33|    services| married|secondary|     no|   3444|    yes|  no|telephone| 21|  oct|     144|       1|   91|       4| failure|    yes|
| 56|  technician| married|secondary|     no|    589|    yes|  no|  unknown| 23|  oct|     518|       1|  147|       2| success|    yes|
| 34|      admin.| married| tertiary|     no|    899|    yes|  no|  unknown| 12|  nov|     114|       1|  170|       3| failure|    yes|
| 53|     retired| married| tertiary|     no|   2269|     no|  no| cellular| 17|  nov|    1091|       2|  150|       1| success|    yes|
| 37|  technician| married|secondary|     no|   5115|    yes|  no| cellular| 17|  nov|    1210|       2|  171|       4| failure|    yes|
| 45|entrepreneur| married|secondary|     no|    781|     no| yes| cellular| 17|  nov|     652|       2|  126|       2| failure|    yes|
| 46|  unemployed|divorced|secondary|     no|   3354|    yes|  no| cellular| 19|  nov|     522|       1|  174|       1| success|    yes|
| 40|  management| married| tertiary|     no|   3352|    yes|  no| cellular| 19|  nov|     639|       2|   27|       1| success|    yes|
```

Figure 32

Count of records after removal of unknown

```
#no of records after removal of unknown values
df2.count()
```

```
2181
```

When summarising new dataframe ,we get accurate mean ,min and max values.Not affected by unknown values.

Figure 33

Summary of new dataframe

summary	age	job	marital	education	default	balance	housing	loan	contact	day	month
count	2181	2181	2181	2181	2181	2181	2181	2181	2181	2181	2181
mean	41.84364970197157	null	null	null	null	1742.946813388354	null	null	null	14.204034846400734	null
stddev	12.855329179952637	null	null	null	null	3397.7939950723485	null	null	null	8.10108738010334	null
min	18	admin.	divorced	primary	no	-938	no	no	cellular	1	apr
25%	32	null	null	null	null	224	null	null	null	8	null
50%	38	null	null	null	null	719	null	null	null	13	null
75%	50	null	null	null	null	2044	null	null	null	20	null
max	88	unemployed	single	tertiary	yes	81204	yes	yes	unknown	31	sep

summary	campaign	pdays	previous	poutcome	deposit
count	2181	2181	2181	2181	2181
mean	1.8211829436038514	202.8578633654287	3.0917010545621273	null	null
stddev	1.2274126864078023	121.3097255486155	2.958246116660583	null	null
min	1	1	1	failure	no
25%	1	97	1	null	null
50%	1	182	2	null	null
75%	2	278	4	null	null
max	12	854	55	success	yes

d) Renaming label column (deposit to Subscribed)for readability

Using **withColumnRenamed()** method “deposit “ renamed as”Subscribed”. So that it makes sense whether a customer has subscribed term deposit or not .

Figure 34

Deposit To Subscribed

```
rename=df2.withColumnRenamed("deposit","Subscribed")
```

```
[46] df_new=rename
```

```
[47] df_new.count()

2181
```

```
[48] df_new.show(5)

+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
|age|       job|marital|education|default|balance|housing|loan|  contact|day|month|duration|campaign|pdays|previous|poutcome|Subscribed|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
| 33|  services|married|secondary|     no|   3444|    yes|  no|telephone| 21|  oct|     144|       1|   91|       4| failure|       yes|
| 56|technician|married|secondary|     no|    589|    yes|  no|  unknown| 23|  oct|     518|       1|  147|       2| success|       yes|
| 34|    admin.|married| tertiary|     no|    899|    yes|  no|  unknown| 12|  nov|     114|       1|  170|       3| failure|       yes|
| 53|   retired|married| tertiary|     no|   2269|     no|  no| cellular| 17|  nov|    1091|       2|  150|       1| success|       yes|
| 37|technician|married|secondary|     no|   5115|    yes|  no| cellular| 17|  nov|    1210|       2|  171|       4| failure|       yes|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
only showing top 5 rows
```

e) Encoding and vectorisation of categorical columns

The data must be converted to a vector and before that the features have to be encoded. (Walker Rowe,2019).First ,I have selected the categorical columns and created an empty stage list for the pipeline model.

Figure 35

Selected categorical columns

```
[39] #Filtering categorical columns
     categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loa

     #Creating an empty list for pipeline and assembler
     list_stages = []
```

Then from the spark ML.feature library imported OneHotEncoder ,StringIndexer and VectorAssembler methods . Stringrelated features are indexed using StringIndexer() and Encoded using OneHotEncoder(), then are assembled together with Numeric columns using VectorAssembler().(Walker Rowe,2019)

Figure 36

Indexing, encoding and vectorising of categorical columns

```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
```

```
[43] #Using FOR LOOP for indexing and encoding all selected categorical columns
     #STRING INDEXER  index all the columns and store in a new column with +INDEXED
     #ONE HOT ENCODER encode all the indexed columns and store in a new column with +ENCODED
     for i in categoricalColumns:
         stringIndexer = StringIndexer(inputCol = i, outputCol = i + '_indexed')
         encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[i+ "_encoded"])
         list_stages += [stringIndexer, encoder]

     #Indexing predictor column 'Subscribed' as label and features
     label_index= StringIndexer(inputCol = 'Subscribed', outputCol = 'label')

     #Creating stages for both numeric and categorical columns
     list_stages += [label_index]
     numericColumns = ['age', 'balance', 'campaign', 'pdays', 'previous']

     #Adding both to assembler
     input_assembler = [c + "_encoded" for c in categoricalColumns] + numericColumns

     #vectorizing to create new features column with indexed and encoded values.
     assembler = VectorAssembler(inputCols=input_assembler, outputCol="features")
     list_stages += [assembler]
```

Now a pipeline model is built by combining all the pipeline stages and it is fitted into the new dataframe df_new. The df_new is stored into another variable df4 to avoid nonetype errors.

Figure 37

Fitting pipeline model

```
[44] from pyspark.ml import Pipeline
```

```
#combining all pipeline stages
pipeline = Pipeline(stages = list_stages)
#fitting the model
pipelineModel = pipeline.fit(df_new)
#transforming the model
df_new= pipelineModel.transform(df_new)
```

```
[49] #Storing in new variable to avoid none type error
     df4=df_new
```

The encoded categorical features are shown below.

Figure 38

Encoded columns and labelled features

```
df4.show(5)
```

```
-+------------+--------------+----------------+----------------+------------------+------------------+----------------+----------------+----------------+----------------+--
d|job_indexed|   job_encoded|marital_indexed| marital_encoded|education_indexed| education_encoded|default_indexed| default_encoded|housing_indexed| housing_encoded|lo
-+------------+--------------+----------------+----------------+------------------+------------------+----------------+----------------+----------------+----------------+--
5|         5.0|(10,[5],[1.0])|             0.0|   (2,[0],[1.0])|               0.0|     (2,[0],[1.0])|             0.0|   (1,[0],[1.0])|             1.0|       (1,[],[])|
5|         5.0|(10,[5],[1.0])|             0.0|   (2,[0],[1.0])|               0.0|     (2,[0],[1.0])|             0.0|   (1,[0],[1.0])|             1.0|       (1,[],[])|
5|         5.0|(10,[5],[1.0])|             0.0|   (2,[0],[1.0])|               0.0|     (2,[0],[1.0])|             0.0|   (1,[0],[1.0])|             1.0|       (1,[],[])|
5|         5.0|(10,[5],[1.0])|             0.0|   (2,[0],[1.0])|               0.0|     (2,[0],[1.0])|             0.0|   (1,[0],[1.0])|             1.0|       (1,[],[])|
5|         5.0|(10,[5],[1.0])|             0.0|   (2,[0],[1.0])|               0.0|     (2,[0],[1.0])|             0.0|   (1,[0],[1.0])|             1.0|       (1,[],[])|
-+------------+--------------+----------------+----------------+------------------+------------------+----------------+----------------+----------------+----------------+--
```

```
-----+----------------+----------------+-----+--------------------+
icoded|poutcome_indexed|poutcome_encoded|label|            features|
-----+----------------+----------------+-----+--------------------+
1.0])|             0.0|   (1,[0],[1.0])|  0.0|(23,[5,10,12,14,1...|
1.0])|             0.0|   (1,[0],[1.0])|  0.0|(23,[5,10,12,14,1...|
1.0])|             0.0|   (1,[0],[1.0])|  0.0|(23,[5,10,12,14,1...|
1.0])|             0.0|   (1,[0],[1.0])|  0.0|(23,[5,10,12,14,1...|
1.0])|             0.0|   (1,[0],[1.0])|  0.0|(23,[5,10,12,14,1...|
-----+----------------+----------------+-----+--------------------+
```

f) MIN -MAX Normalisation of Encoded columns

As I prepare the data for machine learning predictor analytics, the data has to be scaled .Min -Max scaler rescale each feature individually .As data is having different range of values here, The numerical values (encoded values here) are scaled using Min Max scaler(in range 0 to 1).

Figure 39
Min-Max scaling

```
[51] #Scaling of Data
     #Only scaling the encoded columns as they are having different range of values
     from pyspark.ml.feature import MinMaxScaler
     encoded_vars=['features','job_encoded','marital_encoded','loan_encoded','default_encoded','education_encoded','housing_encoded','poutc

     #Min max scaling to scale down between 0 and 1
     minmaxscaler = [MinMaxScaler(inputCol=scale_features ,outputCol=scale_features+ "_SCALED") for scale_features in encoded_vars]

     #PIPELINING FOR ALL THE COLUMNS AND FITTING IT AGAIN TO DF2
     pipeline = Pipeline(stages=minmaxscaler)
     model_scaler= pipeline.fit(df_new)
     scaled_df = model_scaler.transform(df_new)
```

The scaled dataframe is stored in scaled_df variable.

Figure 40
Scaled Dataframe

