Predicting Term Deposit Subscription Based on Customer Demographics, Banking History, and the Current Marketing Campaign : A Classification Problem

For the requirement of the module - Machine Learning,
Higher National Diploma in Data Science.

Submitted on – 31st May 2023

By - Theekshitha Varatharajsarma (COHNDDS231F-012)

TABLE OF CONTENTS

INTRODUCTION

This report shows the outputs of supervised machine learning models used to predict whether a customer will subscribe to a term deposit or not based on customer demographics such as , age, job, education and marital status and previous banking history such as whether the customer has a credit default, whether they have taken loans, etc., and the current marketing campaign's efforts which includes number of contacts performed during this campaign and the duration of the last contact made.

The report is divided into parts and gives step by step explanations with the codes from data preprocessing to performing and evaluating the machine learning model. All the models used are supervised machine learning models, especially classification models, chosen to be the best fit for our dataset and purpose as the dependent variable is binary(yes/no). The models are logistic regression model, random forest model and decision tree model. Confusion matrices are visualized, and the outcomes are explained.

In short, this report gives an overall idea of performing a classification problem and using machine learning – classification models to predict whether a customer will subscribe to a term deposit or not.

VARIABLE DESCRIPTION

1 - age (numeric)

2 - job : type of job

3 - marital : marital status

4 - education : Level of Education

5 - default: has credit in default or not

6 - balance: average yearly balance, in euros

7 - housing: has housing loan or not

8 - loan: has personal loan or not

9 - contact: contact communication type

10 - day: last contact day of the month

11 - month: last contact month of year

12 - duration: last contact duration, in seconds

13 - campaign: number of contacts performed during this campaign and for this client

14 - pdays: number of days that passed by after the client was last contacted from a previous campaign

15 - previous: number of contacts performed before this campaign and for this client (numeric)

16 - poutcome: outcome of the previous marketing

17. Target - Whether the client subscribed a term deposit or not.

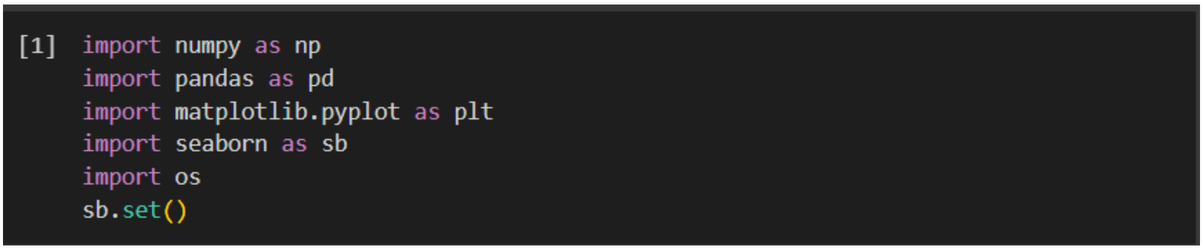
Derived from : UCI Machine Learning Repository

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

CODE EXPLANATION

Importing Libraries

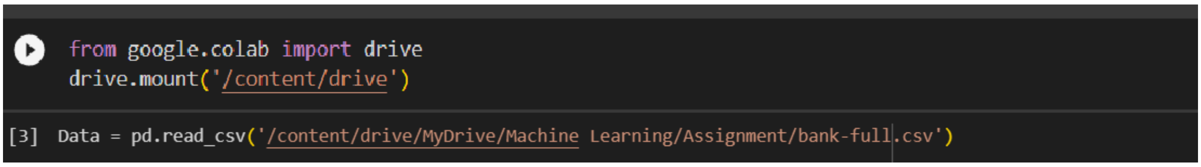
1. Importing some of the necessary libraries to run the codes.



```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sb
     import os
     sb.set()
```

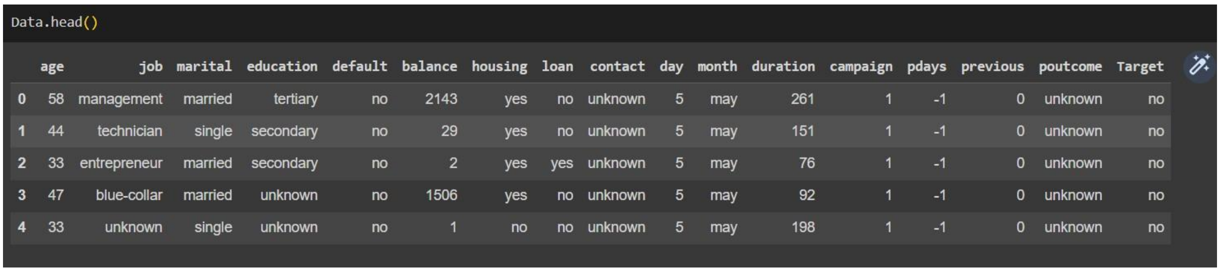
Loading the Dataset

2. Mounting the google drive where the dataset is stored and then loading it into google colab.



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

[3]  Data = pd.read_csv('/content/drive/MyDrive/Machine Learning/Assignment/bank-full.csv')
```

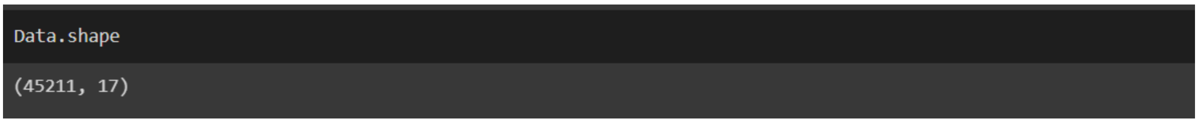
3. Visualizing the first five rows(head) of the dataset



```
Data.head()
```

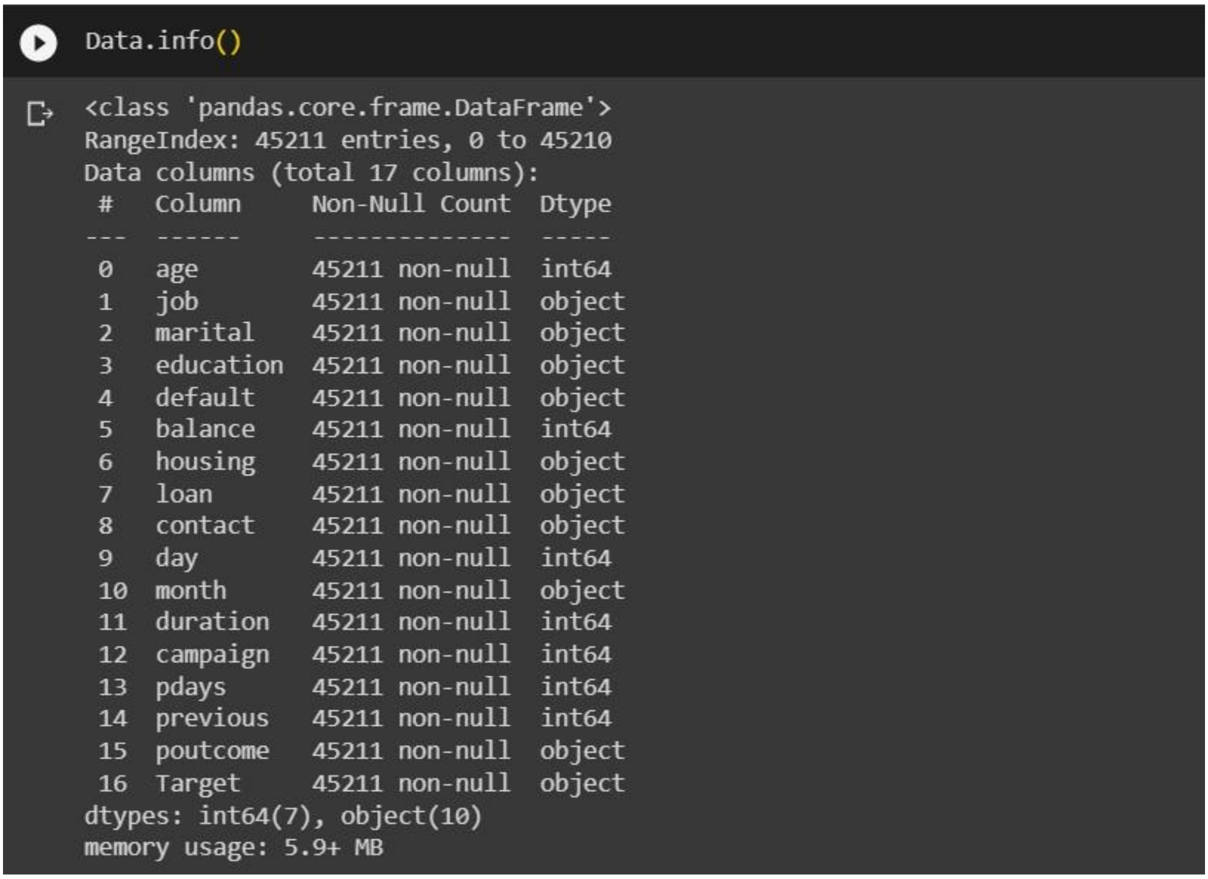
	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	Target
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

4. Printing the number of rows and columns of the dataset (shape of the dataset).



```
Data.shape

(45211, 17)
```

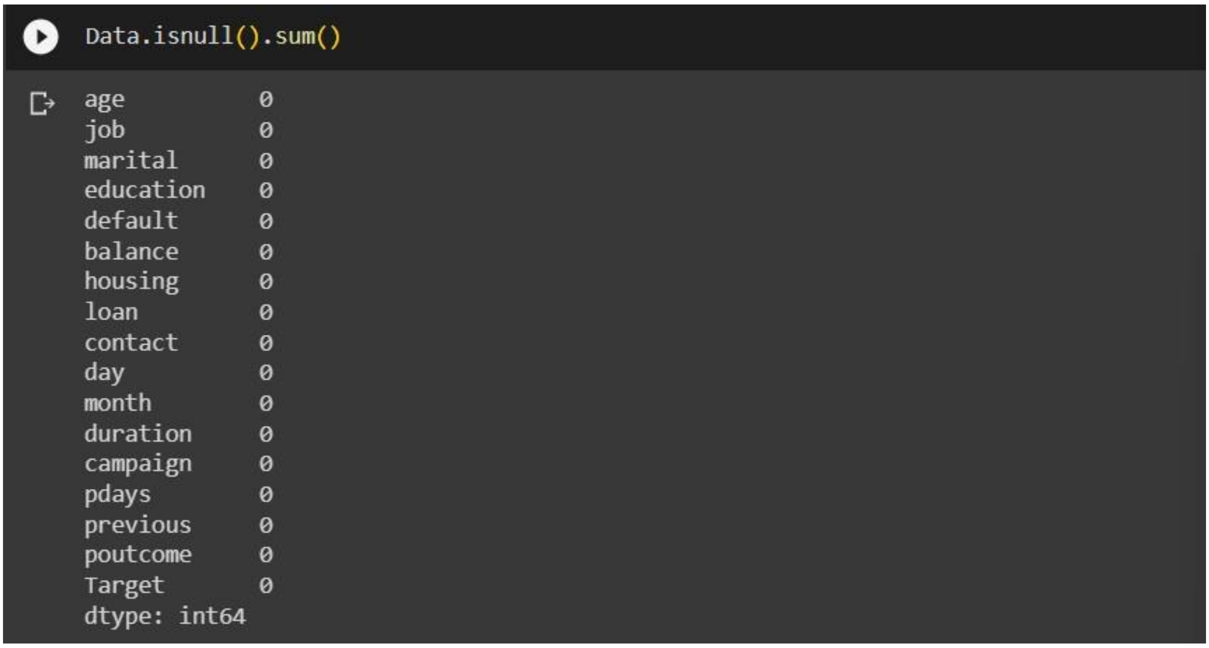
5. Printing information about the dataset that includes all the column names, non-null value counts and variable types stored in each column.



```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        45211 non-null  int64 
 1   job        45211 non-null  object
 2   marital    45211 non-null  object
 3   education  45211 non-null  object
 4   default    45211 non-null  object
 5   balance    45211 non-null  int64 
 6   housing    45211 non-null  object
 7   loan       45211 non-null  object
 8   contact    45211 non-null  object
 9   day        45211 non-null  int64 
 10  month      45211 non-null  object
 11  duration   45211 non-null  int64 
 12  campaign   45211 non-null  int64 
 13  pdays      45211 non-null  int64 
 14  previous   45211 non-null  int64 
 15  poutcome   45211 non-null  object
 16  Target     45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

6. Printing the sum of null values in each column

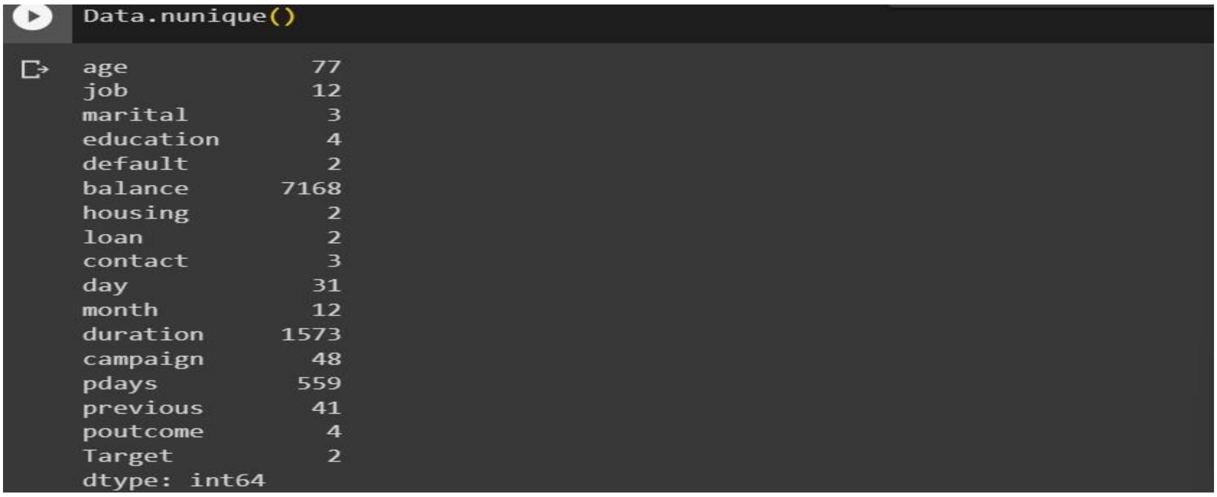


```
Data.isnull().sum()
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
Target       0
dtype: int64
```

Checking Unique Values

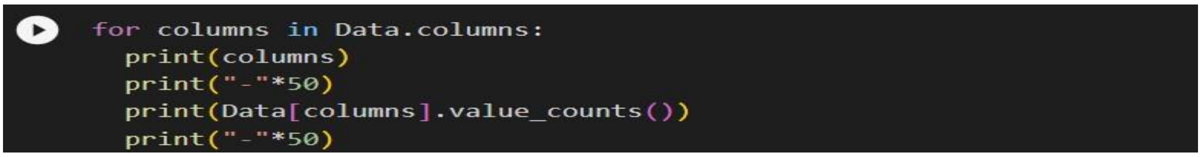
7. Printing the number of unique values in each column.



```
Data.nunique()
```

```
age              77
job              12
marital           3
education         4
default           2
balance        7168
housing           2
loan              2
contact           3
day              31
month            12
duration       1573
campaign         48
pdays           559
previous         41
poutcome          4
Target            2
dtype: int64
```

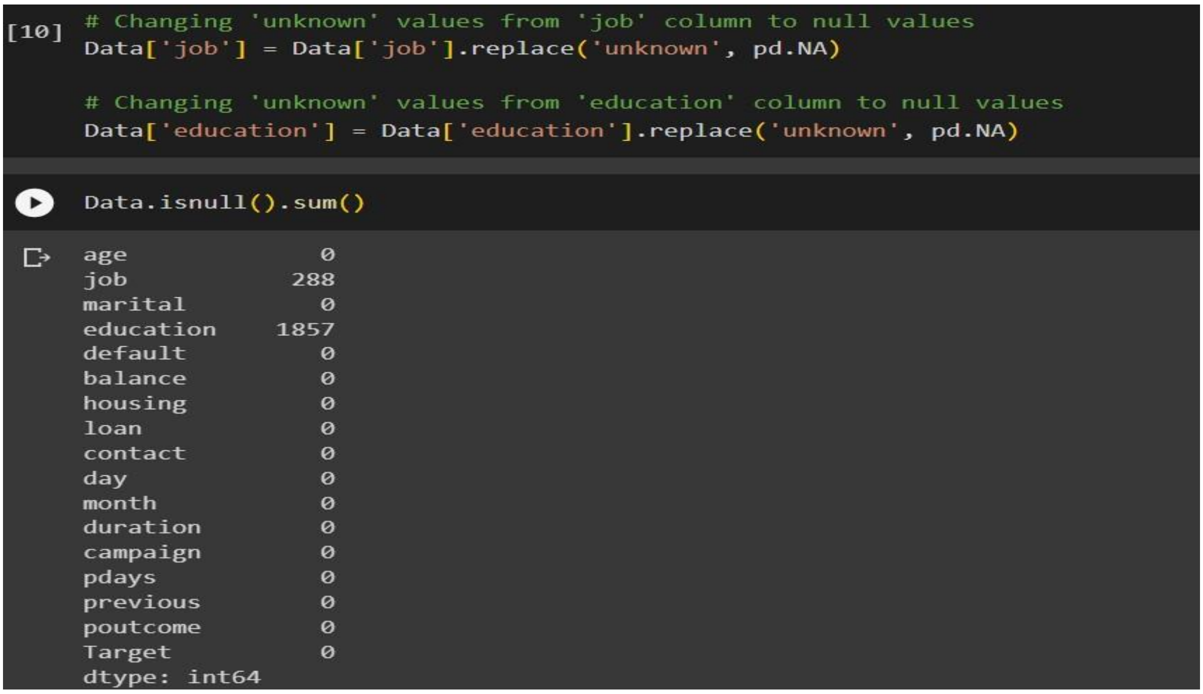
8. Printing number of unique values, what they are and the count of each unique value for each column.



```
for columns in Data.columns:
  print(columns)
  print("-"*50)
  print(Data[columns].value_counts())
  print("-"*50)
```

Dropping Null Values

9. Changing the ‘unknown’ values in ‘job’ and ‘education’ columns to NA format and dropping these values.



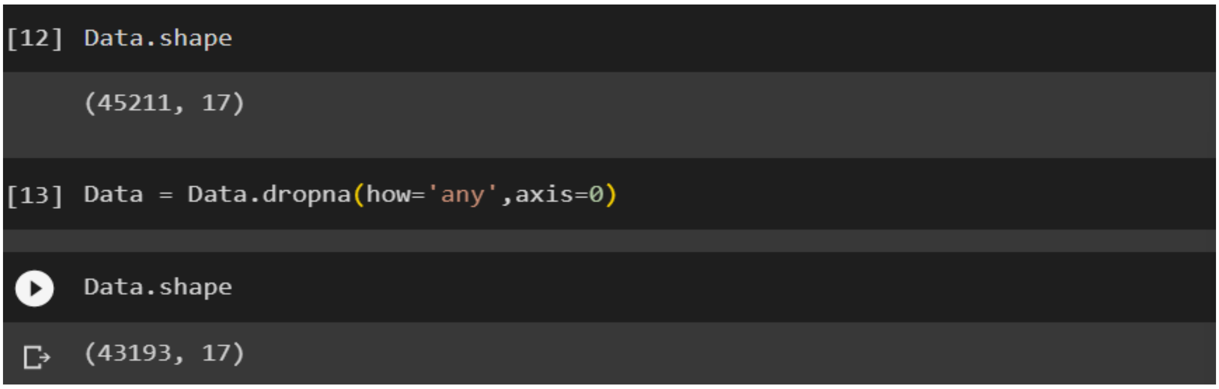
```
[10] # Changing 'unknown' values from 'job' column to null values
     Data['job'] = Data['job'].replace('unknown', pd.NA)

     # Changing 'unknown' values from 'education' column to null values
     Data['education'] = Data['education'].replace('unknown', pd.NA)
```

```
Data.isnull().sum()
```

```
age             0
job           288
marital         0
education    1857
default         0
balance         0
housing         0
loan            0
contact         0
day             0
month           0
duration        0
campaign        0
pdays           0
previous        0
poutcome        0
Target          0
dtype: int64
```

I have not removed the unknown values in the ‘poutcome’ column which shows the previous marketing outcome as unknown, failure, other, or success. This is because in the 45211 observations, the unknown values are 36959 and removing this high amount of data could highly affect the performance of the model.

