Project: Bank Marketing (Campaign)

Name: Suyog Nemade

Email: suyognemade005@gmail.com

Country: USA

Batch Code: LISUM38

Specialization: Data Science

Submitted to: Data Glacier

(Individual project)

Table of Contents

1. Problem Description

ABC Bank wants to sell it's term deposit product to customers and before launching the product they want to develop a model which help them in understanding whether a particular customer will buy their product or not (based on customer's past interaction with bank or other Financial Institution). This is an application of the organization's marketing data.

2. Business Understanding

In predicting the results of marketing campaign for each customer and interpreting which all features affect the results, will help the organization understand how to make campaign more efficient. Moreover, in categorizing which segment of customers subscribed the term deposit, helps to identify who is more likely to buy the product in future thereby developing more targeted marketing campaigns. This can be achieved using ML model that shortlists the customer whose chance of buying the product is more so that their marketing channel (tele marketing, SMS/email marketing etc) can focus only to those customers. This will save resource and their time.

3. Data understanding

- <u>Types of data</u>

The dataset contains 20 independent variables and 1 dependent variable. The independent variables are the following:

1 - age (numeric)
2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','selfemployed',' services','student','technician','unemployed','unknown')
3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4 - education (categorical:

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

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

6 - housing: has housing loan? (categorical: 'no','yes','unknown') 7 - loan: has personal loan? (categorical: 'no','yes','unknown') # related with the last contact of the current campaign:

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

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

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

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model. 12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

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

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

# social and economic context attributes

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

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

19 - euribor3m: euribor 3 month rate - daily indicator (numeric) 20 - nr.employed: number of employees - quarterly indicator (numeric) Output variable (desired target):

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

The independent variables are a mix of categorical and numerical values.

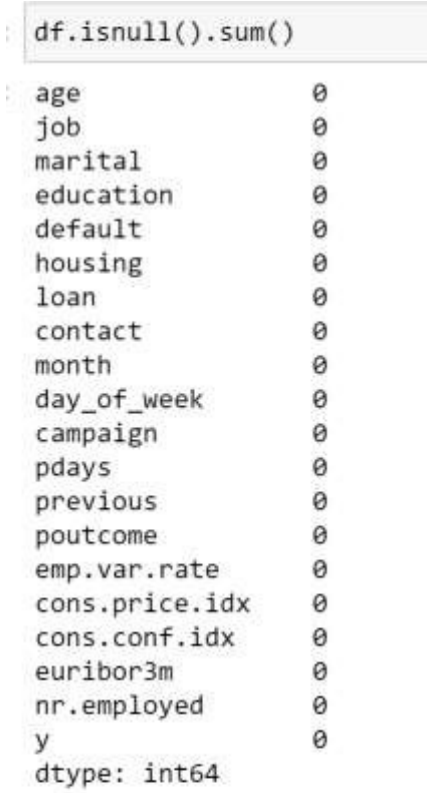
As instructed in the dataset, the **duration** feature is dropped from the dataset.

- <u>Problems in the data</u>

The dataset contains a total of 41188 rows, out of which there were a total of 12 duplicate rows. The pandas **drop_duplicates** method is used to drop the duplicate rows.

<u>Missing Values:</u>

There are no missing values in the dataset.

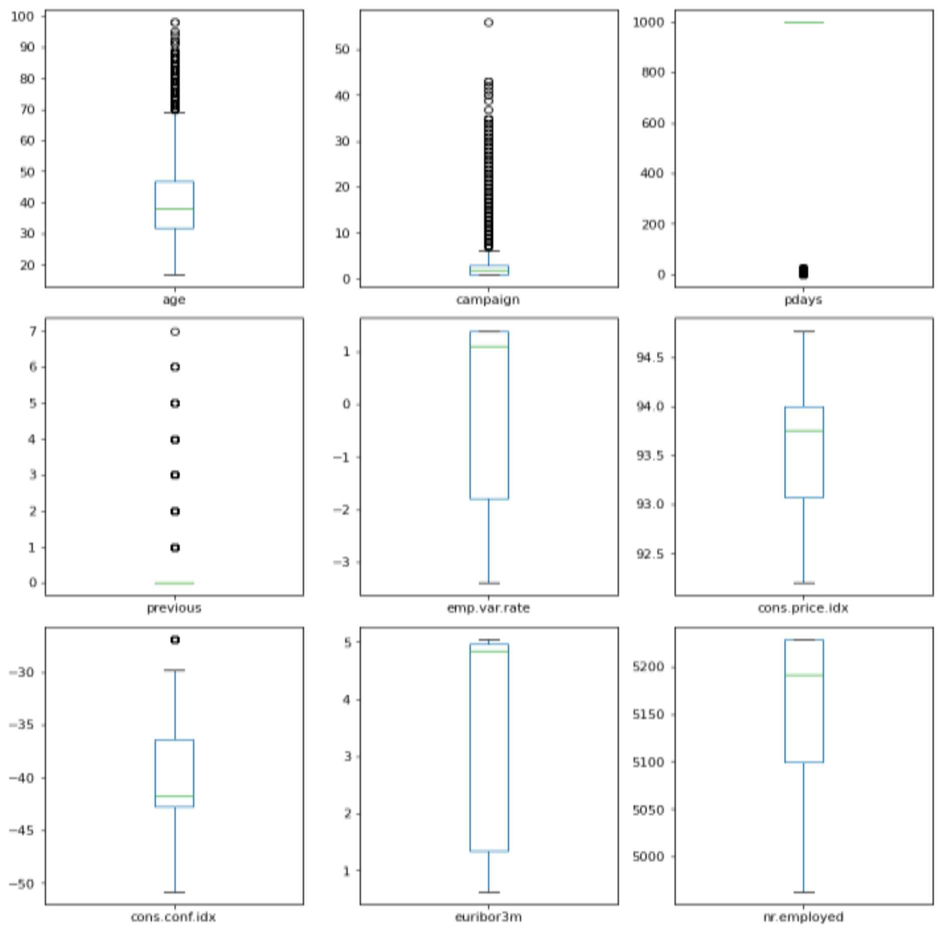


```
df.isnull().sum()
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       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
```

<u>Outlier detection</u>

Outliers are detected using box plots.

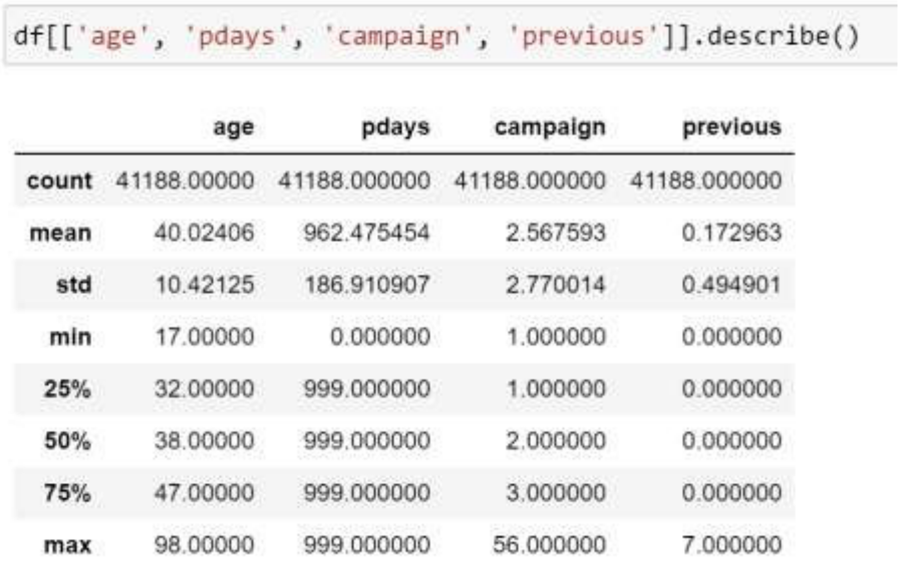


Outliers represent those values which are at an abnormal distance from the central distribution of data points. These values affect the statistics of the data mainly mean and mode. Therefore it's important to deal with outliers in the data cleaning stage so that the performance of the model don't get compromised.

In the box plot, outliers are seen in the features **age, campaign, pdays and previous** which are indicated as data points outside the whiskers of the box plot.

- <u>Approaches to overcome the problems</u>

Considering the statistics of the outlier features

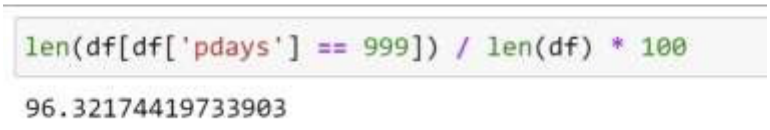


```
df[['age', 'pdays', 'campaign', 'previous']].describe()
```

	age	pdays	campaign	previous
count	41188.00000	41188.000000	41188.000000	41188.000000
mean	40.02406	962.475454	2.567593	0.172963
std	10.42125	186.910907	2.770014	0.494901
min	17.00000	0.000000	1.000000	0.000000
25%	32.00000	999.000000	1.000000	0.000000
50%	38.00000	999.000000	2.000000	0.000000
75%	47.00000	999.000000	3.000000	0.000000
max	98.00000	999.000000	56.000000	7.000000

i. **age:** The maximum value of age is 98 which seems to be realistic. Therefore, it's not dropped.

