bank_marketing_campaign_predictive_analytics

August 12, 2023

1 Bank Marketing Campaign Predictive Analytics

1.1 Abstract

Predictive analytics plays a crucial role in modern bank marketing campaigns. By harnessing the power of data and advanced analytical techniques, this project aims to develop a predictive model to enhance the effectiveness of marketing campaigns in the banking industry. The project leverages historical customer data, including demographics, transaction history, and previous marketing campaign responses, to build a predictive model that can accurately identify potential customers who are more likely to respond positively to future marketing efforts. Through the application of machine learning algorithms and statistical modeling techniques, this project aims to predict customer behavior and preferences, allowing banks to optimize their marketing strategies and resources. By identifying the most promising leads, the predictive model assists banks in allocating marketing budgets effectively, tailoring personalized offers, and designing targeted campaigns to maximize customer engagement and conversion rates. The developed predictive model not only helps in identifying potential customers but also enables the bank to understand the key factors that drive customer responses. By analyzing the significant predictors, such as customer demographics, transaction patterns, and previous campaign interactions, banks can gain valuable insights into customer preferences and behaviors. This information facilitates the development of customer-centric marketing strategies, enabling banks to offer personalized products and services that meet individual needs and increase customer satisfaction. The outcomes of this project have the potential to revolutionize bank marketing campaigns by providing data-driven insights and predictions. By leveraging predictive analytics, banks can optimize their marketing efforts, reduce costs, and improve overall campaign efficiency. Moreover, the project contributes to the enhancement of customer experiences, fostering long-term customer relationships, and increasing customer loyalty. In conclusion, this project showcases the power of predictive analytics in bank marketing campaigns. By utilizing historical customer data and advanced analytical techniques, the project aims to develop a predictive model that enables banks to identify potential customers, understand their preferences, and design targeted marketing strategies. The integration of predictive analytics in bank marketing has the potential to transform customer acquisition and retention processes, leading to improved business outcomes and customer satisfaction in the banking industry.

1.2 Keywords

Pandas, NumPy, Matplotlib, Seaborn, Feature Extraction, Algorithm, accuracy prediction technique

1.3 Technology

Data Science & Machine Learning

1.4 Problem Statement

There has been a revenue decline for the Portuguese bank and they would like to know what actions to take. After investigation, we found out that the root cause is that their clients are not depositing as frequently as before. Knowing that term deposits allow banks to hold onto a deposit for a specific amount of time, so banks can invest in higher gain financial products to make a profit. In addition, banks also hold better chance to persuade term deposit clients into buying other products such as funds or insurance to further increase their revenues. As a result, the Portuguese bank would like to identify existing clients that have higher chance to subscribe for a term deposit and focus marketing effort on such clients.

1.5 About Dataset

It is a dataset that describing Portugal bank marketing campaigns results.Conducted campaigns were based mostly on direct phone calls, offering bank client to place a term deposit. If after all marking afforts client had agreed to place deposit - target variable marked 'yes', otherwise 'no'

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

1.6 What I will do with all this information?

With all this info, I will analyze the Bank lead's dataset and create a classification algorithm with full end feature engineering and EDA

1.7 Project Summary

My name is Sunil Ghanchi and I'm a Data Science & Machine Learning Intern of Brainybeam Info-Tech PVT LTD. The Portugal Bank approached our service and requested us to create a classfication algorithm to automatically place their prospective leads on having a term deposit in their bank. We will be creating a classification algorithm and also suggest them the insights we derive from this dataset and also help them to narrow down their leads into marketing funnel and in the end make a term deposit.

1.8 Objectives of project

- Meet and Greet Data
- Prepare the Data for consumption (Feature Engineering and Selection)
- Perform Exploratory Analysis (Visualizations)
- Model the Data using Machine Learning
- Validate and implement data model
- Optimize and Strategize

1.9 Prepare Data for Consumption

1.9.1 Import Libraries

We will import all the necessary libraries that we are going to use in this project

[176]:
```
#manipulation library
import pandas as pd
import numpy as np

#visulization library
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
%matplotlib inline

#machine learning library
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.pipeline import make_pipeline

#metrices library
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

#ignore warning library
import warnings
warnings.filterwarnings('ignore')
```

1.10 Meet and Greet Data

In this phase we will import csv data and analyis it

[6]:
```
bank = pd.read_csv(r"D:\BrainyBeam Internship\Project\bank-additional-full.
↪csv", sep=';')
```

[9]:
```
bank_copy = bank.copy()
bank_copy
```

[9]:
```
       age          job  marital            education  default housing loan  \
0       56    housemaid  married             basic.4y       no      no   no
1       57     services  married          high.school  unknown      no   no
2       37     services  married          high.school       no     yes   no
3       40       admin.  married             basic.6y       no      no   no
4       56     services  married          high.school       no      no  yes
…     …           …       …                   …      …      …  …
41183   73      retired  married  professional.course       no     yes   no
41184   46  blue-collar  married  professional.course       no      no   no
```

```
41185   56      retired  married    university.degree       no      yes   no
41186   44   technician  married  professional.course       no       no   no
41187   74      retired  married  professional.course       no      yes   no

         contact month day_of_week  …  campaign  pdays  previous  \
0      telephone   may         mon  …         1    999         0
1      telephone   may         mon  …         1    999         0
2      telephone   may         mon  …         1    999         0
3      telephone   may         mon  …         1    999         0
4      telephone   may         mon  …         1    999         0
…            …   …           …   …       …     …         …
41183   cellular   nov         fri  …         1    999         0
41184   cellular   nov         fri  …         1    999         0
41185   cellular   nov         fri  …         2    999         0
41186   cellular   nov         fri  …         1    999         0
41187   cellular   nov         fri  …         3    999         1

          poutcome emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0      nonexistent          1.1          93.994          -36.4      4.857
1      nonexistent          1.1          93.994          -36.4      4.857
2      nonexistent          1.1          93.994          -36.4      4.857
3      nonexistent          1.1          93.994          -36.4      4.857
4      nonexistent          1.1          93.994          -36.4      4.857
…              …          …             …            …        …
41183  nonexistent         -1.1          94.767          -50.8      1.028
41184  nonexistent         -1.1          94.767          -50.8      1.028
41185  nonexistent         -1.1          94.767          -50.8      1.028
41186  nonexistent         -1.1          94.767          -50.8      1.028
41187      failure         -1.1          94.767          -50.8      1.028

       nr.employed    y
0           5191.0   no
1           5191.0   no
2           5191.0   no
3           5191.0   no
4           5191.0   no
…              …  …
41183       4963.6  yes
41184       4963.6   no
41185       4963.6   no
41186       4963.6  yes
41187       4963.6   no

[41188 rows x 21 columns]
```

[11]:
```python
print("The shape of bank csv is (Rows, Columns):", bank_copy.shape)
bank_copy.info()
```

```
The shape of bank csv is (Rows, Columns): (41188, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

[27]:
```
print("Sum of how many null values we have in each columns:",bank_copy.isnull().
 ↪sum(), sep='\n')
```

```
Sum of how many null values we have in each columns:
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
```

```
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

Dataset In our dataset we have 41188 instances and 21 features. We also check down the sum of null value, so we have not a single null value in our dataset. Let's Analyis the each columns what it contains.

Bank Client data

1. Age: Age of the lead (numeric)
2. Job : type of job (Categorical)
3. Marital : Marital status (Categorical)
4. Education : Educational Qualification of the lead (Categorical)
5. Default: Does the lead has any default(unpaid)credit (Categorical)
6. Housing: Does the lead has any housing loan? (Categorical)
7. Loan: Does the lead has any personal loan? (Categorical)

Related with the last contact of the current campaign

8. Contact: Contact communication type (Categorical)
9. Month: last contact month of year (Categorical)
10. day_of_week: last contact day of the week (categorical)
11. duration: last contact duration, in seconds (numeric).

Important: Duration 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.