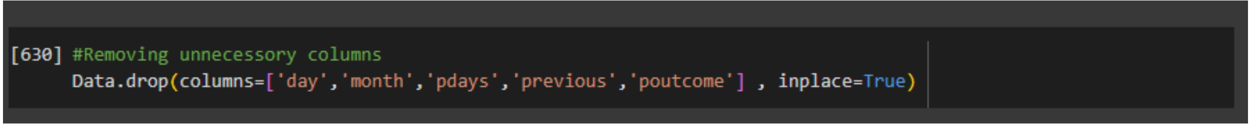
10. Printing the shape of the dataset, removing the null values, and printing the shape after removing.



```
[12] Data.shape

     (45211, 17)

[13] Data = Data.dropna(how='any',axis=0)

     Data.shape

     (43193, 17)
```

Removing Unnecessary Columns

11. Removing the unnecessary columns.



```
[630] #Removing unnecessory columns
      Data.drop(columns=['day','month','pdays','previous','poutcome'] , inplace=True)
```

‘Pdays’ is removed because of the majority of the data being -1 and other values being positive (182 – 530) which will act as extreme outliers.

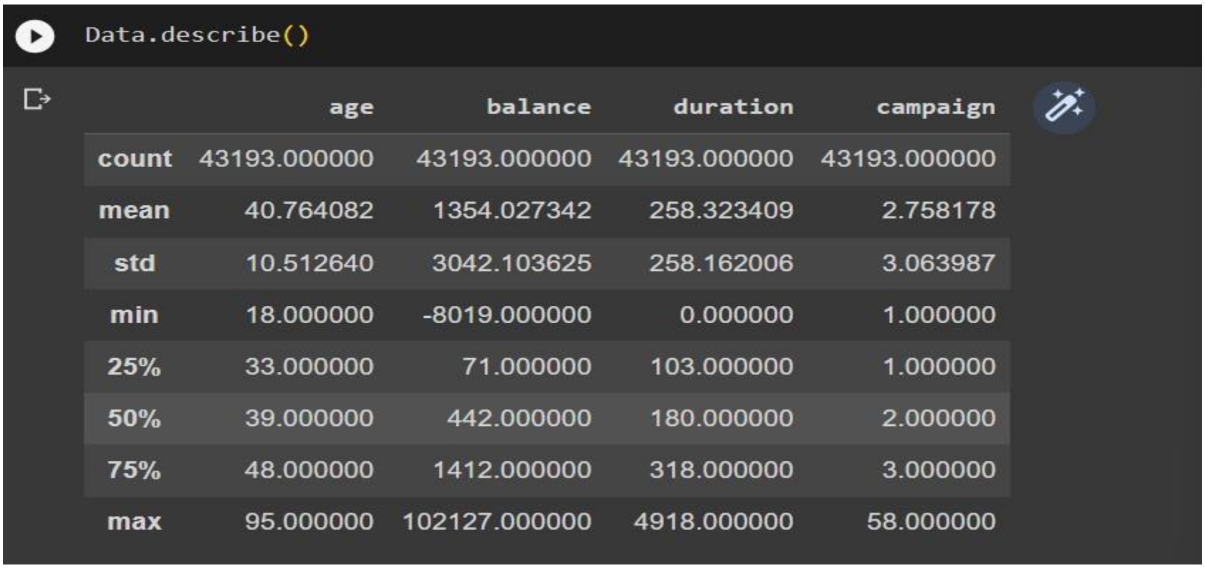
‘Previous’ also has 36954 values of 0 (No previous contact before current campaign) making this field irrelevant. Also present with extreme outliers and very low number of observations in every other values except 0.

‘Poutcome’ also has majority of observations in unknown and other (36959 + 1840)

Other variables are just irrelevant to our objective of just focusing on basic customer demographics, banking history and, current marketing efforts.

Descriptive Analysis

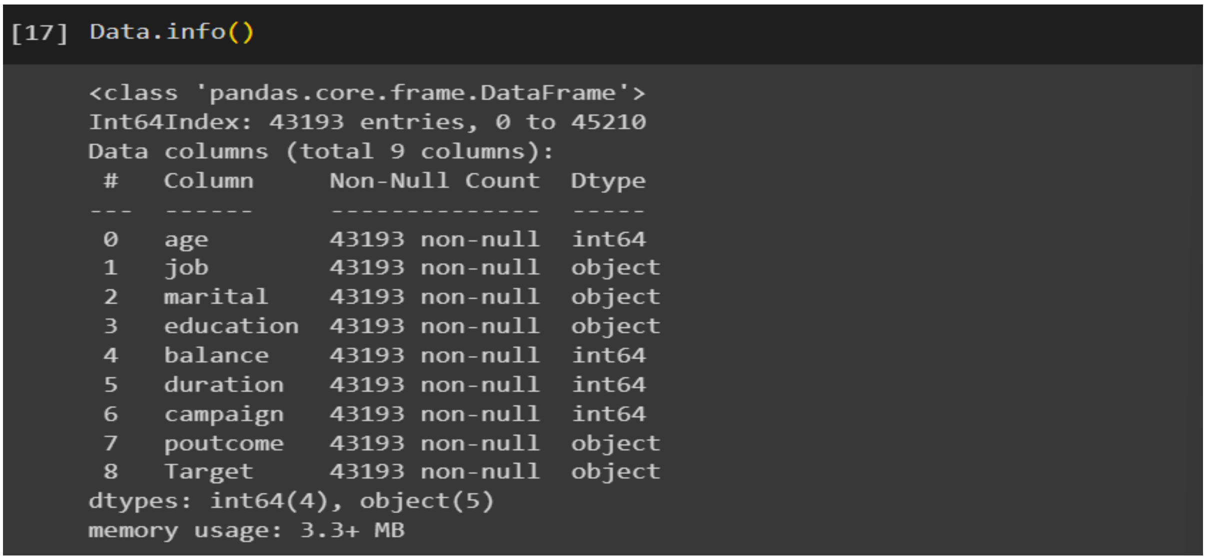
12. Performing descriptive analysis and printing the count, mean, standard deviation, minimum, maximum as well as the 25th, 50th and 75th percentiles for each column containing numerical variables.



```
Data.describe()
```

	age	balance	duration	campaign
count	43193.000000	43193.000000	43193.000000	43193.000000
mean	40.764082	1354.027342	258.323409	2.758178
std	10.512640	3042.103625	258.162006	3.063987
min	18.000000	-8019.000000	0.000000	1.000000
25%	33.000000	71.000000	103.000000	1.000000
50%	39.000000	442.000000	180.000000	2.000000
75%	48.000000	1412.000000	318.000000	3.000000
max	95.000000	102127.000000	4918.000000	58.000000

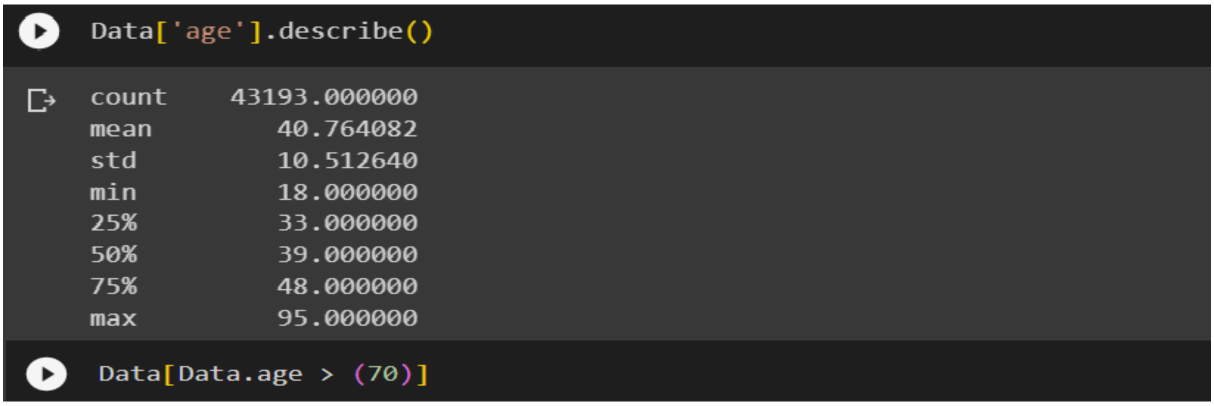
13. Printing the information of the dataset containing selected columns without null values.



```
[17] Data.info()

     <class 'pandas.core.frame.DataFrame'>
     Int64Index: 43193 entries, 0 to 45210
     Data columns (total 9 columns):
      #   Column     Non-Null Count  Dtype
     ---  ------     --------------  -----
      0   age        43193 non-null  int64
      1   job        43193 non-null  object
      2   marital    43193 non-null  object
      3   education  43193 non-null  object
      4   balance    43193 non-null  int64
      5   duration   43193 non-null  int64
      6   campaign   43193 non-null  int64
      7   poutcome   43193 non-null  object
      8   Target     43193 non-null  object
     dtypes: int64(4), object(5)
     memory usage: 3.3+ MB
```

Checking For Outliers.

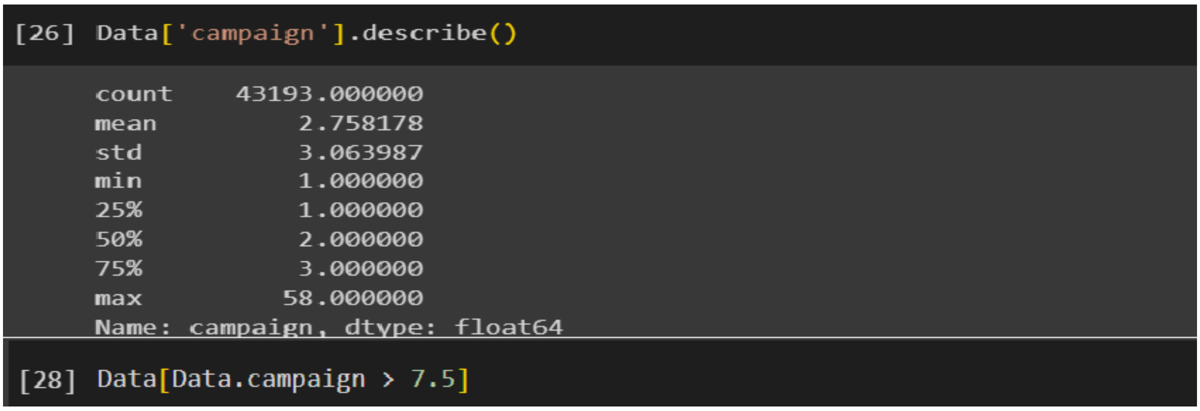
13. Checking for outliers in each numerical column by printing the descriptive analysis, boxplot, etc. and printing the rows that might contain outliers.



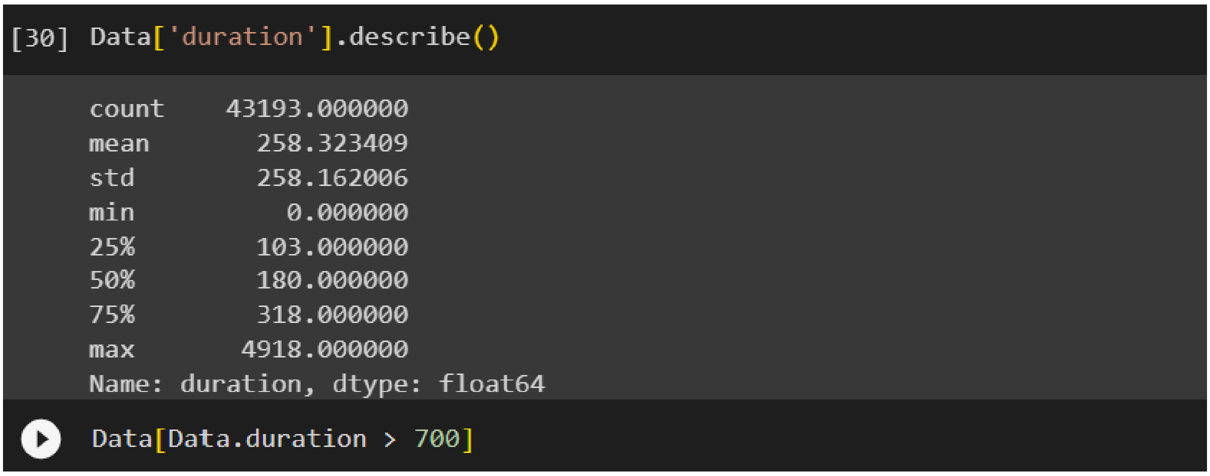
```
Data['age'].describe()

count    43193.000000
mean        40.764082
std         10.512640
min         18.000000
25%         33.000000
50%         39.000000
75%         48.000000
max         95.000000
```

```
Data[Data.age > (70)]
```

Here, there is a huge difference between 75th percentile and maximum values which could mean there are outliers present. The same pattern could be seen in campaign and duration columns.

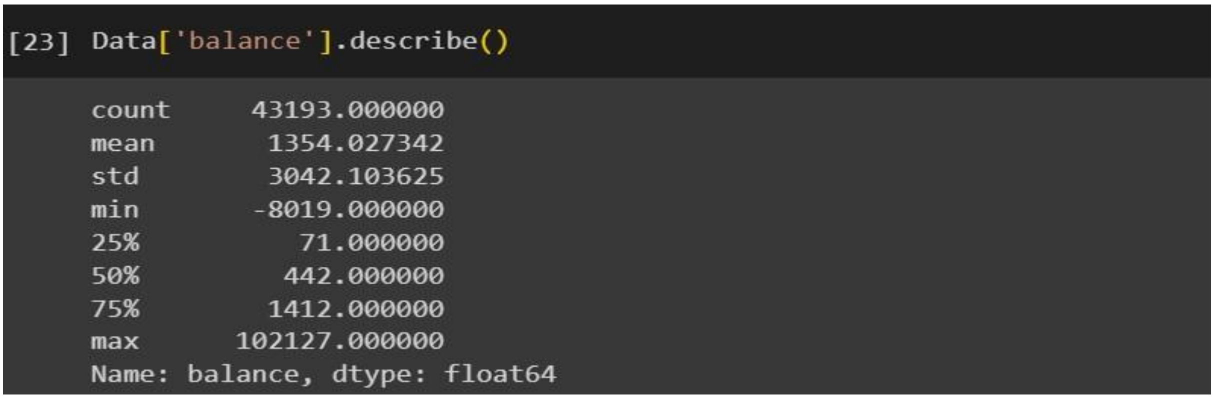


```
[26] Data['campaign'].describe()

     count    43193.000000
     mean         2.758178
     std          3.063987
     min          1.000000
     25%          1.000000
     50%          2.000000
     75%          3.000000
     max         58.000000
     Name: campaign, dtype: float64

[28] Data[Data.campaign > 7.5]
```



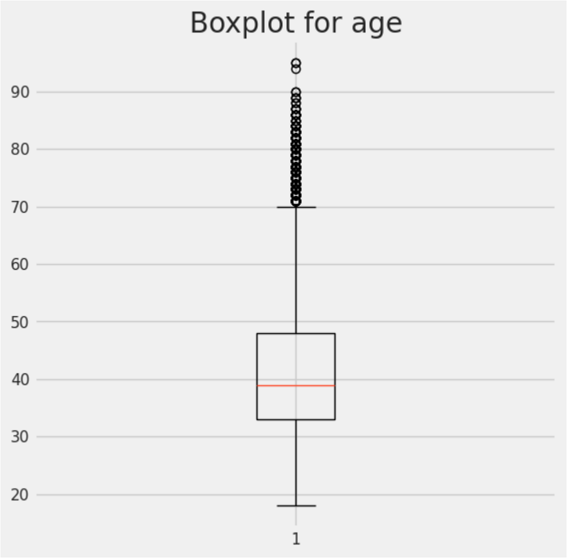
```
[30] Data['duration'].describe()

     count    43193.000000
     mean       258.323409
     std        258.162006
     min          0.000000
     25%        103.000000
     50%        180.000000
     75%        318.000000
     max       4918.000000
     Name: duration, dtype: float64

     Data[Data.duration > 700]
```

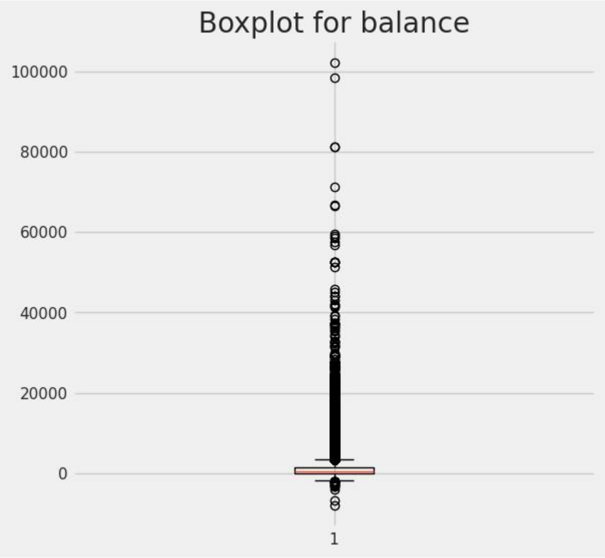
Again, in the balance column, there is a big difference between 75th percentile and maximum as well as 25th percentile and minimum which might suggest there are outliers.