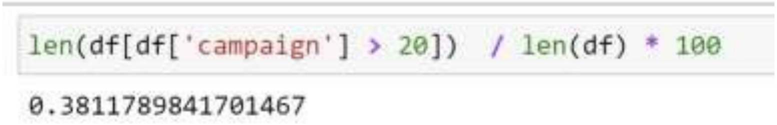
ii. **pdays:** number of days that passed by after the client was last contacted from a previous campaign. From the statistics, the maximum value of **pdays** is 999. In the features description, this value means the client was not previously contacted. Moreover around 96% of rows contains this value, therefore dropping rows containing 999 seems unrealistic.



```
len(df[df['pdays'] == 999]) / len(df) * 100
```

```
96.32174419733903
```

iii. **campaign:** 'campaign' holds the number of contacts performed during this campaign and for this client. From the statistics, the maximum value is given as 56 which is clearly noise. Numbers for 'campaign' above 20 is around 0.38%.



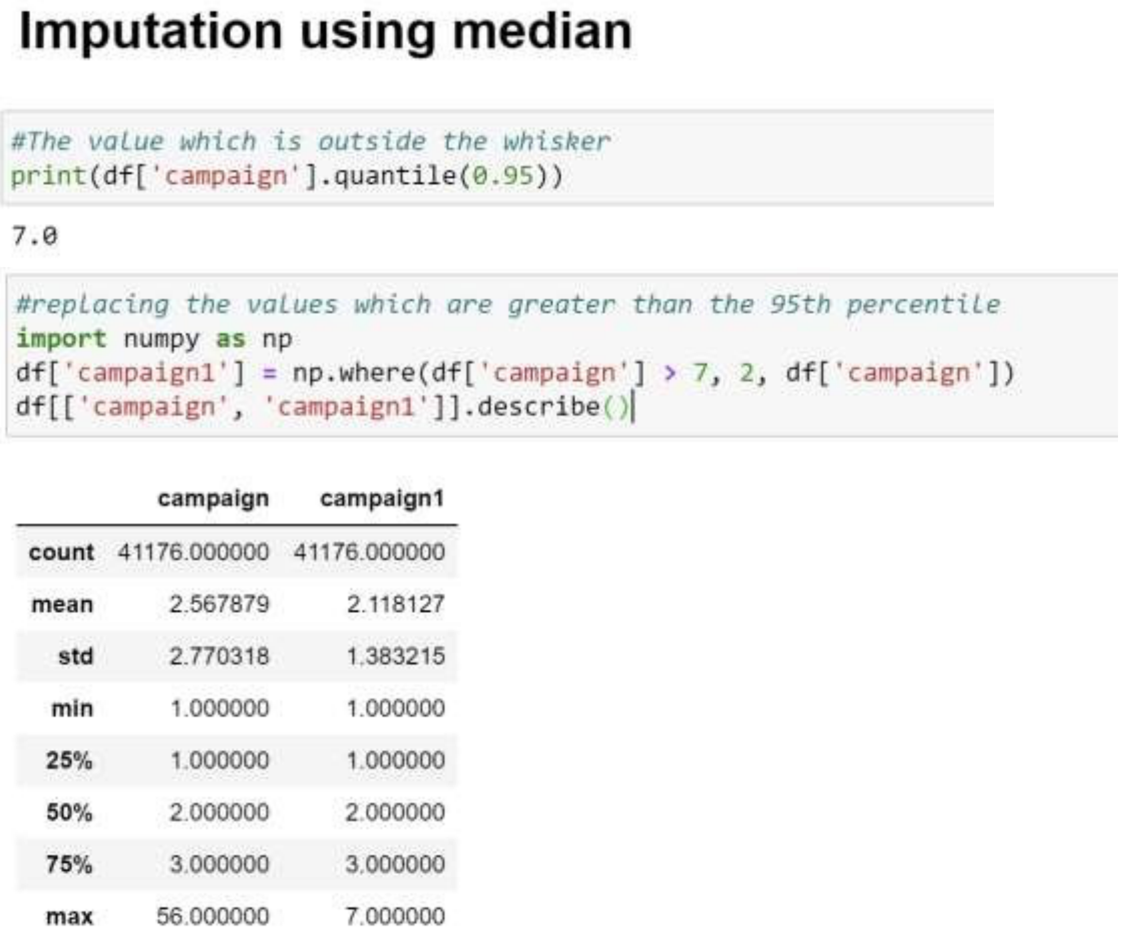
```
len(df[df['campaign'] > 20])  / len(df) * 100
```

```
0.3811789841701467
```

Therefore, I suggest to impute those rows with average of campaign values.

iv. **previous:** 'previous' holds the number of contacts performed before this campaign and for this client. Since the maximum value of 7 doesn't seem to be a noise, we I chose to ignore this outlier.

4. Treating outliers



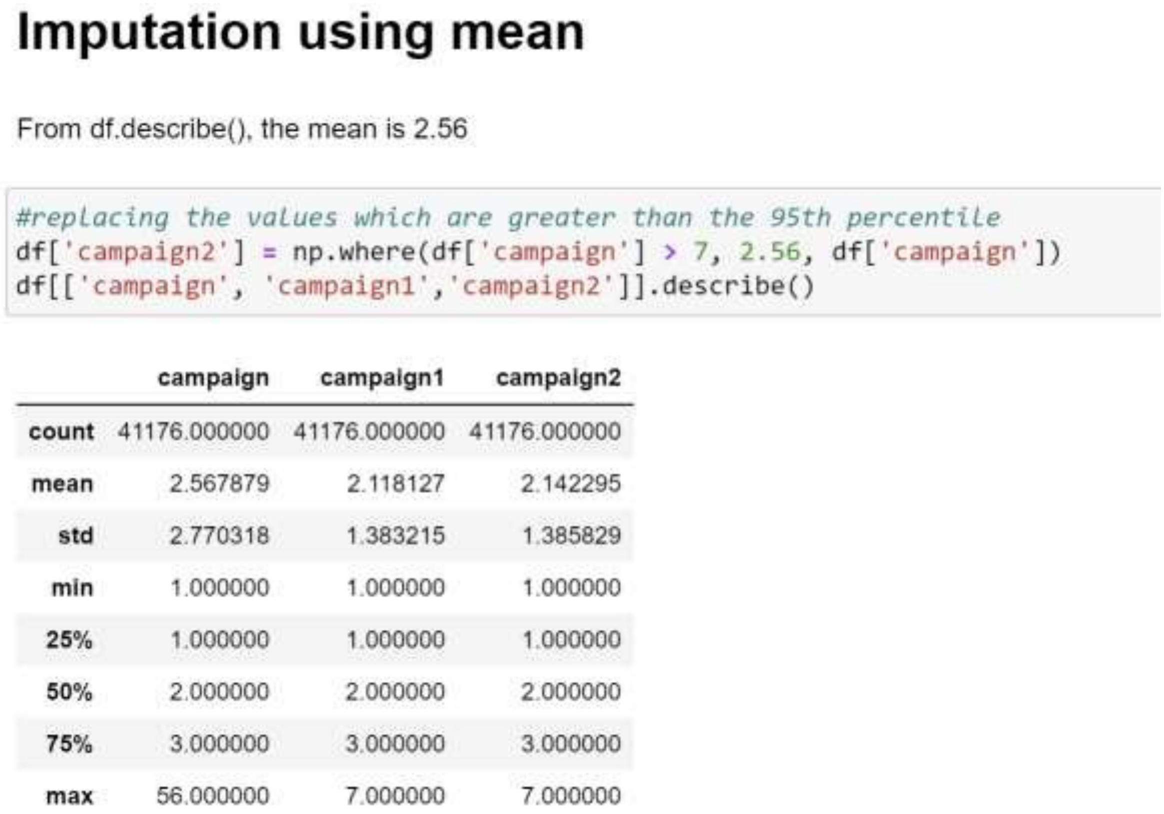
Imputation using median

```
#The value which is outside the whisker
print(df['campaign'].quantile(0.95))
```

```
7.0
```

```
#replacing the values which are greater than the 95th percentile
import numpy as np
df['campaign1'] = np.where(df['campaign'] > 7, 2, df['campaign'])
df[['campaign', 'campaign1']].describe()
```

	campaign	campaign1
count	41176.000000	41176.000000
mean	2.567879	2.118127
std	2.770318	1.383215
min	1.000000	1.000000
25%	1.000000	1.000000
50%	2.000000	2.000000
75%	3.000000	3.000000
max	56.000000	7.000000