Other attributes

12. campaign: number of contacts performed during this campaign and for this client (numeric)
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)

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

Let's take the general overview of our dataset

[17]:
```
bank_copy.head()
```

[17]:
```
   age        job  marital    education  default housing loan    contact  \
0   56  housemaid  married     basic.4y       no      no   no  telephone
1   57   services  married  high.school  unknown      no   no  telephone
2   37   services  married  high.school       no     yes   no  telephone
3   40     admin.  married     basic.6y       no      no   no  telephone
4   56   services  married  high.school       no      no  yes  telephone

  month day_of_week  …  campaign  pdays  previous     poutcome emp.var.rate  \
0   may         mon  …         1    999         0  nonexistent          1.1
1   may         mon  …         1    999         0  nonexistent          1.1
2   may         mon  …         1    999         0  nonexistent          1.1
3   may         mon  …         1    999         0  nonexistent          1.1
4   may         mon  …         1    999         0  nonexistent          1.1

   cons.price.idx  cons.conf.idx  euribor3m  nr.employed   y
0          93.994          -36.4      4.857       5191.0  no
1          93.994          -36.4      4.857       5191.0  no
2          93.994          -36.4      4.857       5191.0  no
3          93.994          -36.4      4.857       5191.0  no
4          93.994          -36.4      4.857       5191.0  no

[5 rows x 21 columns]
```

[18]:
```
bank_copy.dtypes
```

[18]:
```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
```

```
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
```

[19]:
```
#statistical paramaters
bank_copy.describe()
```

[19]:
```
               age      duration      campaign         pdays      previous  \
count  41188.00000  41188.000000  41188.000000  41188.000000  41188.000000
mean      40.02406    258.285010      2.567593    962.475454      0.172963
std       10.42125    259.279249      2.770014    186.910907      0.494901
min       17.00000      0.000000      1.000000      0.000000      0.000000
25%       32.00000    102.000000      1.000000    999.000000      0.000000
50%       38.00000    180.000000      2.000000    999.000000      0.000000
75%       47.00000    319.000000      3.000000    999.000000      0.000000
max       98.00000   4918.000000     56.000000    999.000000      7.000000

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m   nr.employed
count  41188.000000    41188.000000   41188.000000  41188.000000  41188.000000
mean       0.081886       93.575664     -40.502600      3.621291   5167.035911
std        1.570960        0.578840       4.628198      1.734447     72.251528
min       -3.400000       92.201000     -50.800000      0.634000   4963.600000
25%       -1.800000       93.075000     -42.700000      1.344000   5099.100000
50%        1.100000       93.749000     -41.800000      4.857000   5191.000000
75%        1.400000       93.994000     -36.400000      4.961000   5228.100000
max        1.400000       94.767000     -26.900000      5.045000   5228.100000
```

[23]:
```
#let's print the categories and it's respective count values
print("Job:", bank_copy.job.value_counts(), sep='\n')
print("-"*40)
print("Marital:", bank_copy.marital.value_counts(), sep='\n')
print("-"*40)
print("Education:", bank_copy.education.value_counts(), sep='\n')
print("-"*40)
print("Default:", bank_copy.default.value_counts(), sep='\n')
print("-"*40)
print("Housing:", bank_copy.housing.value_counts(), sep='\n')
print("-"*40)
print("Loan:", bank_copy.loan.value_counts(), sep='\n')
print("-"*40)
print("Contact:", bank_copy.contact.value_counts(), sep='\n')
print("-"*40)
print("Month:", bank_copy.month.value_counts(), sep='\n')
print("-"*40)
```

```python
print("Days:", bank_copy.day_of_week.value_counts(), sep='\n')
print("-"*40)
print("Previous Outcome:", bank_copy.poutcome.value_counts(), sep='\n')
print("-"*40)
print("Outcome of this Compaign:", bank_copy.y.value_counts(), sep='\n')
print("-"*40)
```

```
Job:
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
----------------------------------------
Marital:
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
----------------------------------------
Education:
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
----------------------------------------
Default:
no         32588
unknown     8597
yes            3
Name: default, dtype: int64
----------------------------------------
Housing:
yes        21576
```

```
no         18622
unknown      990
Name: housing, dtype: int64
----------------------------------------
Loan:
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
----------------------------------------
Contact:
cellular     26144
telephone    15044
Name: contact, dtype: int64
----------------------------------------
Month:
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: month, dtype: int64
----------------------------------------
Days:
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: day_of_week, dtype: int64
----------------------------------------
Previous Outcome:
nonexistent    35563
failure         4252
success         1373
Name: poutcome, dtype: int64
----------------------------------------
Outcome of this Compaign:
no     36548
yes     4640
Name: y, dtype: int64
----------------------------------------
```

Insights:

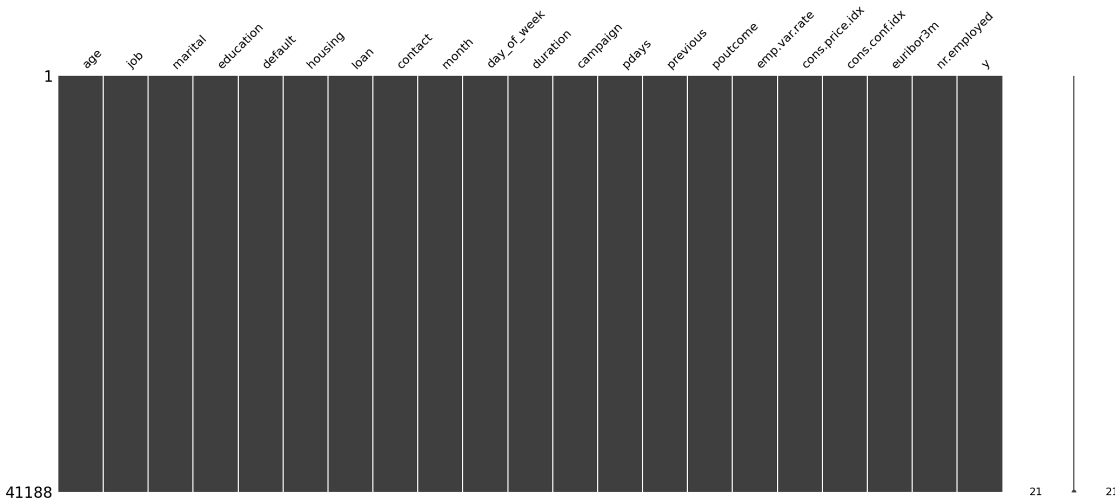
- We got unknown category in each feature, we should figure out how to deal with that
- This campaign only operated during weekdays
- I can't understand what is non-existent category in previous outcome aka poutcome, so I will ignore it because we don't want it as of now

1.11 Data Cleaning

Checking Missing Values with graph and func

[24]:
```python
import missingno as msno
msno.matrix(bank_copy)
```

[24]:
```
<AxesSubplot: >
```



As from visulize we don't have any null values, for confirmation in numbers we saw above, let's do it again

[26]:
```python
print("Sum of how many null values we have in each columns:",bank_copy.isnull().
↪sum(), sep='\n')
```

```
Sum of how many null values we have in each columns:
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
```

```
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

So by this we confirm that we don't have any null values.

1.12 Data Visulization

We have much numerical data, let's plot the graph to visulize for our machine learning models and also figure out which feature are important and drop the unimportant features.

Duration of Calls Vs Job Roles

[30]:
```
bank_copy
```

[30]:
```
       age          job  marital            education  default housing loan  \
0       56    housemaid  married             basic.4y       no      no   no
1       57     services  married          high.school  unknown      no   no
2       37     services  married          high.school       no     yes   no
3       40       admin.  married             basic.6y       no      no   no
4       56     services  married          high.school       no      no  yes
…     …          …        …                    …       …      …  …
41183   73      retired  married  professional.course       no     yes   no
41184   46  blue-collar  married  professional.course       no      no   no
41185   56      retired  married    university.degree       no     yes   no
41186   44   technician  married  professional.course       no      no   no
41187   74      retired  married  professional.course       no     yes   no

         contact month day_of_week  …  campaign  pdays  previous  \
0      telephone   may         mon  …         1    999         0
1      telephone   may         mon  …         1    999         0
2      telephone   may         mon  …         1    999         0
3      telephone   may         mon  …         1    999         0
4      telephone   may         mon  …         1    999         0
…           …   …          …  …       …     …        …
41183   cellular   nov         fri  …         1    999         0
41184   cellular   nov         fri  …         1    999         0
41185   cellular   nov         fri  …         2    999         0
41186   cellular   nov         fri  …         1    999         0
```

```
41187   cellular   nov         fri  …         3    999         1

           poutcome  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0       nonexistent           1.1          93.994          -36.4      4.857
1       nonexistent           1.1          93.994          -36.4      4.857
2       nonexistent           1.1          93.994          -36.4      4.857
3       nonexistent           1.1          93.994          -36.4      4.857
4       nonexistent           1.1          93.994          -36.4      4.857
…               …           …             …             …         …
41183   nonexistent          -1.1          94.767          -50.8      1.028
41184   nonexistent          -1.1          94.767          -50.8      1.028
41185   nonexistent          -1.1          94.767          -50.8      1.028
41186   nonexistent          -1.1          94.767          -50.8      1.028
41187       failure          -1.1          94.767          -50.8      1.028

       nr.employed    y
0           5191.0   no
1           5191.0   no
2           5191.0   no
3           5191.0   no
4           5191.0   no
…              …   …
41183       4963.6  yes
41184       4963.6   no
41185       4963.6   no
41186       4963.6  yes
41187       4963.6   no