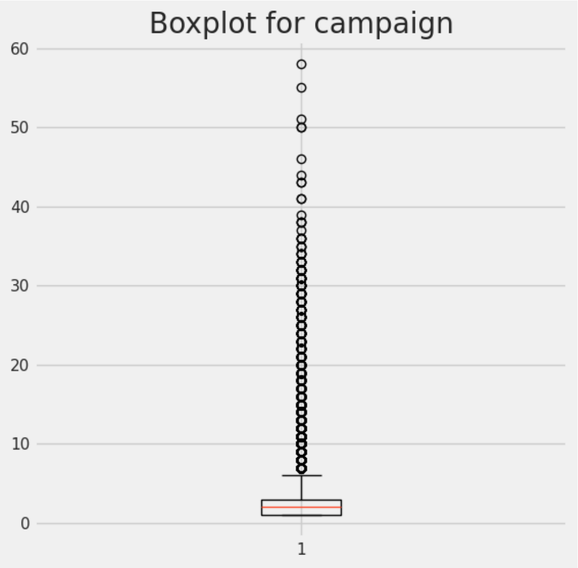


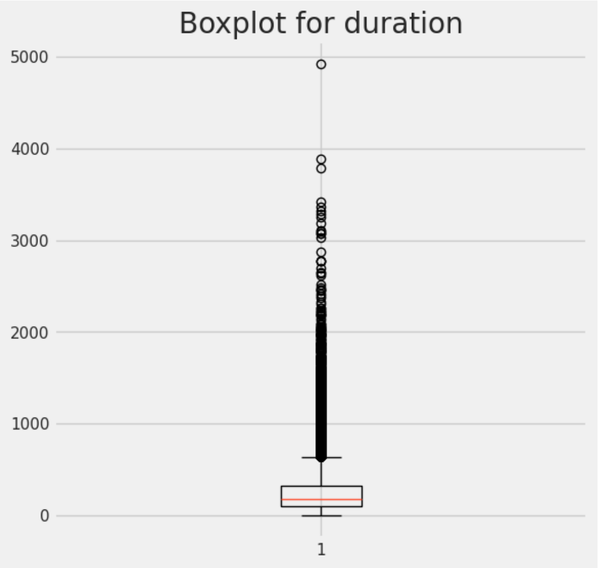
```
[23] Data['balance'].describe()

     count     43193.000000
     mean       1354.027342
     std        3042.103625
     min       -8019.000000
     25%          71.000000
     50%         442.000000
     75%        1412.000000
     max      102127.000000
     Name: balance, dtype: float64
```

The boxplots are below:



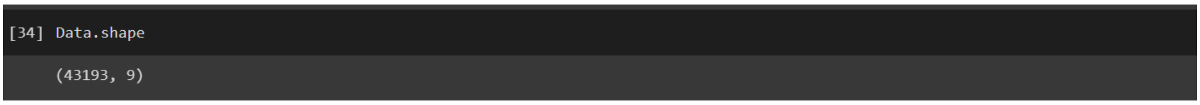






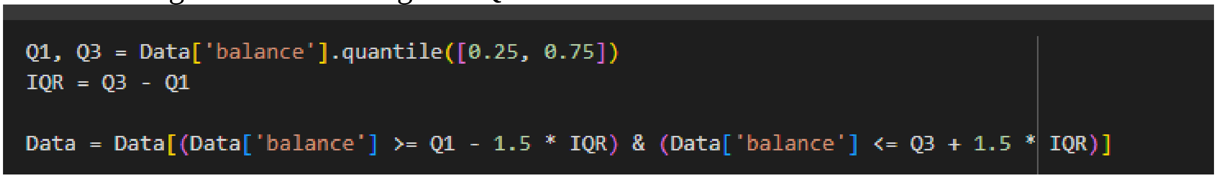
Removing Outliers

14. Printing the shape of the dataset again before removing outliers.



```
[34] Data.shape

     (43193, 9)
```

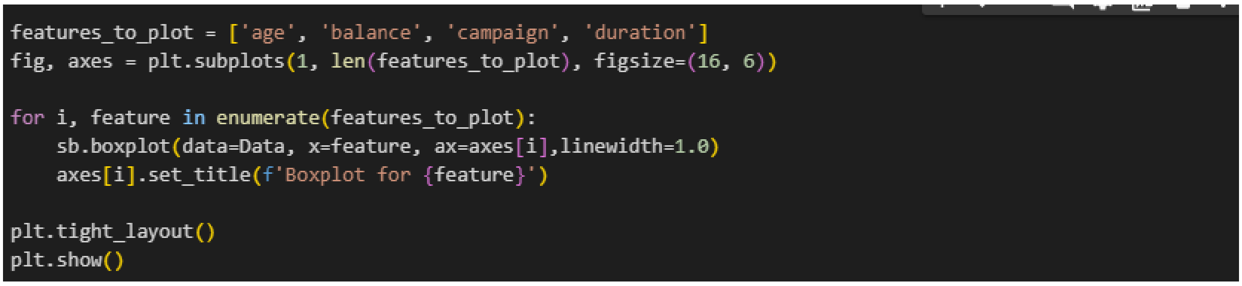
15. Removing the outliers using the IQR method.



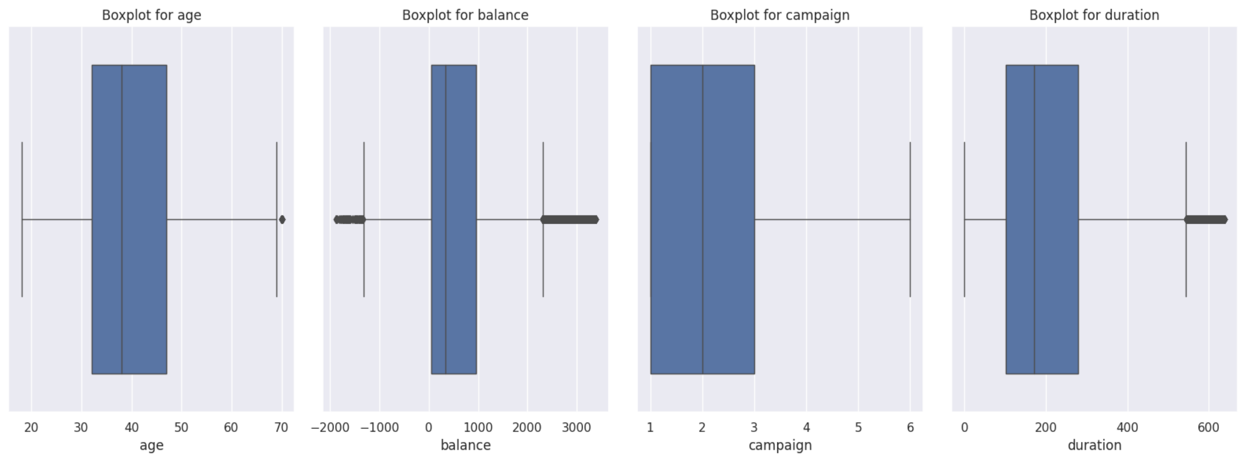
```
Q1, Q3 = Data['balance'].quantile([0.25, 0.75])
IQR = Q3 - Q1

Data = Data[(Data['balance'] >= Q1 - 1.5 * IQR) & (Data['balance'] <= Q3 + 1.5 * IQR)]
```

Same procedure has been followed to all columns.

16. Boxplots after outlier removal

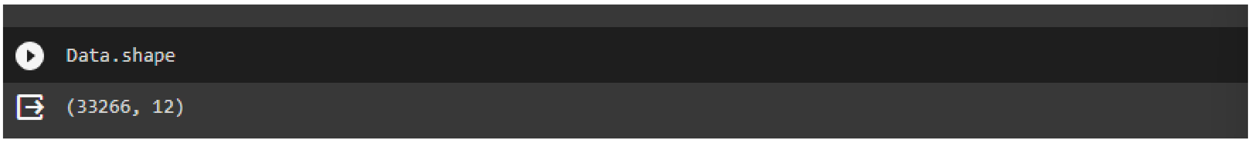


```
features_to_plot = ['age', 'balance', 'campaign', 'duration']
fig, axes = plt.subplots(1, len(features_to_plot), figsize=(16, 6))

for i, feature in enumerate(features_to_plot):
    sb.boxplot(data=Data, x=feature, ax=axes[i],linewidth=1.0)
    axes[i].set_title(f'Boxplot for {feature}')

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



Removed the outliers using the iqr method. Balance and Duration still appear to have some outliers. This will be modified only if needed (After checking the model performance). Not doing the removal again at this moment to not lose more data.

17. Printing the shape of the dataset after removing outliers.

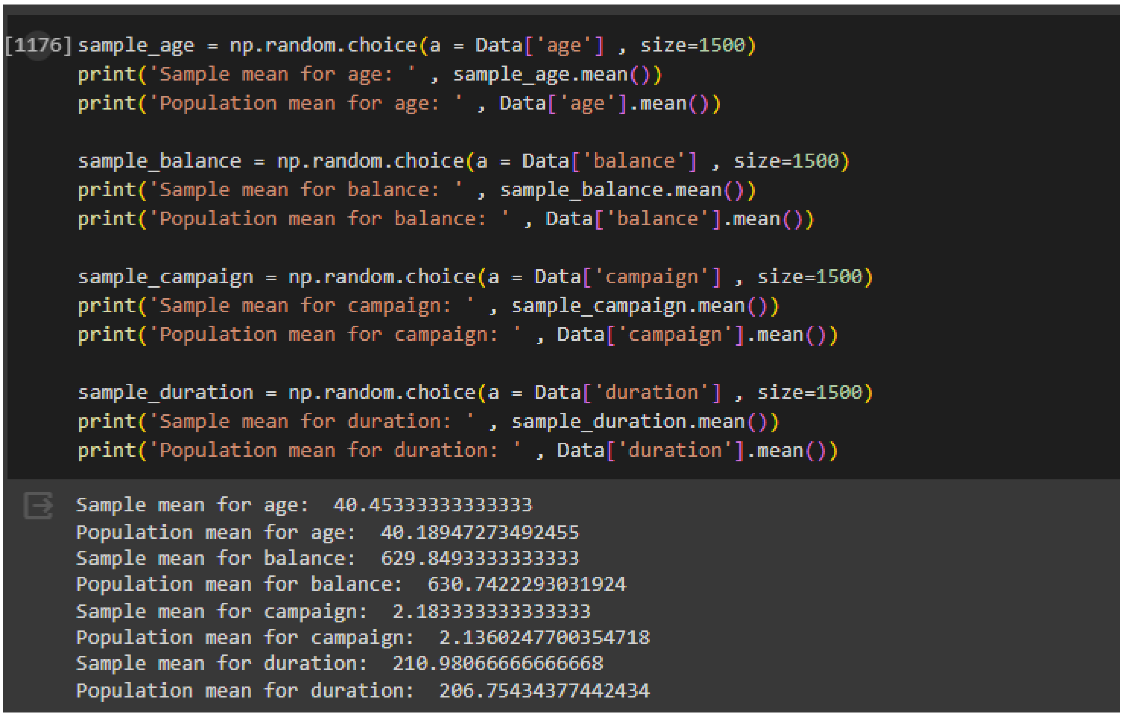


```
Data.shape
(33266, 12)
```

Inferential Statistics

18. Performing inferential statistics.

Here, I have taken 1500 random samples for all columns containing numerical data and calculated the sample mean. I, then printed the population mean with it as well.

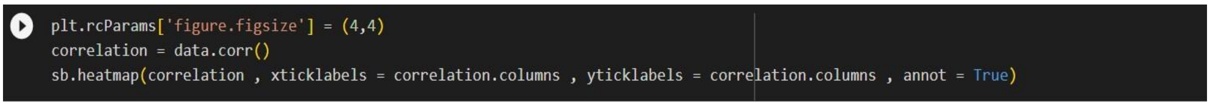


```
[1176] sample_age = np.random.choice(a = Data['age'] , size=1500)
       print('Sample mean for age: ' , sample_age.mean())
       print('Population mean for age: ' , Data['age'].mean())

       sample_balance = np.random.choice(a = Data['balance'] , size=1500)
       print('Sample mean for balance: ' , sample_balance.mean())
       print('Population mean for balance: ' , Data['balance'].mean())

       sample_campaign = np.random.choice(a = Data['campaign'] , size=1500)
       print('Sample mean for campaign: ' , sample_campaign.mean())
       print('Population mean for campaign: ' , Data['campaign'].mean())

       sample_duration = np.random.choice(a = Data['duration'] , size=1500)
       print('Sample mean for duration: ' , sample_duration.mean())
       print('Population mean for duration: ' , Data['duration'].mean())
```

```
Sample mean for age:  40.45333333333333
Population mean for age:  40.18947273492455
Sample mean for balance:  629.8493333333333
Population mean for balance:  630.7422293031924
Sample mean for campaign:  2.1833333333333333
Population mean for campaign:  2.1360247700354718
Sample mean for duration:  210.98066666666668
Population mean for duration:  206.75434377442434
```

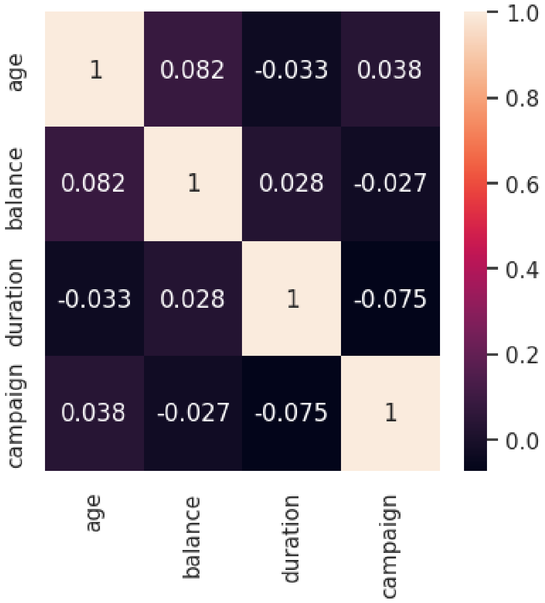
The sample seems to be representative of the population.

Checking Correlation.



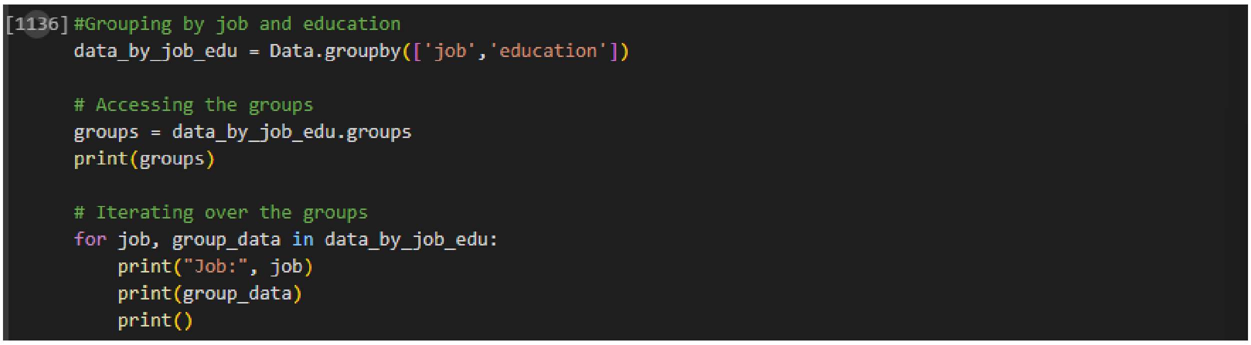
```
plt.rcParams['figure.figsize'] = (4,4)
correlation = data.corr()
sb.heatmap(correlation , xticklabels = correlation.columns , yticklabels = correlation.columns , annot = True)
```

1. Identifying correlations between variables with a heatmap.

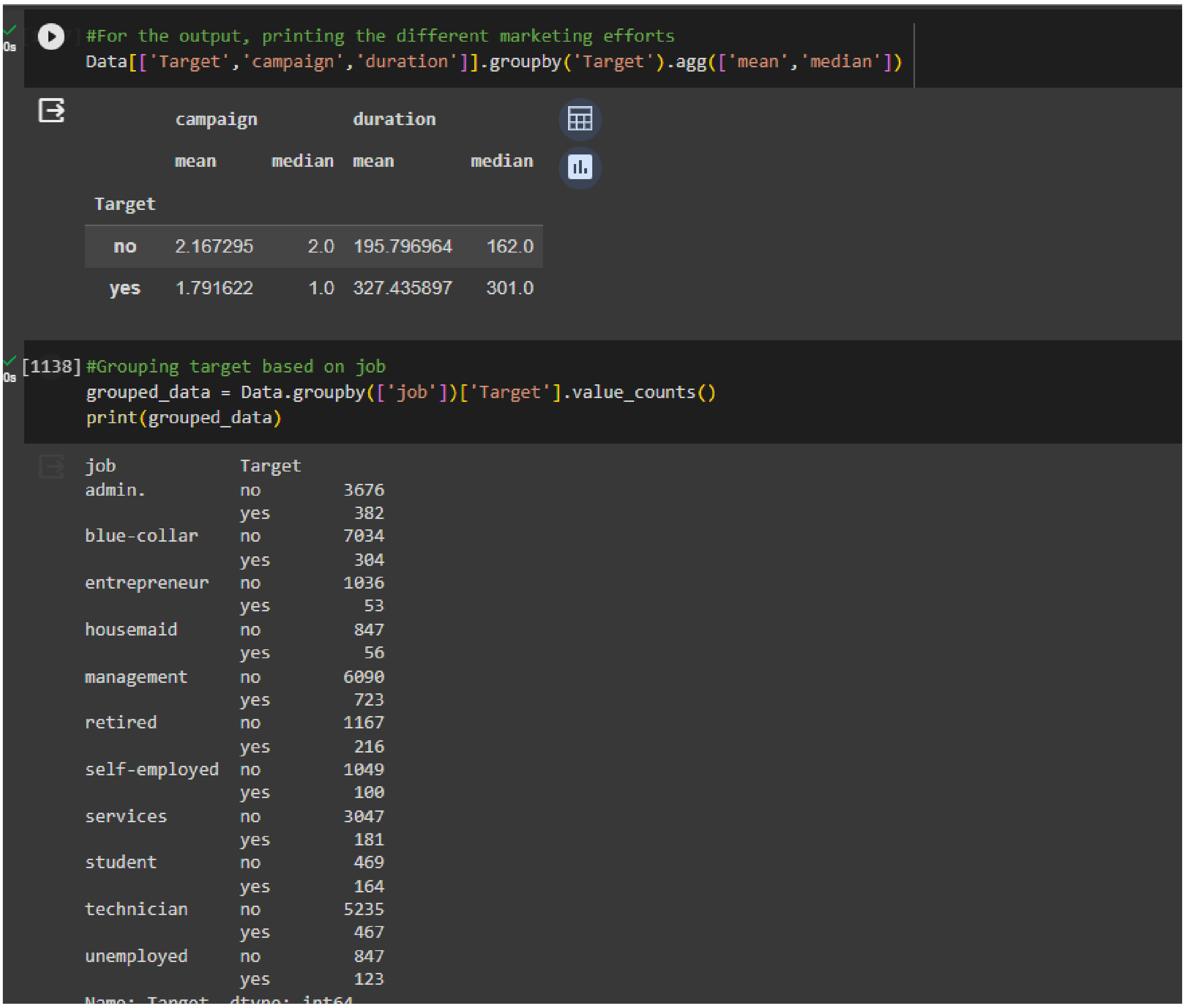


It could be seen that there are no highest correlated variables that may affect the performanceof the model.

Grouping the Data