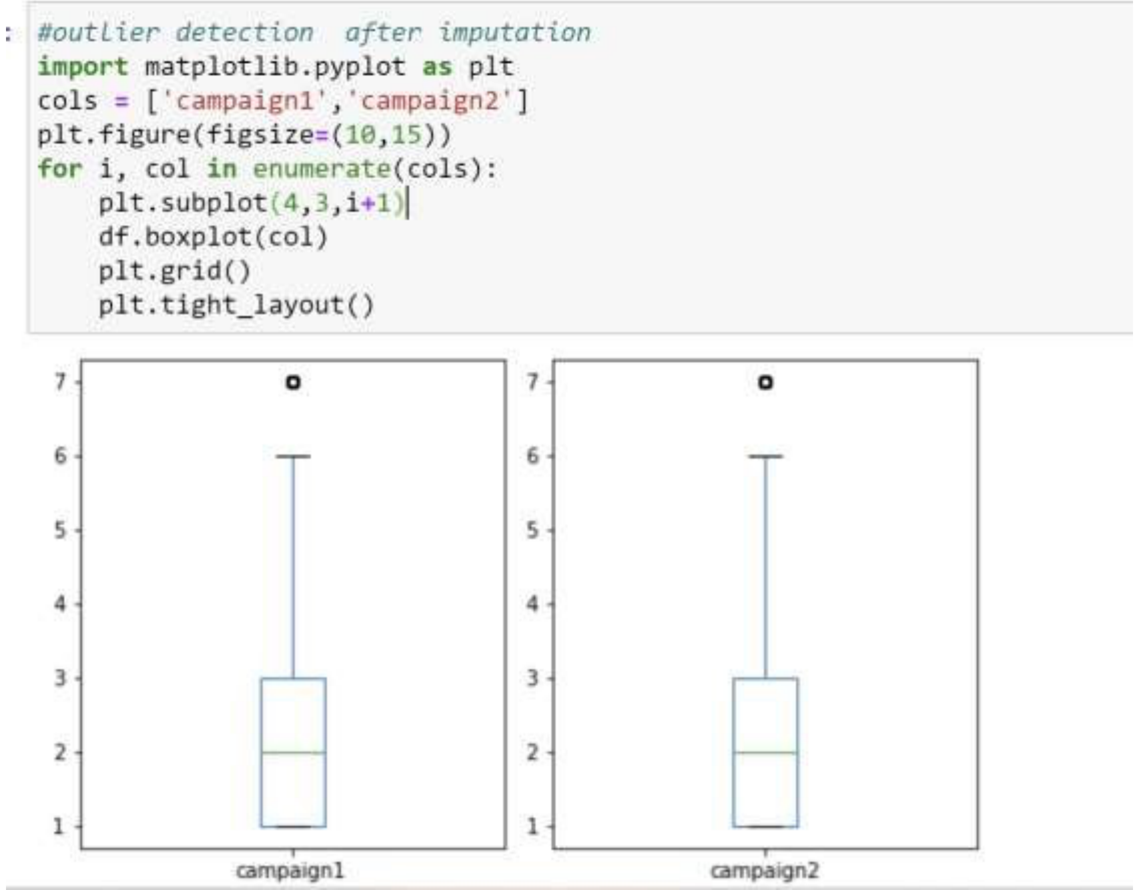
Imputation using mean

From df.describe(), the mean is 2.56

```
#replacing the values which are greater than the 95th percentile
df['campaign2'] = np.where(df['campaign'] > 7, 2.56, df['campaign'])
df[['campaign', 'campaign1','campaign2']].describe()
```

	campaign	campaign1	campaign2
count	41176.000000	41176.000000	41176.000000
mean	2.567879	2.118127	2.142295
std	2.770318	1.383215	1.385829
min	1.000000	1.000000	1.000000
25%	1.000000	1.000000	1.000000
50%	2.000000	2.000000	2.000000
75%	3.000000	3.000000	3.000000
max	56.000000	7.000000	7.000000

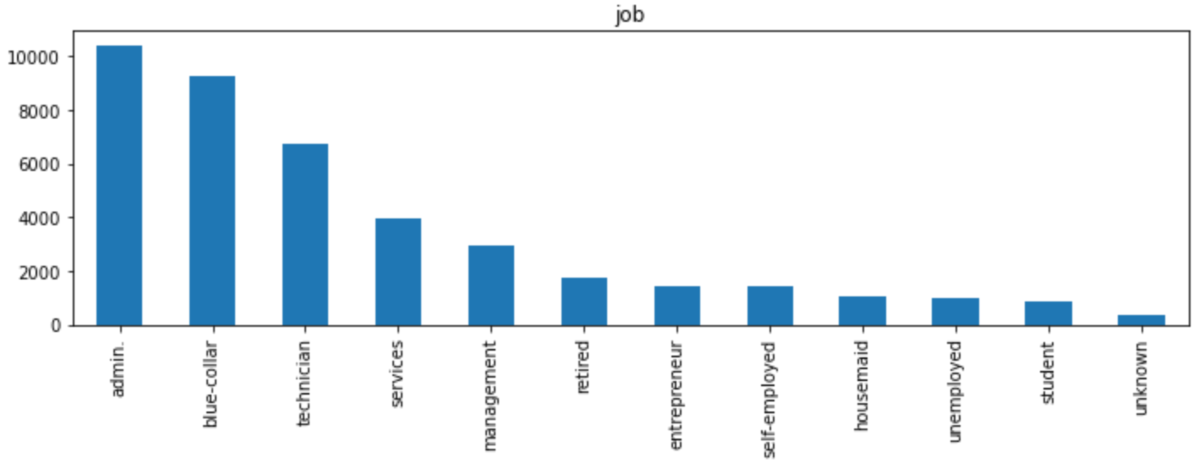
Statistics of the dataset after both median and mean imputation remains more or less the same.

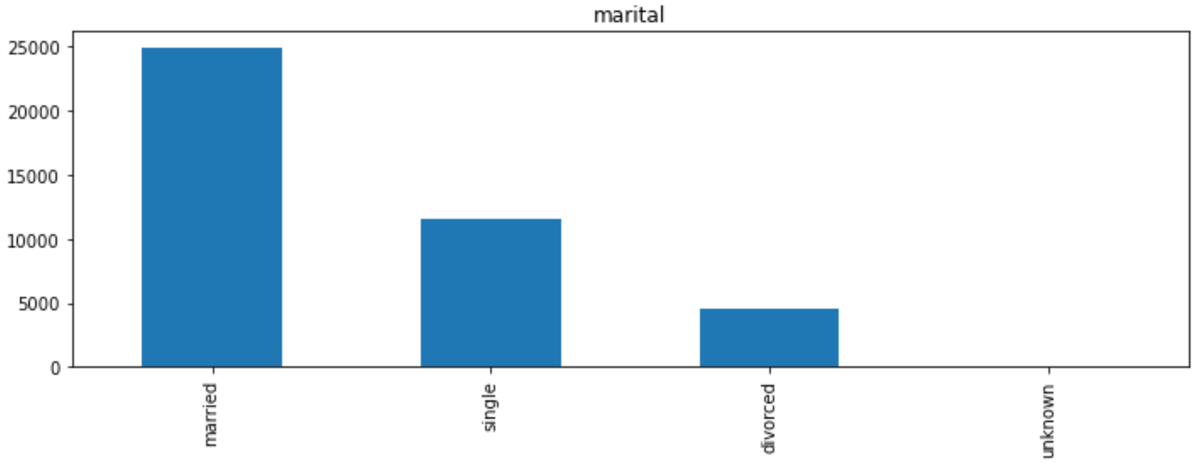


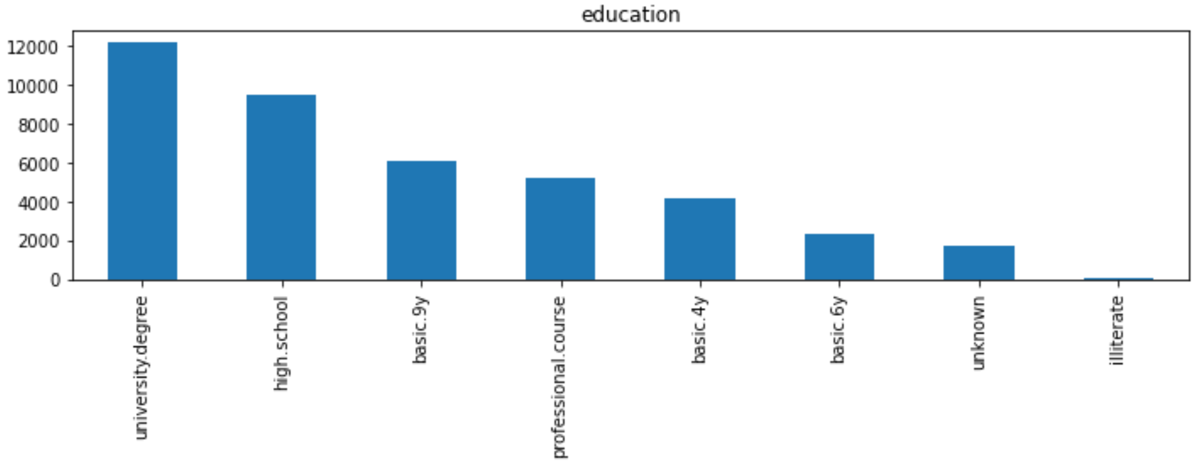
```
#outlier detection  after imputation
import matplotlib.pyplot as plt
cols = ['campaign1','campaign2']
plt.figure(figsize=(10,15))
for i, col in enumerate(cols):
    plt.subplot(4,3,i+1)
    df.boxplot(col)
    plt.grid()
    plt.tight_layout()
```