```
[52] #DISPLAYING ALL THE NORMALIZED VALUES
     scaled_df.show(5)
```

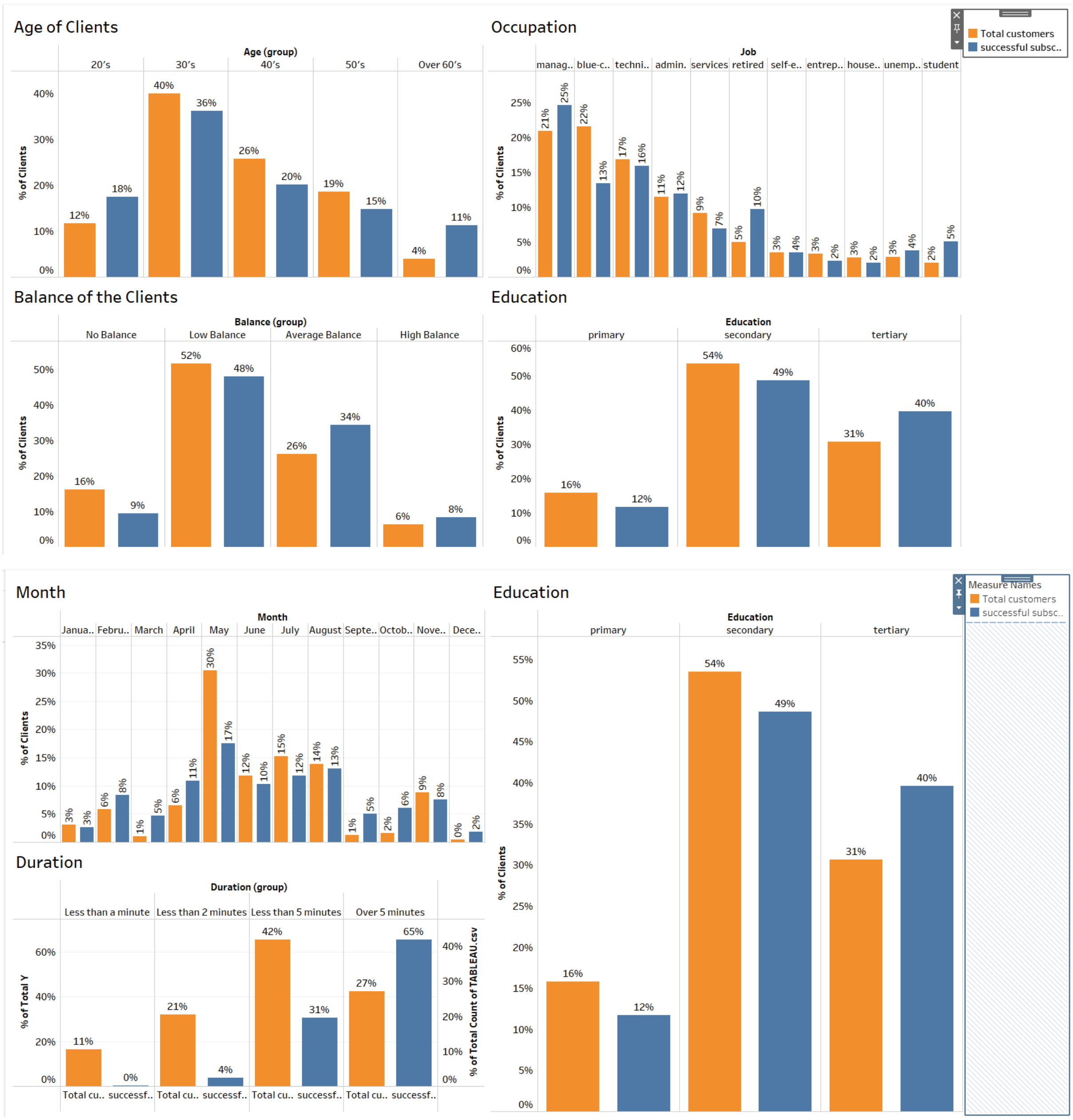
```
-------+-------------------+------------------+----------------------+-------------------+----------------------+------------------------+----------------------+-
eatures|    features_SCALED|job_encoded_SCALED|marital_encoded_SCALED|loan_encoded_SCALED|default_encoded_SCALED|education_encoded_SCALED|housing_encoded_SCALED|p
-------+-------------------+------------------+----------------------+-------------------+----------------------+------------------------+----------------------+-
14,1...|(23,[5,10,12,14,1...|   (10,[5],[1.0])|             [1.0,0.0]|              [1.0]|                 [1.0]|               [1.0,0.0]|                 [0.0]|
14,1...|(23,[5,10,12,14,1...|   (10,[5],[1.0])|             [1.0,0.0]|              [1.0]|                 [1.0]|               [1.0,0.0]|                 [0.0]|
14,1...|(23,[5,10,12,14,1...|   (10,[5],[1.0])|             [1.0,0.0]|              [1.0]|                 [1.0]|               [1.0,0.0]|                 [0.0]|
14,1...|(23,[5,10,12,14,1...|   (10,[5],[1.0])|             [1.0,0.0]|              [1.0]|                 [1.0]|               [1.0,0.0]|                 [0.0]|
14,1...|(23,[5,10,12,14,1...|   (10,[5],[1.0])|             [1.0,0.0]|              [1.0]|                 [1.0]|               [1.0,0.0]|                 [0.0]|
-------+-------------------+------------------+----------------------+-------------------+----------------------+------------------------+----------------------+-
```

3.4 EXPLORATORY DATA ANALYSIS

The exploratory data analysis is done using Tableau visualizations.

Figure 41

Majority classes in Attributes-created in Tableau Dashboard



When comparing the percentage distribution of each attribute for every class, the most common client age category is 30 to 40 years (40%). In the job attribute ,Blue collar is major (22%). Highest percentage of clients (60%) is married in Marital, and most of them has secondary class (54%) Education. Clients with no credit are more (Default attribute -98%). Those having average yearly balance is between -8019 and 10000 (low balance) is more (Balance-52%).Month May has highest ratio (30%) during the year. In the attribute Duration, (73%) lasted long ,300 seconds (5 min). (KomboElvis,2020)

Figure 42
Age v/s job -created in Tableau-linegraph

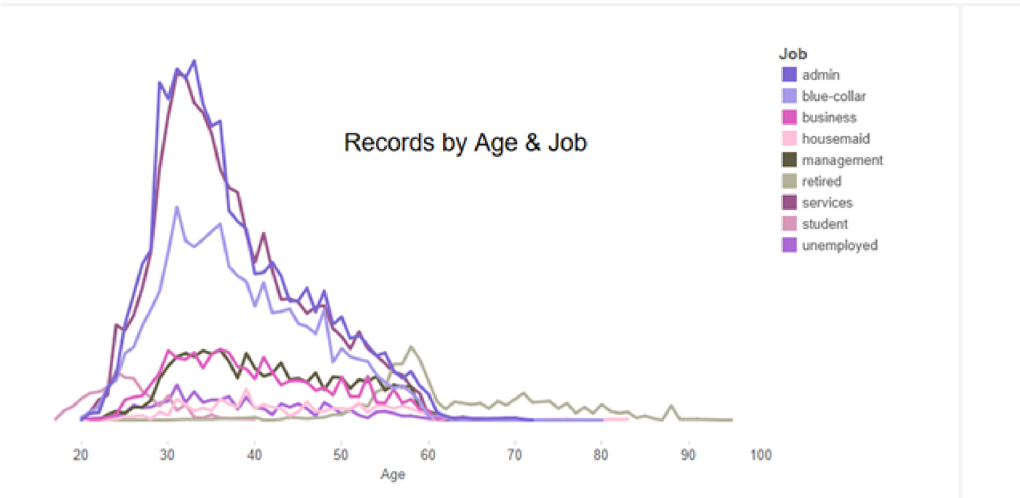
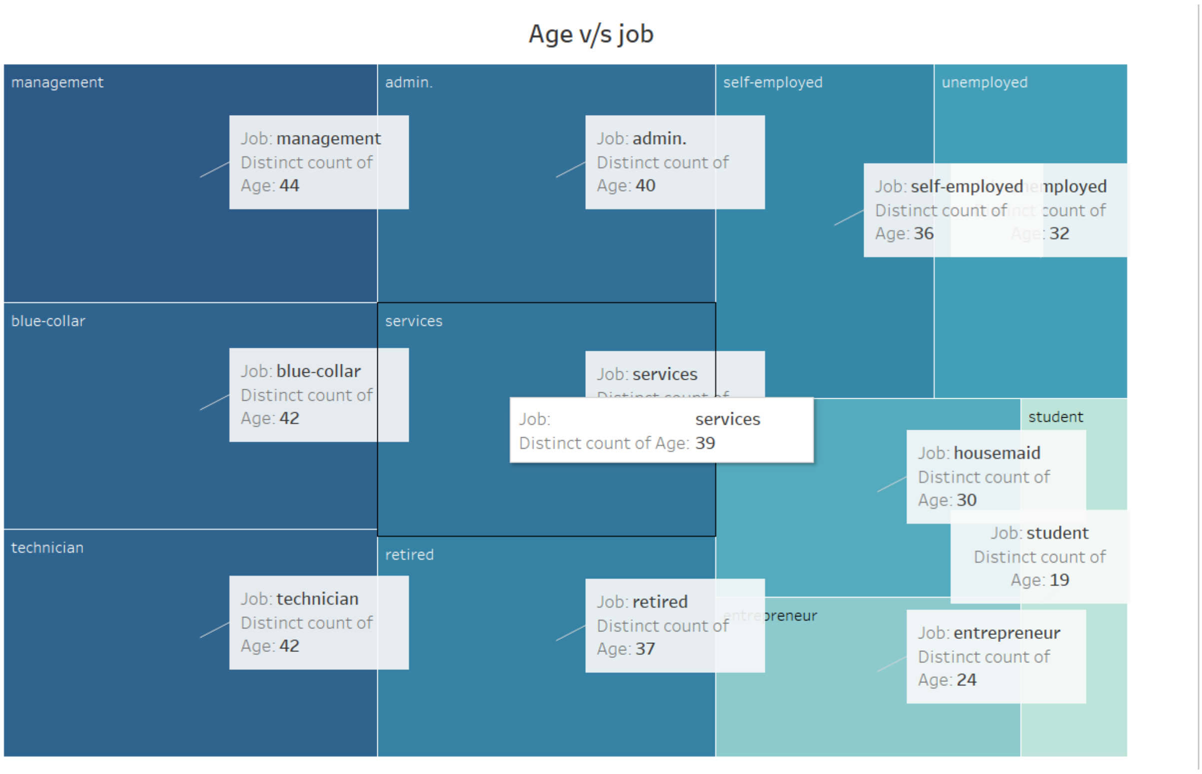


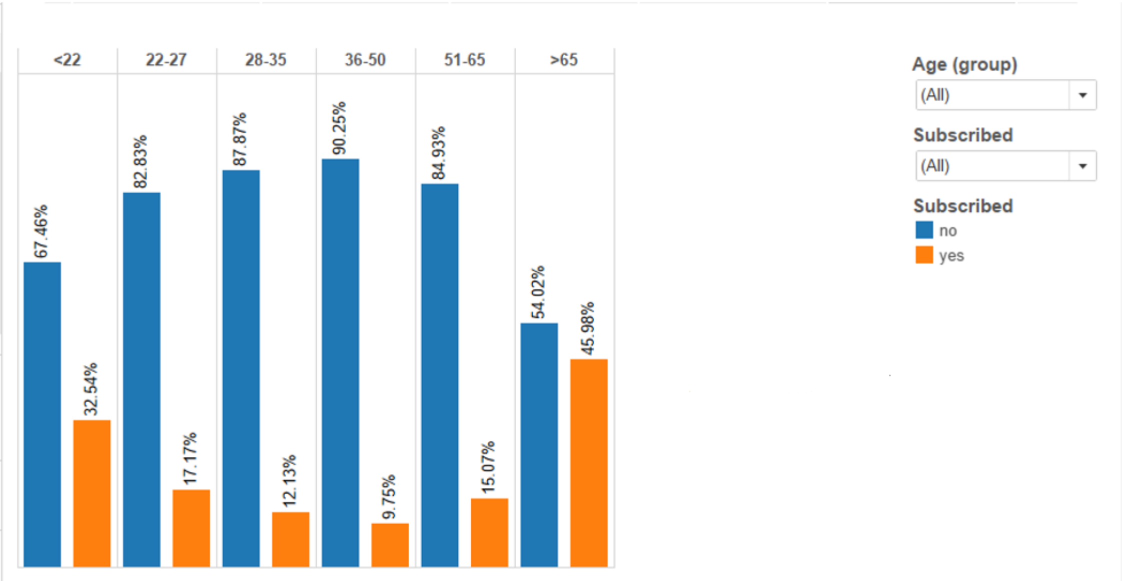
Figure 43
Job v/s age -created in Tableau Treemap.



Job attribute has various kinds of job such as admin, unemployed, management, housemaid ,entrepreneur, student, blue collar, self-employed, retired, technician and services. It is found that age group from 25-50 are populated in blue-collar and services job. Few percentage is unemployed as well. Age group60-100 are all either retired or unemployed.

We can see how they responded to subscription of deposit.

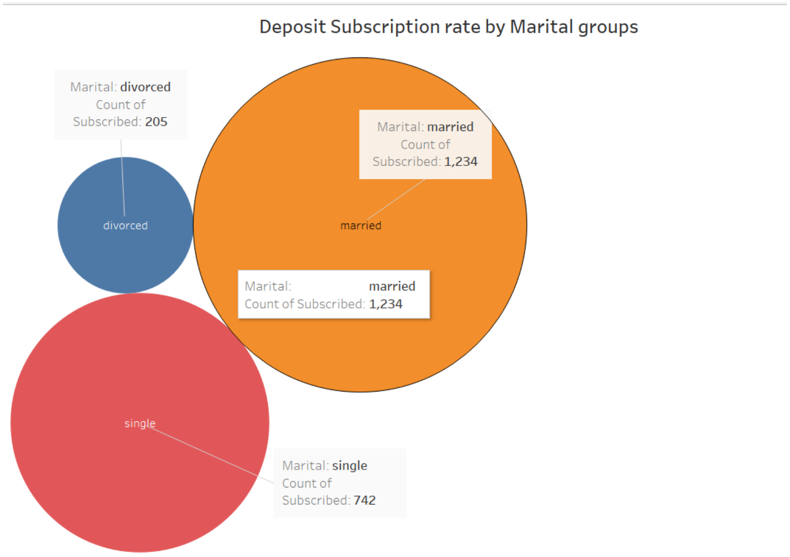
Figure 44
Age v/s Subscription of Deposit -created in Tableau