[41188 rows x 21 columns]
```

[32]:
```python
job = bank_copy.groupby('job').sum()['duration']
```

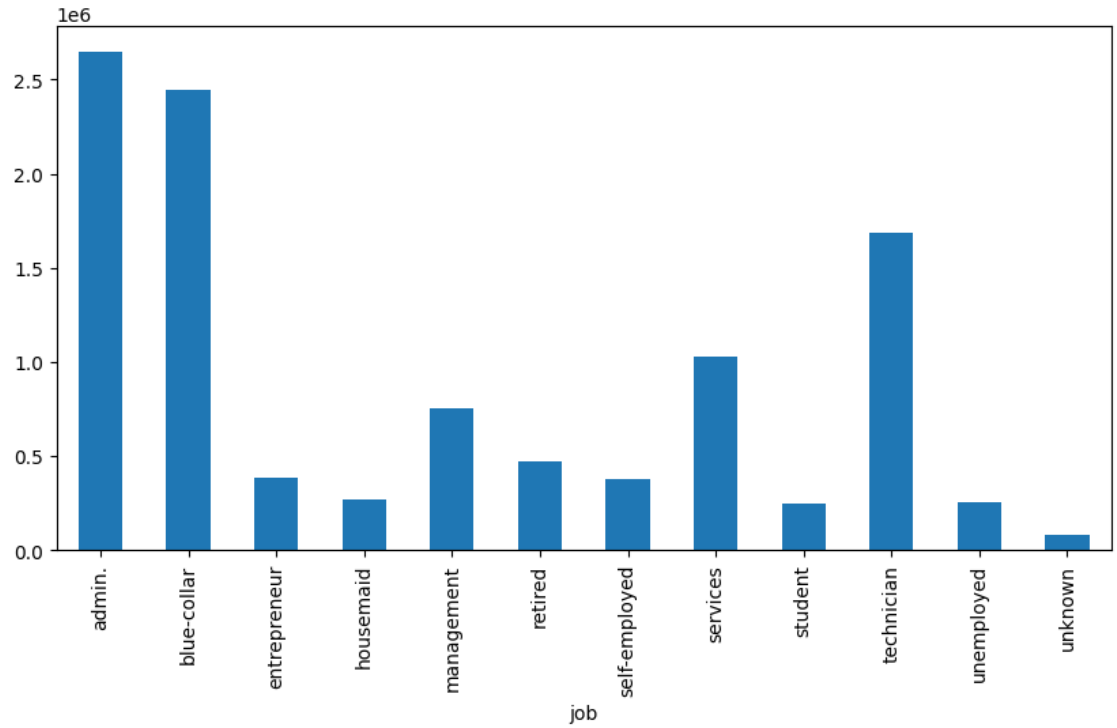
[33]:
```python
job
```

[33]:
```
job
admin.           2650441
blue-collar      2448075
entrepreneur      383318
housemaid         265482
management        751638
retired           470785
self-employed     375346
services         1025582
student           248223
technician       1687316
unemployed        252944
unknown            79093
```

Name: duration, dtype: int64

[34]:
```
job.plot.bar(x='job', y='duration', figsize=(10,5))
```

[34]: <AxesSubplot: xlabel='job'>



Insights

- We can see this by above graph that Admin job category is on top followed by Blue-Collar

Campaign Vs Duration

[36]:
```
campaign = bank_copy.groupby('campaign').sum()['duration']
```

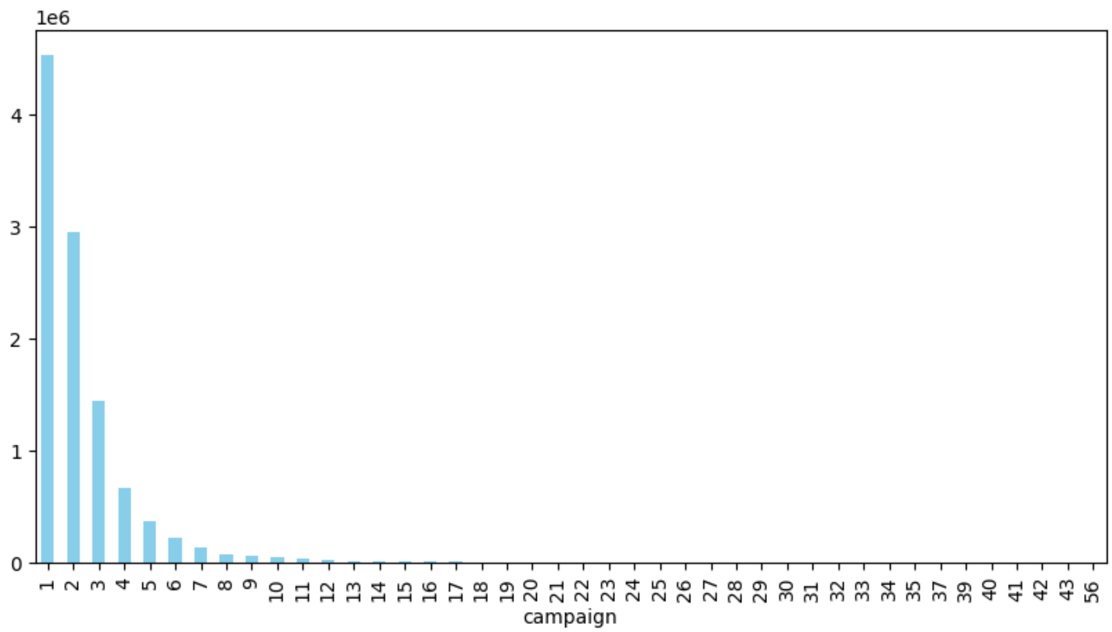
[37]:
```
campaign
```

[37]:
```
campaign
1       4529150
2       2956397
3       1442081
4        666732
5        364187
6        221210
7        140475
8         75810
```