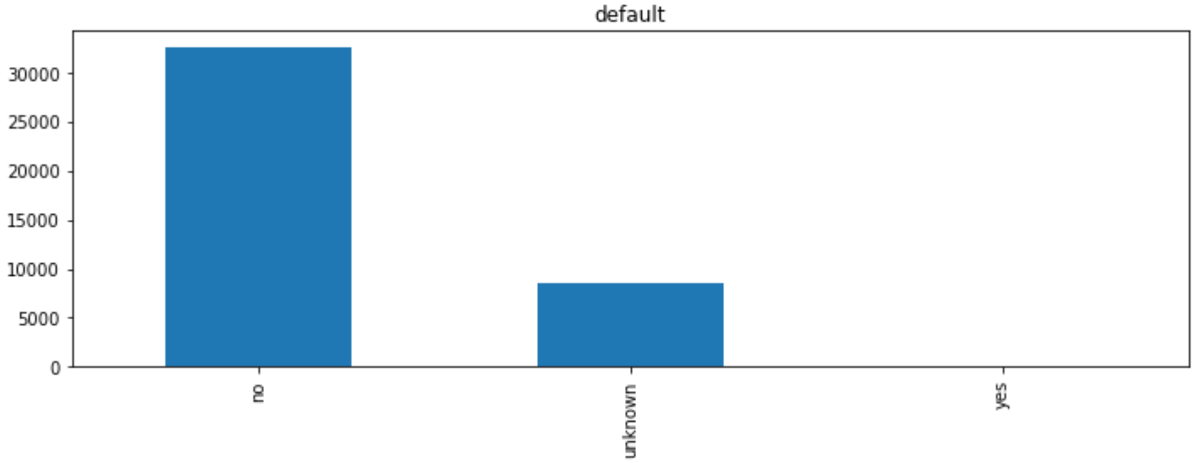
5. EDA and recommendation

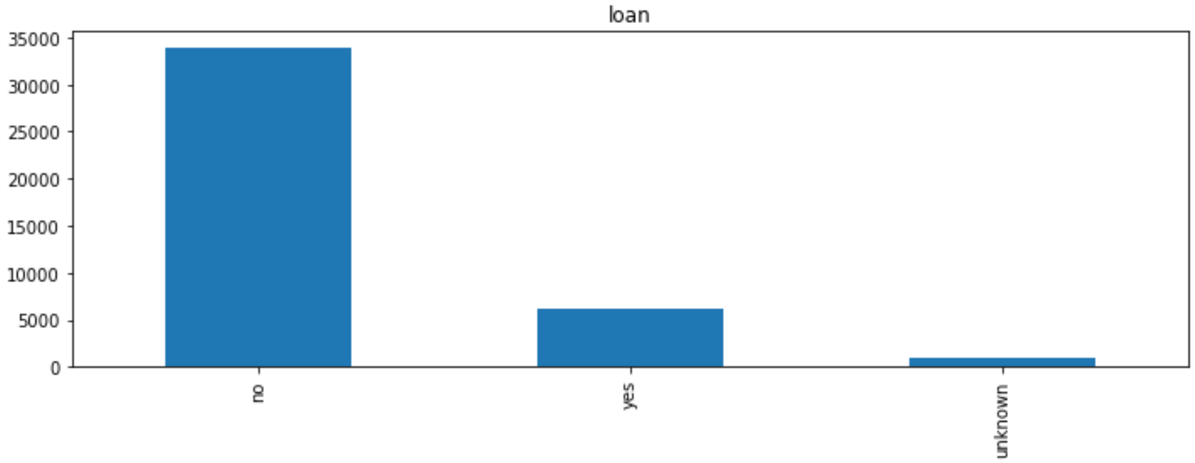
- Exploring categorical values

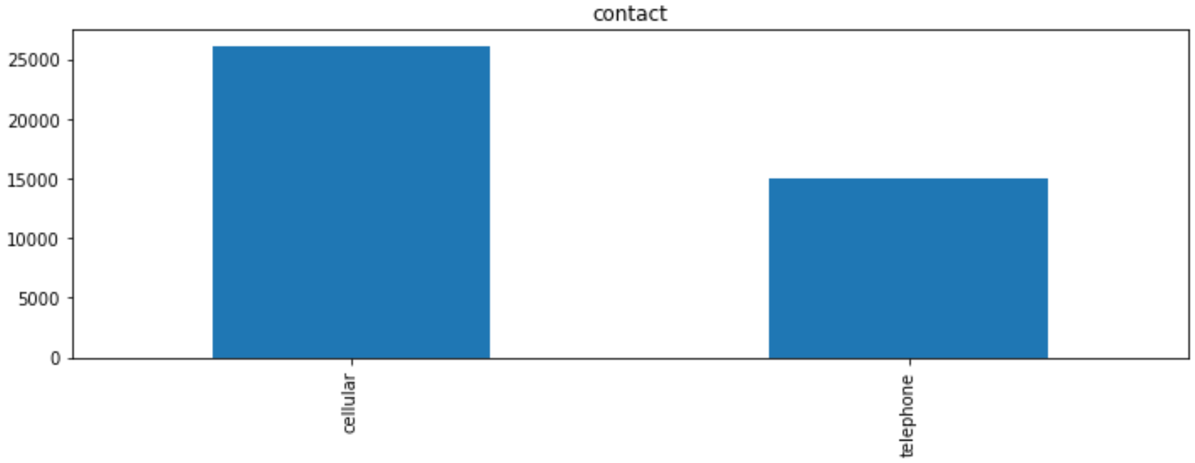


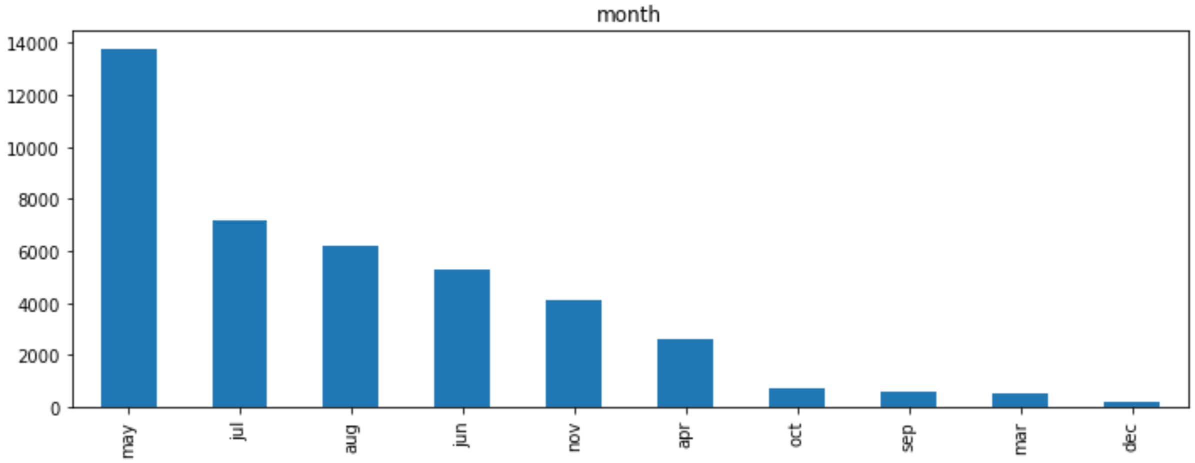

marital
25000
20000
15000
10000
5000
0
married
single
divorced
unknown

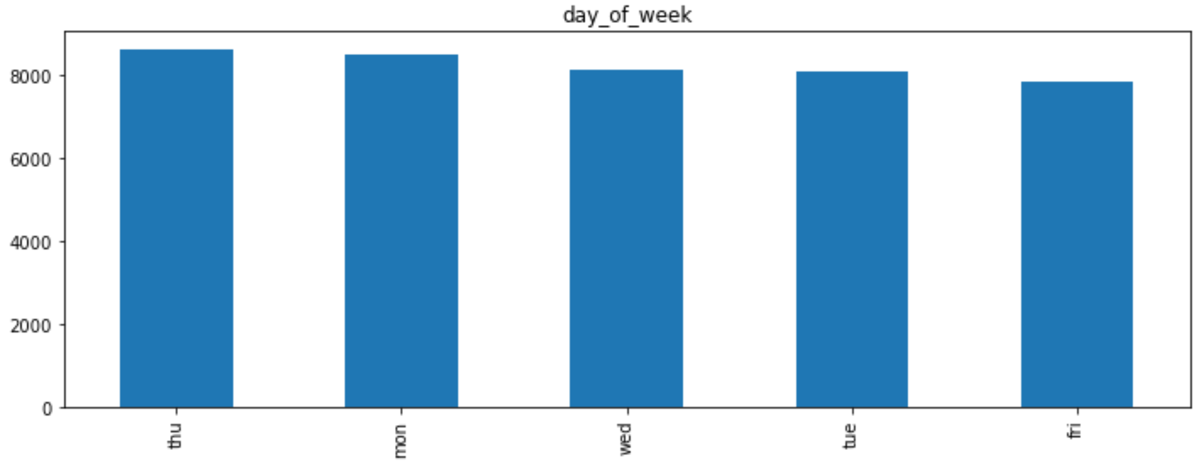

education
12000
10000
8000
6000
4000
2000
0
university.degree
high.school
basic.9y
professional.course
basic.4y
basic.6y
unknown
illiterate