```
[1136] #Grouping by job and education
       data_by_job_edu = Data.groupby(['job','education'])

       # Accessing the groups
       groups = data_by_job_edu.groups
       print(groups)

       # Iterating over the groups
       for job, group_data in data_by_job_edu:
           print("Job:", job)
           print(group_data)
           print()
```

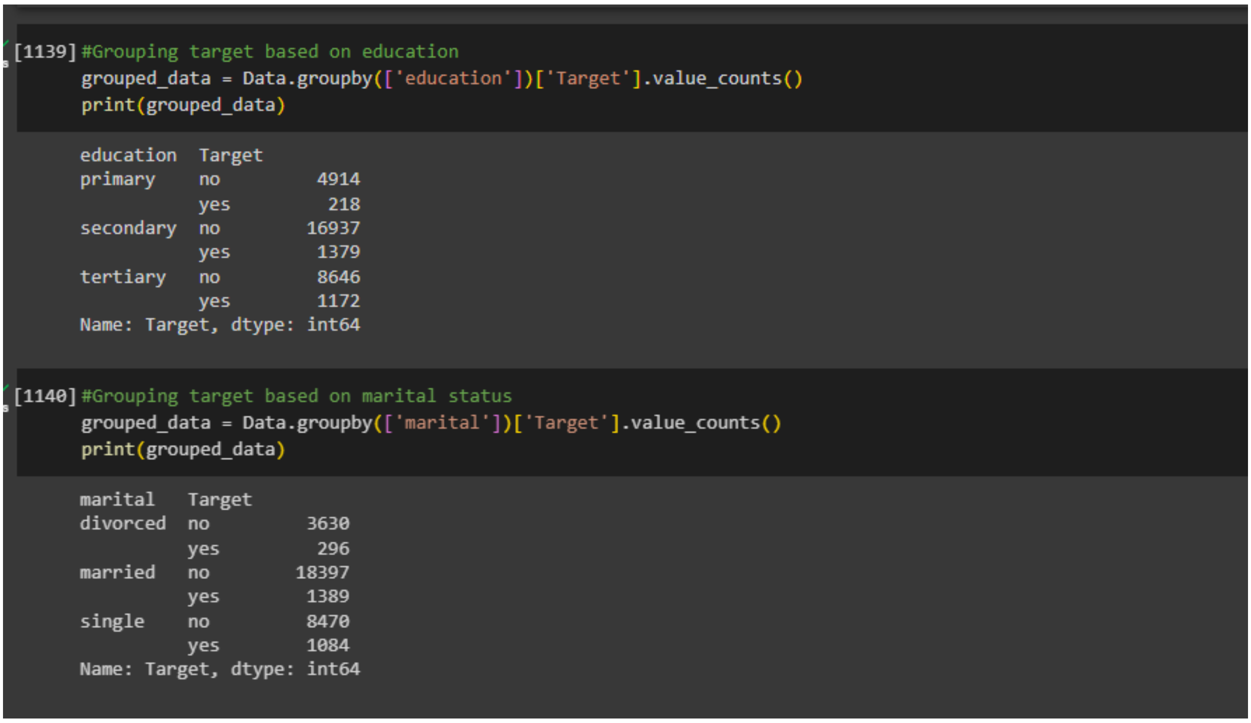


```
#For the output, printing the different marketing efforts
Data[['Target','campaign','duration']].groupby('Target').agg(['mean','median'])
```

```
         campaign           duration
         mean      median   mean        median
Target
no       2.167295  2.0      195.796964  162.0
yes      1.791622  1.0      327.435897  301.0
```

```
[1138] #Grouping target based on job
       grouped_data = Data.groupby(['job'])['Target'].value_counts()
       print(grouped_data)
```

```
job            Target
admin.         no        3676
               yes        382
blue-collar    no        7034
               yes        304
entrepreneur   no        1036
               yes         53
housemaid      no         847
               yes         56
management     no        6090
               yes        723
retired        no        1167
               yes        216
self-employed  no        1049
               yes        100
services       no        3047
               yes        181
student        no         469
               yes        164
technician     no        5235
               yes        467
unemployed     no         847
               yes        123
Name: Target, dtype: int64
```

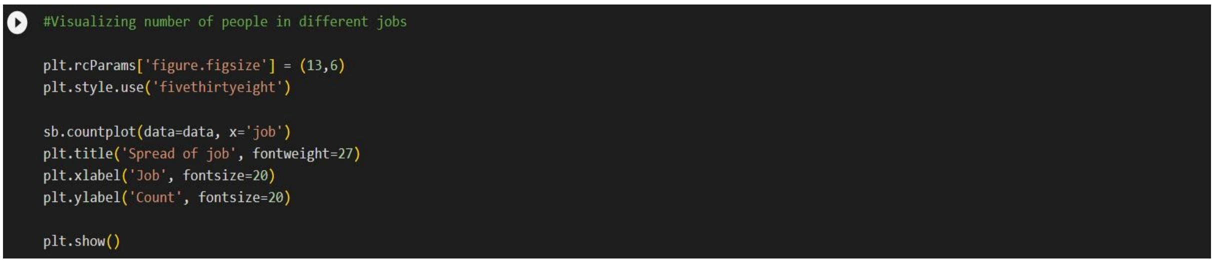


```
[1139] #Grouping target based on education
       grouped_data = Data.groupby(['education'])['Target'].value_counts()
       print(grouped_data)

       education  Target
       primary    no         4914
                  yes         218
       secondary  no        16937
                  yes        1379
       tertiary   no         8646
                  yes        1172
       Name: Target, dtype: int64

[1140] #Grouping target based on marital status
       grouped_data = Data.groupby(['marital'])['Target'].value_counts()
       print(grouped_data)

       marital   Target
       divorced  no         3630
                 yes         296
       married   no        18397
                 yes        1389
       single    no         8470
                 yes        1084
       Name: Target, dtype: int64
```

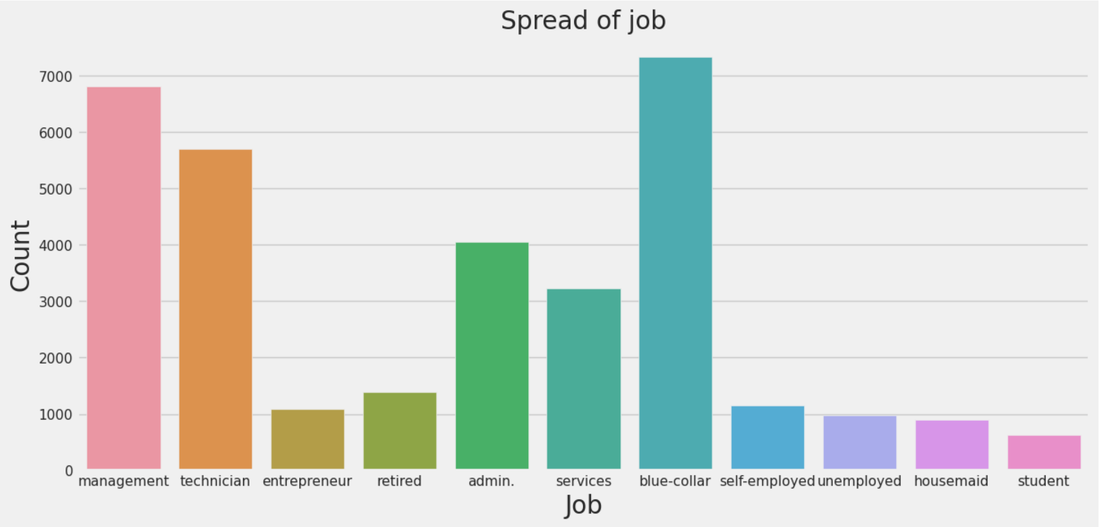
This step was helpful in understanding the data better and to get better data alignment.

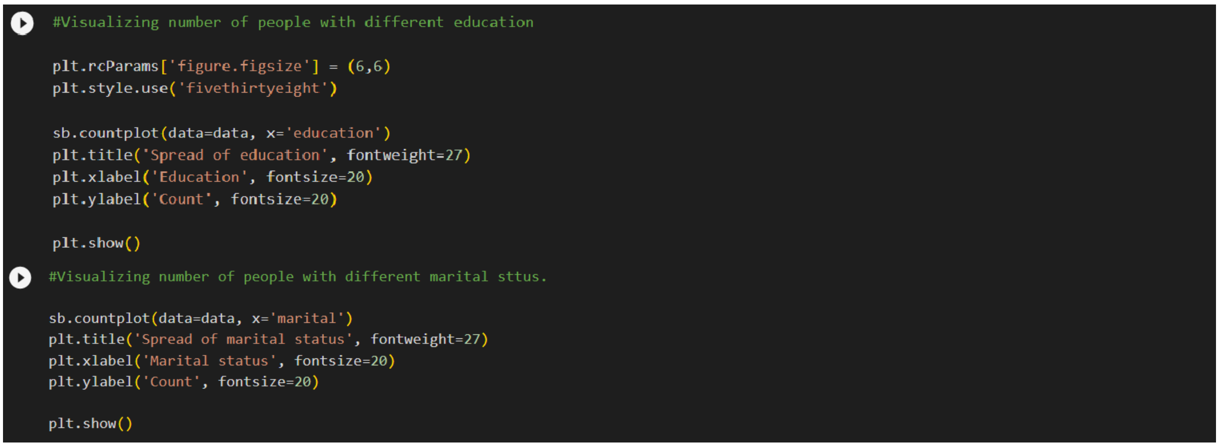
Data Visualization

1. Visualizing the data in different forms to identify the spread of the data and to understand the relationship among two or more features.



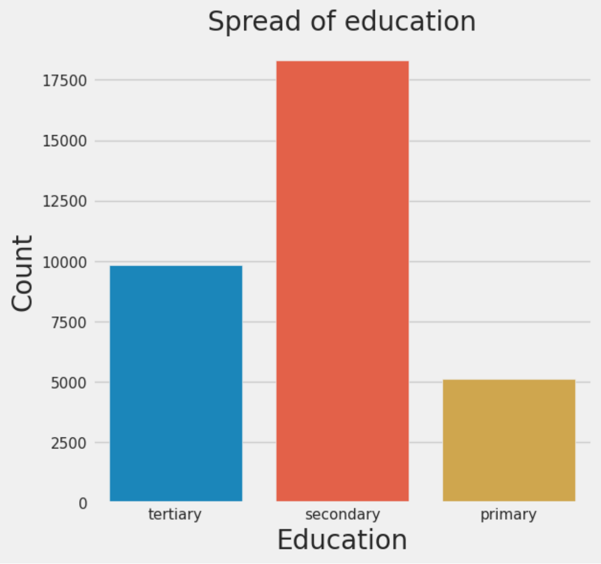
```
#Visualizing number of people in different jobs

plt.rcParams['figure.figsize'] = (13,6)
plt.style.use('fivethirtyeight')

sb.countplot(data=data, x='job')
plt.title('Spread of job', fontweight=27)
plt.xlabel('Job', fontsize=20)
plt.ylabel('Count', fontsize=20)

plt.show()
```

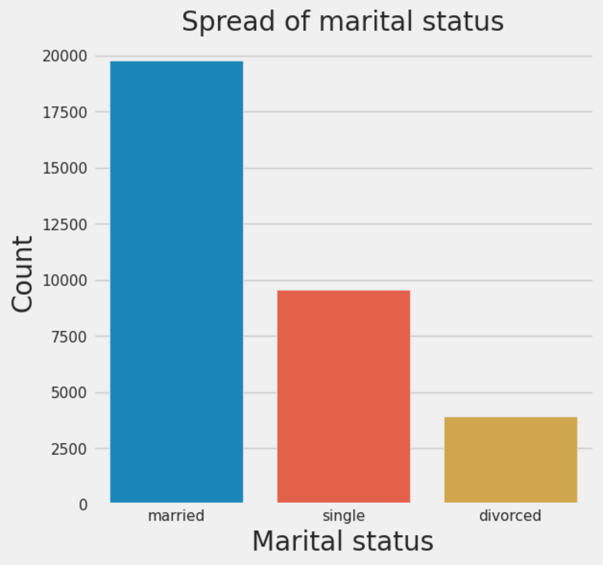


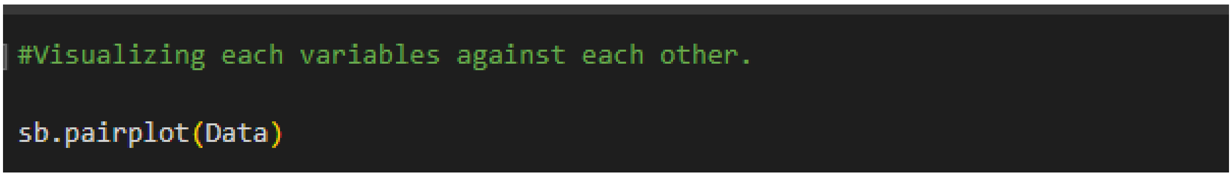


```
#Visualizing number of people with different education

plt.rcParams['figure.figsize'] = (6,6)
plt.style.use('fivethirtyeight')

sb.countplot(data=data, x='education')
plt.title('Spread of education', fontweight=27)
plt.xlabel('Education', fontsize=20)
plt.ylabel('Count', fontsize=20)

plt.show()
```

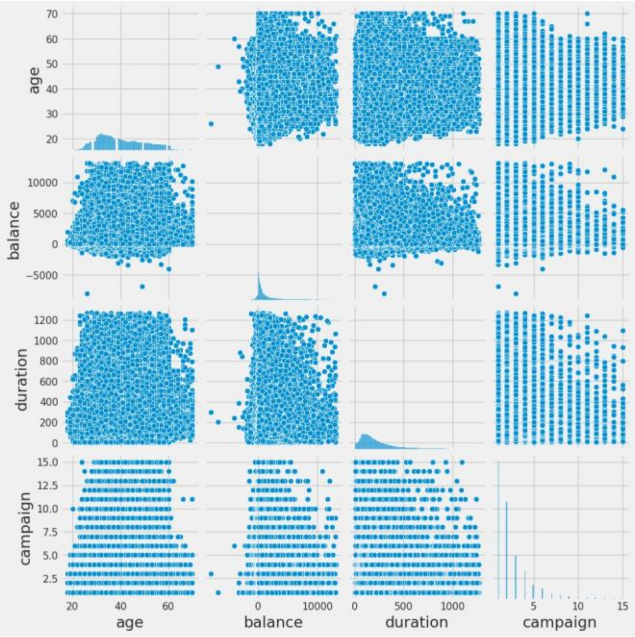
```
#Visualizing number of people with different marital sttus.

sb.countplot(data=data, x='marital')
plt.title('Spread of marital status', fontweight=27)
plt.xlabel('Marital status', fontsize=20)
plt.ylabel('Count', fontsize=20)

plt.show()
```

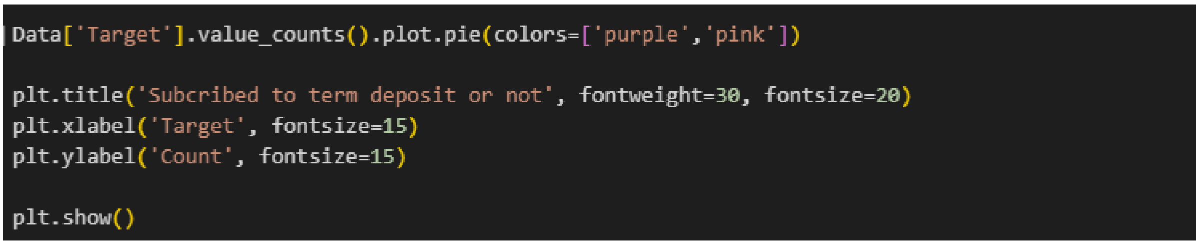




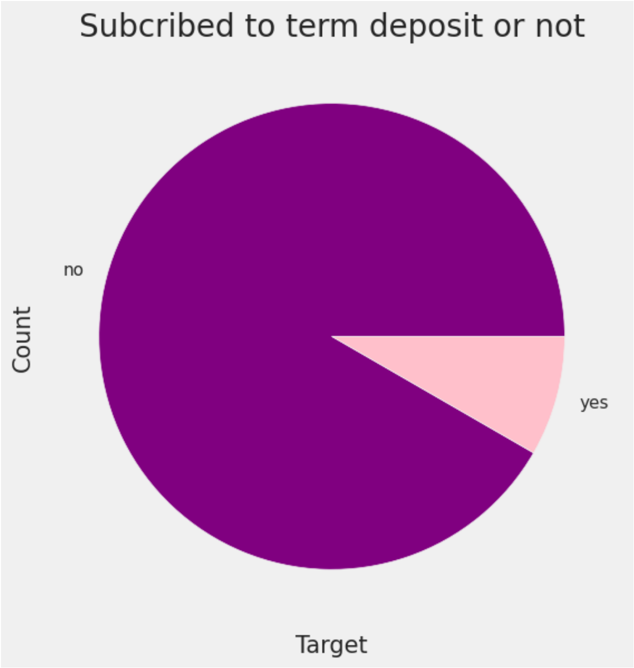