Age groups from 22-50 have subscribed the deposit accounts in high percentage .51-65 age group, during their retirement stage or being in the state of unemployed have also subscribed the deposit.

The marital attribute is described in classes married,single ,divorced.Class divorced means divorced or widowed.

Figure 45
Marital v/s Subscribed -created in Tableau Piecharts



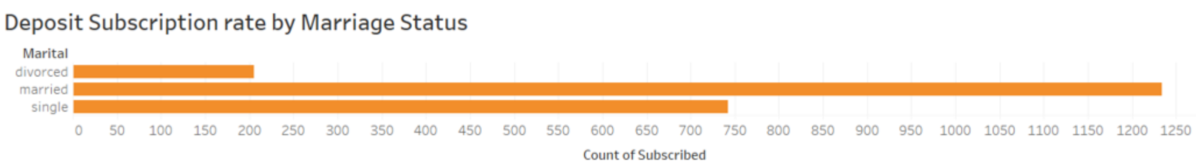
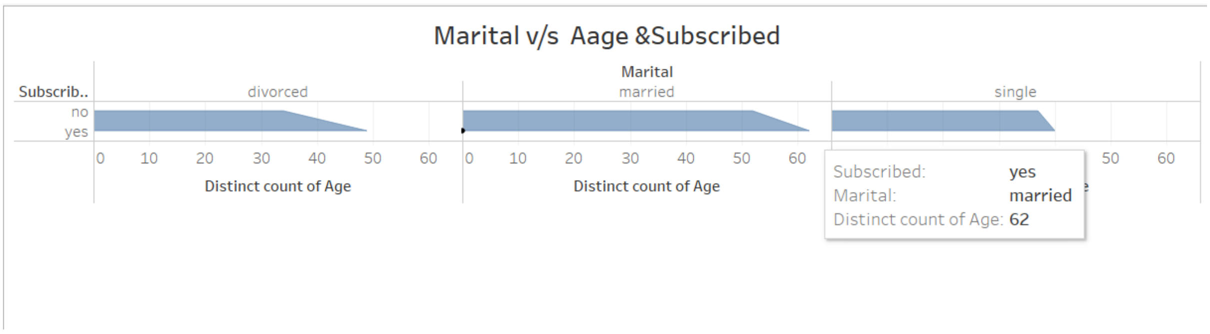
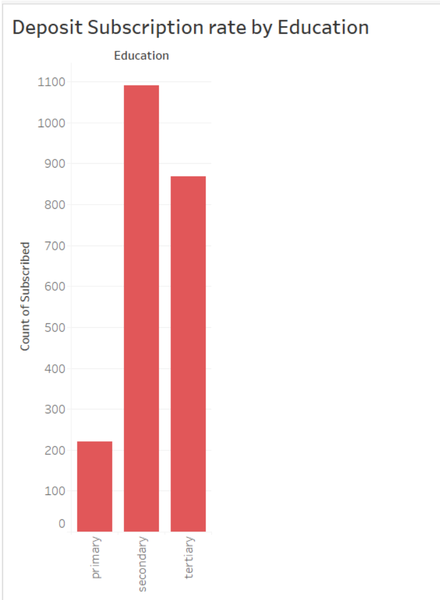


Figure 46
Marital v/s Subscribed & Age -created in Tableau



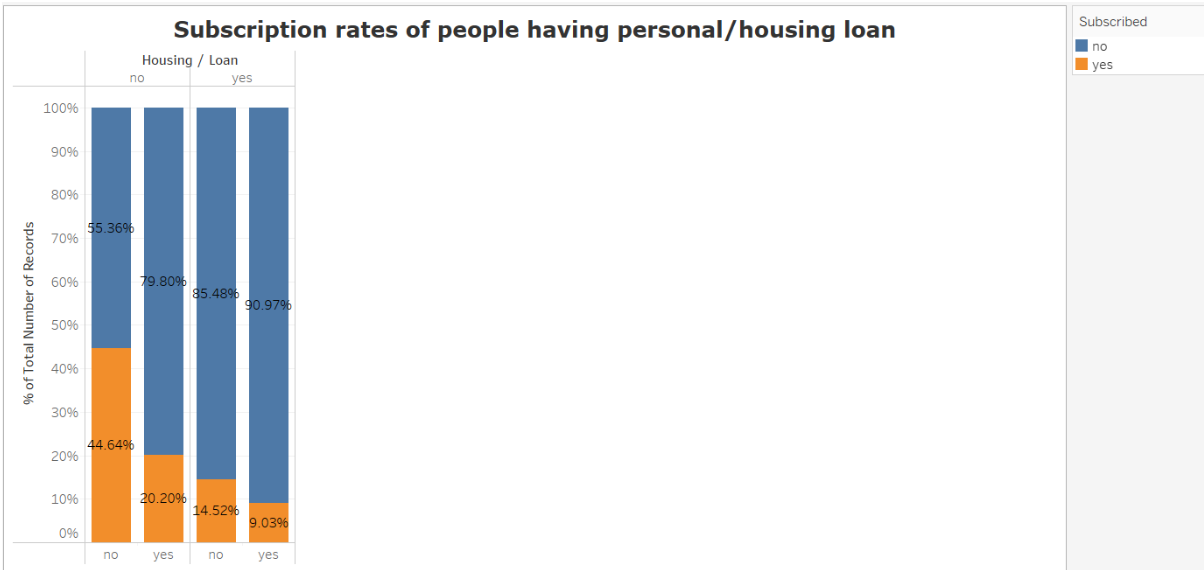
The married class(age 20-55) have subscribed most of the deposits. Single people also shown interest in a good scale.

Figure 47
Education v/s Subscribed -created in Tableau



Education classes are primary, secondary and tertiary.Secondary Education group has taken more deposits than tertiary level groups. Primary groups have the least response which indicates Education has significant impact on money saving interest.

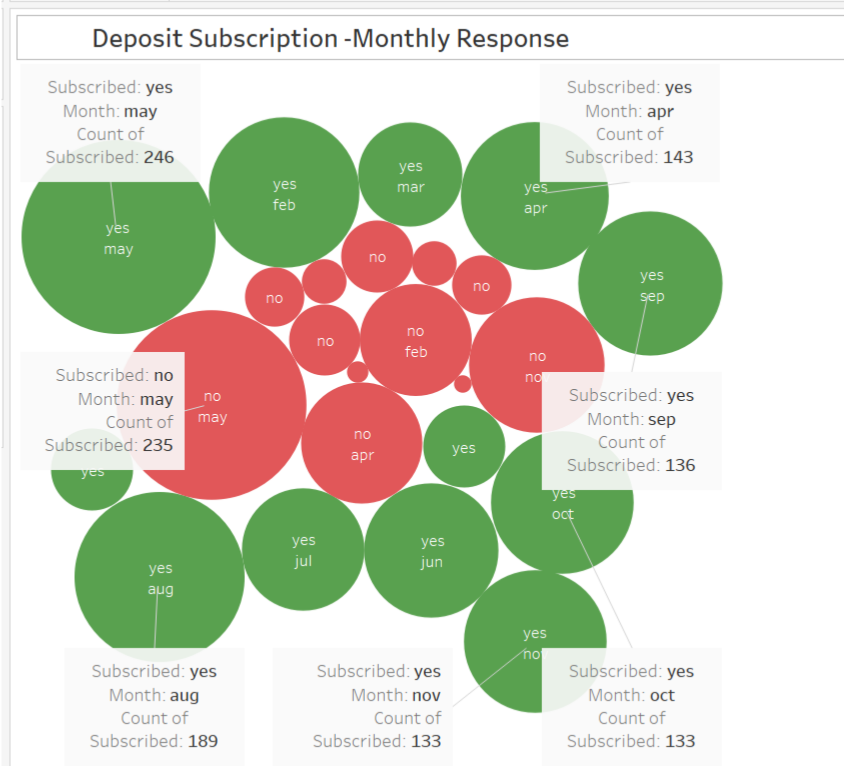
Figure 48
Personal Loan and Housing loan v/s Subscribed -created in Tableau



The clients with no loan accounts housing (taken housing loan or not)and loan(has any personal loan or not) subscribed more deposits.Negative correlation is found.

Figure 49

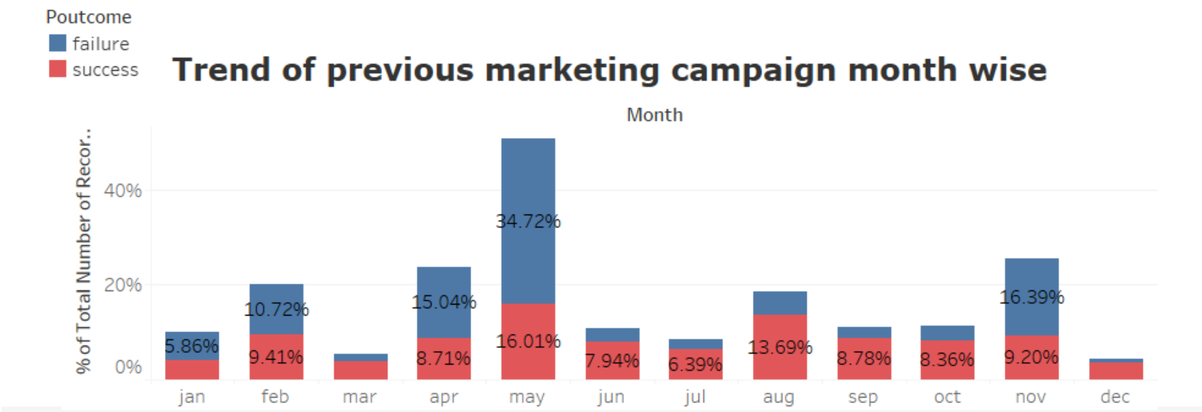
Month v/s Subscribed (current campaign)-created in Tableau



The month and day attributes are clearly months of year and days in a week. The data analysis shows in the month of May has the best positive response. August and April also performed better.

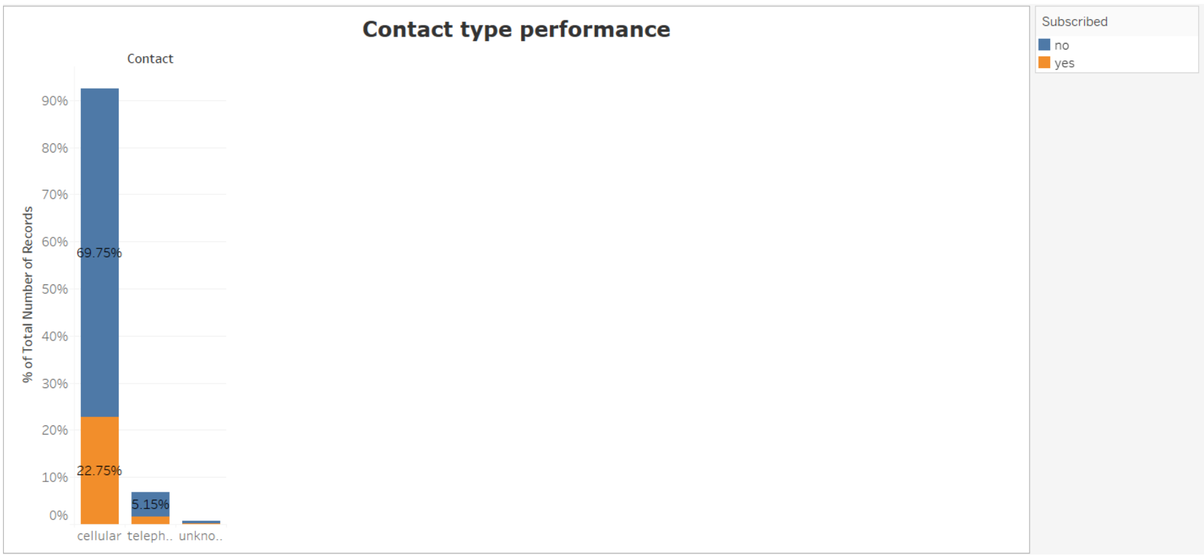
Figure 50

Month v/s Poutcome (Previous campaign subscription)-created in Tableau



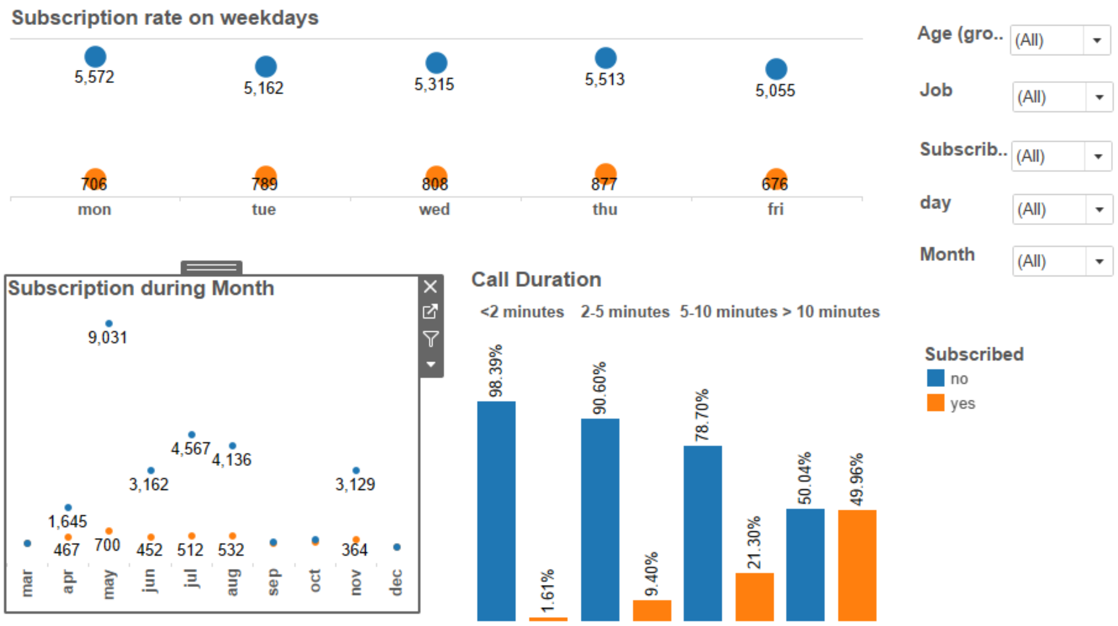
In the previous campaign also, May and august performed better.

Figure 51
Contact Types- Performance-created in Tableau



The contact communication classes are telephone, cellular and unknown .Unknown group is relevant here because many have subscribed but not clearly known on the means of communication made through marketing.Telephone mode has the most successful subscription rate.

Figure 52
Day, Month, Duration v/s Subscribed plots-created in Tableau Dashboard



Earlier the week is the best time to call potential clients. Thursdays and Wednesdays get more response.Call Duration and subscribed rate and directly proportional. Higher the duration, higher the chance of subscription.

The attribute poutcome represent the previous outcome of marketing campaign on the same age groups .It indicates whether it was a failure or success then .This attribute compared with the present marketing outcome “Subscribed”.

Figure 53

Current campaign performance -Tableau

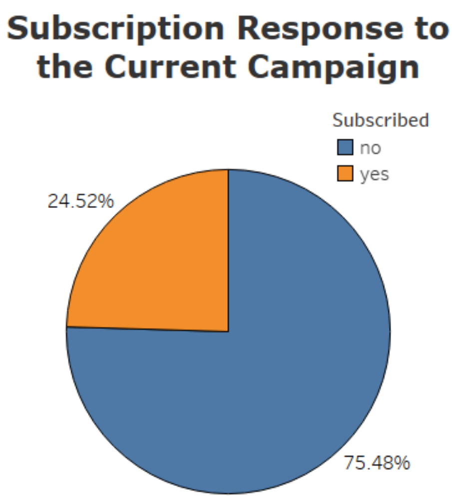
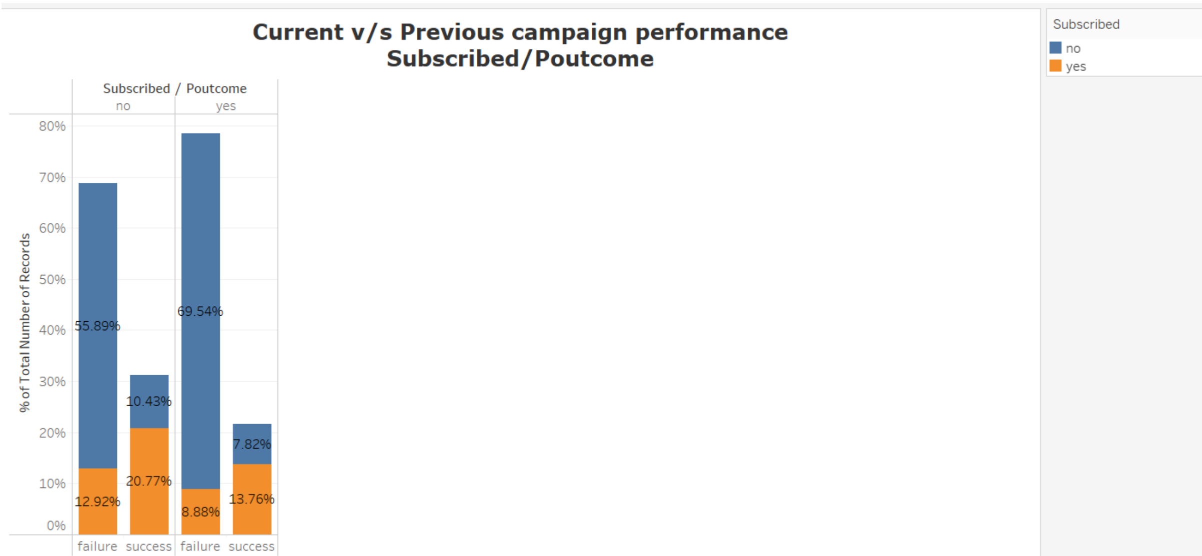


Figure 54

Current v/s previous campaign performance -Tableau



The current campaign performed better than previous with more subscriptions. Although the negative response slightly increased than before.

3.5 MACHINE LEARNING MODEL BUILDING AND EVALUATION

1)Train set and Test set split

Splitting Dataset into train set and test set in the proportion of 70% and 30 % respectively.

Figure 55

Train/Test split