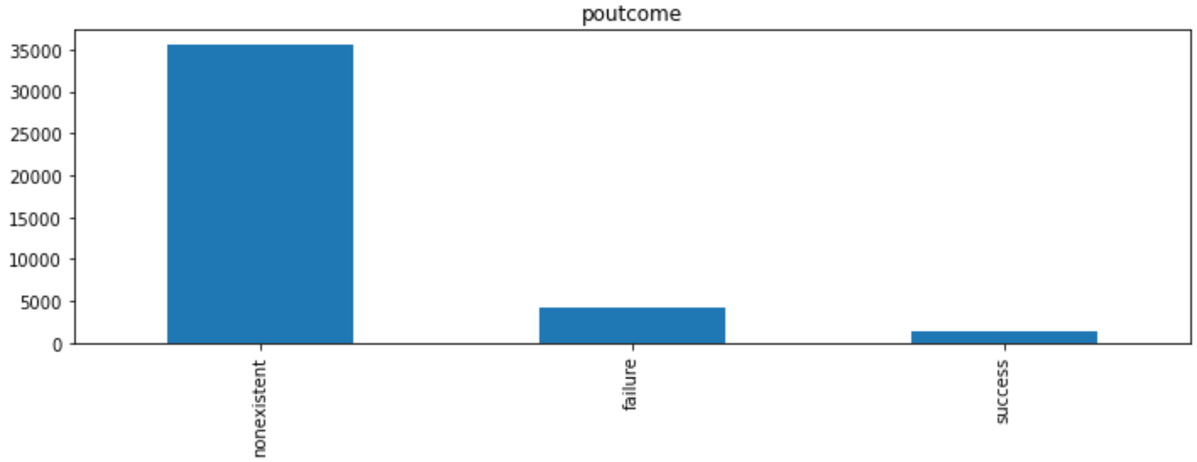

default
30000
25000
20000
15000
10000
5000
0
no
unknown
yes

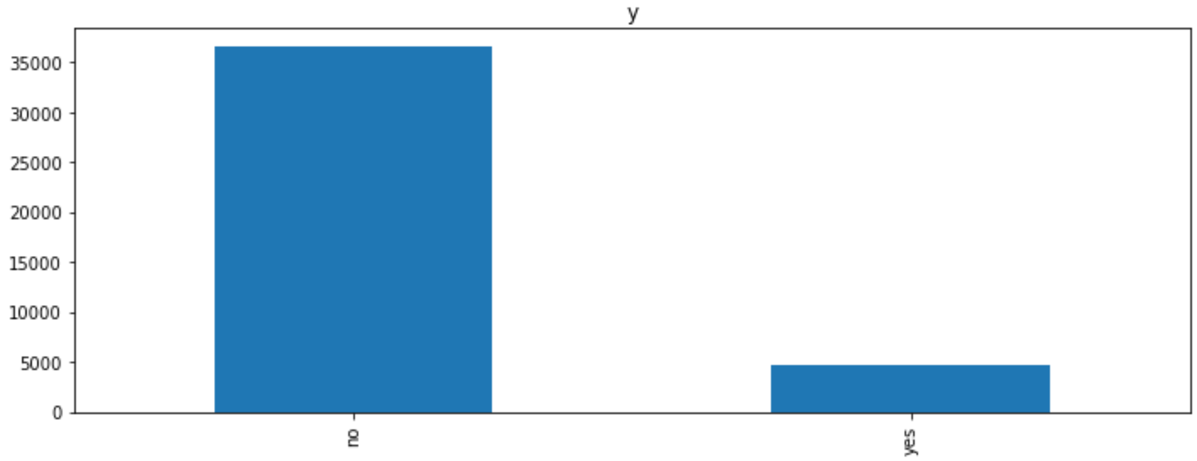

loan
35000
30000
25000
20000
15000
10000
5000
0
no
yes
unknown


contact
25000
20000
15000
10000
5000
0
cellular
telephone

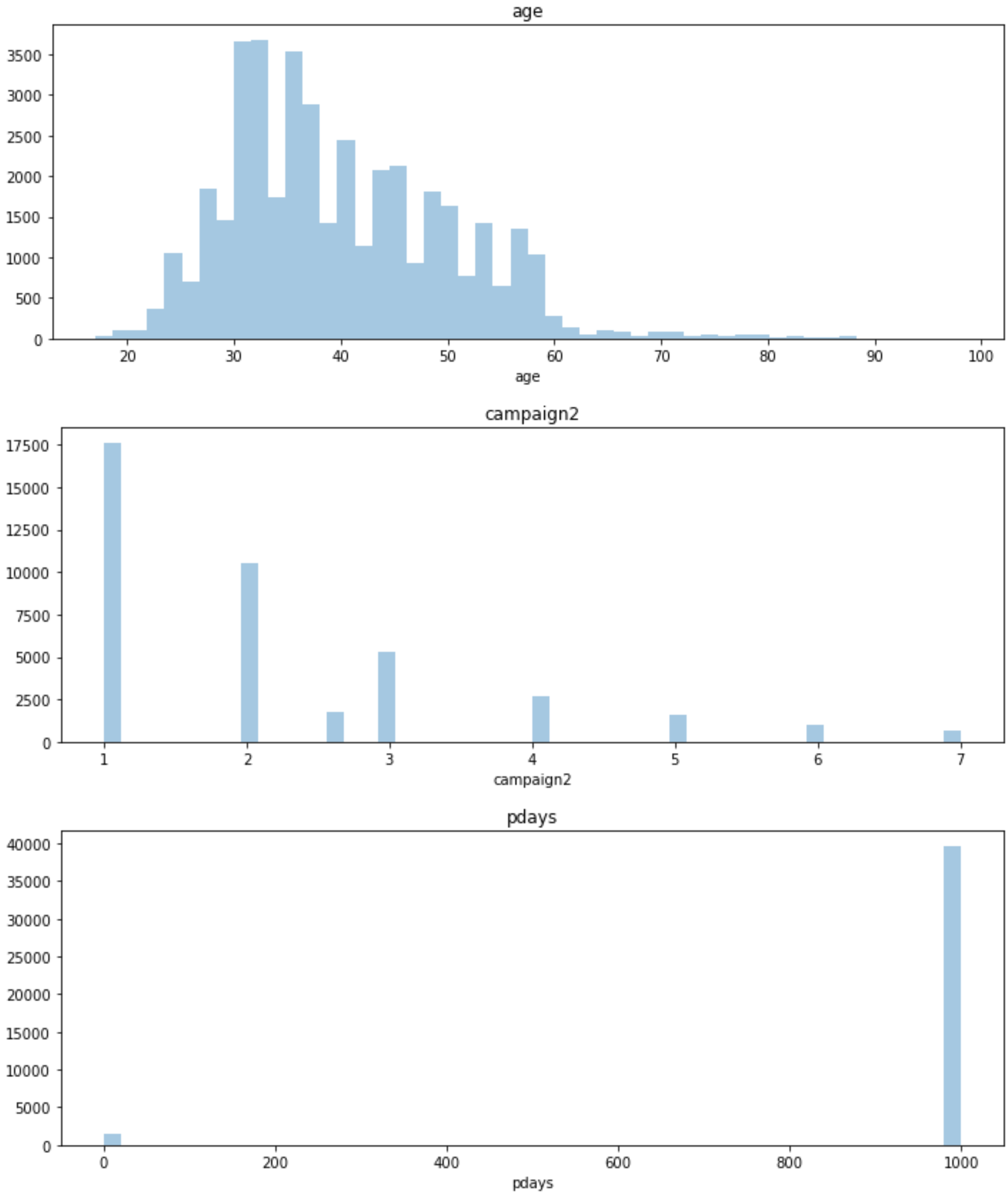

month
14000
12000
10000
8000
6000
4000
2000
0
may
jul
aug
jun
nov
apr
oct
sep
mar
dec