```
#Visualizing each variables against each other.

sb.pairplot(Data)
```

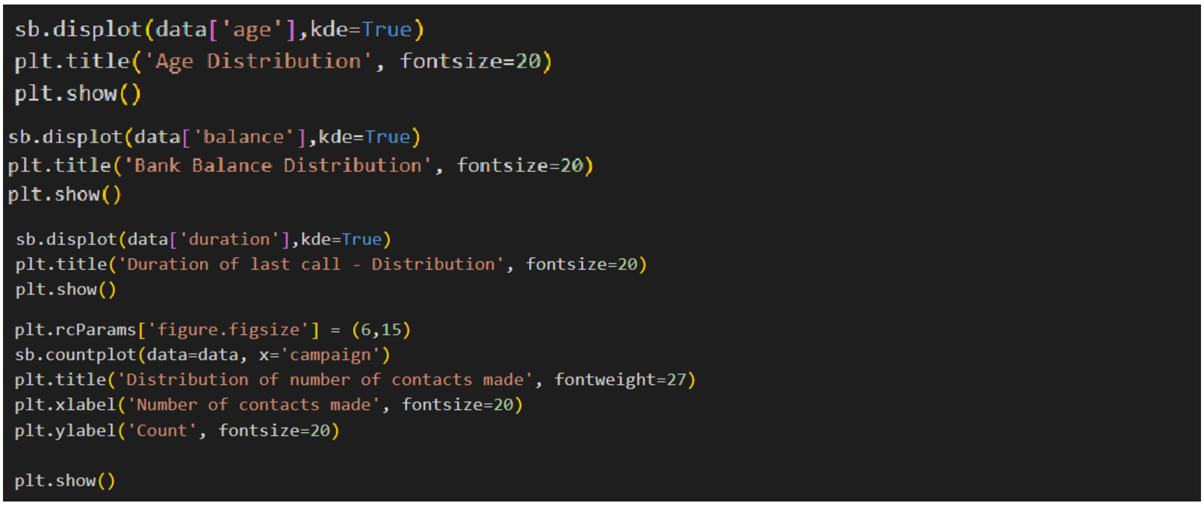




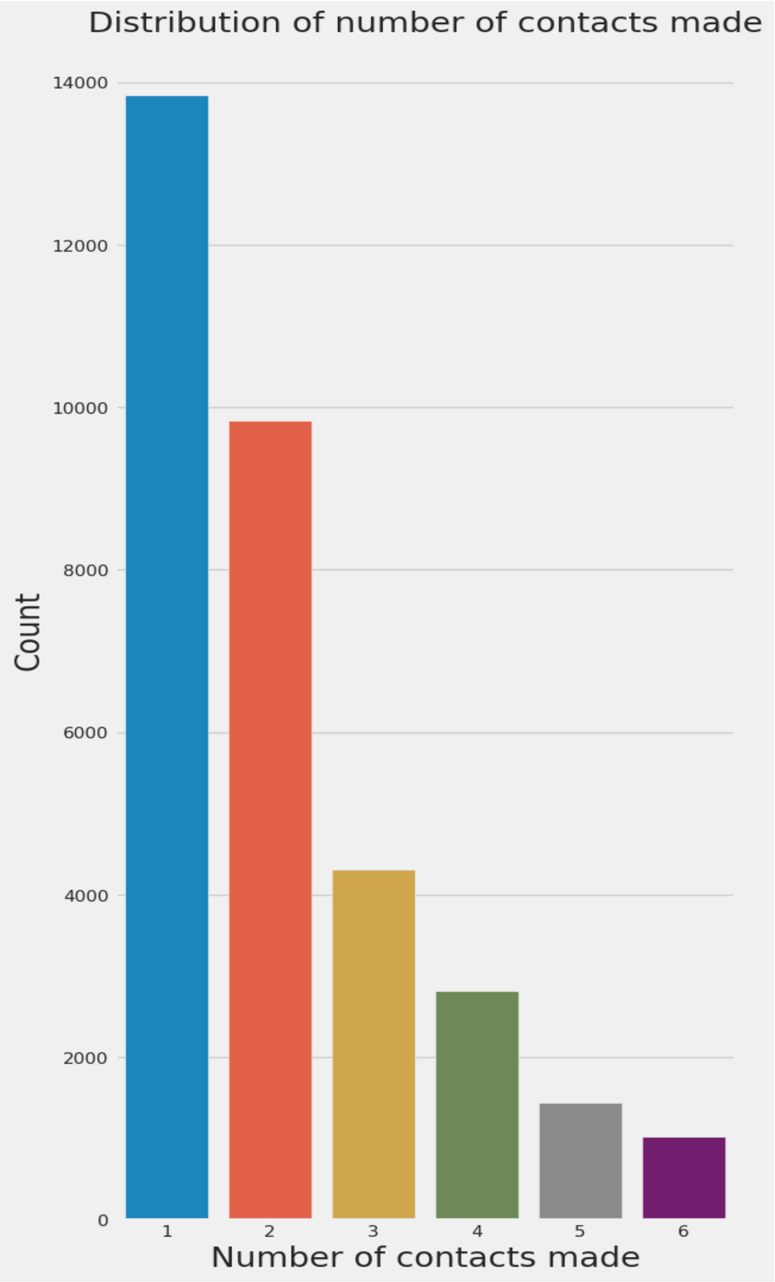
```
Data['Target'].value_counts().plot.pie(colors=['purple','pink'])

plt.title('Subcribed to term deposit or not', fontweight=30, fontsize=20)
plt.xlabel('Target', fontsize=15)
plt.ylabel('Count', fontsize=15)

plt.show()
```

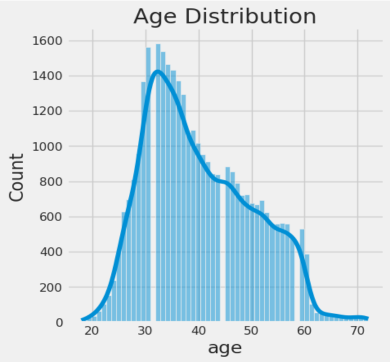


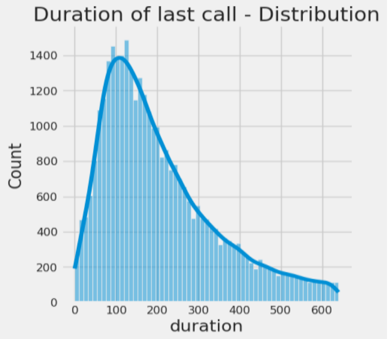
Class imbalance present : That has been corrected using oversampling the minority class in later procedures.

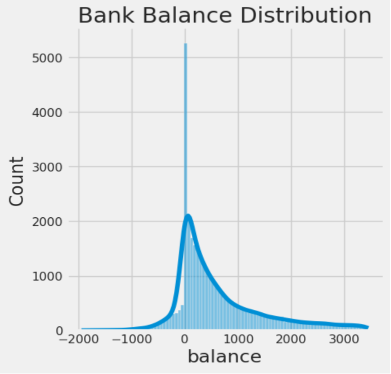


```
sb.displot(data['age'],kde=True)
plt.title('Age Distribution', fontsize=20)
plt.show()
sb.displot(data['balance'],kde=True)
plt.title('Bank Balance Distribution', fontsize=20)
plt.show()
sb.displot(data['duration'],kde=True)
plt.title('Duration of last call - Distribution', fontsize=20)
plt.show()
plt.rcParams['figure.figsize'] = (6,15)
sb.countplot(data=data, x='campaign')
plt.title('Distribution of number of contacts made', fontweight=27)
plt.xlabel('Number of contacts made', fontsize=20)
plt.ylabel('Count', fontsize=20)

plt.show()
```

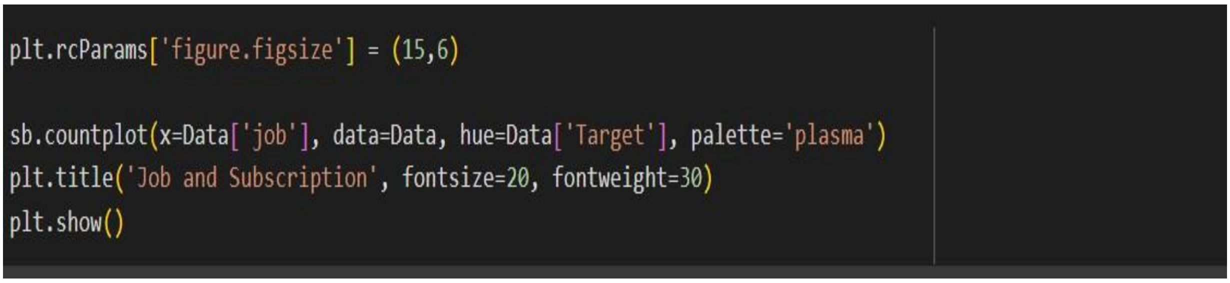




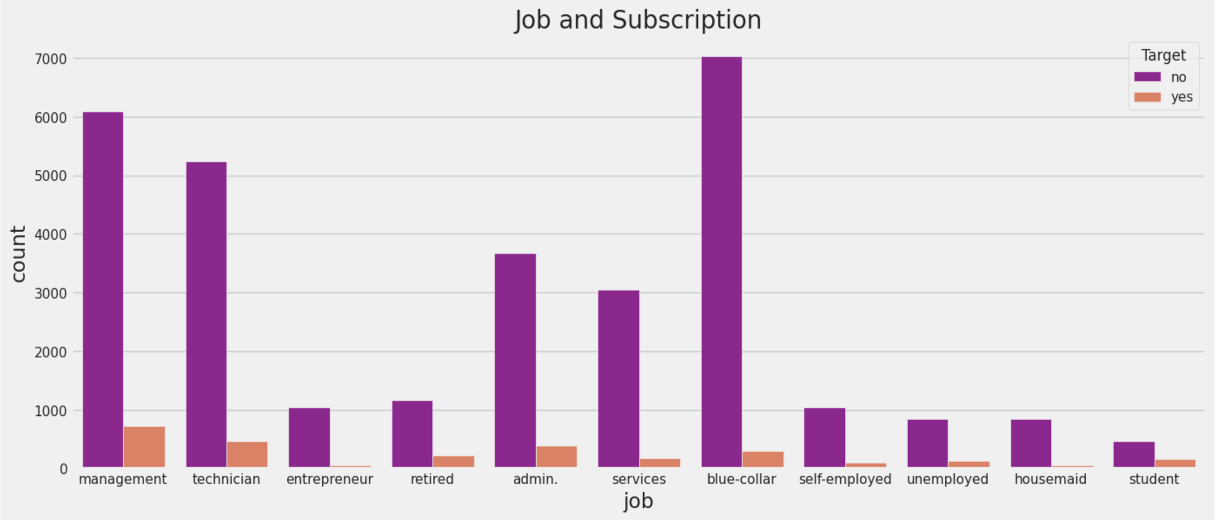


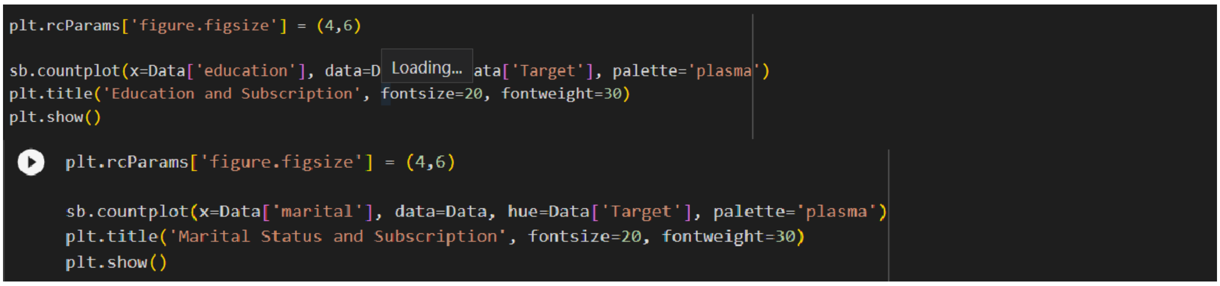


To make distributions normal, Normalization has been performed in the later steps.



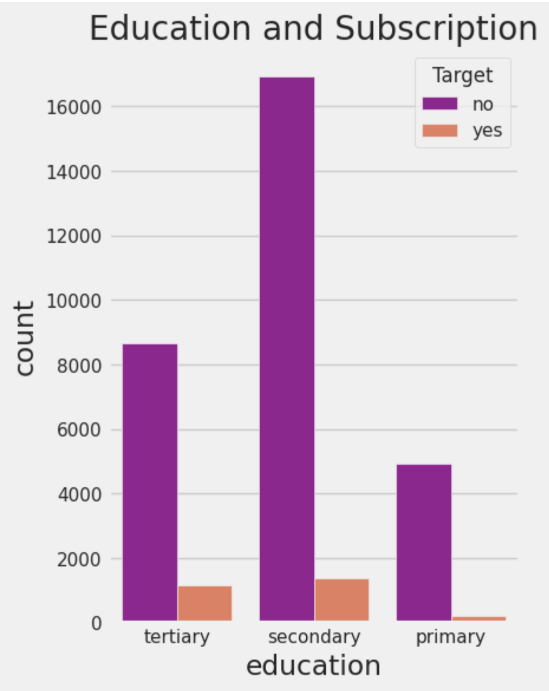
```
plt.rcParams['figure.figsize'] = (15,6)

sb.countplot(x=Data['job'], data=Data, hue=Data['Target'], palette='plasma')
plt.title('Job and Subscription', fontsize=20, fontweight=30)
plt.show()
```

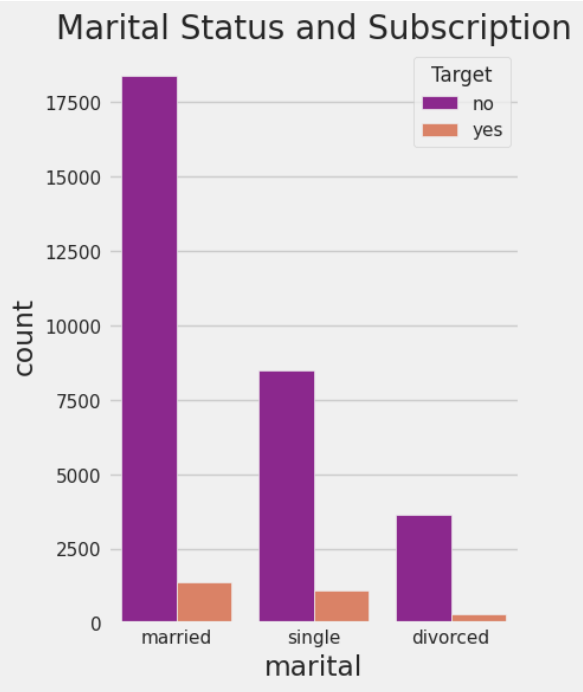


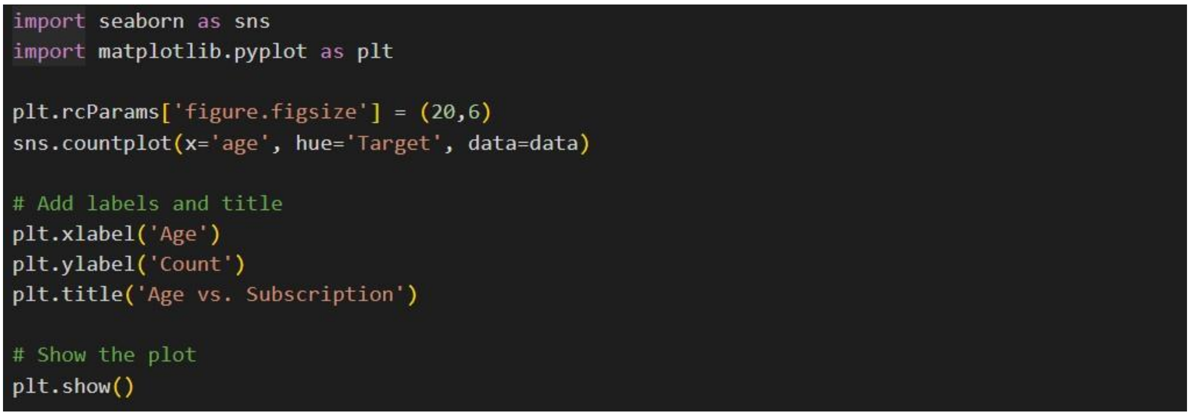