```
#Splitting scaled dataset in 70% AND 30% ratio
train, test = df5.randomSplit([0.7, 0.3], seed = 742)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 1540
Test Dataset Count: 641
```

Supervised Learning

1)Logistic Regression

This model is suitable for describing and testing the hypothesis relationship between categorical outcome variable and one or more categorical or continuous predictors.The values of the estimated parameters are adjusted iteratively until the greatest probability value of them is obtained.**(Gulcan Ogundur,2020).**Here direct marketing data set "Subscribed" is a flag attribute (yes or no) then the option of forward binomial procedure in the partitioned data is selected.

Figure 56

Performance metrics,LR

```
[ ] #Multi class Classification Evaluator for accuracy
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator

    eval1 = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='label', metricName='accuracy')
    acc = eval1.evaluate(LR_predictions)
    print("accuracy=%g" %(acc))

    accuracy=0.75507
```

```
[ ] # CONFUSION MATRIX
    from pyspark.mllib.evaluation import MulticlassMetrics
    pred_label=LR_predictions.select( 'label', 'prediction').rdd
    metrics = MulticlassMetrics(pred_label)
    print(metrics.confusionMatrix())

    DenseMatrix([[365.,  76.],
                 [ 81., 119.]])
```

```
[ ] #PRECISION, RECALL and F1SCORE
    cm=metrics.confusionMatrix().toArray()
    precision=(cm[0][0])/(cm[0][0]+cm[1][0])
    recall=(cm[0][0])/(cm[0][0]+cm[0][1])
    f1score =((2*precision*recall )/ (precision + recall))

    print("Logistic regression:--precision,recall,f1score",precision,recall,f1score)

    Logistic regression:--precision,recall,f1score 0.8183856502242153 0.8276643990929705 0.8229988726042842
```

2)Decision Tree Classifier

This model recursively separates data into branches ,build a tree for improving the prediction accuracy. Decision nodes have to split, testing the values of some functions of data attributes. Each branch of the decision node is different outcome of the test.

Figure 57

Performance metrics-DT

```
[ ] #AREA UNDER ROC curve
    evaluator = BinaryClassificationEvaluator()
    print("Test Area Under ROC: " + str(evaluator.evaluate(prediction1, {evaluator.metricName: "areaUnderROC"})))

    Test Area Under ROC: 0.7771760377141542

[ ] #CALCULATING ACCURACY
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator

    eval2 = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='label', metricName='accuracy')
    acc1 = eval2.evaluate(prediction1)
    print("accuracy=%g" %(acc1))

    accuracy=0.764431

[ ] #CONFUSION MATRIX
    from pyspark.mllib.evaluation import MulticlassMetrics
    pred_label1=prediction1.select( 'label', 'prediction').rdd
    metrics1 = MulticlassMetrics(pred_label1)
    print(metrics1.confusionMatrix())

    DenseMatrix([[364.,  69.],
                 [ 82., 126.]])
```

```
[ ] # RECALL, PRECISION AND F1SCORE
    cm=metrics1.confusionMatrix().toArray()
    precision=(cm[0][0])/(cm[0][0]+cm[1][0])
    recall=(cm[0][0])/(cm[0][0]+cm[0][1])
    f1score =((2*precision*recall )/ (precision + recall))

    print("Decision Tree:precision,recall,f1score",precision,recall,f1score)

    Decision Tree:precision,recall,f1score 0.8161434977578476 0.8406466512702079 0.8282138794084187

[ ] #PRINTING ALL THE IMPORTANT FEATURES
    DT_Model.featureImportances

    SparseVector(23, {1: 0.0075, 3: 0.0282, 15: 0.1401, 17: 0.7534, 21: 0.0597, 22: 0.0112})
```

3)Naïve Bayes Classifier

Naive Bayes classifiers are a family of simple" probabilistic classifiers" based on applying Bayes' theorem with naive independence assumptions between the features.**(Tanvi Penumudy,2021).**Multinominal Naïve Bayes is used here ,because the classification of one feature doesnot depend on other.**(ShriRam,2021)**

Figure 58

Performance Metrics-NB

```
+-----+--------------------+----------+--------------------+
|label|       rawPrediction|prediction|         probability|
+-----+--------------------+----------+--------------------+
|  0.0|[-18.128101289108...|       0.0|[0.71227863685887...|
|  0.0|[-18.905025587194...|       0.0|[0.83308795651024...|
|  0.0|[-17.210598336493...|       0.0|[0.83078085549233...|
|  0.0|[-16.368239992271...|       0.0|[0.82011846673742...|
|  0.0|[-18.122511379455...|       0.0|[0.51656396890790...|
|  0.0|[-18.379947725811...|       0.0|[0.52612872952829...|
|  0.0|[-17.219309198880...|       0.0|[0.54173239419087...|
|  0.0|[-19.904541403547...|       0.0|[0.80183300993376...|
|  0.0|[-22.618681384194...|       0.0|[0.76733814845131...|
|  0.0|[-20.750414292819...|       0.0|[0.80281500588117...|
+-----+--------------------+----------+--------------------+
only showing top 10 rows