```
9      59862
10     46959
11     36767
12     23161
13     16126
14      9287
15      7752
16      5985
17     11557
18      2819
19      4282
20      1867
21      1982
22      1930
23      2067
24      1672
25       367
26      2445
27      1110
28       946
29      1180
30       483
31       235
32       121
33       150
34       111
35       248
37        17
39        44
40        31
41        25
42       271
43        81
56       261
Name: duration, dtype: int64
```

[40]:
```
campaign.plot.bar(x='campaign', y='duration', figsize=(10,5), color='skyblue')
```

[40]:
```
<AxesSubplot: xlabel='campaign'>
```



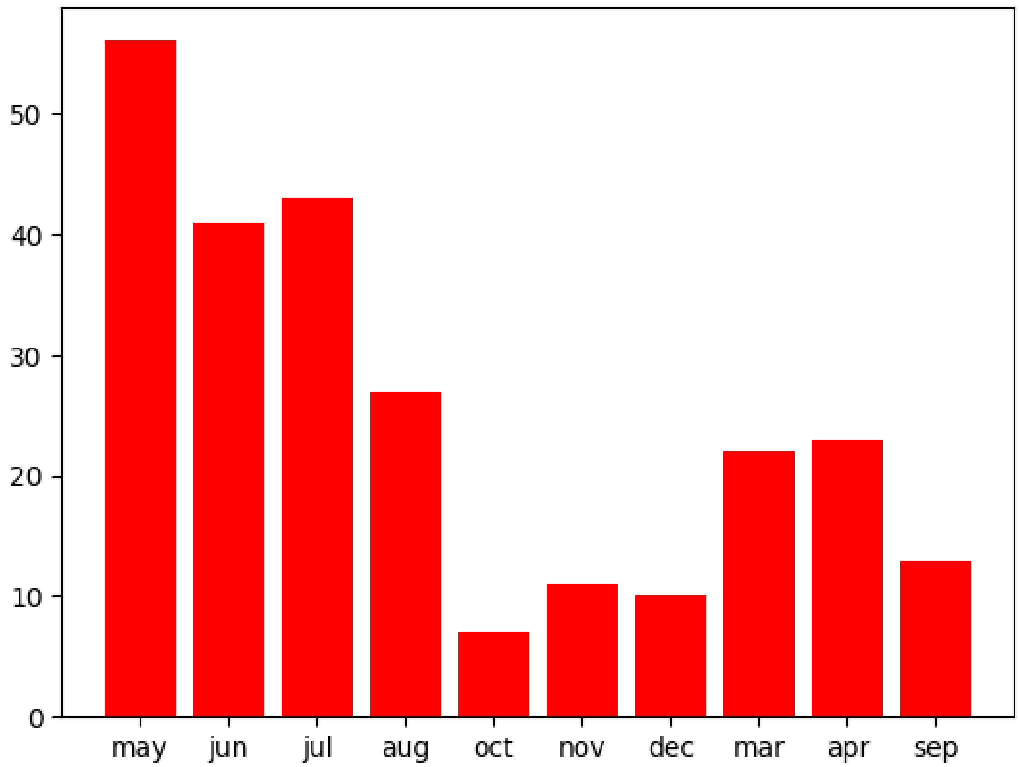
Insights

- By this we can see that in the initial days of Campaign there were many positive leads
- Duration is faded as the Campaign extended

Campaign Vs Month

[45]:
```
plt.bar(bank_copy['month'], bank_copy['campaign'], color='red')
```

[45]: <BarContainer object of 41188 artists>

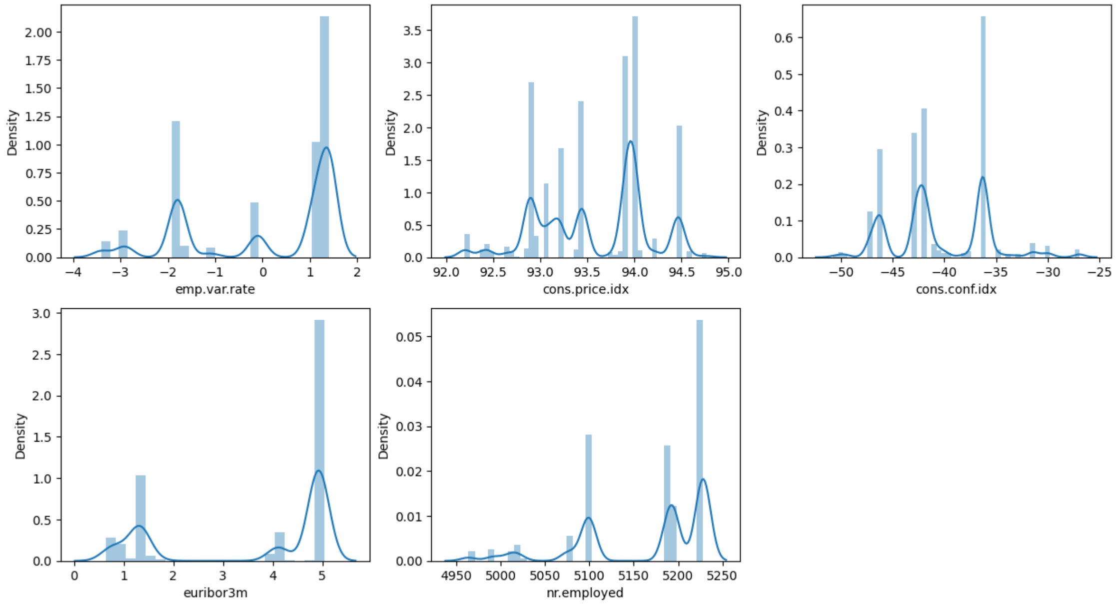


Insights

- As we can see that in the Starting period of new quarter of banking (may, june, july), the campaign were mostly concentrated.
- That period is also the starting period of Schools and college for new classes so there is a possibilites that parents make deposits in name of their children.
- Campaign is also active in end of bank period.

Distribution of Quarterly Indicators

[49]:
```
fig, ax=plt.subplots(2, 3, figsize=(15, 8))
sns.distplot(bank_copy['emp.var.rate'], ax=ax[0,0])
sns.distplot(bank_copy['cons.price.idx'], ax=ax[0,1])
sns.distplot(bank_copy['cons.conf.idx'], ax=ax[0,2])
sns.distplot(bank_copy['euribor3m'], ax=ax[1,0])
sns.distplot(bank_copy['nr.employed'], ax=ax[1,1])
ax[1, 2].axis('off')
plt.show()
```

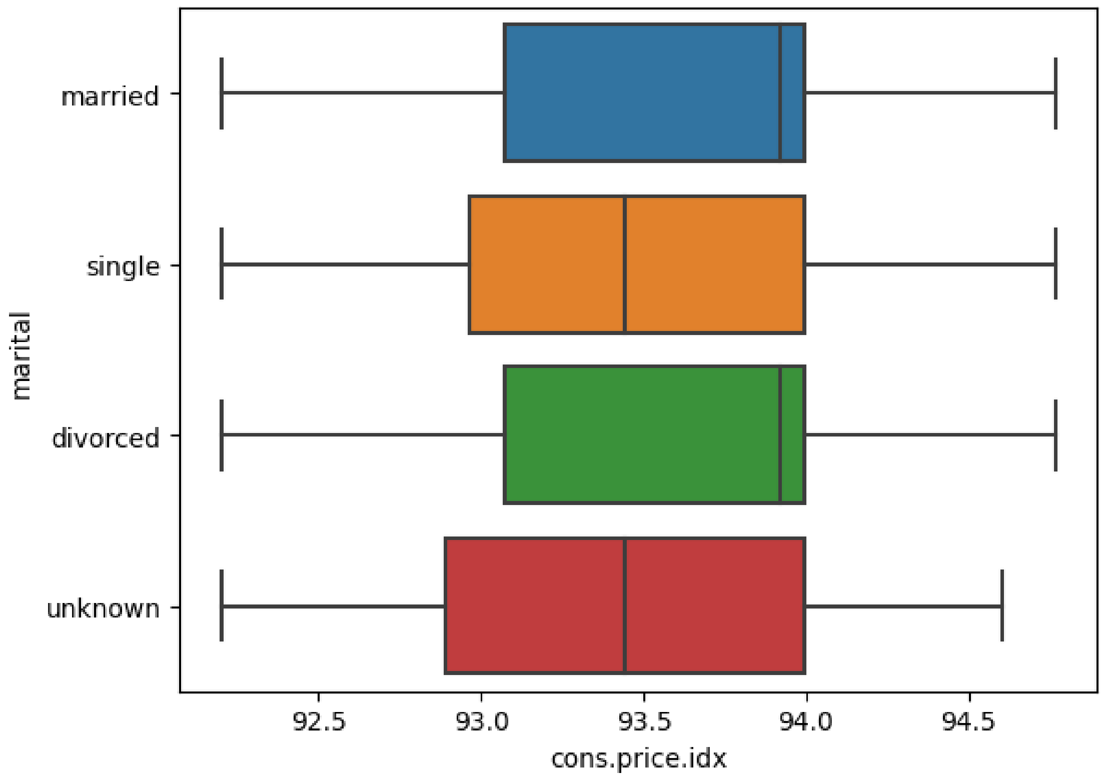


Insights

- We can see there is a high employee variation rate which signifies that they have made the campaign when there were high shifts in job due to conditions of economy
- The Consumer price index is also good which shows the leads where having good price to pay for goods and services may be that could be the reason to stimulate these leads into making a deposit and plant the idea of savings
- Consumer confidence index is pretty low as they don't have much confidence on the fluctuating economy
- The 3 month Euribor interest rate is the interest rate at which a selection of European banks lend one another funds denominated in euros whereby the loans have a maturity of 3 months. In our case the interest rates are high for lending their loans
- The number of employees were also at peak which can increase their income index that could be the reason the campaign targetted the leads who were employeed to make a deposit

Marital Status Vs Price Index

[52]:
```
sns.boxplot(x= bank_copy['cons.price.idx'], y= bank_copy['marital'])
plt.show()
```



Insights

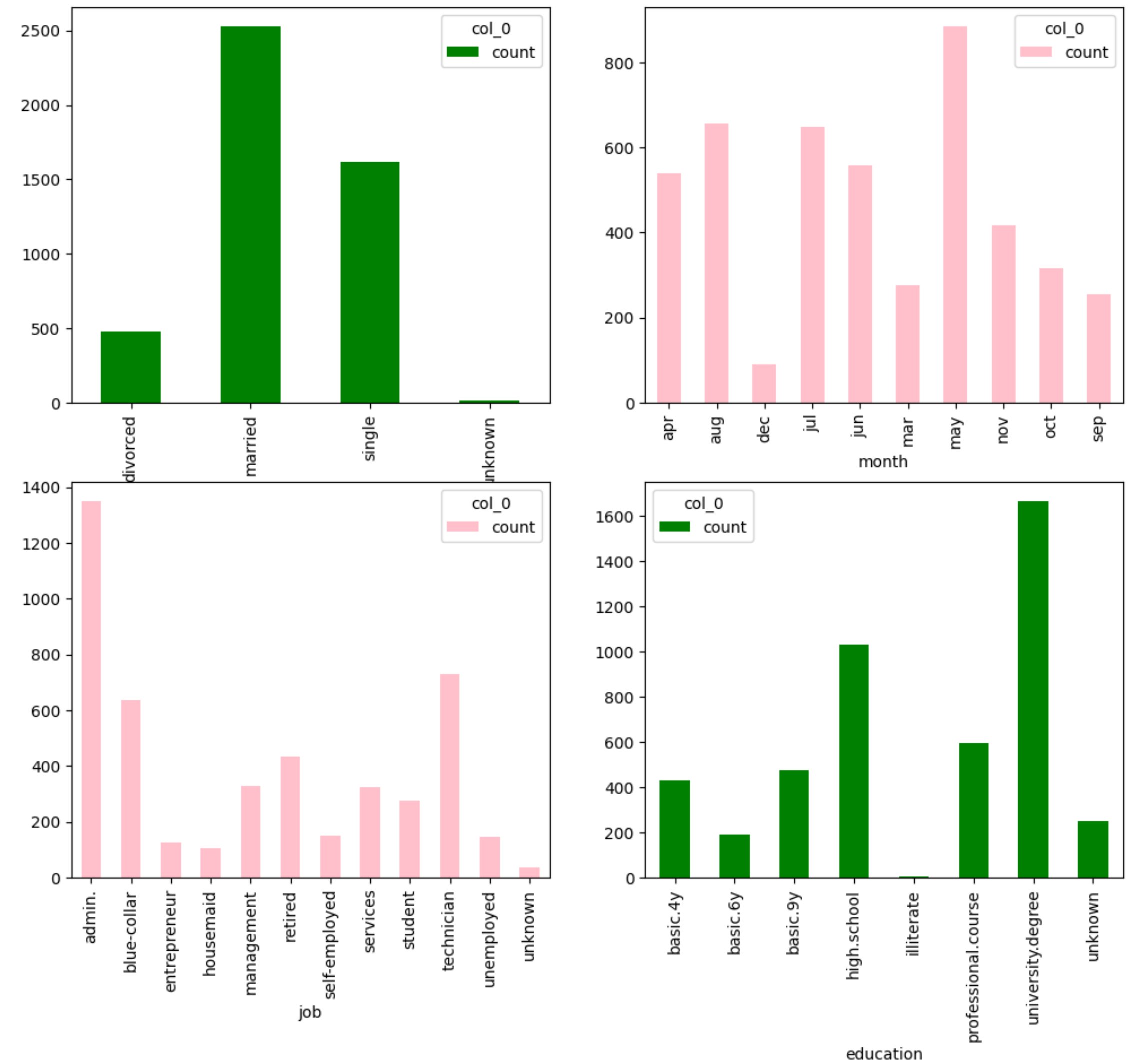
- There is no much difference in price index.
- Married have an upper hand in they have index contributing as couple.

Positive Deopsits Vs Attributes

[66]:
```
bank_yes = bank_copy[bank_copy['y']=='yes']