```
plt.rcParams['figure.figsize'] = (4,6)

sb.countplot(x=Data['education'], data=D Loading... ata['Target'], palette='plasma')
plt.title('Education and Subscription', fontsize=20, fontweight=30)
plt.show()
```

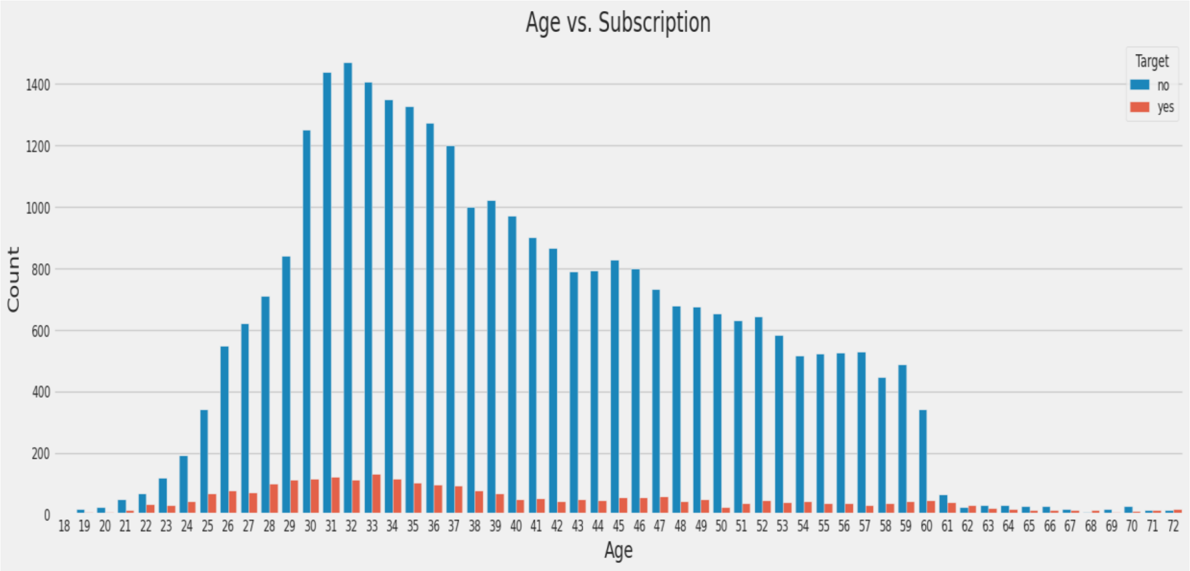
```
plt.rcParams['figure.figsize'] = (4,6)

sb.countplot(x=Data['marital'], data=Data, hue=Data['Target'], palette='plasma')
plt.title('Marital Status and Subscription', fontsize=20, fontweight=30)
plt.show()
```

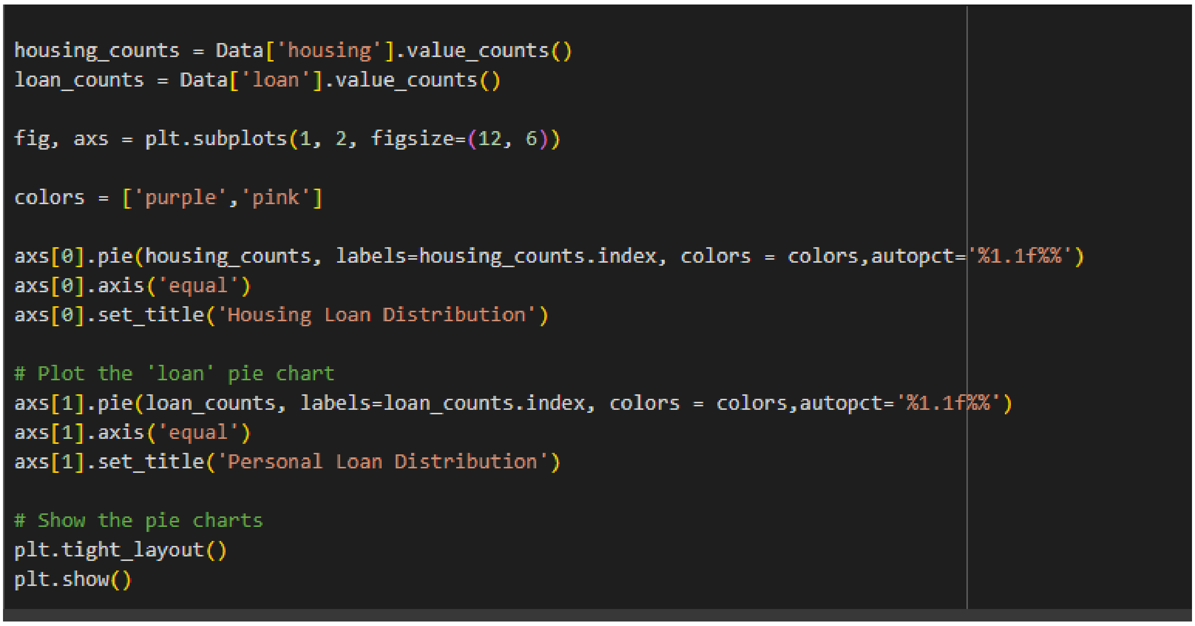




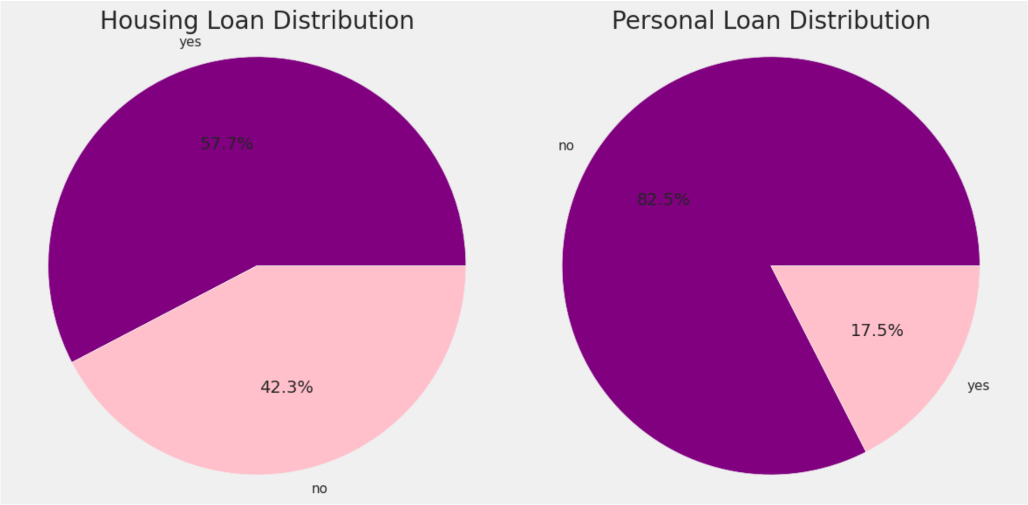


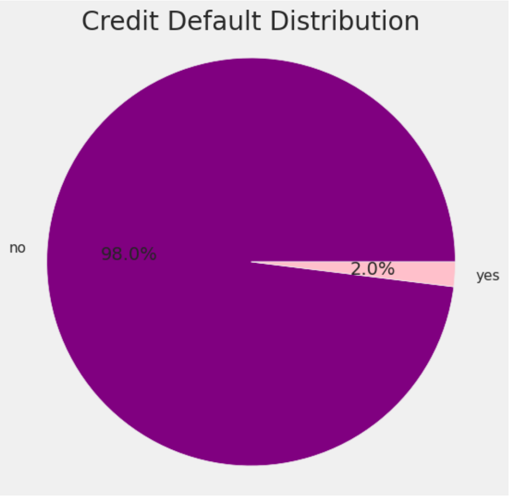
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (20,6)
sns.countplot(x='age', hue='Target', data=data)

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Age vs. Subscription')

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

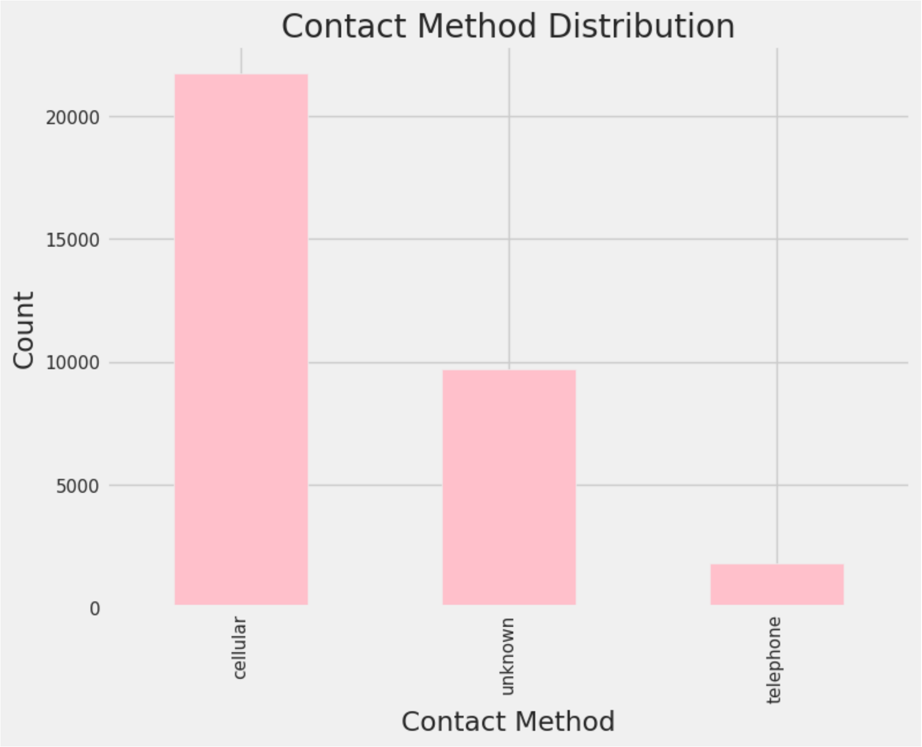




```
housing_counts = Data['housing'].value_counts()
loan_counts = Data['loan'].value_counts()

fig, axs = plt.subplots(1, 2, figsize=(12, 6))

colors = ['purple','pink']

axs[0].pie(housing_counts, labels=housing_counts.index, colors = colors,autopct='%1.1f%%')
axs[0].axis('equal')
axs[0].set_title('Housing Loan Distribution')

# Plot the 'loan' pie chart
axs[1].pie(loan_counts, labels=loan_counts.index, colors = colors,autopct='%1.1f%%')
axs[1].axis('equal')
axs[1].set_title('Personal Loan Distribution')

# Show the pie charts
plt.tight_layout()
plt.show()
```

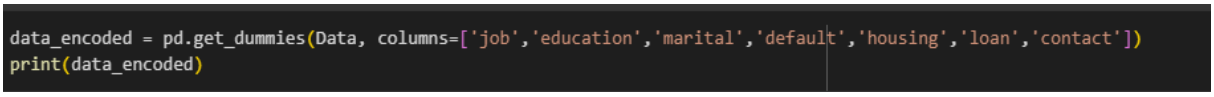

Housing Loan Distribution
yes
57.7%
42.3%
no
Personal Loan Distribution
no
82.5%
17.5%
yes


Credit Default Distribution
no
98.0%
2.0%
yes

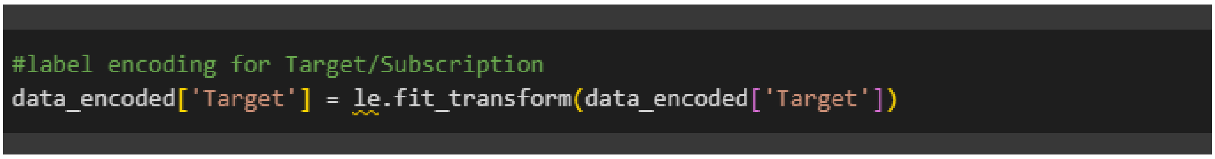

Contact Method Distribution
Count
20000
15000
10000
5000
0
cellular
unknown
telephone
Contact Method

Categorical Variables in Numerical Representation

2. Changing categorical variables into numerical representation – performing label encoding and getting dummy variables. This step helps to make thedataset ready for training.



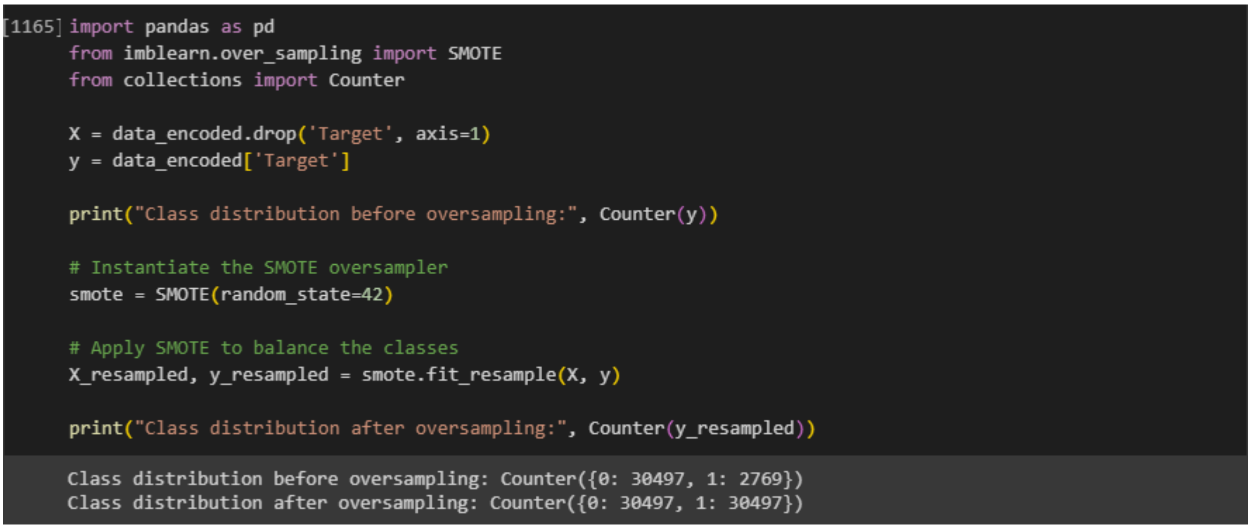
```
data_encoded = pd.get_dummies(Data, columns=['job','education','marital','default','housing','loan','contact'])
print(data_encoded)
```



```
#label encoding for Target/Subscription
data_encoded['Target'] = le.fit_transform(data_encoded['Target'])
```

Dealing with Class Imbalance.

In the Target (Y) Variable, balancing the class distribution in both classes, oversampling has been performed to avoid the model being biased towards the majority class.

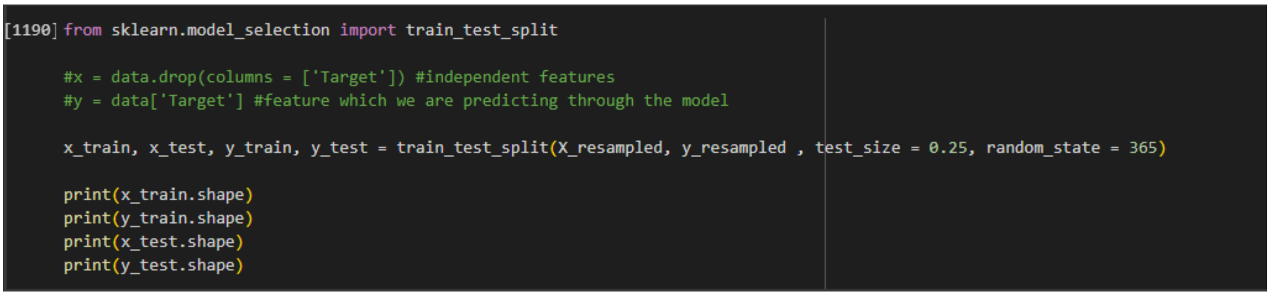


```
[1165] import pandas as pd
       from imblearn.over_sampling import SMOTE
       from collections import Counter

       X = data_encoded.drop('Target', axis=1)
       y = data_encoded['Target']

       print("Class distribution before oversampling:", Counter(y))

       # Instantiate the SMOTE oversampler
       smote = SMOTE(random_state=42)

       # Apply SMOTE to balance the classes
       X_resampled, y_resampled = smote.fit_resample(X, y)

       print("Class distribution after oversampling:", Counter(y_resampled))

       Class distribution before oversampling: Counter({0: 30497, 1: 2769})
       Class distribution after oversampling: Counter({0: 30497, 1: 30497})
```

Train – Test Split

3. Splitting the dataset into train and test.

The training set is used to train the model where the model learns patterns and relationships and the test set is used to evaluate the model’s performance. This could help avoid overfitting and evaluate more accurately.

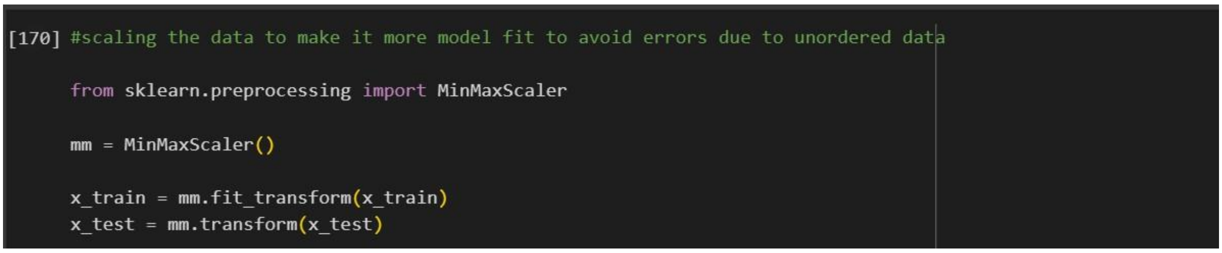


```
[1190] from sklearn.model_selection import train_test_split

       #x = data.drop(columns = ['Target']) #independent features
       #y = data['Target'] #feature which we are predicting through the model

       x_train, x_test, y_train, y_test = train_test_split(X_resampled, y_resampled , test_size = 0.25, random_state = 365)

       print(x_train.shape)
       print(y_train.shape)
       print(x_test.shape)
       print(y_test.shape)
```

Scaling the Features

4. Scaling the features.

It ensures all the features have the same magnitude ensuring equal weights (Enabling all features to contribute equally and to avoid bias)

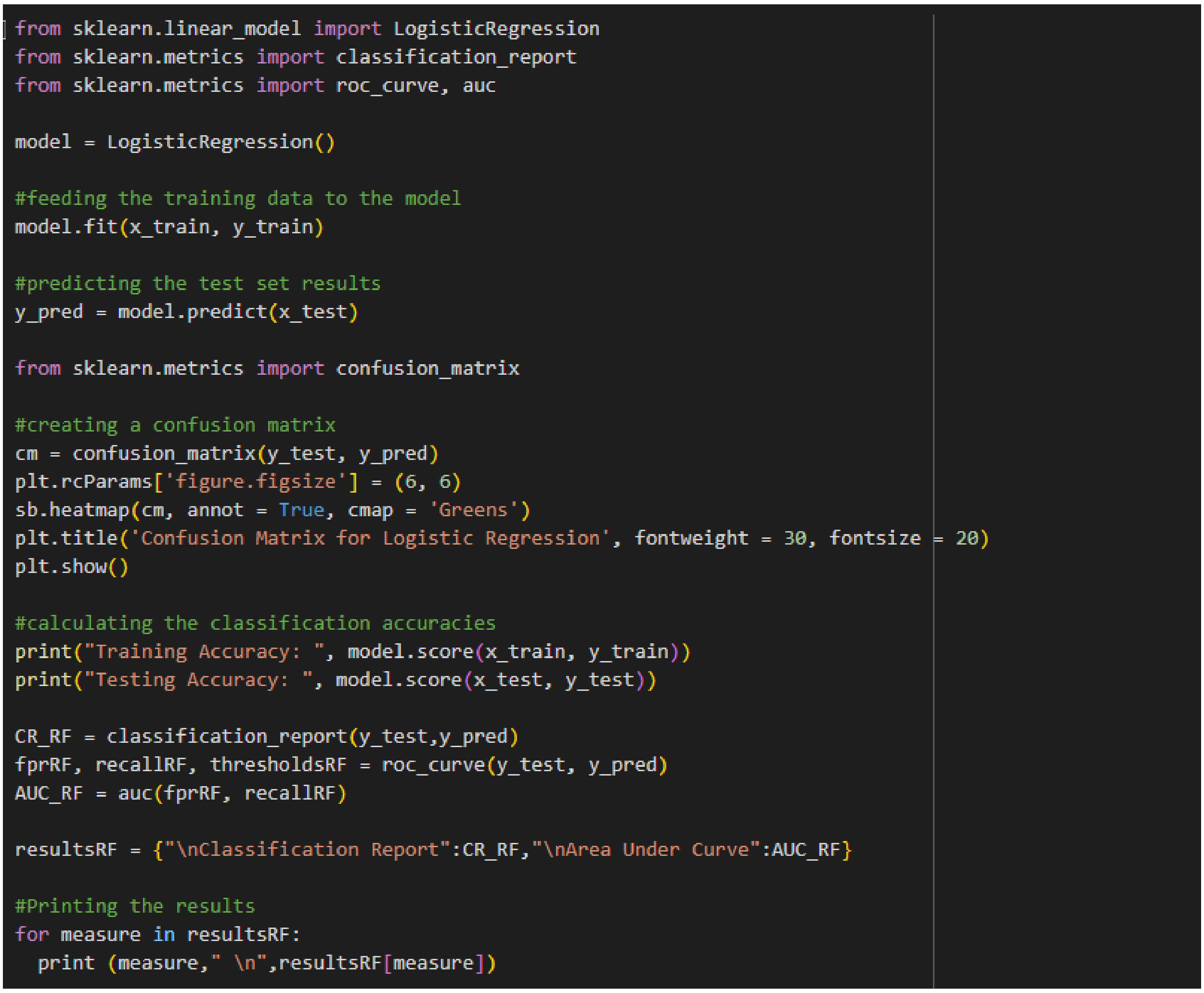