Test Area Under ROC: 0.5786054421768712
```

```
[ ] #CALCULATING ACCURACY
    print("accuracy=%g" %(nb_accuracy))
```

```
accuracy=0.74415
```

```
#Confusion matrix
from pyspark.mllib.evaluation import MulticlassMetrics
pred_and_label=nb_predictions.select( 'label', 'prediction').rdd
metrics2 = MulticlassMetrics(pred_and_label)
print(metrics2.confusionMatrix())
```

```
DenseMatrix([[386., 109.],
             [ 55.,  91.]])
```

4. EXPERIMENTAL RESULTS

Model Evaluation

Receiver Operating Characteristic (ROC) plots the specificity, true negative against sensitivity and true positive rate given different threshold.(Hany Elsalamony,2013) From this, Area Under the Curve (AUC) is calculated ,better evaluation metric than accuracy score. ()

Figure 59
Confusion Matrix

CONFUSION MATRIX

		Predicted	
		Positive (yes)	Negative (no)
Actual	Positive (yes)	TP	FP
	Negative (no)	TN	FN

From Hany Elsalamony(2013)

True Positive is Correctly predicted Event values,whereas False Positive is Incorrectly predicted event values.True Negative- predicts correctly for no-event values.While False Negative predicted no-event values incorrectly.

Precision is an important metric here .As False positive is critically misread in the prediction of subscription here .The subscribers wrongly predicted to be "Yes"is wrong analysis. Hence Precision should be more for the model.

Figure 60

AUC and accuracy score of each classification model

Model	AUC_score	Accuracy_score	Precision
LR	.81	.75	.81
DT	.77	.76	.81
NB	.57	.74	.87

The LR model and Decision Tree performs similar regarding the accuracy and precision .But Better the ROC curve,better the model.AUC_score is more for LR.Also false positives are less for LR .Hence LR is chosen as best model.

Figure 61

ROC curve and AUC score of LR model

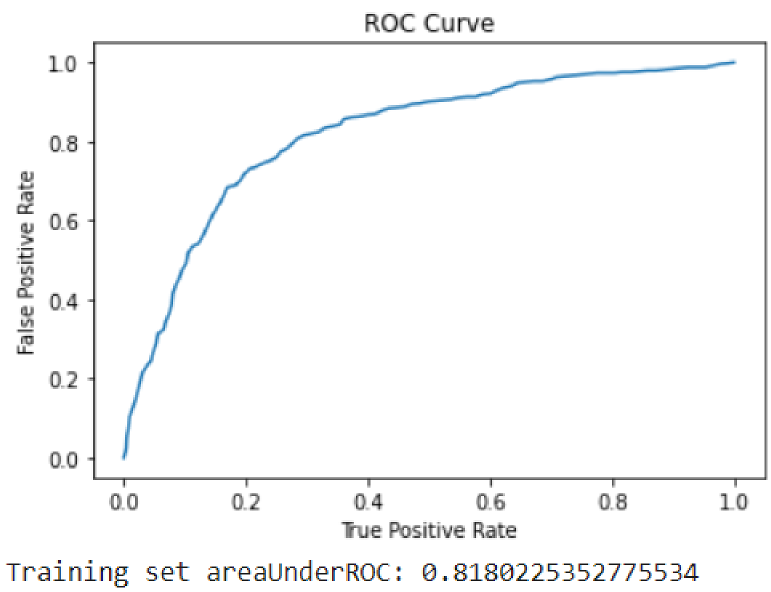
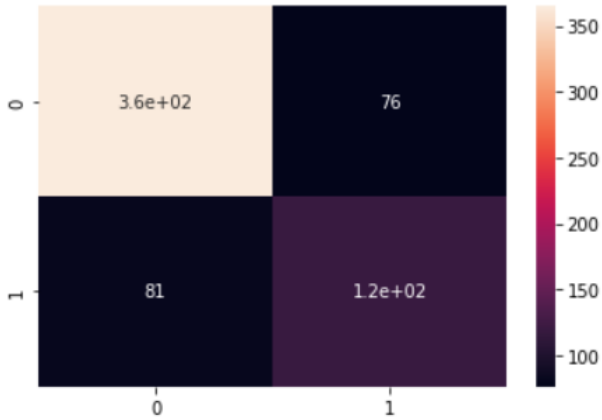


Figure 62

Heat map of LR model

```
[ ] #PLOTTING HEATMAP OF ALL THE METRICS PARAMETERS USING SEABORN PACKAGE
    import seaborn as sns
    sns.heatmap(cm, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1af3a86c10>
```



5. DISCUSSIONS

For the success and survival, a Bank need best marketing strategies. Big data analytics along with predictive analytics here proved excellence in analysing complex data and large procedures ,minimalize number of faulty decisions (false positives and false negatives).Here are few insights and analytical trends found.

EDA findings and Recommendations

- There were no null or duplicates values in dataset.
- The “unknown “values handled by removing them.
- The categorical columns were indexed, encoded, and scaled between 0 and 1 for predictor analytics.
- EDA analysis done through Tableau visualizations and found the following insights.
- Catch the customers when they are young. Age (22- 35) and married are targets.
- The calls should be made on Wednesdays and Thursdays in a week to get fair response (duration).
- The spring and summer months get successful response.
- The call lengths are expected to vary by job groups.
- Telephone is the winner contact medium.
- Duration of call is not used for prediction models as a feature, but from the tableau analysis, higher the duration ,higher subscription .So marketing team should make engaging and longer calls.
- Out of 11162 records, 5289 subscriptions are success.5873 clients did not subscribe. The campaign was not bad this time compared to previous one.

6. CONCLUSION

This paper evaluate and compare the classification performance of three classification models on the Bank marketing data set to categorize bank deposit subscription response. The effectiveness of campaign can be increased by the influential features found through EDA. The classification performances of the **three models** evaluated using Classification accuracy and AUC score. Experimental results shows Logistic Regression model has achieved slightly better performance than Naive Bayes(NB),Decision Tree(DT).Decision Tree also has similar performance results.

7. REFERENCES

[1] WalkerRowe.(October 24,2019). *Using Python and Spark Machine Learning to Do Classification.*.bmc.https://www.bmc.com/blogs/python-spark-machine-learning-classification

[2] JanioMartinezBachmann.(2019).*BankMarketingCampaign..Kaggle* https://www.kaggle.com/code/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/notebook#What-is-a-Term-Deposit?

[3] WalkerRowe.(November28 ,2019). *Python Spark ML K-Means Example.*bmc.https://www.bmc.com/blogs/python-spark-k-means-example/

[4] Geeksforgeeks. (January 13,2021).*10 Types of Tableau Charts For Data Visualization.* https://www.geeksforgeeks.org/10-types-of-tableau-charts-for-data-visualization/

[5] Mohammad Waseem.(March 28,2022). *How To Implement Classification In Machine Learning?.*Edureka. https://www.edureka.co/blog/classification-in-machine-learning/

[6] Dushanthi Manthushika.(May7,2021).*Pyspark with Google Colab.*Medium.https://medium.com/linkit-intecs/pyspark-with-google-colab-d964fd693ca7

[7] Gulcan Ogundur.(May 4,2020). *LogisticRegression with Pyspark.* Medium.https://medium.com/swlh/logistic-regression-with-pyspark-60295d41221

[8] Soumya Goyal.(May 2,2022).*How to Setup Pyspark on Windows??.* Medium .https://medium.com/datamics/how-to-install-pyspark-on-windows-faf7ac293ecf

[9] Bank Marketing Campaign(2019) . *Dataset Description.* https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing

[10] JasonWong.(November 14,202). *Machine Learning Pipelines With Scikit-Learn.*TowardsDataScience.https://towardsdatascience.com/machine-learning-pipelines-with-scikit-learn-d43c32a6aa52

[11]KomboElvis. (August19,2020). *BankTerm DepositMarketingStrategy*.medium. https://medium.com/analytics-vidhya/bank-term-deposit-marketing-strategy-b9684e46c7cc

[12] Henrique Ap. Laureano .*(2018).Bank Marketing Dataset: An overview of classification algorithms.*github.https://henriquelaureano.github.io/courses/ml-kaust/project_report.pdf

[13] Hany Elsalamony.(December,2013).*Bank Direct Marketing Analysis of Data Mining Techniquesarticle.*ResearchGate.https://www.researchgate.net/publication/263054095_Bank_Direct_Marketing_Analysis_of_Data_Mining_Techniques

[14] Tanvi Penumudi.(January 17,2021). *Naïve Bayes from Scratch*.medium.https://medium.com/swlh/naive-bayes-from-scratch-c0c93ed4b826

[15]ShriRam(January 3,2021).*MultinominalNaiveBayesExplained.*upGrad https://www.upgrad.com/blog/multinomial-naive-bayes-explained/

8. APPENDIX

Word count-3000including references

Figure 63
Tableau renamed column

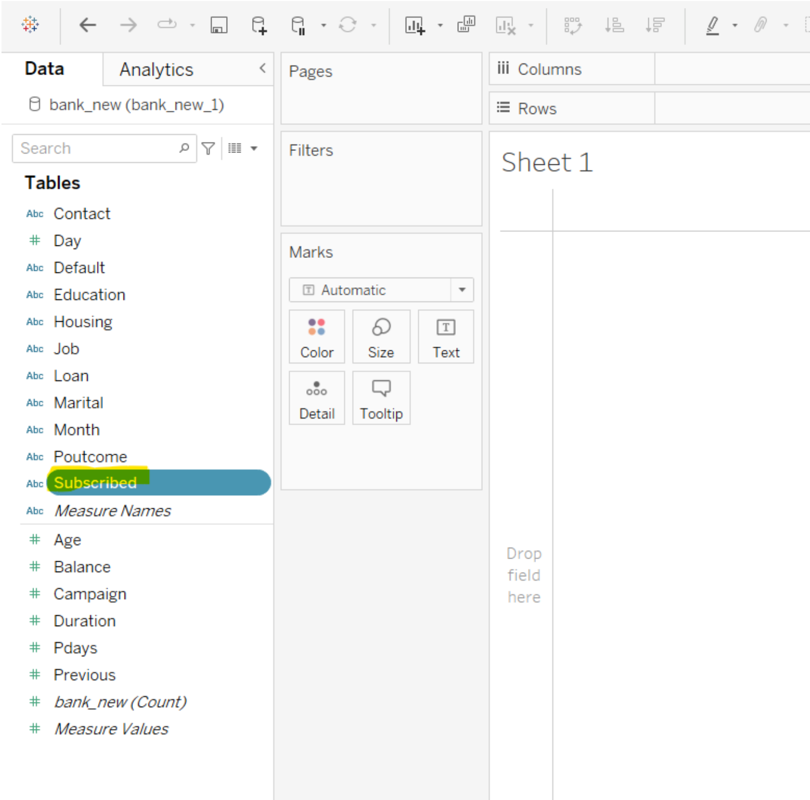


Figure 64
Month v/s Subscribed

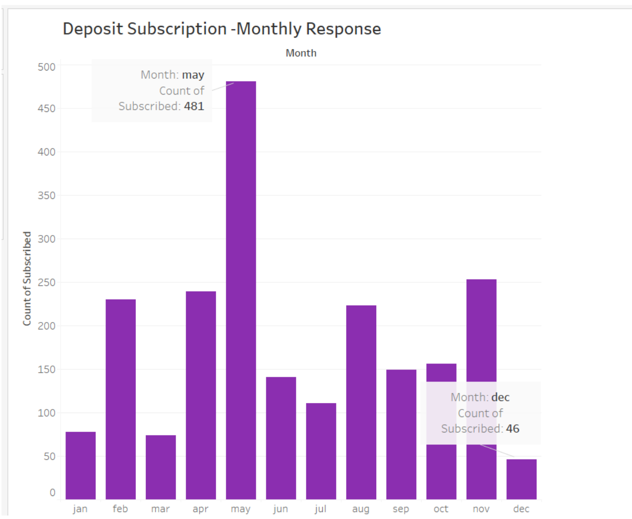


Figure 65

Age v/s Deposit

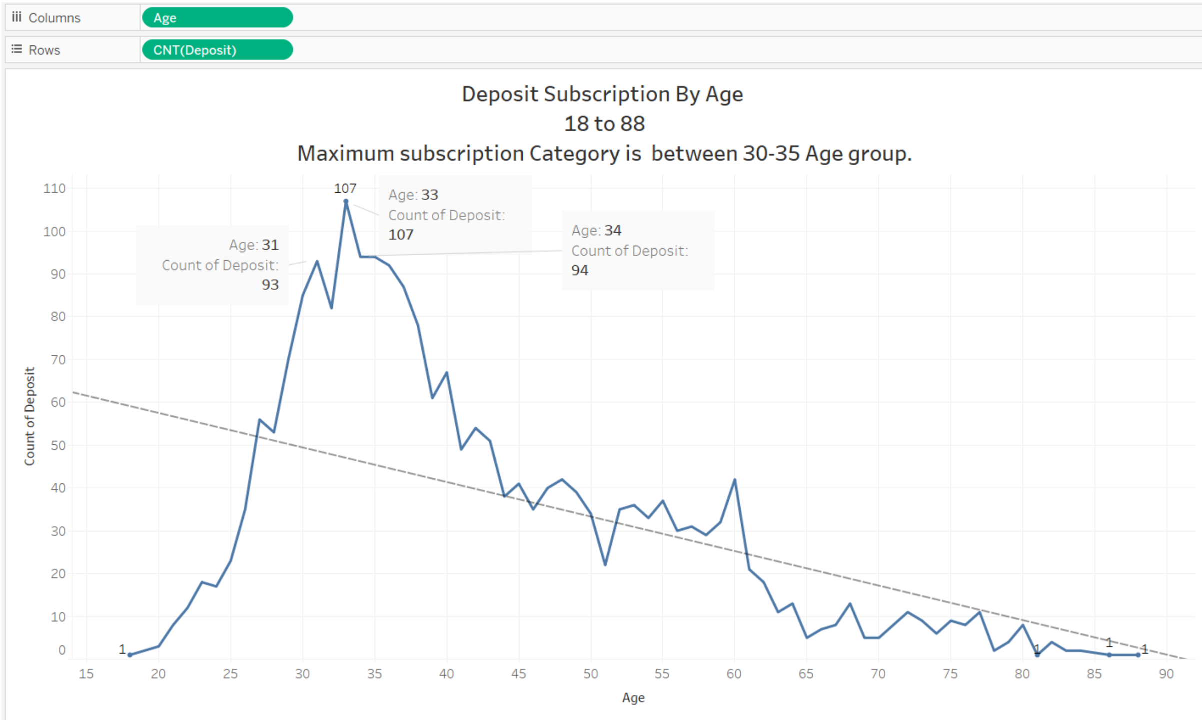
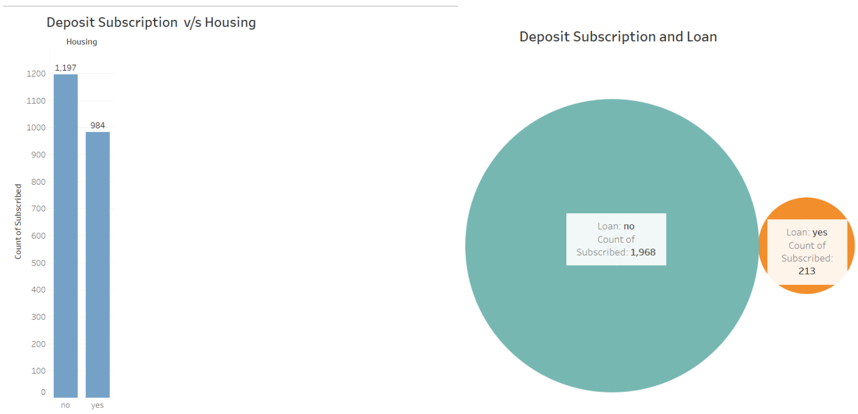
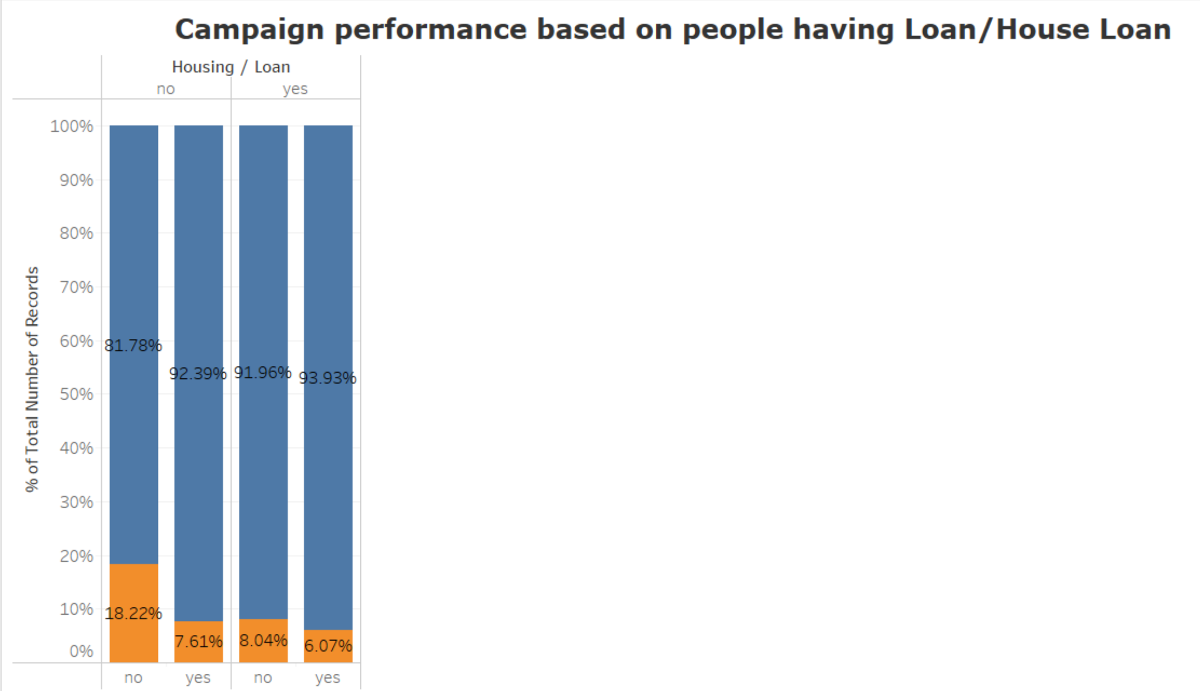


Figure 66

Loan and Housing loan v/s subscription





SOURCE CODE

Colab Notebook:-
https://colab.research.google.com/drive/1R0MiU-KmfSq9y3TQEcCi_9Nm-SgHTE6S?usp=sharing

Data-
https://www.kaggle.com/code/janiobachmann/bank-marketing-campaign-opening-a-term-deposit/data