df1 = pd.crosstab(index = bank_yes['marital'], columns='count')
df2 = pd.crosstab(index = bank_yes['month'], columns='count')
df3 = pd.crosstab(index = bank_yes['job'], columns='count')
df4 = pd.crosstab(index = bank_yes['education'], columns='count')

fig, ax= plt.subplots(2, 2, figsize=(12,10))
df1.plot.bar(ax=ax[0,0], color='green')
df2.plot.bar(ax=ax[0,1], color='pink')
df3.plot.bar(ax=ax[1,0], color='pink')
df4.plot.bar(ax=ax[1,1], color='green')
plt.show()
```

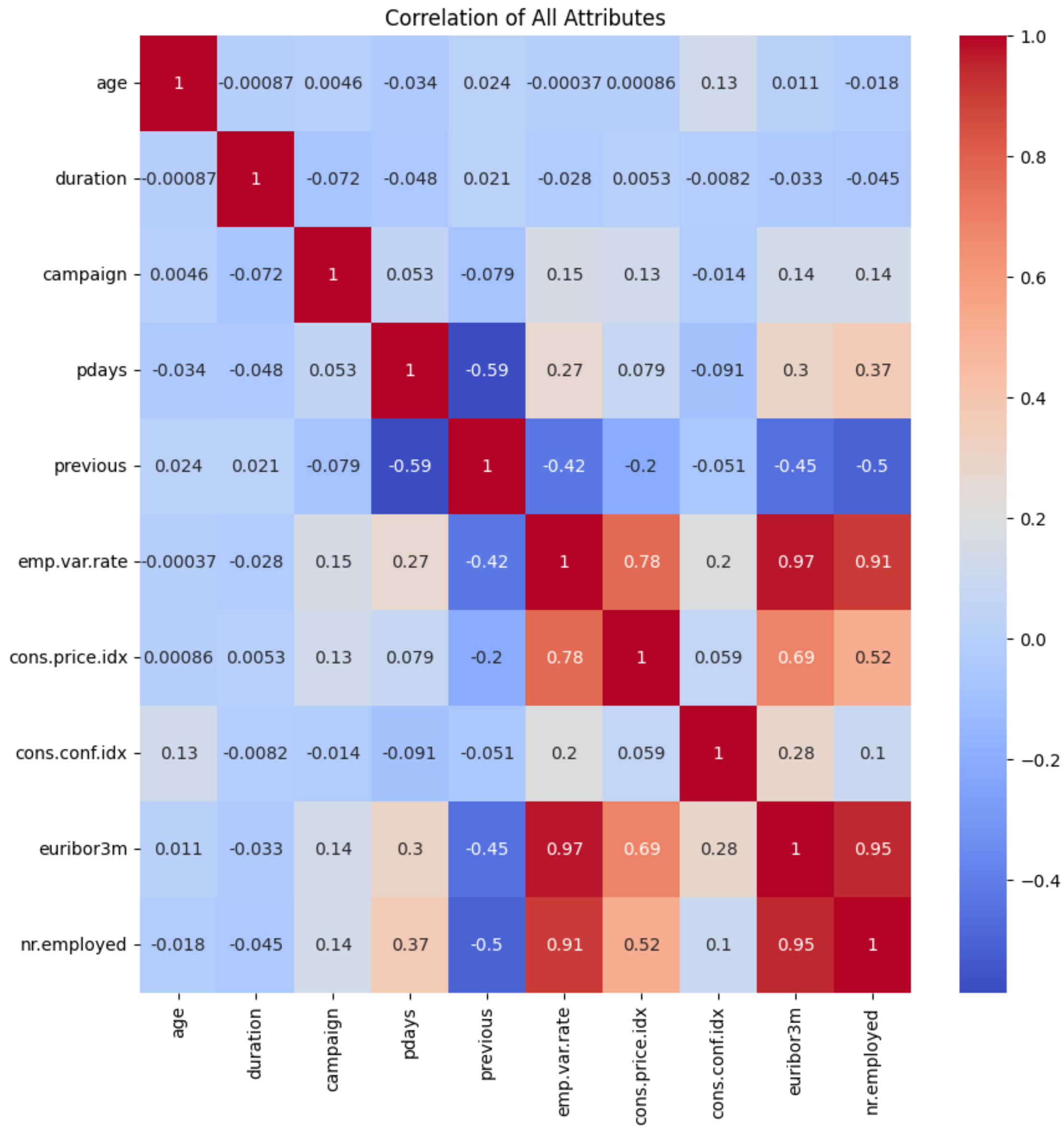


Insights

- We see that married have high deposits among all
- In may month there were much deposits as it is starting of banking period
- In job role Admin has high deposits followed by Technician
- In education Degree student has much deposits

Correlation of All Attributes

[68]:
```
plt.figure(figsize=(10,10))
sns.heatmap(bank_copy.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation of All Attributes")
plt.show()
```



Insights

- The indicators have correlation among themselves
- Number of employees rate is highly correlated with employee variation rate
- Consumer price index is highly correlated with bank interest rate(higher the price index, higher the interest rate)
- Employee variation rate also correlates with the bank interest rates

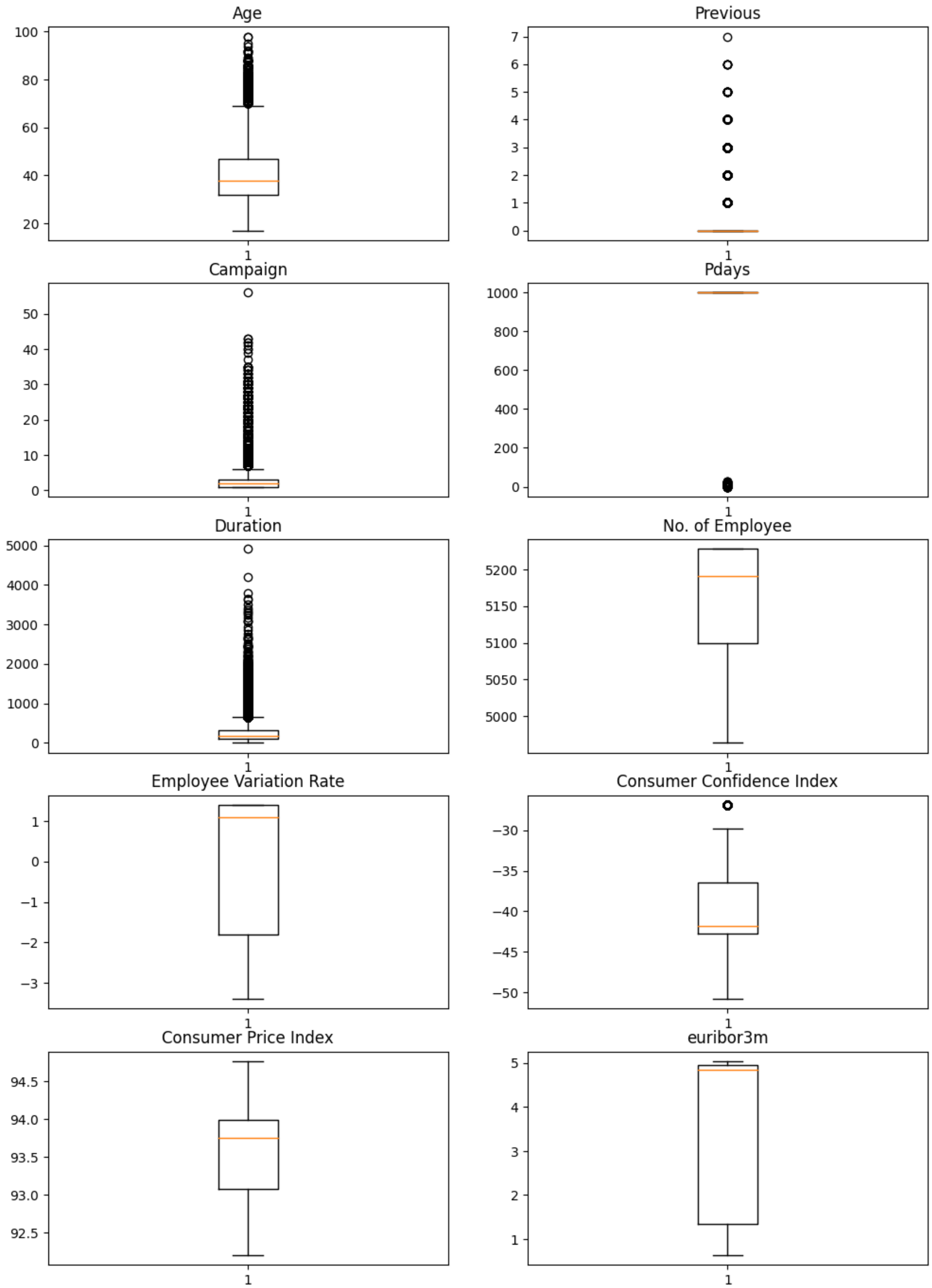
1.13 Feature Engineering

1.13.1 Handling Outliers

Let's check the outliers with Boxplot

[80]:

```
fig, ax= plt.subplots(5, 2, figsize=(12,17))
ax[0,0].boxplot(bank_copy['age'])
ax[0,0].set_title("Age")
ax[0,1].boxplot(bank_copy['previous'])
ax[0,1].set_title("Previous")
ax[1,0].boxplot(bank_copy['campaign'])
ax[1,0].set_title("Campaign")
ax[1,1].boxplot(bank_copy['pdays'])
ax[1,1].set_title("Pdays")
ax[2,0].boxplot(bank_copy['duration'])
ax[2,0].set_title("Duration")
ax[2,1].boxplot(bank_copy['nr.employed'])
ax[2,1].set_title("No. of Employee")
ax[3,0].boxplot(bank_copy['emp.var.rate'])
ax[3,0].set_title("Employee Variation Rate")
ax[3,1].boxplot(bank_copy['cons.conf.idx'])
ax[3,1].set_title("Consumer Confidence Index")
ax[4,0].boxplot(bank_copy['cons.price.idx'])
ax[4,0].set_title("Consumer Price Index")
ax[4,1].boxplot(bank_copy['euribor3m'])
ax[4,1].set_title("euribor3m")
plt.show()
```


Age
Previous
Campaign
Pdays
Duration
No. of Employee
Employee Variation Rate
Consumer Confidence Index
Consumer Price Index
euribor3m

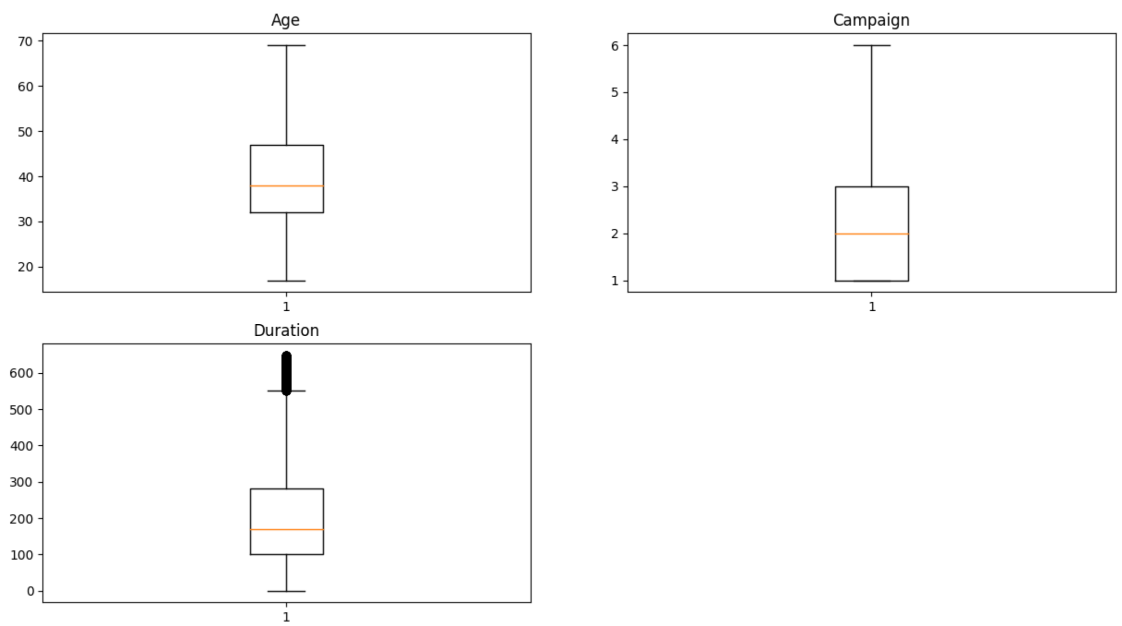
Insights

- We see that many features doesn't have much outliers except for age,duration and campaign. So, let's fix only those features using IQR method.

[81]:
```
numerical_features = ['age', 'campaign', 'duration']
for cols in numerical_features:
    Q1 = bank_copy[cols].quantile(0.25)
    Q3 = bank_copy[cols].quantile(0.75)
    IQR = Q3 - Q1

    filter = (bank_copy[cols] >= Q1 - 1.5 * IQR) & (bank_copy[cols] <= Q3 + 1.5␣
↪* IQR)
    bank_copy=bank_copy.loc[filter]
```

[82]:
```
fig, ax = plt.subplots(2, 2, figsize=(15, 8))
ax[0, 0].boxplot(bank_copy['age'])
ax[0, 0].set_title("Age")
ax[0, 1].boxplot(bank_copy['campaign'])
ax[0, 1].set_title("Campaign")
ax[1, 0].boxplot(bank_copy['duration'])
ax[1, 0].set_title("Duration")
ax[1, 1].axis('off')
plt.show()
```



Insights

- We can see that we remove outliers from this features and now can move forward

1.13.2 Education Category Clubbing

[84]:
```
bank_feature = bank_copy.copy()
lst = ['basic.9y', 'basic.6y', 'basic.4y']
bank_feature['education'].replace(lst, 'middle.school', inplace=True)
bank_feature['education'].value_counts()
```

[84]:
```
middle.school          10688
university.degree      10559
high.school             8287
professional.course     4554
unknown                 1459
illiterate                14
Name: education, dtype: int64
```

Insights

- Yeah, We club it and see the value count ot Education Category

1.13.3 Encoding Month and Day of Week

Endoing the categories of Month and Day or week in respective numbers

[87]:
```
month_dict={'may':5,'jul':7,'aug':8,'jun':6,'nov':11,'apr':4,'oct':10,'sep':
 ↪9,'mar':3,'dec':12}
bank_feature['month']= bank_feature['month'].map(month_dict)