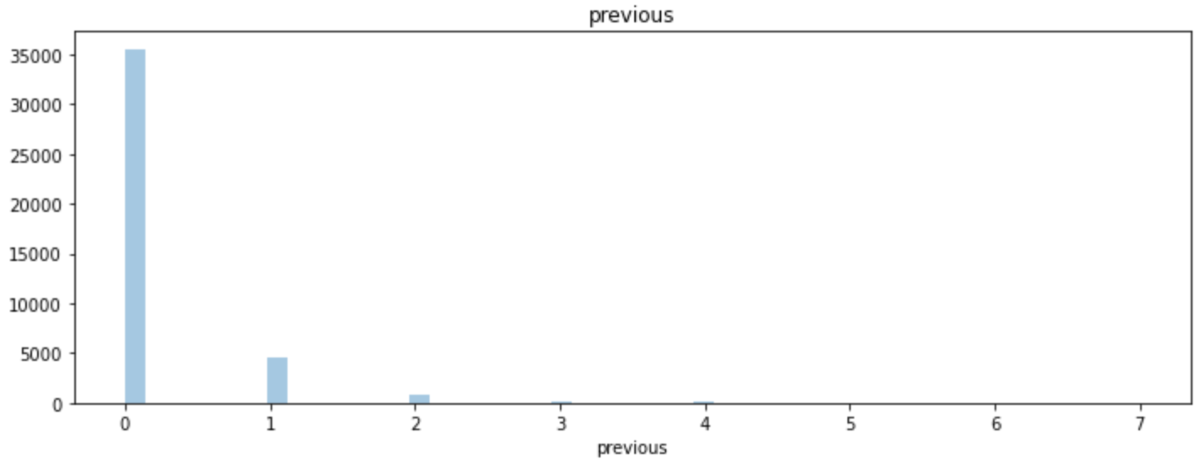


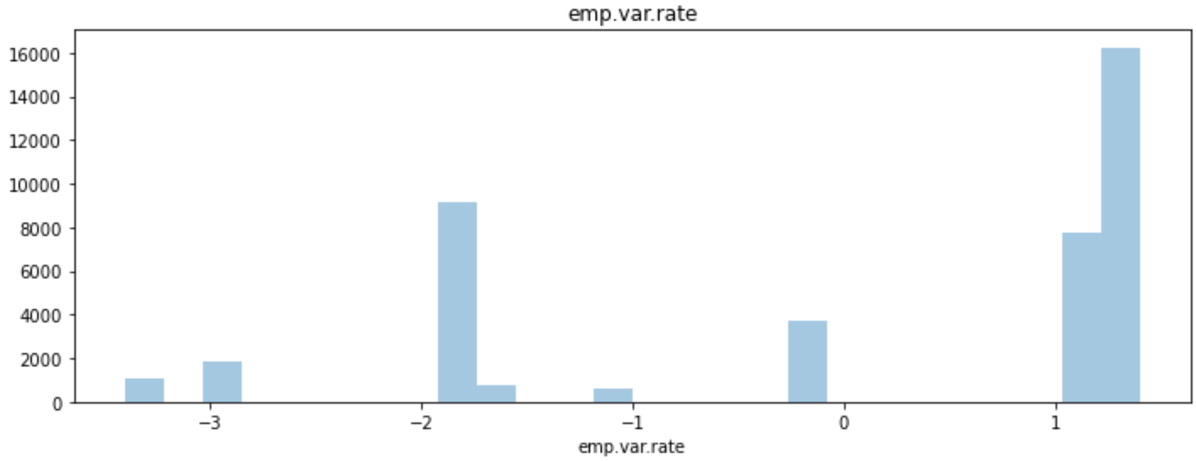


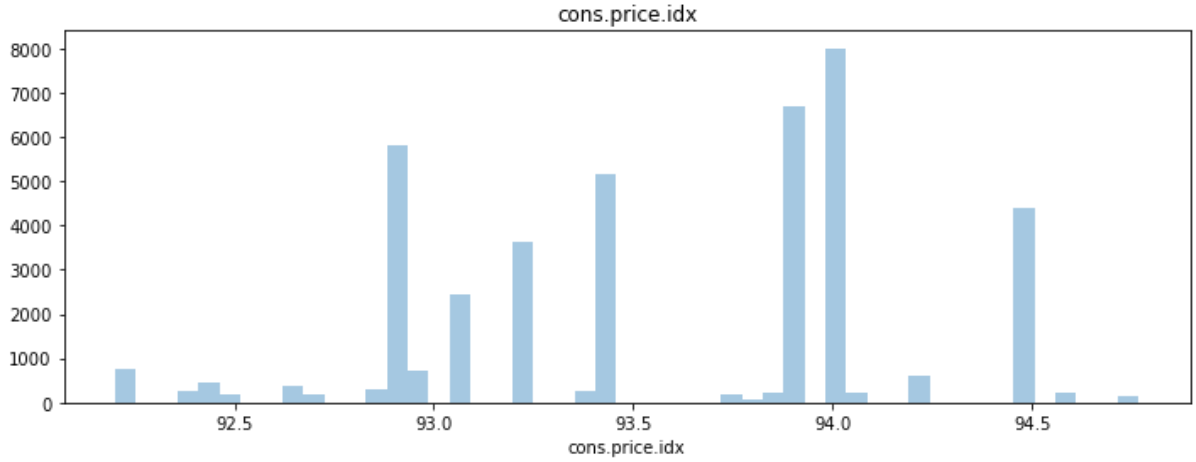


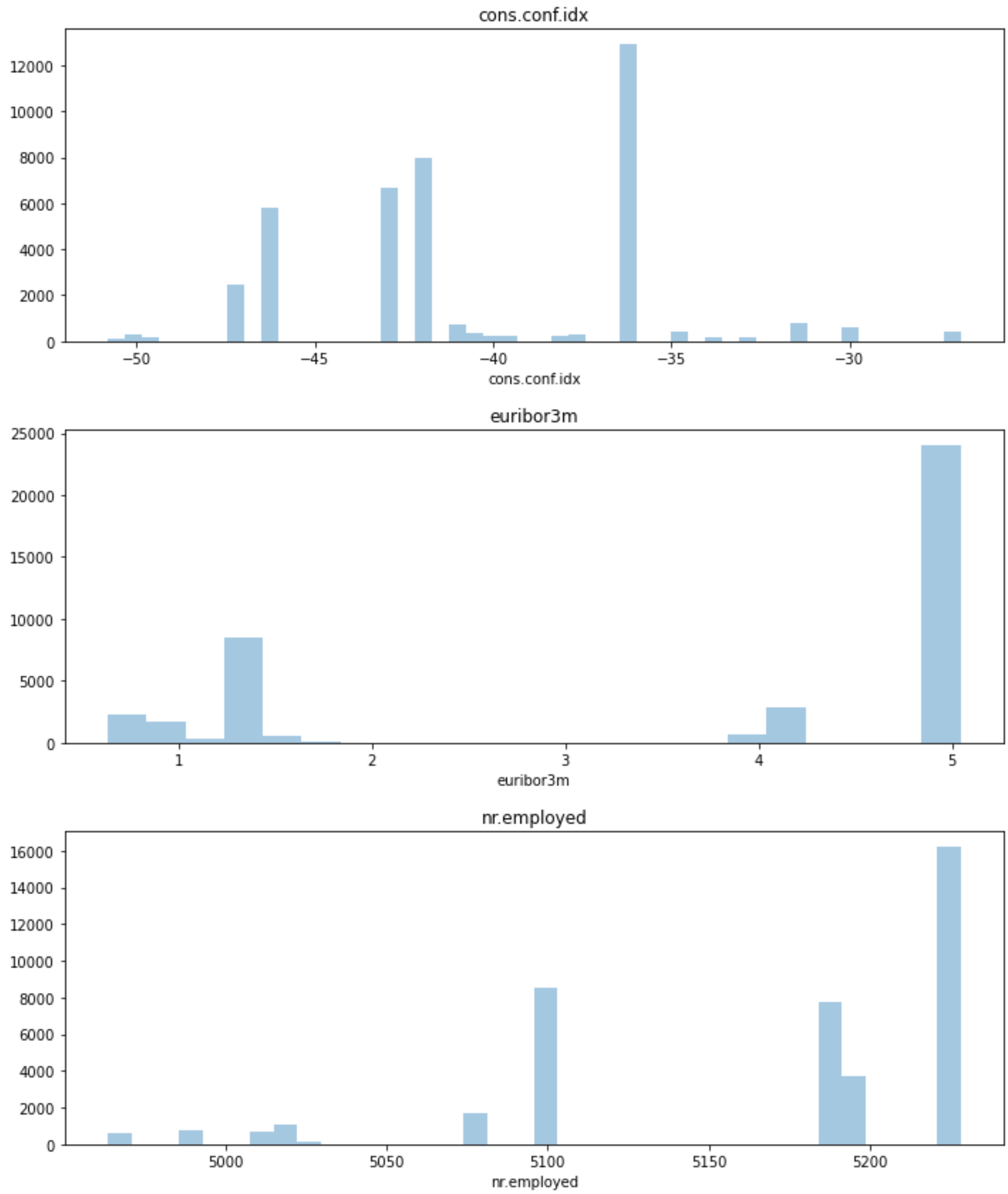
- Exploring numerical features


age
3500
3000
2500
2000
1500
1000
500
0
20
30
40
50
60
70
80
90
100
age
campaign2
17500
15000
12500
10000
7500
5000
2500
0
1
2
3
4
5
6
7
campaign2
pdays
40000
35000
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
pdays

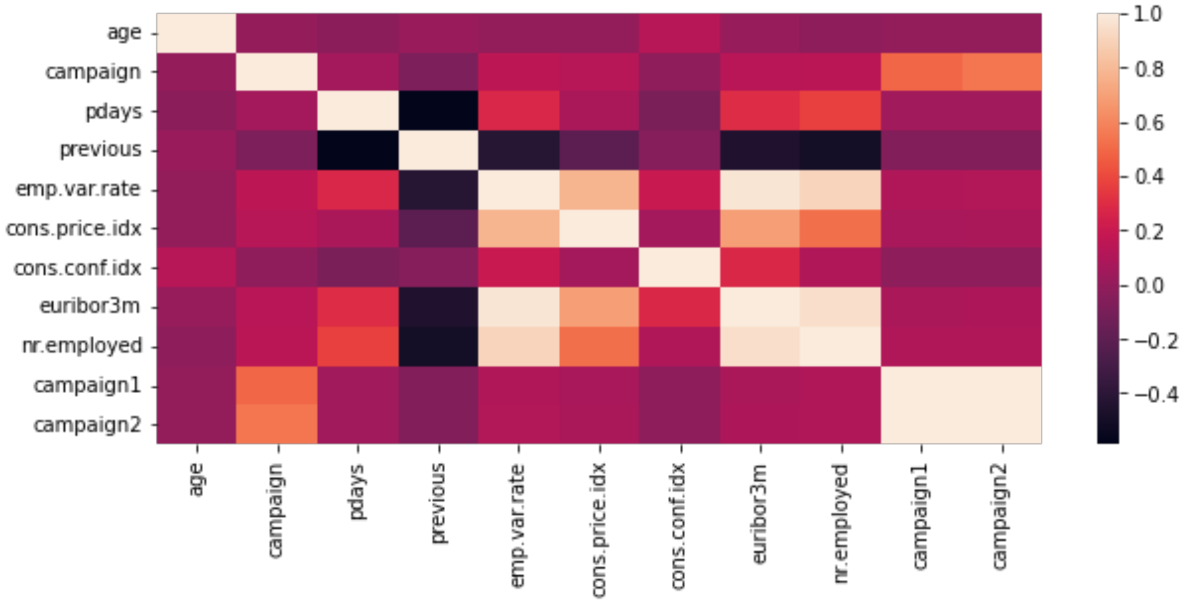

previous
35000
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
previous


emp.var.rate
16000
14000
12000
10000
8000
6000
4000
2000
0
−3
−2
−1
0
1
emp.var.rate