```
## **Mounting Google Drive**
from google.colab import drive
import os
drive.mount('/content/drive/')
os.chdir("/content/drive/My Drive/7153")
## **Pyspark  installation**
#installing wget for browser link instllation
!pip install wget
#installing java run time
!apt-get install openjdk-8-jdk-headless -qq
```

```
#checking the existing installed java version
!java -version
#decompressing the zipped file in current directory in gdrive
!tar xf spark-3.0.0-bin-hadoop3.2.tgz
#setting the environment variables for spark and java
import os
os.environ["JAVA_HOME"]="/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"]="/content/spark-3.0.0/spark-3.0.0-bin-hadoop3.2"
#installing findspark
!pip install -q findspark
#installing the matching pyspark version
!pip install pyspark==3.0.0
import pyspark
#checking pyspark version
print(pyspark.__version__)
3.0.0
#adding pyspark to sys.path at run time
import findspark
findspark.init("spark-3.0.0-bin-hadoop3.2")
#adding pyspark to sys.path at run time
import findspark
findspark.init("spark-3.0.0-bin-hadoop3.2")

#Testing Pyspark Setup and installation
from pyspark.sql import SparkSession
from pyspark import SparkConf,SparkContext
spark = SparkSession.builder.master("local").appName("Search").config(conf=SparkConf()).getOrCreate()
#creating a dataframe with columnames and value
df=spark.createDataFrame([{"language,usercount" :("java,2000")}])
df.show(1)
```

```
+------------------+
|language,usercount|
+------------------+
|         java,2000|
+------------------+
#Loading Dataset and deriving information
#sparksession object
spark = SparkSession.builder.appName('Bank Marketing Analytics').getOrCreate()
#creating spark dataframe from csv file
df = spark.read.csv('bank_new.csv', header = True, inferSchema = True)
df.show()
```

```
+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
|age|       job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|deposit|
+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
| 59|    admin.| married|secondary|     no|   2343|    yes|  no|unknown|  5|  may|    1042|       1|   -1|       0| unknown|    yes|
| 56|    admin.| married|secondary|     no|     45|     no|  no|unknown|  5|  may|    1467|       1|   -1|       0| unknown|    yes|
| 41|technician| married|secondary|     no|   1270|    yes|  no|unknown|  5|  may|    1389|       1|   -1|       0| unknown|    yes|
| 55|  services| married|secondary|     no|   2476|    yes|  no|unknown|  5|  may|     579|       1|   -1|       0| unknown|    yes|
| 54|    admin.| married| tertiary|     no|    184|     no|  no|unknown|  5|  may|     673|       2|   -1|       0| unknown|    yes|
| 42|management|  single| tertiary|     no|      0|    yes| yes|unknown|  5|  may|     562|       2|   -1|       0| unknown|    yes|
| 56|management| married| tertiary|     no|    830|    yes| yes|unknown|  6|  may|    1201|       1|   -1|       0| unknown|    yes|
| 60|   retired|divorced|secondary|     no|    545|    yes|  no|unknown|  6|  may|    1030|       1|   -1|       0| unknown|    yes|
| 37|technician| married|secondary|     no|      1|    yes|  no|unknown|  6|  may|     608|       1|   -1|       0| unknown|    yes|
| 28|  services|  single|secondary|     no|   5090|    yes|  no|unknown|  6|  may|    1297|       3|   -1|       0| unknown|    yes|
| 38|    admin.|  single|secondary|     no|    100|    yes|  no|unknown|  7|  may|     786|       1|   -1|       0| unknown|    yes|
| 30|blue-collar| married|secondary|     no|    309|    yes|  no|unknown|  7|  may|    1574|       2|   -1|       0| unknown|    yes|
| 29|management| married| tertiary|     no|    199|    yes| yes|unknown|  7|  may|    1689|       4|   -1|       0| unknown|    yes|
| 46|blue-collar|  single| tertiary|     no|    460|    yes|  no|unknown|  7|  may|    1102|       2|   -1|       0| unknown|    yes|
| 31|technician|  single| tertiary|     no|    703|    yes|  no|unknown|  8|  may|     943|       2|   -1|       0| unknown|    yes|
| 35|management|divorced| tertiary|     no|   3837|    yes|  no|unknown|  8|  may|    1084|       1|   -1|       0| unknown|    yes|
| 32|blue-collar|  single|  primary|     no|    611|    yes|  no|unknown|  8|  may|     541|       3|   -1|       0| unknown|    yes|
| 49|  services| married|secondary|     no|     -8|    yes|  no|unknown|  8|  may|    1119|       1|   -1|       0| unknown|    yes|
| 41|    admin.| married|secondary|     no|     55|    yes|  no|unknown|  8|  may|    1120|       2|   -1|       0| unknown|    yes|
| 49|    admin.|divorced|secondary|     no|    168|    yes| yes|unknown|  8|  may|     513|       1|   -1|       0| unknown|    yes|
+---+----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
```

```
type(df)
pyspark.sql.dataframe.DataFrame
row=df.count()
column=len(df.columns)
print(f"dimension of dataframe is {(row,column)}")
print(f"number of rows are {row}")
print(f"number of columns are {column}")
dimension of dataframe is (11162, 17)
number of rows are 11162
number of columns are 17
df.summary().show()
```