```
[170] #scaling the data to make it more model fit to avoid errors due to unordered data

      from sklearn.preprocessing import MinMaxScaler

      mm = MinMaxScaler()

      x_train = mm.fit_transform(x_train)
      x_test = mm.transform(x_test)
```

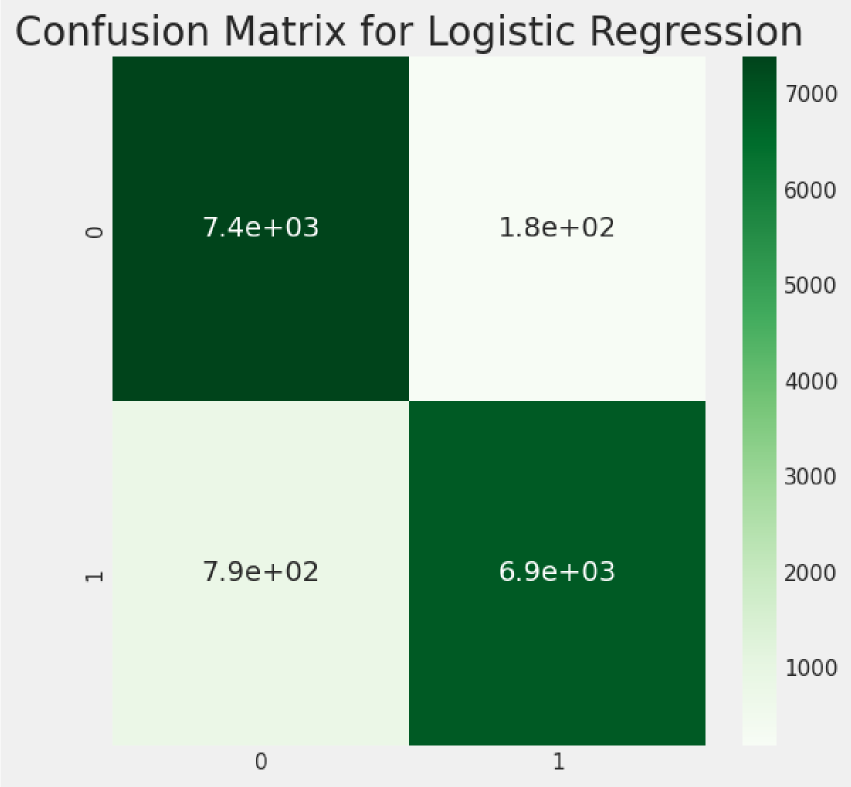
Logistic Regression Model

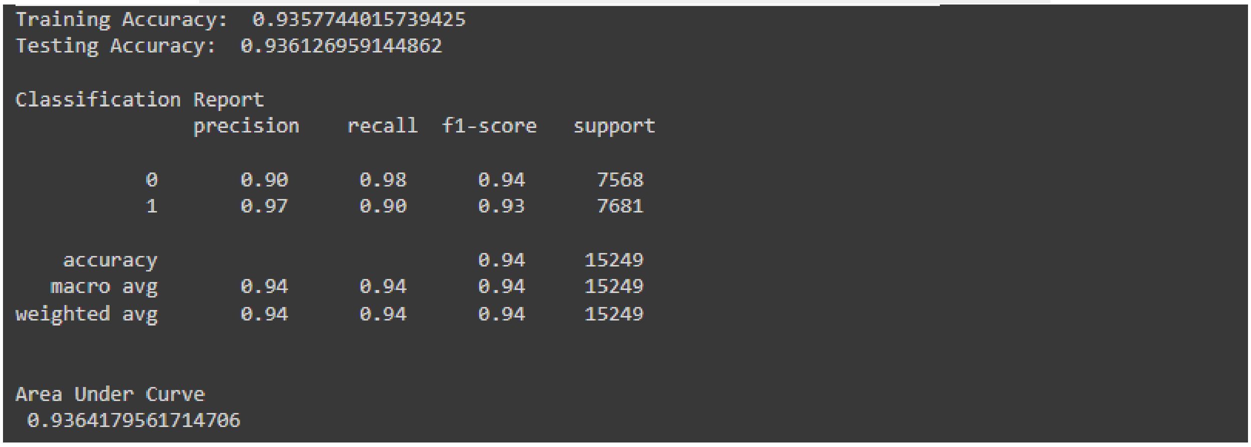
In all the models, class 0 gives ‘No’ which means the customer will not subscribe to the term deposit and class 1 gives ‘Yes’ which means the customer will subscribe to the term deposit

5. Training the Logistic Regression model and printing the performance matrices.



```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc

model = LogisticRegression()

#feeding the training data to the model
model.fit(x_train, y_train)

#predicting the test set results
y_pred = model.predict(x_test)

from sklearn.metrics import confusion_matrix

#creating a confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.rcParams['figure.figsize'] = (6, 6)
sb.heatmap(cm, annot = True, cmap = 'Greens')
plt.title('Confusion Matrix for Logistic Regression', fontweight = 30, fontsize = 20)
plt.show()

#calculating the classification accuracies
print("Training Accuracy: ", model.score(x_train, y_train))
print("Testing Accuracy: ", model.score(x_test, y_test))

CR_RF = classification_report(y_test,y_pred)
fprRF, recallRF, thresholdsRF = roc_curve(y_test, y_pred)
AUC_RF = auc(fprRF, recallRF)

resultsRF = {"\nClassification Report":CR_RF,"\nArea Under Curve":AUC_RF}

#Printing the results
for measure in resultsRF:
  print (measure," \n",resultsRF[measure])
```





```
Training Accuracy:  0.9357744015739425
Testing Accuracy:  0.936126959144862

Classification Report
               precision    recall  f1-score   support

           0       0.90      0.98      0.94      7568
           1       0.97      0.90      0.93      7681

    accuracy                           0.94     15249
   macro avg       0.94      0.94      0.94     15249
weighted avg       0.94      0.94      0.94     15249


Area Under Curve
 0.9364179561714706
```

The confusion matrix shows the number of true positives, false negatives, true negatives, and false negatives. We can see the most of the values are getting classified correctly.

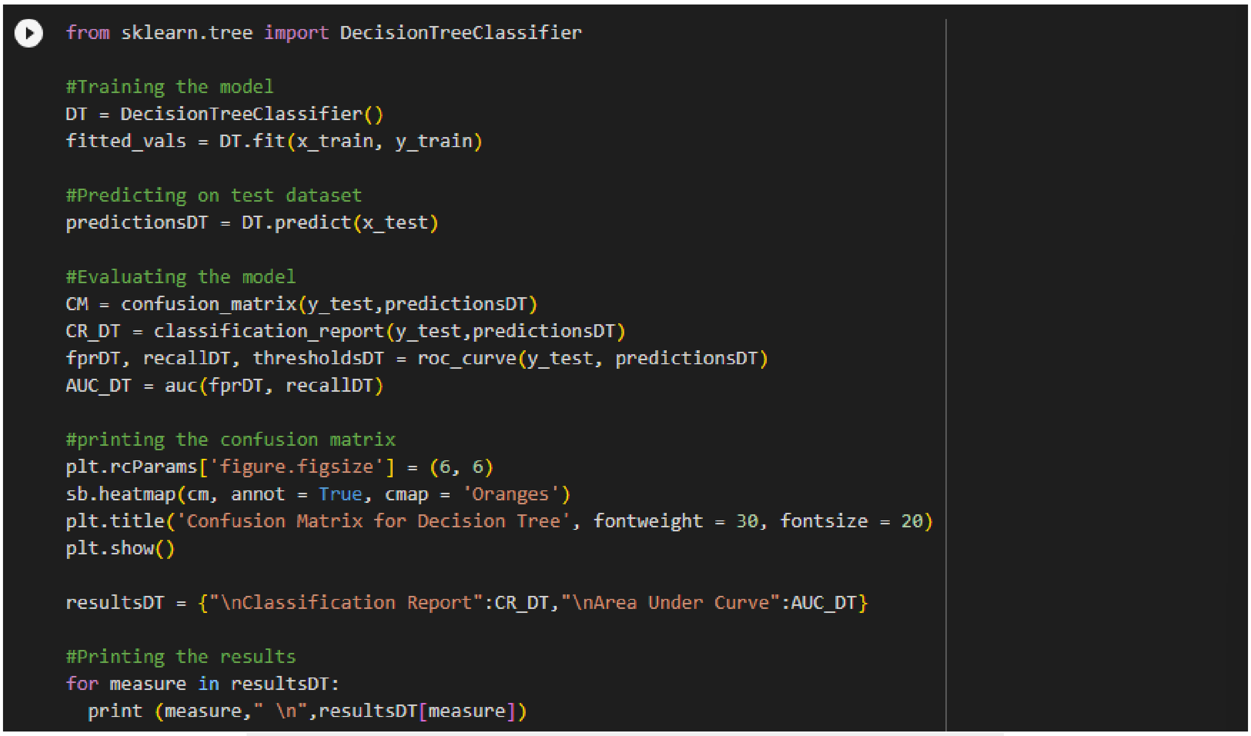
The training accuracy is 0.9357 which means the model achieved an accuracy of 93.57% on the training data. The testing accuracy is 0.9361. The model achieved accuracy of 93.61% on the testing data. The model performs quite well on both test and train data.

Precision is the measurement of accuracy of positive predictions. For class 0, it is 0.90. For class 1 it is 0.97. This means that 91% of the times it predicted class 0 correctly and 97% of times, predicted class 1 correctly. Recall is the ability to find all positive instances. Recall also looks good with values 0.90 for class 1 and 0.90 for class 1 indicating the model gets actual positive instances in a high percentage.. F1 score isa balanced measure that measures the performance of the model. It is a harmonic mean of both precision and recall. F1 scores (0.94,0.93) show high values and a good balance.

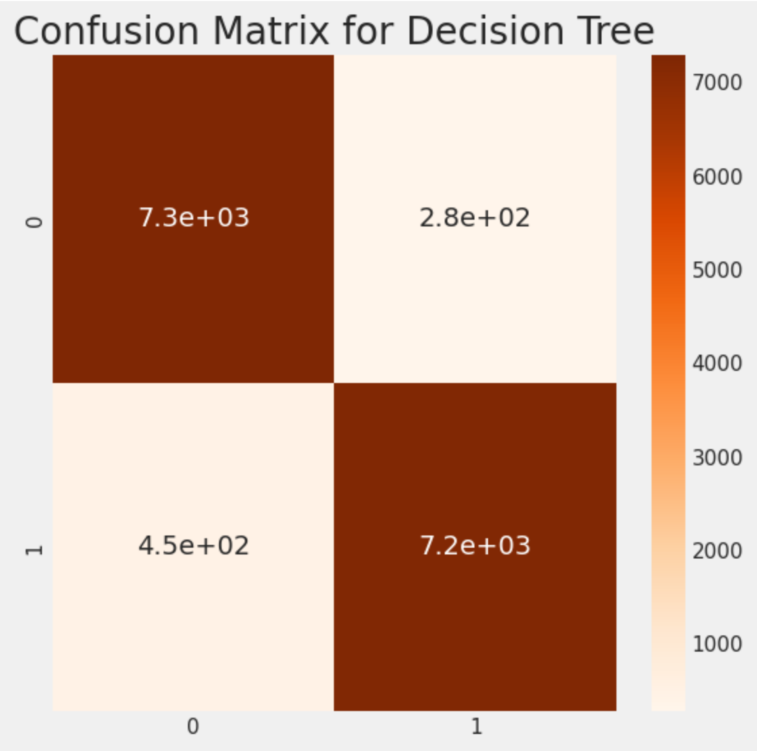
Overall accuracy is 94% meaning the model classified the values correctly 94% of times. AUC stands for Area Under Curve. This shows the ability the model has to differentiate between positive and negative classes. It is 0.9364 . A high value that shows the model performs well in separating classes. In summary, the model performs well.

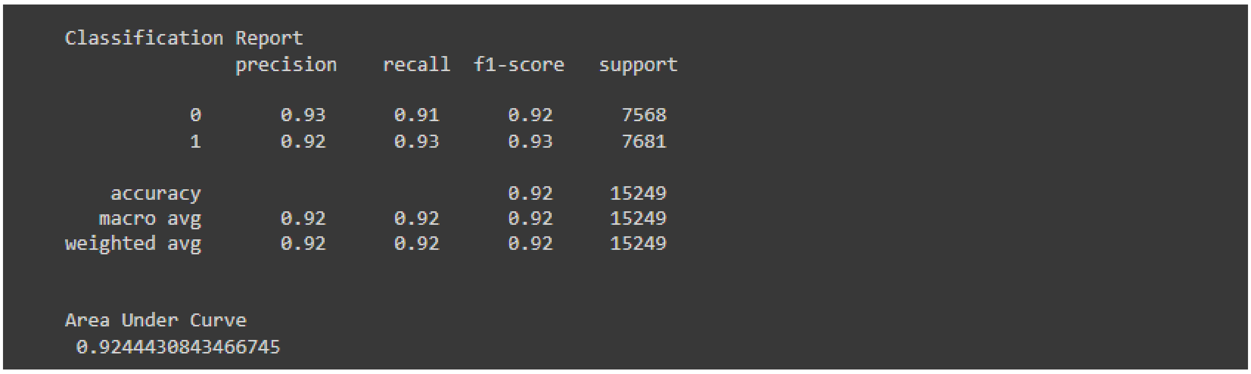
Decision Tree Model

6. Training the Decision Tree model and printing the performance matrices



```
from sklearn.tree import DecisionTreeClassifier

#Training the model
DT = DecisionTreeClassifier()
fitted_vals = DT.fit(x_train, y_train)

#Predicting on test dataset
predictionsDT = DT.predict(x_test)

#Evaluating the model
CM = confusion_matrix(y_test,predictionsDT)
CR_DT = classification_report(y_test,predictionsDT)
fprDT, recallDT, thresholdsDT = roc_curve(y_test, predictionsDT)
AUC_DT = auc(fprDT, recallDT)

#printing the confusion matrix
plt.rcParams['figure.figsize'] = (6, 6)
sb.heatmap(cm, annot = True, cmap = 'Oranges')
plt.title('Confusion Matrix for Decision Tree', fontweight = 30, fontsize = 20)
plt.show()

resultsDT = {"\nClassification Report":CR_DT,"\nArea Under Curve":AUC_DT}

#Printing the results
for measure in resultsDT:
  print (measure," \n",resultsDT[measure])
```



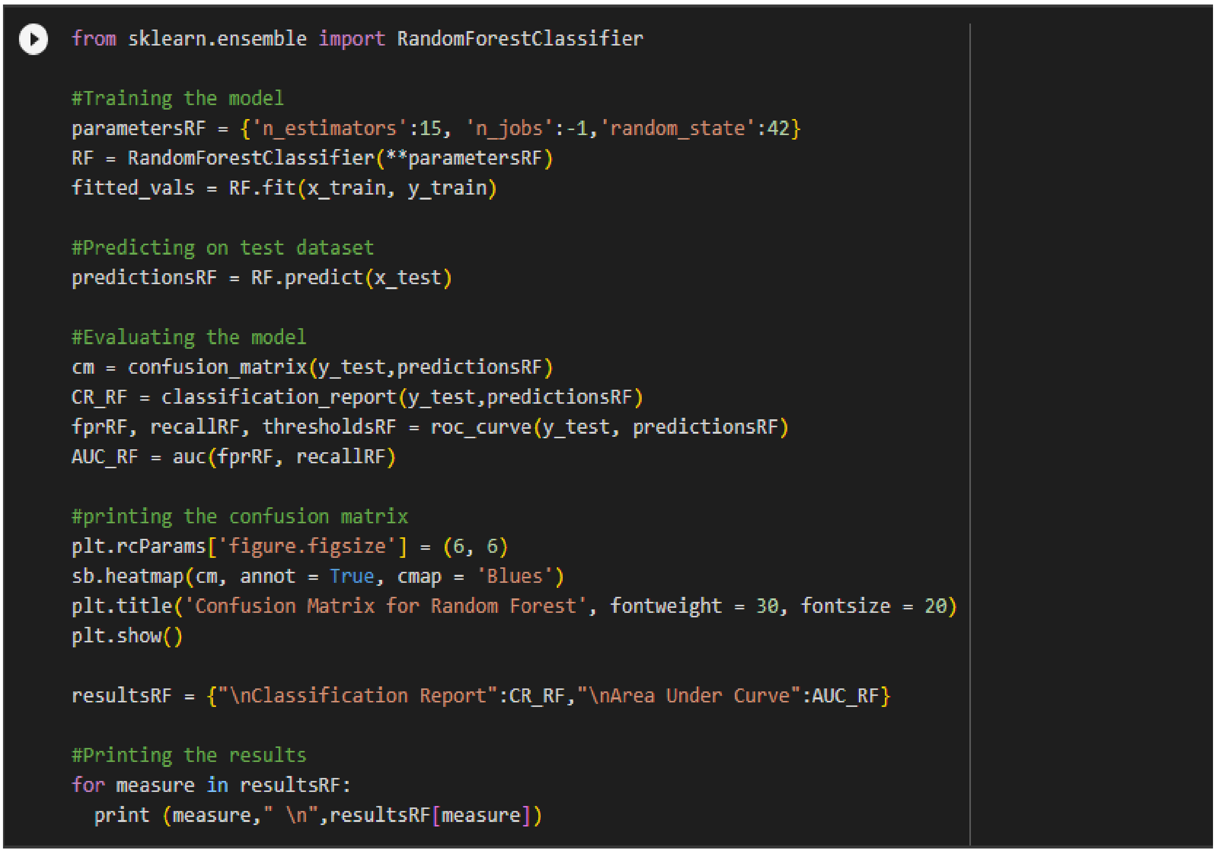