day_dict={'thu':5,'mon':2,'wed':4,'tue':3,'fri':6}
bank_feature['day_of_week']= bank_feature['day_of_week'].map(day_dict)
```

[91]:
```
bank_feature.loc[:, ['month','day_of_week']].head()
```

[91]:

	month	day_of_week
0	5	2
1	5	2
2	5	2
3	5	2
4	5	2

We have encoded the month and days of week into numerical from categorical

1.13.4 Encoding 999 as 0 in pdays

Encoding 999 in pdays feature(i.e clients who haven't been contacted for the previous campaign) into 0

[92]:
```
bank_feature.loc[bank_feature['pdays'] == 999, 'pdays'] = 0
```

[93]:
```
bank_feature.pdays.value_counts()
```

[93]:
```
0     34305
3       367
6       343
4       105
9        54
2        51
12       50
7        48
10       44
5        38
13       28
1        23
11       22
15       20
14       15
8        14
16       10
17        8
18        6
22        3
21        2
25        1
26        1
27        1
20        1
19        1
Name: pdays, dtype: int64
```

Insights

- We have converted all 999 occurences as 0 in pdays

1.13.5 Ordinal Number Encoding

In this step we will encode the 'yes, no, unknown' into 1,0,-1 in respective features

[94]:
```
dict = {'yes': 1, 'no': 0, 'unknown': -1}

bank_feature['housing'].replace(dict, inplace=True)
bank_feature['loan'].replace(dict, inplace=True)
bank_feature['default'].replace(dict, inplace=True)
```

[95]:
```
dict1 = {'yes': 1, 'no': 0}

bank_feature['y'].replace(dict1, inplace=True)
```

[97]:
```
bank_feature['y']
```

[97]:
```
0        0
1        0
2        0
3        0
4        0
        ..
41181    1
41182    0
41184    0
41185    0
41186    1
Name: y, Length: 35561, dtype: int64
```

[101]:
```python
bank_feature.loc[:, ['housing', 'loan', 'default']].head()
```

[101]:

	housing	loan	default
0	0	0	0
1	0	0	-1
2	1	0	0
3	0	0	0
4	0	1	0

We have encoded the yes/no ,unknown into respective numbers

1.13.6 Ordinal Encoding

[102]:
```python
dummy_contact = pd.get_dummies(bank_feature['contact'], prefix='encode',␣
↪drop_first=True)
dummy_outcome = pd.get_dummies(bank_feature['poutcome'], prefix='encode',␣
↪drop_first=True)

bank_feature = pd.concat([bank_feature, dummy_contact, dummy_outcome], axis=1)

bank_feature.drop(['contact', 'poutcome'], axis=1, inplace=True)
```

[106]:
```python
bank_feature.loc[:,['encode_telephone', 'encode_nonexistent',␣
↪'encode_success']].head()
```

[106]:

	encode_telephone	encode_nonexistent	encode_success
0	1	1	0
1	1	1	0
2	1	1	0
3	1	1	0
4	1	1	0

Insights

- We have performed One-hot encoding to change the values from categorical to numerical and drop the original features

1.13.7 Frequency Encoding

Let's use frequency encoding with job and education features in our dataset

[107]:
```
bank_job = bank_feature['job'].value_counts().to_dict()
bank_education = bank_feature['education'].value_counts().to_dict()
```

We convert the frequency into Key-pairs, now map them

[110]:
```
bank_feature['job'].replace(bank_job, inplace=True)
bank_feature['education'].replace(bank_education, inplace=True)
```

[111]:
```
bank_feature.loc[:,['job', 'education']].head()
```

[111]:

	job	education
0	899	10688
1	3456	8287
2	3456	8287
3	9110	10688
4	3456	8287

We encoded the job and education into key-pairs

1.13.8 Target Guided Ordinal Encoding

Let's encode marital feature based on the target 'y', before that we find mean of target value with respect to marital feature

[117]:
```
bank_feature.groupby(['marital'])['y'].mean()
```

[117]:
```
marital
divorced    0.063988
married     0.069050
single      0.113226
unknown     0.129032
Name: y, dtype: float64
```

[118]:
```
ordinal_labels = bank_feature.groupby(['marital'])['y'].mean().sort_values().
 ↪index
ordinal_labels
```

[118]:
```
Index(['divorced', 'married', 'single', 'unknown'], dtype='object',
name='marital')
```

We have sorted the categories based on the mean with respect to our outcome

[119]:
```
ordinal_labels1 = {}
for i, k in enumerate(ordinal_labels):
    ordinal_labels1[k] = i
```

[120]:
```
ordinal_labels1
```

[120]:
```
{'divorced': 0, 'married': 1, 'single': 2, 'unknown': 3}
```

We changed the value into key-pairs, now map them

[121]:
```
bank_feature['marital_ordinal'] = bank_feature['marital'].map(ordinal_labels1)
bank_feature.drop(['marital'], axis=1, inplace=True)
```

[129]:
```
bank_feature.marital_ordinal.value_counts()
```

[129]:
```
1    21506
2    10086
0     3907
3       62
Name: marital_ordinal, dtype: int64
```

We see that values are encoded

1.13.9 Standardization of numerical values

[136]:
```
bank_scale = bank_feature.copy()

categorical_variables =  ['job', 'education', 'default', 'housing', 'loan',␣
 ↪'month', 'day_of_week', 'y', 'encode_telephone', 'encode_nonexistent',␣
 ↪'encode_success', 'marital_ordinal']

feature_scalar = []
for feature in bank_scale.columns:
    if feature not in categorical_variables:
        feature_scalar.append(feature)

from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

scalar.fit(bank_scale[feature_scalar])
```

[136]:
```
StandardScaler()
```

[138]:
```
scaled_data = pd.concat([bank_scale[['job', 'education', 'default', 'housing',␣
 ↪'loan', 'month', 'day_of_week',
                            'y', 'encode_telephone', 'encode_nonexistent',
                                   'encode_success', 'marital_ordinal']].
 ↪reset_index(drop=True),
                        pd.DataFrame(scalar.
 ↪transform(bank_scale[feature_scalar]), columns=feature_scalar)], axis=1)
```

[139]:
```
scaled_data.head()
```

[139]:
```
    job  education  default  housing  loan  month  day_of_week  y  \
0   899      10688        0        0     0      5            2  0
1  3456       8287       -1        0     0      5            2  0
2  3456       8287        0        1     0      5            2  0
3  9110      10688        0        0     0      5            2  0
4  3456       8287        0        0     1      5            2  0

   encode_telephone  encode_nonexistent  …       age  duration  campaign  \
0                 1                   1  …  1.694643  0.383434 -0.813061
1                 1                   1  …  1.797965 -0.413575 -0.813061
2                 1                   1  … -0.268482  0.134369 -0.813061
3                 1                   1  …  0.041485 -0.399342 -0.813061
4                 1                   1  …  1.694643  0.710777 -0.813061

      pdays  previous  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  \
0 -0.161001 -0.354645      0.660543        0.741263       0.891988    0.72072
1 -0.161001 -0.354645      0.660543        0.741263       0.891988    0.72072
2 -0.161001 -0.354645      0.660543        0.741263       0.891988    0.72072
3 -0.161001 -0.354645      0.660543        0.741263       0.891988    0.72072
4 -0.161001 -0.354645      0.660543        0.741263       0.891988    0.72072

   nr.employed
0     0.340002
1     0.340002
2     0.340002
3     0.340002
4     0.340002