```
+-------+------------------+-------+--------+---------+-------+------------------+-------+-----+--------+-------
|summary|               age|    job| marital|education|default|           balance|housing| loan| contact|
+-------+------------------+-------+--------+---------+-------+------------------+-------+-----+--------+-------
|  count|             11162|  11162|   11162|    11162|  11162|             11162|  11162|11162|   11162|
|   mean|41.231947679627304|   null|    null|     null|   null|1528.5385235620856|   null| null|    null|15.658036
| stddev|11.913369192215518|   null|    null|     null|   null| 3225.413325946149|   null| null|    null| 8.420739
|    min|                18| admin.|divorced|  primary|     no|             -6847|     no|   no|cellular|
|    25%|                32|   null|    null|     null|   null|               122|   null| null|    null|
|    50%|                39|   null|    null|     null|   null|               550|   null| null|    null|
|    75%|                49|   null|    null|     null|   null|              1708|   null| null|    null|
|    max|                95|unknown|  single|  unknown|    yes|             81204|    yes|  yes| unknown|
+-------+------------------+-------+--------+---------+-------+------------------+-------+-----+--------+-------
```

```
df.printSchema()
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
 df.describe()
DataFrame[summary: string, age: string, job: string, marital: string, education: string, default: string,
balance: string, housing: string, loan: string, contact: string, day: string, month: string, duration: string,
campaign: string, pdays: string, previous: string, poutcome: string, deposit: string]
#Pre-Processing Of Dataset

#1.Identifying Duplicates
#1.Finding the duplicates if any .Here returns same no of records so  no duplicates in dataframe
#df.dropDuplicates().count()
df.distinct().count()
11162
df.count()
11162
#no duplicates
```

```
#2.Identifying null values
#checking null values
df_col=df.columns
from pyspark.sql.functions import col,isnan,when,count
null_col=df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_col]
  )
null_col.show()
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
#no null values occured
#3.Removal of unknown values
#Creating a temporary Table named "bank" for filtering the columns
df.registerTempTable("bank")
#Filtering unknown values from all the columns using  "AND" "OR" in sparksql
#with this query ,The "other"attrinbute in poutcome also gets removed as it is not valid for data analysis
sqlfilter=spark.sql("SELECT * FROM bank WHERE job!='unknown' AND education!='unknown' AND marital!='unknown' AND
 loan!='unknown' AND (poutcome =='failure' OR poutcome == 'success')")
#Storing in new variable to avoid nonetype error
df2=sqlfilter
df2.show()
```

age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
33	services	married	secondary	no	3444	yes	no	telephone	21	oct	144	1	91	4	failure	yes
56	technician	married	secondary	no	589	yes	no	unknown	23	oct	518	1	147	2	success	yes
34	admin.	married	tertiary	no	899	yes	no	unknown	12	nov	114	1	170	3	failure	yes
53	retired	married	tertiary	no	2269	no	no	cellular	17	nov	1091	2	150	1	success	yes
37	technician	married	secondary	no	5115	yes	no	cellular	17	nov	1210	2	171	4	failure	yes
45	entrepreneur	married	secondary	no	781	no	yes	cellular	17	nov	652	2	126	2	failure	yes
46	unemployed	divorced	secondary	no	3354	yes	no	cellular	19	nov	522	1	174	1	success	yes
40	management	married	tertiary	no	3352	yes	no	cellular	19	nov	639	2	27	1	success	yes

```
#no of records after removal of unknown values
df2.count()
2181
#summarise new dataframe
df2.summary().show
+-------+-----------------+----------+--------+---------+-------+------------------+-------+----+--------+-----------------+-----+------------------+----
|summary|              age|       job| marital|education|default|           balance|housing|loan| contact|              day|month|          duration|    cam
+-------+-----------------+----------+--------+---------+-------+------------------+-------+----+--------+-----------------+-----+------------------+----
|  count|             2181|      2181|    2181|     2181|   2181|              2181|   2181|2181|    2181|             2181| 2181|              2181|
|   mean| 41.84364970197157|     null|    null|     null|   null| 1742.946813388354|   null|null|    null|14.204034846400734| null| 343.5295735900963|1.82118294360
| stddev|12.855329179952637|     null|    null|     null|   null|3397.7939950723485|   null|null|    null| 8.10108738010334| null|275.48193079367053|1.22741268640
|    min|               18|    admin.|divorced|  primary|     no|              -938|     no|  no|cellular|                1|  apr|                 4|
|    25%|               32|      null|    null|     null|   null|               224|   null|null|    null|                8| null|               164|
|    50%|               38|      null|    null|     null|   null|               719|   null|null|    null|               13| null|               263|
|    75%|               50|      null|    null|     null|   null|              2044|   null|null|    null|               20| null|               432|
|    max|               88|unemployed|  single| tertiary|    yes|             81204|    yes| yes| unknown|               31|  sep|              2184|
+-------+-----------------+----------+--------+---------+-------+------------------+-------+----+--------+-----------------+-----+------------------+----

#printing the distinct column values
df2.select('poutcome').distinct().collect()
[Row(poutcome='success'), Row(poutcome='failure')]
#.Renaming label column (deposit to Subscribed)for readability
rename=df2.withColumnRenamed("deposit","Subscribed")
df_new=rename
df_new.count()
2181
df_new.show(5)
```

```
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
|age|       job|marital|education|default|balance|housing|loan|  contact|day|month|duration|campaign|pdays|previous|poutcome|Subscribed|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
| 33|  services|married|secondary|     no|   3444|    yes|  no|telephone| 21|  oct|     144|       1|   91|       4| failure|       yes|
| 56|technician|married|secondary|     no|    589|    yes|  no|  unknown| 23|  oct|     518|       1|  147|       2| success|       yes|
| 34|    admin.|married| tertiary|     no|    899|    yes|  no|  unknown| 12|  nov|     114|       1|  170|       3| failure|       yes|
| 53|   retired|married| tertiary|     no|   2269|     no|  no| cellular| 17|  nov|    1091|       2|  150|       1| success|       yes|
| 37|technician|married|secondary|     no|   5115|    yes|  no| cellular| 17|  nov|    1210|       2|  171|       4| failure|       yes|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+
only showing top 5 rows
## **5.Indexing and Encoding the categorical variables**
#Filtering categorical columns
categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'poutcome']

#Creating an empty list for pipeline and assembler
list_stages = []
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
#Using FOR LOOP for indexing and encoding all selected categorical columns
#STRING INDEXER  index all the columns and store in a new column with +INDEXED
#ONE HOT ENCODER encode all the indexed columns and store in a new column with +ENCODED
for i in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = i, outputCol = i + '_indexed')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[i+ "_encoded"])
    list_stages += [stringIndexer, encoder]

#Indexing predictor column 'Subscribed' as label and features
label_index= StringIndexer(inputCol = 'Subscribed', outputCol = 'label')

#Creating stages for both numeric and categorical columns
list_stages += [label_index]
numericColumns = ['age', 'balance', 'campaign', 'pdays', 'previous']

#Adding both to assembler
input_assembler = [c + "_encoded" for c in categoricalColumns] + numericColumns
```

```
#Adding both to assembler
input_assembler = [c + "_encoded" for c in categoricalColumns] + numericColumns

#vectorizing to create new features column with indexed and encoded values.
assembler = VectorAssembler(inputCols=input_assembler, outputCol="features")
list_stages += [assembler]
from pyspark.ml import Pipeline
#combining all pipeline stages
pipeline = Pipeline(stages = list_stages)
#fitting the model
pipelineModel = pipeline.fit(df_new)
#transforming the model
df_new= pipelineModel.transform(df_new)
#Storing in new variable to avoid none type error
df4=df_new
df4.show(5)
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+--------------+--
|age|       job|marital|education|default|balance|housing|loan|  contact|day|month|duration|campaign|pdays|previous|poutcome|Subscribed|job_indexed|   job_encoded|marital_indexed|ma
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+--------------+--
| 33|  services|married|secondary|     no|   3444|    yes|  no|telephone| 21|  oct|     144|       1|   91|       4| failure|       yes|        5.0|(10,[5],[1.0])|           0.0|
| 56|technician|married|secondary|     no|    589|    yes|  no|  unknown| 23|  oct|     518|       1|  147|       2| success|       yes|        1.0|(10,[1],[1.0])|           0.0|
| 34|    admin.|married| tertiary|     no|    899|    yes|  no|  unknown| 12|  nov|     114|       1|  170|       3| failure|       yes|        2.0|(10,[2],[1.0])|           0.0|
| 53|   retired|married| tertiary|     no|   2269|     no|  no| cellular| 17|  nov|    1091|       2|  150|       1| success|       yes|        4.0|(10,[4],[1.0])|           0.0|
| 37|technician|married|secondary|     no|   5115|    yes|  no| cellular| 17|  nov|    1210|       2|  171|       4| failure|       yes|        1.0|(10,[1],[1.0])|           0.0|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+--------------+--
only showing top 5 rows
```

```
## **6.Normalisation of encoded columns**
#Scaling of Data
#Only scaling the encoded columns as they are having different range of values
from pyspark.ml.feature import MinMaxScaler
encoded_vars=['features','job_encoded','marital_encoded','loan_encoded','default_encoded','education_encoded','housing_encoded','poutcome_encoded']

#Min max scaling to scale down between 0 and 1
minmaxscaler = [MinMaxScaler(inputCol=scale_features ,outputCol=scale_features+ "_SCALED") for scale_features in encoded_vars]

#PIPELINING FOR ALL THE COLUMNS AND FITTING IT AGAIN TO DF2
pipeline = Pipeline(stages=minmaxscaler)
model_scaler=  pipeline.fit(df_new)
scaled_df = model_scaler.transform(df_new)
#DISPLAYING ALL THE NORMALIZED VALUES
scaled_df.show(5)
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+-------------+--
|age|       job|marital|education|default|balance|housing|loan|  contact|day|month|duration|campaign|pdays|previous|poutcome|Subscribed|job_indexed|   job_encoded|marital_indexed|ma
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+-------------+--
| 33|  services|married|secondary|     no|   3444|    yes|  no|telephone| 21|  oct|     144|       1|   91|       4| failure|       yes|        5.0|(10,[5],[1.0])|          0.0|
| 56|technician|married|secondary|     no|    589|    yes|  no|  unknown| 23|  oct|     518|       1|  147|       2| success|       yes|        1.0|(10,[1],[1.0])|          0.0|
| 34|    admin.|married| tertiary|     no|    899|    yes|  no|  unknown| 12|  nov|     114|       1|  170|       3| failure|       yes|        2.0|(10,[2],[1.0])|          0.0|
| 53|   retired|married| tertiary|     no|   2269|     no|  no| cellular| 17|  nov|    1091|       2|  150|       1| success|       yes|        4.0|(10,[4],[1.0])|          0.0|
| 37|technician|married|secondary|     no|   5115|    yes|  no| cellular| 17|  nov|    1210|       2|  171|       4| failure|       yes|        1.0|(10,[1],[1.0])|          0.0|
+---+----------+-------+---------+-------+-------+-------+----+---------+---+-----+--------+--------+-----+--------+--------+----------+-----------+--------------+-------------+--
only showing top 5 rows
#SELECTING ONLY THE REQUIRED COLUMNS FOR FURTHER SUPERVISED AND UNSUPERVISED LEARNING

df5=scaled_df.select('Subscribed','label','features','job_encoded_SCALED','marital_encoded_SCALED','loan_encoded_SCALED','default_encoded_SCALED','education_encoded_SCALED','housing
## **Supervised Learning-Splitting data into train set and test set**
#Splitting scaled dataset in 70% AND 30% ratio
train, test = df5.randomSplit([0.7, 0.3], seed = 742)
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
Training Dataset Count: 1540
Test Dataset Count: 641
```