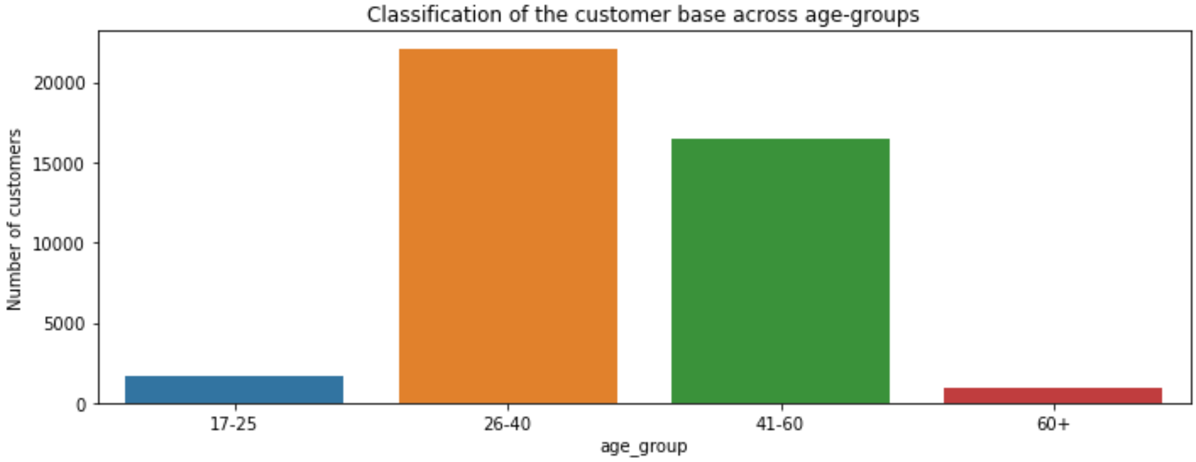

cons.price.idx
8000
7000
6000
5000
4000
3000
2000
1000
0
92.5
93.0
93.5
94.0
94.5
cons.price.idx



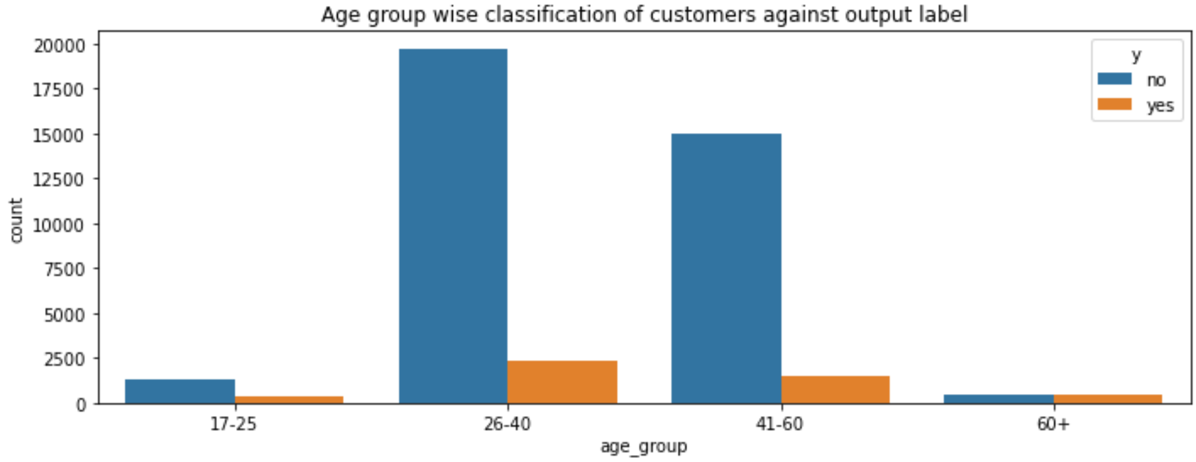
- Correlation matrix



- Classification of the customer base across age-groups

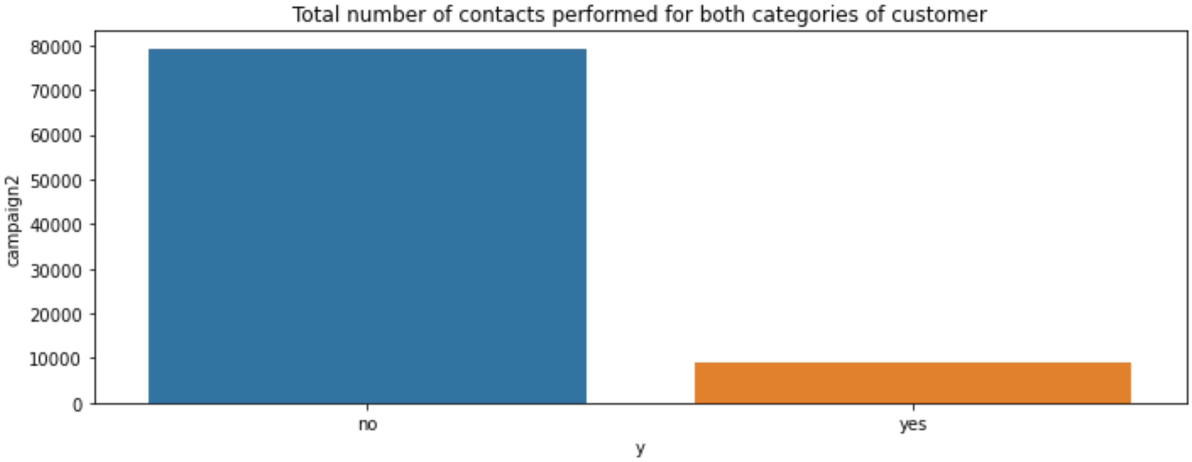


- Looking at relation between different age groups and the output label y



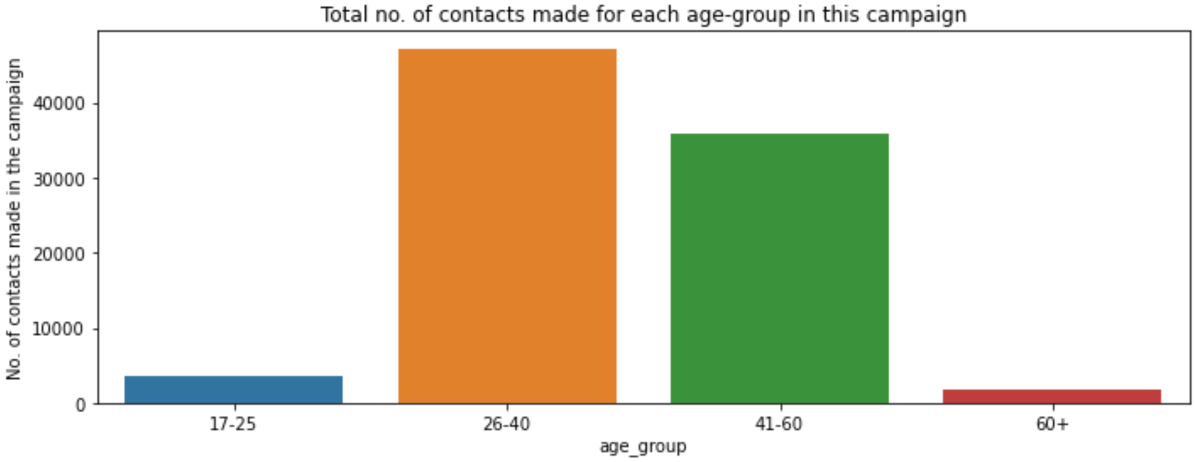
Observation: In the age-groups of 26-40 and 41-60 yrs, majority of the people are not subscribed to the term deposit plan

- Looking at relation between Number of contacts made to the customer (campaign) and the output label y