[5 rows x 22 columns]
```

We have scaled our numerical features as you can see from the head.

1.13.10 Feature Selection

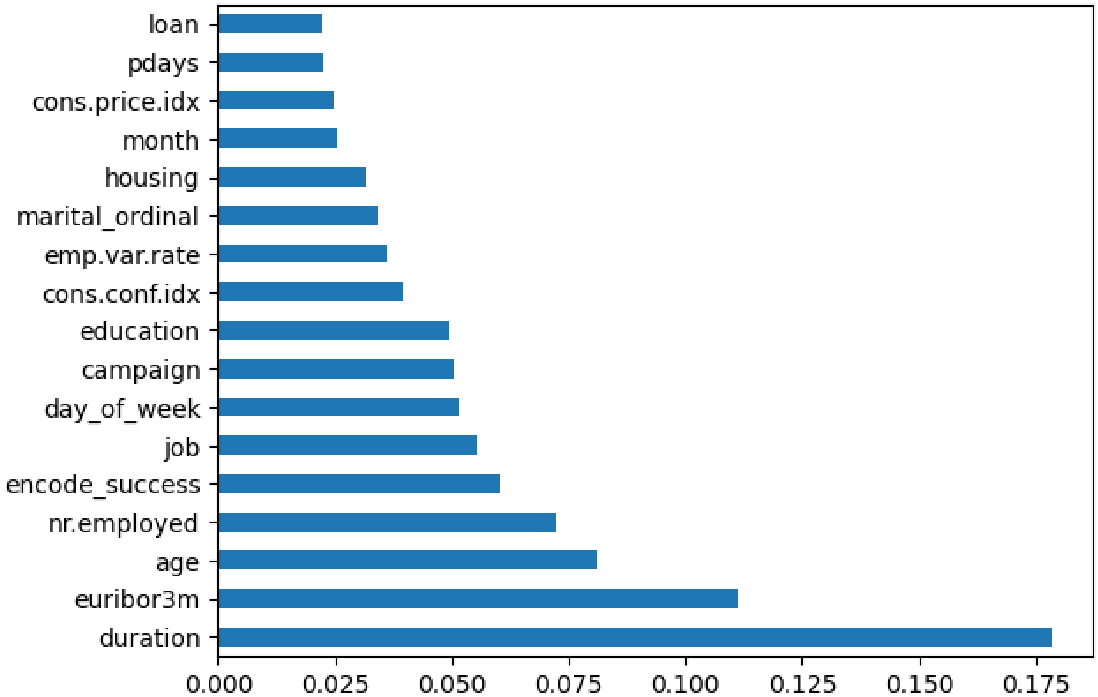
First, we'll find out which feature are most important for our model to work well. Then, we'll remove any unnecessary feature to make our model perform even better.

[141]:
```
x = scaled_data.drop(['y'], axis=1)
y = scaled_data['y']

et = ExtraTreesClassifier()
et.fit(x, y)
```

[141]:
```
ExtraTreesClassifier()
```

[142]:
```python
feature_imp = pd.Series(et.feature_importances_, index=x.columns)
feature_imp.nlargest(17).plot(kind='barh')
plt.show()
```



From the bar plot we can see the importances of features based on it's impact towards output. Let's take up the top 15 features

1.13.11 Train Test Split

Let's drop the required and split the data into train and test

[144]:
```python
x = scaled_data.drop(['pdays','month','cons.price.idx','loan','housing','emp.
↪var.rate','y'], axis=1)
y = scaled_data['y']

x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.
↪8,random_state=10)
print("X-Training data size: ", x_train.shape)
print("X-Test data size: ", x_test.shape)
print("Y-Training data size: ", y_train.shape)
print("Y-Test data size: ", y_test.shape)
```

```
X-Training data size:  (28448, 15)
X-Test data size:  (7113, 15)
Y-Training data size:  (28448,)
```

```
Y-Test data size:  (7113,)
```

1.13.12 Modeling the data

Let's move into the important phase of building our machine learning model. Before we decide on 'which algorithm is best for prediction,' let's focus on 'why.' This step is really crucial.

Why? Why do we need to understand 'why'? Because our main goal is to predict whether someone will make a deposit based on the provided information. The result can be either 'yes' (1) or 'no' (0). So, we need to figure out the 'why' before the 'which.'

What? Now, let's talk about the 'what.' To decide which classification model is the best fit, we won't jump straight into testing models. Instead, we'll start by writing quality code. We'll create a process called 'cross-validation' to check the accuracy of all the models together. This way, we can find the best model without wasting time. After comparing their accuracies, we'll pick the model with the highest accuracy.

1.13.13 Model Selection

Let's do the process and select the best model

Logistic Regression

[150]:
```python
from sklearn.linear_model import LogisticRegression
```

[159]:
```python
lr = LogisticRegression(random_state=0)
```

[167]:
```python
cv_scores_lr = cross_val_score(lr, x_train, y_train, cv=5)
```

[168]:
```python
cv_scores_lr
```

[168]:
```
array([0.92759227, 0.92618629, 0.92407733, 0.93144665, 0.92564598])
```

[169]:
```python
for fold, score in enumerate(cv_scores_lr, start=1):
    print(f"Fold {fold}: Accuracy = {score:.4f}")

mean_accuracy = cv_scores_lr.mean()
print(f"Mean Accuracy of Logistic Regression: {mean_accuracy:.4f}")
```

```
Fold 1: Accuracy = 0.9276
Fold 2: Accuracy = 0.9262
Fold 3: Accuracy = 0.9241
Fold 4: Accuracy = 0.9314
Fold 5: Accuracy = 0.9256
Mean Accuracy of Logistic Regression: 0.9270
```

Decision Tree Classifier

[157]:
```python
from sklearn.tree import DecisionTreeClassifier
```

[158]:
```
dc = DecisionTreeClassifier()
```

[170]:
```
cv_scores_dc = cross_val_score(dc, x_train, y_train, cv=5)
```

[171]:
```
cv_scores_dc
```

[171]:
```
array([0.91581722, 0.91652021, 0.9142355 , 0.9191422 , 0.92019687])
```

[172]:
```
for fold, score in enumerate(cv_scores_dc, start=1):
    print(f"Fold {fold}: Accuracy = {score:.4f}")

mean_accuracy = cv_scores_dc.mean()
print(f"Mean Accuracy of Decision Tree Classfier: {mean_accuracy:.4f}")
```

```
Fold 1: Accuracy = 0.9158
Fold 2: Accuracy = 0.9165
Fold 3: Accuracy = 0.9142
Fold 4: Accuracy = 0.9191
Fold 5: Accuracy = 0.9202
Mean Accuracy of Decision Tree Classfier: 0.9172
```

1.13.14 Logistic Regression

Let's fit the model in Logistic Regression to figure out Accuracy of our model

[156]:
```
lrs = LogisticRegression(random_state=0)
lrs.fit(x_train,y_train)
lrs_predict = lrs.predict(x_test)
print("The accuracy of Logistic Regression: {:.2f}".format(lrs.
 ↪score(x_test,y_test)))
```

```
The accuracy of Logistic Regression: 0.92
```

1.13.15 Decision Tree Classifier

Let's fit the model in Decision Tree Classifier to figure out Accuracy of our model

[173]:
```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
dtc_predict = dtc.predict(x_test)
print("The accuracy of Decision Tree Classifier: {:.2f}".format(dtc.
 ↪score(x_test,y_test)))
```

```
The accuracy of Decision Tree Classifier: 0.91
```

We see that both the accuracy are preety much good, for Logistic Regression is 92% and for Decision Tree Classifier is 91%

1.13.16 Classfication Report

Logistic Regression Report Let's see the report of Classification for Logistic Regression

[177]:
```
reprot_lrs = classification_report(y_test, lrs_predict)
```

[179]:
```
print(reprot_lrs)
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      6501
           1       0.54      0.29      0.38       612

    accuracy                           0.92      7113
   macro avg       0.74      0.63      0.67      7113
weighted avg       0.90      0.92      0.91      7113
```

Decision Tree Classifier Now, Let's see the report of Classification for Decision Tree Classifier

[180]:
```
report_dtc = classification_report(y_test, dtc_predict)
```

[181]:
```
print(report_dtc)
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95      6501
           1       0.50      0.50      0.50       612

    accuracy                           0.91      7113
   macro avg       0.73      0.72      0.72      7113
weighted avg       0.91      0.91      0.91      7113
```

1.13.17 Conclusion

After analyzing the data and choosing the right model, we've found that the length of calls (duration) is a key factor in deciding if someone will go for a deposit. Basically, if a person is more interested, they tend to have longer calls. Also, their job and education play a big role in their decision.

Here's what the bank can do to improve their deposit success:

- Sort Jobs by Importance: Group jobs based on their importance in companies. For the top-tier jobs, like managers, reach out shortly after starting the campaign. These folks are more likely to say yes.
- Listen and Personalize: Really pay attention to what people say during calls. Use that info to create personalized deposit plans that match their needs. This might make the calls longer and boost the chances of getting a deposit.
- Time Things Right: Plan the campaign to start when the bank's new period kicks off, usually between May and July. In the past, this time has shown good results, so it's a smart time to connect with potential customers.

- Sync with the Economy: Keep an eye on the economy. If it's not doing well nationally, maybe hold off on spending too much on the campaign. It's smart to adjust your plans based on how the economy is doing.