```
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      7568
           1       0.92      0.93      0.93      7681

    accuracy                           0.92     15249
   macro avg       0.92      0.92      0.92     15249
weighted avg       0.92      0.92      0.92     15249


Area Under Curve
 0.9244430843466745
```

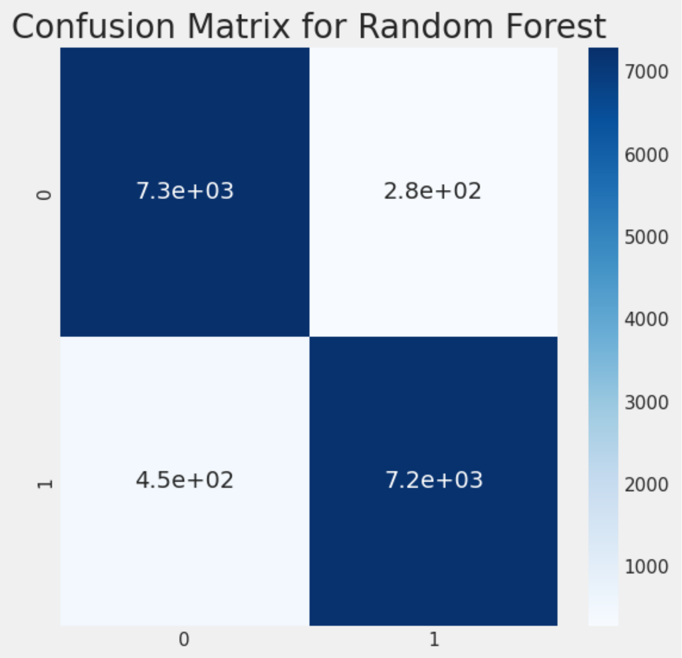
Precision for class 0 is 93% and class 1 is 92%. Model has low rate of false positives in both classes. Recall for class 0 is 91% and class 1 is 0.93%. Model is good at capturing actual positives. F1 score for class 0 is 92% and class 1 is 93%. Both high and balanced values. The overall accuracy is 92% means the model classifies correctly 92% of times. AUC is 0.9244 indicating the model distinguishes well between the classes.

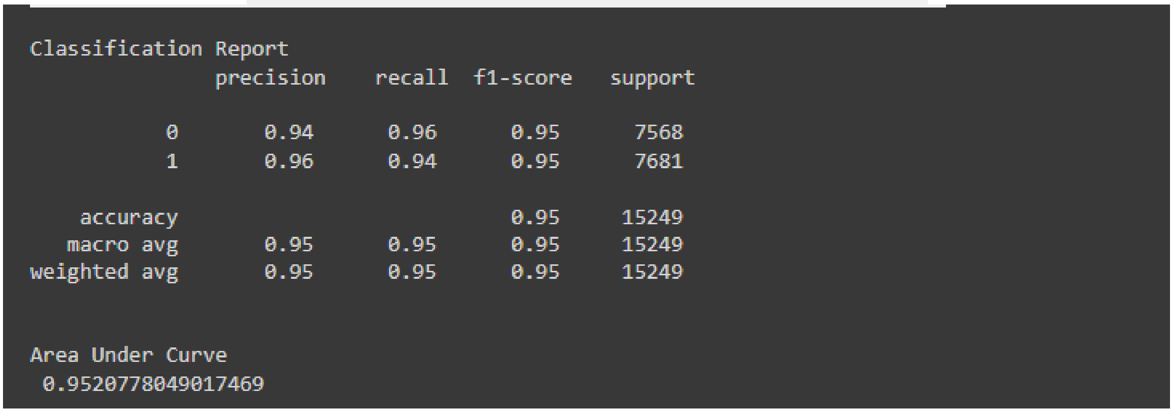
Random Forest Model

7. Training the Random Forest model and printing the performance matrices



```
from sklearn.ensemble import RandomForestClassifier

#Training the model
parametersRF = {'n_estimators':15, 'n_jobs':-1,'random_state':42}
RF = RandomForestClassifier(**parametersRF)
fitted_vals = RF.fit(x_train, y_train)

#Predicting on test dataset
predictionsRF = RF.predict(x_test)

#Evaluating the model
cm = confusion_matrix(y_test,predictionsRF)
CR_RF = classification_report(y_test,predictionsRF)
fprRF, recallRF, thresholdsRF = roc_curve(y_test, predictionsRF)
AUC_RF = auc(fprRF, recallRF)

#printing the confusion matrix
plt.rcParams['figure.figsize'] = (6, 6)
sb.heatmap(cm, annot = True, cmap = 'Blues')
plt.title('Confusion Matrix for Random Forest', fontweight = 30, fontsize = 20)
plt.show()

resultsRF = {"\nClassification Report":CR_RF,"\nArea Under Curve":AUC_RF}

#Printing the results
for measure in resultsRF:
  print (measure," \n",resultsRF[measure])
```





```
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.96      0.95      7568
           1       0.96      0.94      0.95      7681

    accuracy                           0.95     15249
   macro avg       0.95      0.95      0.95     15249
weighted avg       0.95      0.95      0.95     15249


Area Under Curve
 0.9520778049017469
```

The confusion matrix shows how majority of the values are getting correctly classified.

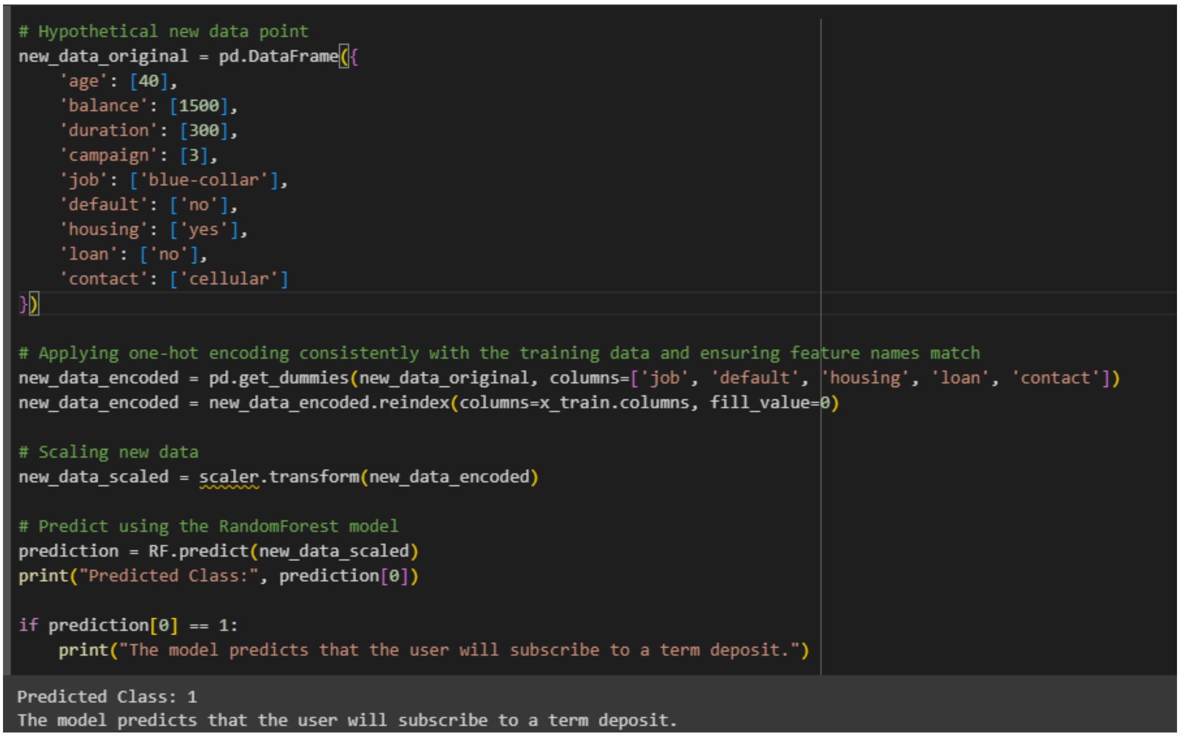
The precision for 0 is 94% and 1 is 96%. These are high precision values showing the model has a very low rate of false positives. Recall for 0 is 96% and 1 is 94%. These are high recall values indicating a very low rate of false negatives. F1 score for both classes is 95% which is a high value and also a good balance between the two classes. Overall accuracy is 95%. The AUC value of 0.9521 shows the model has strong discriminatory power (The model is well-abled to distinguish between the positive and negative class.)

In conclusion, the random forest model performs very well and it has even better evaluation metrics than the logistic regression and the decision tree models. Hence, finalizing the random forest model for predictions.

Predicting New Data

The following line of code gives a new hypothetical data point and predicts the Target variable using the Random Forest Model.

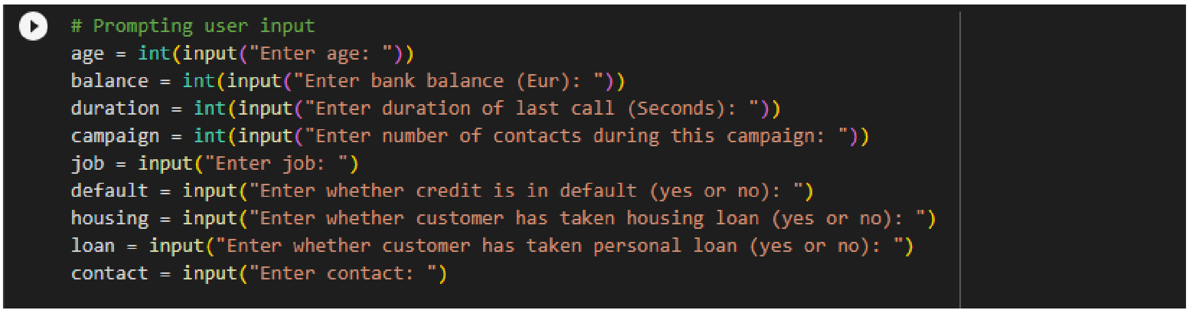
It makes sure to stay consistent with the columns of the trained data (with encoded and dummy columns) and since scaling is performed to training and testing data, the new data is also scaled before the prediction.



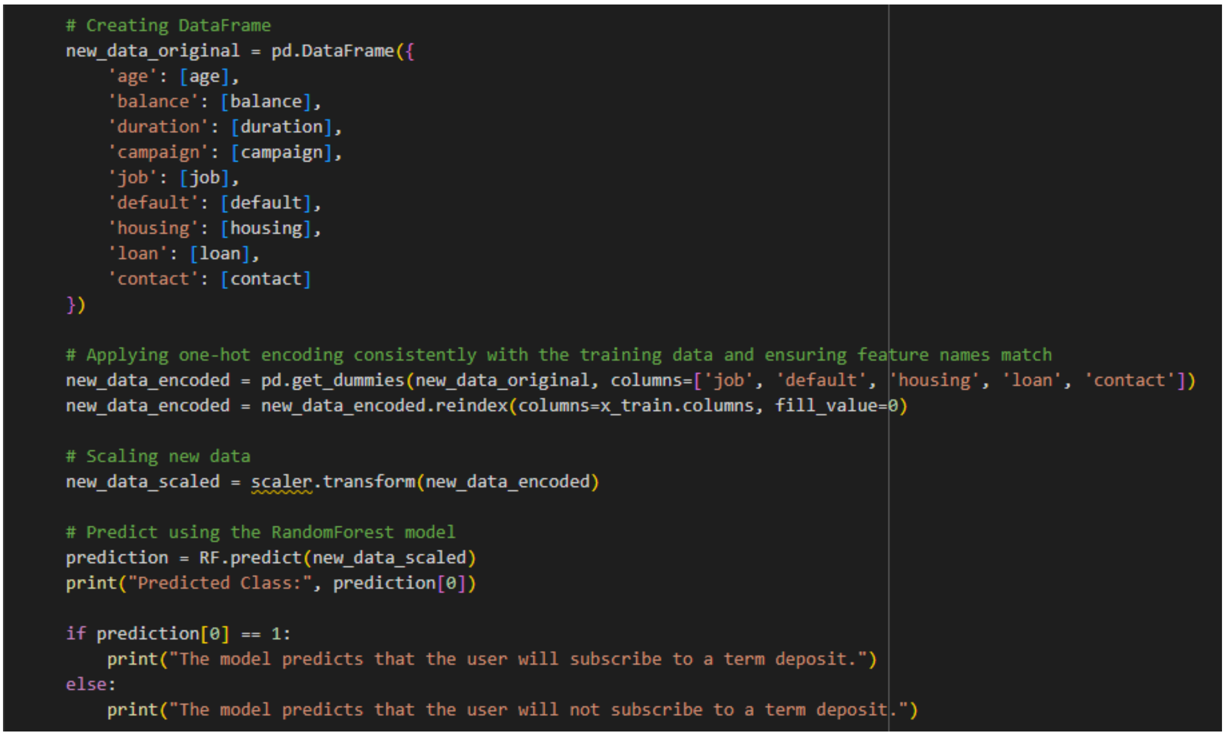
```
# Hypothetical new data point
new_data_original = pd.DataFrame({
    'age': [40],
    'balance': [1500],
    'duration': [300],
    'campaign': [3],
    'job': ['blue-collar'],
    'default': ['no'],
    'housing': ['yes'],
    'loan': ['no'],
    'contact': ['cellular']
})

# Applying one-hot encoding consistently with the training data and ensuring feature names match
new_data_encoded = pd.get_dummies(new_data_original, columns=['job', 'default', 'housing', 'loan', 'contact'])
new_data_encoded = new_data_encoded.reindex(columns=x_train.columns, fill_value=0)

# Scaling new data
new_data_scaled = scaler.transform(new_data_encoded)

# Predict using the RandomForest model
prediction = RF.predict(new_data_scaled)
print("Predicted Class:", prediction[0])

if prediction[0] == 1:
    print("The model predicts that the user will subscribe to a term deposit.")
```

```
Predicted Class: 1
The model predicts that the user will subscribe to a term deposit.
```

The following code does the same procedure as before but asks the new data as user input and predict the Target Class.

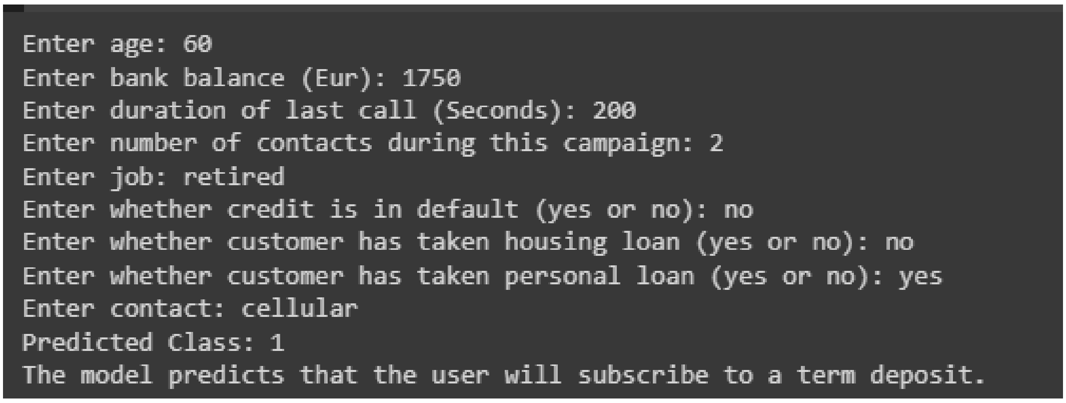


```
# Prompting user input
age = int(input("Enter age: "))
balance = int(input("Enter bank balance (Eur): "))
duration = int(input("Enter duration of last call (Seconds): "))
campaign = int(input("Enter number of contacts during this campaign: "))
job = input("Enter job: ")
default = input("Enter whether credit is in default (yes or no): ")
housing = input("Enter whether customer has taken housing loan (yes or no): ")
loan = input("Enter whether customer has taken personal loan (yes or no): ")
contact = input("Enter contact: ")
```



```
# Creating DataFrame
new_data_original = pd.DataFrame({
    'age': [age],
    'balance': [balance],
    'duration': [duration],
    'campaign': [campaign],
    'job': [job],
    'default': [default],
    'housing': [housing],
    'loan': [loan],
    'contact': [contact]
})

# Applying one-hot encoding consistently with the training data and ensuring feature names match
new_data_encoded = pd.get_dummies(new_data_original, columns=['job', 'default', 'housing', 'loan', 'contact'])
new_data_encoded = new_data_encoded.reindex(columns=x_train.columns, fill_value=0)

# Scaling new data
new_data_scaled = scaler.transform(new_data_encoded)

# Predict using the RandomForest model
prediction = RF.predict(new_data_scaled)
print("Predicted Class:", prediction[0])

if prediction[0] == 1:
    print("The model predicts that the user will subscribe to a term deposit.")
else:
    print("The model predicts that the user will not subscribe to a term deposit.")
```

Result:



```
Enter age: 60
Enter bank balance (Eur): 1750
Enter duration of last call (Seconds): 200
Enter number of contacts during this campaign: 2
Enter job: retired
Enter whether credit is in default (yes or no): no
Enter whether customer has taken housing loan (yes or no): no
Enter whether customer has taken personal loan (yes or no): yes
Enter contact: cellular
Predicted Class: 1
The model predicts that the user will subscribe to a term deposit.
```

SUMMARY

In short, the report outlines a comprehensive procedure for solving a classification problem : Predicting Term Deposit Subscription Based on Customer Demographics, Banking History, and the Current Marketing Campaign.

It follows a systematic approach explaining multiple steps, including data preprocessing (such as handling missing values and removing outliers), exploratory data analysis (including checking for correlations and visualizing the data), addressing class imbalance, splitting the data into training and testing sets, feature scaling, building and evaluating three classification models.

The models trained and tested are, logistic regression, decision tree and random forest and the random forest model has been chosen as the final model to classify the target variables/ make predictions due to its strong performance with high precision, recall, F1 score, accuracy and AUC.