```
#Classification-Logistic Regression

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

#Building LR model and fitting to train set
LR = LogisticRegression(featuresCol = 'features_SCALED', labelCol = 'label', maxIter=10)
LR_model = LR.fit(train)
import matplotlib.pyplot as plt
import numpy as np
#sorting the coefficients
beta_coef = np.sort(LR_model.coefficients)

#plotting the coefficients
plt.plot(beta_coef)
plt.ylabel('Beta Coefficients')
plt.show()
#Displaying the coefficient and intercept of the model
print(beta_coef)
print("Coefficients: " + str(LR_model.coefficients))
print("Intercept: " + str(LR_model.intercept))
[-2.56245614 -2.39329099 -0.72894848 -0.66562526 -0.61195461 -0.48246488
 -0.42261548 -0.30638277 -0.25365577 -0.11593598 -0.06005363 -0.03727342
 -0.01719841 -0.00880428  0.01508273  0.05560106  0.05692917  0.10278279
  0.21061537  0.38093549  0.49488607  2.02643564  2.35802913]
Coefficients: [-0.11593597634032905,0.05560106483969587,0.05692917050013604,0.38093548808297556,-0.2536557713914945,0.015082725188940384,-0.6656252635387444,-0.6119546125677278,
0.21061536524791594,0.49488607488632846,-0.0372734171172424,-0.060053626608081015,-0.008804281762072257,-0.3063827661093845,-0.42261548168677826,-0.7289484838949549,-0.482464884889
Intercept: -0.7793344296330789
#ROC Curve
#ROC computation
#Summary function gives all the parameters
trainingSummary = lrModel.summary
roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.ylabel('False Positive Rate')
```

```
plt.xlabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
#RECALL VS PRECISION graph
pr = trainingSummary.pr.toPandas()
plt.plot(pr['recall'],pr['precision'])
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
#CALCULATING PREDICTION AND PROBABILITY FOR ALL THE FEATURES
LR_predictions= LR_model.transform(test)
LR_predictions.select( 'features','label', 'rawPrediction', 'prediction', 'probability').show(10)
+--------------------+-----+--------------------+----------+--------------------+
|            features|label|       rawPrediction|prediction|         probability|
+--------------------+-----+--------------------+----------+--------------------+
|(23,[0,10,12,14,1...|  1.0|[2.42431948692267...|       0.0|[0.91866308610377...|
|(23,[0,10,12,14,1...|  1.0|[-0.1394982070808...|       1.0|[0.46518189256400...|
|(23,[0,10,12,14,1...|  1.0|[-0.2966828137176...|       1.0|[0.42636859649879...|
|(23,[0,10,13,14,1...|  1.0|[0.98137542427916...|       0.0|[0.72738104531559...|
|(23,[0,10,13,14,1...|  1.0|[0.84721961733564...|       0.0|[0.69998356870205...|
|(23,[0,10,13,14,1...|  1.0|[0.91382764481071...|       0.0|[0.71378277825887...|
|(23,[0,10,13,14,1...|  1.0|[-0.2793269885308...|       1.0|[0.43061878150569...|
|(23,[0,10,13,14,1...|  1.0|[-1.2793503095994...|       1.0|[0.21766083555062...|
|(23,[0,10,13,14,1...|  1.0|[0.03527376707084...|       0.0|[0.50881752752762...|
|(23,[0,10,13,14,1...|  1.0|[-0.0639181477997...|       1.0|[0.48402590123423...|
+--------------------+-----+--------------------+----------+--------------------+
only showing top 10 rows
#Using BinaryClassificationEvaluator  for TEST AREA UNDER ROC .For binary class,default metric is Area under ROC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator_ = BinaryClassificationEvaluator()
print('Test Area Under ROC is', evaluator_.evaluate(LR_predictions))
Test Area Under ROC is 0.8041048637461209
```

```
#Comparing  LABEL AND PREDICTION for understanding accuracy
acc_df=LR_predictions.select("label","prediction").show(5)
+-----+----------+
|label|prediction|
+-----+----------+
|  1.0|       0.0|
|  1.0|       1.0|
|  1.0|       1.0|
|  1.0|       0.0|
|  1.0|       0.0|
+-----+----------+
only showing top 5 rows
#Multi class Classification Evaluator for accuracy
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
eval1 = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='label', metricName='accuracy')
acc = eval1.evaluate(LR_predictions)
print("accuracy=%g" %(acc))
accuracy=0.75507
# CONFUSION MATRIX
from pyspark.mllib.evaluation import MulticlassMetrics
pred_label=LR_predictions.select( 'label', 'prediction').rdd
metrics = MulticlassMetrics(pred_label)
print(metrics.confusionMatrix())
DenseMatrix([[365.,  76.],
             [ 81., 119.]])
#PRECISION, RECALL and F1SCORE
cm=metrics.confusionMatrix().toArray()
precision=(cm[0][0])/(cm[0][0]+cm[1][0])
recall=(cm[0][0])/(cm[0][0]+cm[0][1])
f1score =((2*precision*recall )/ (precision + recall))
print("Logistic regression:--precision,recall,f1score",precision,recall,f1score)
Logistic regression:--precision,recall,f1score 0.8183856502242153 0.8276643990929705 0.8229988726042842
```

```
#DECISION TREE CLASSIFIER
#Import Decision Tree Classifier model
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(featuresCol = 'features', labelCol = 'label', maxDepth = 3)
#Fitting to train and test data
DT_Model = dt.fit(train)
prediction1= DT_Model.transform(test)
prediction1.select( 'label', 'rawPrediction', 'prediction', 'probability').show(10)


+-----+-------------+----------+--------------------+
|label|rawPrediction|prediction|         probability|
+-----+-------------+----------+--------------------+
|  1.0| [655.0,63.0]|       0.0|[0.91225626740947...|
|  1.0|[167.0,294.0]|       1.0|[0.36225596529284...|
|  1.0|[167.0,294.0]|       1.0|[0.36225596529284...|
|  1.0| [199.0,98.0]|       0.0|[0.67003367003367...|
|  1.0| [199.0,98.0]|       0.0|[0.67003367003367...|
|  1.0| [199.0,98.0]|       0.0|[0.67003367003367...|
|  1.0| [199.0,98.0]|       0.0|[0.67003367003367...|
|  1.0| [199.0,98.0]|       0.0|[0.67003367003367...|
|  1.0|[167.0,294.0]|       1.0|[0.36225596529284...|
|  1.0|[167.0,294.0]|       1.0|[0.36225596529284...|
+-----+-------------+----------+--------------------+
only showing top 10 rows

#CALCULATING AREA UNDER ROC
evaluator = BinaryClassificationEvaluator()
print("Test Area Under ROC: " + str(evaluator.evaluate(prediction1, {evaluator.metricName: "areaUnderROC"})))
Test Area Under ROC: 0.7771760377141542
#CALCULATING ACCURACY USING MULTICLASS EVALUATOR
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
eval2 = MulticlassClassificationEvaluator(predictionCol='prediction',labelCol='label', metricName='accuracy')
acc1 = eval2.evaluate(prediction1)
print("accuracy=%g" %(acc1))
```

```
accuracy=0.764431
#PRINTING CONFUSION MATRIX FOR DECISION TREE MODEL
from pyspark.mllib.evaluation import MulticlassMetrics
pred_label1=prediction1.select( 'label', 'prediction').rdd
metrics1 = MulticlassMetrics(pred_label1)
print(metrics1.confusionMatrix())
DenseMatrix([[364.,  69.],
             [ 82., 126.]])
#METRICS FUNCTION TO EVALUATE RECALL, PRECISION AND F1SCORE
cm=metrics1.confusionMatrix().toArray()
precision=(cm[0][0])/(cm[0][0]+cm[1][0])
recall=(cm[0][0])/(cm[0][0]+cm[0][1])
f1score =((2*precision*recall )/ (precision + recall))
print("Decision Tree:precision,recall,f1score",precision,recall,f1score)
Decision Tree:precision,recall,f1score 0.8161434977578476 0.8406466512702079 0.8282138794084187
#PRINTING ALL THE IMPORTANT FEATURES
DT_Model.featureImportances
SparseVector(23, {1: 0.0075, 3: 0.0282, 15: 0.1401, 17: 0.7534, 21: 0.0597, 22: 0.0112})
#Naive Bayes Classifier
#IMPORTING NAIVE BAYES PACKAGE
from pyspark.ml.classification import NaiveBayes

#SELECTING NORMALIZED COLUMNS FOR MODELLING
nb=predictions.select('label','job_encoded_SCALED','marital_encoded_SCALED','loan_encoded_SCALED','default_encoded_SCALED','education_encoded_SCALED','housing_encoded_SCALED',
'poutcome_encoded_SCALED','features_SCALED',"prediction")
#Renaming features_SCALED  to  'features'
nb = nb.selectExpr("label as label","features_SCALED as features")
nb.show(3)
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|(23,[5,10,12,14,1...|
|  0.0|(23,[1,10,12,14,1...|
|  0.0|(23,[2,10,13,14,1...|
+-----+--------------------+
only showing top 3 rows
```

```
#SPLITTING THE LABEL AND FEATURES DATA AS TRAIN AND TEST DATA
train_1, test_1 = nb.randomSplit([0.7, 0.3], seed = 742)
#Naive bayes fitting to train and test data
#USING MULTINOMIAL METHOD BECAUSE BERNOULLI REQUIRES ONLY BINARY INPUT
nb1=NaiveBayes(modelType="multinomial")
nbmodel=nb1.fit(train_1)
nb_predictions=nbmodel.transform(test_1)

nb_evaluator=MulticlassClassificationEvaluator(labelCol="label",predictionCol="prediction",metricName="accuracy")

#EVALUATING PREDICTION AND PROBABILITY VALUES
nb_accuracy=nb_evaluator.evaluate(nb_predictions)
nb_predictions.select( 'label', 'rawPrediction', 'prediction', 'probability').show(10)


print("Test Area Under ROC: " + str(evaluator.evaluate(nb_predictions, {evaluator.metricName: "areaUnderROC"})))
+-----+--------------------+----------+--------------------+
|label|       rawPrediction|prediction|         probability|
+-----+--------------------+----------+--------------------+
|  0.0|[-18.128101289108...|       0.0|[0.71227863685887...|
|  0.0|[-18.905025587194...|       0.0|[0.83308795651024...|
|  0.0|[-17.210598336493...|       0.0|[0.83078085549233...|
|  0.0|[-16.368239992271...|       0.0|[0.82011846673742...|
|  0.0|[-18.122511379455...|       0.0|[0.51656396890790...|
|  0.0|[-18.379947725811...|       0.0|[0.52612872952829...|
|  0.0|[-17.219309198880...|       0.0|[0.54173239419087...|
|  0.0|[-19.904541403547...|       0.0|[0.80183300993376...|
|  0.0|[-22.618681384194...|       0.0|[0.76733814845131...|
|  0.0|[-20.750414292819...|       0.0|[0.80281500588117...|
+-----+--------------------+----------+--------------------+
only showing top 10 rows

Test Area Under ROC: 0.5786054421768712
```

```
#CALCULATING ACCURACY
print("accuracy=%g" %(nb_accuracy))
accuracy=0.74415
#Confusion matrix
from pyspark.mllib.evaluation import MulticlassMetrics
pred_and_label=nb_predictions.select( 'label', 'prediction').rdd
metrics2 = MulticlassMetrics(pred_and_label)
print(metrics2.confusionMatrix())
DenseMatrix([[386., 109.],
             [ 55.,  91.]])
#CALCULATING PRECISION, RECALL AND F1SCORE
cm2=metrics2.confusionMatrix().toArray()
precision=(cm2[0][0])/(cm2[0][0]+cm2[1][0])
recall=(cm2[0][0])/(cm2[0][0]+cm2[0][1])
f1score =((2*precision*recall )/ (precision + recall))

print("NAIVE BAYES model:precision,recall,f1score",precision,recall,f1score)
NAIVE BAYES model:precision,recall,f1score 0.8752834467120182 0.7797979797979798 0.8247863247863249
```