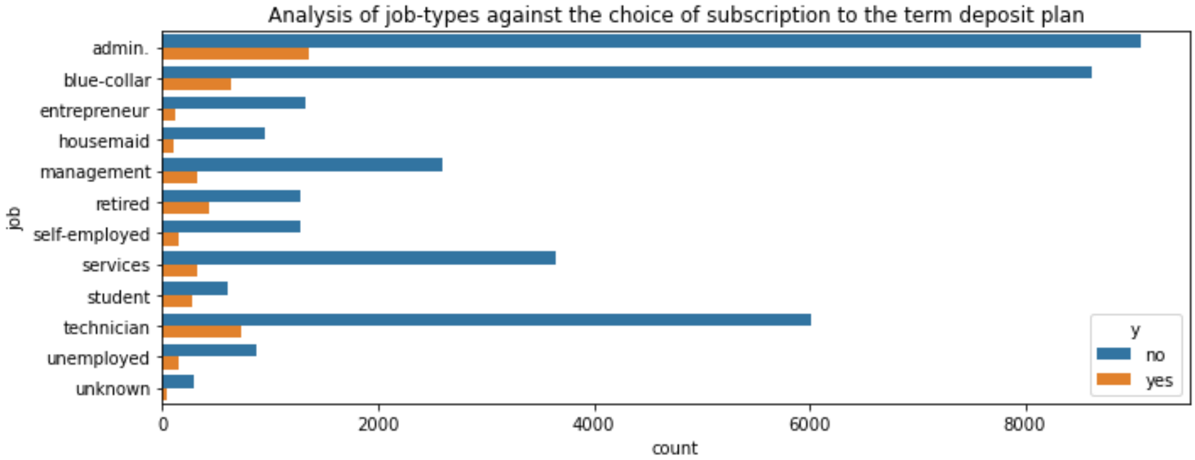
Observation: When a greater number of contacts is made to the customer, they haven't subscribed to the term deposit plan

- Looking at relation between 'age_group' and 'campaign' that is number of contacts performed for each age group



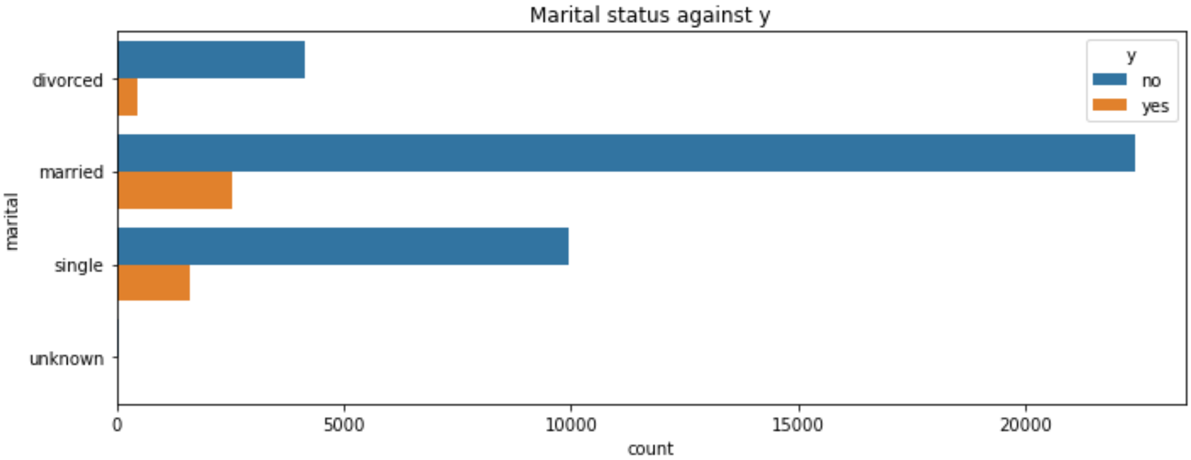
Observation: The 26-40 and 41-60 age-groups witness majority of the contacts made in this campaign. These two age-groups seem to the target groups for the bank.

- Looking at relation between job and the output label y



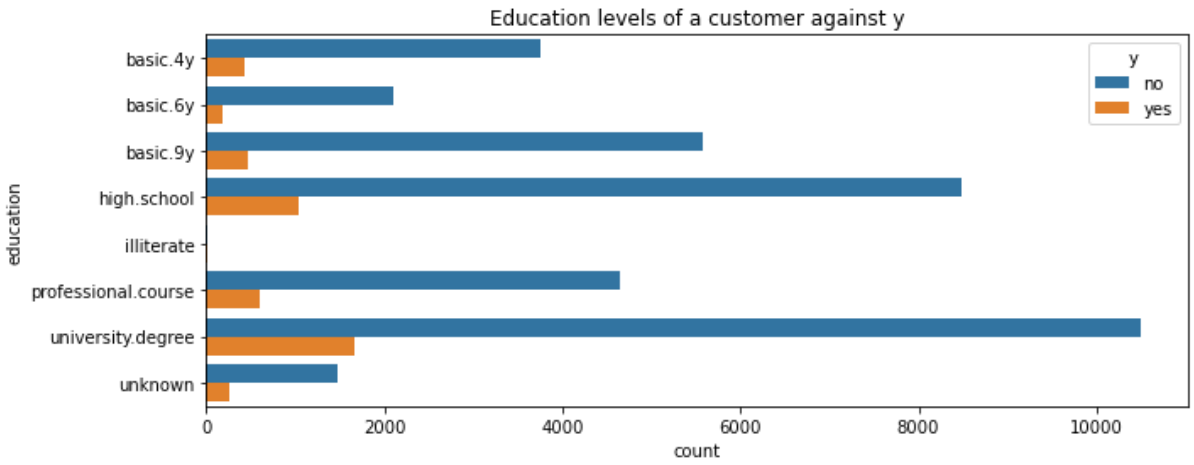
Observation: Looking at the jobs, 'admin', 'blue-collar' and 'technician' are the prominent jobs and most of the customers in these jobs have rejected the term deposit plan.

- Analysing marital status and the output label



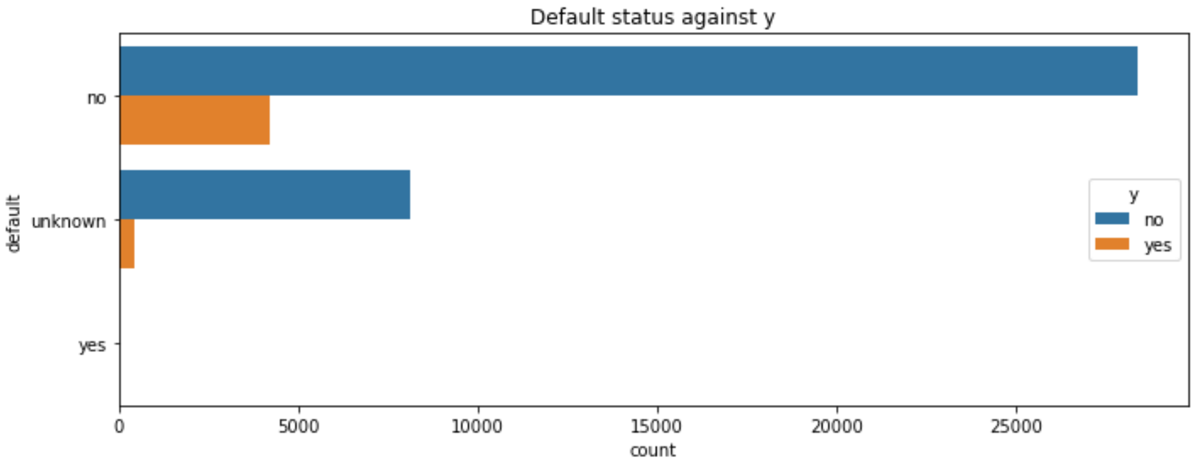
Observation: married and single customers are the majority of the customer base and comparatively married customers have taken the term deposit

- Analysing the different education levels of a customer against the choice of subscription

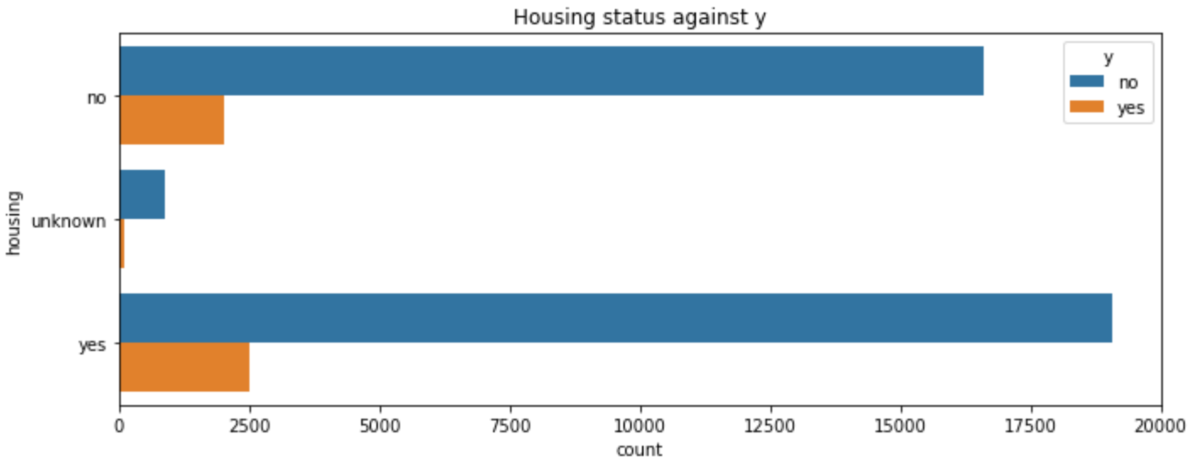


Observation: Customers with university degree have subscribed to the term deposit more

- Analysing the default status against the choice of subscription

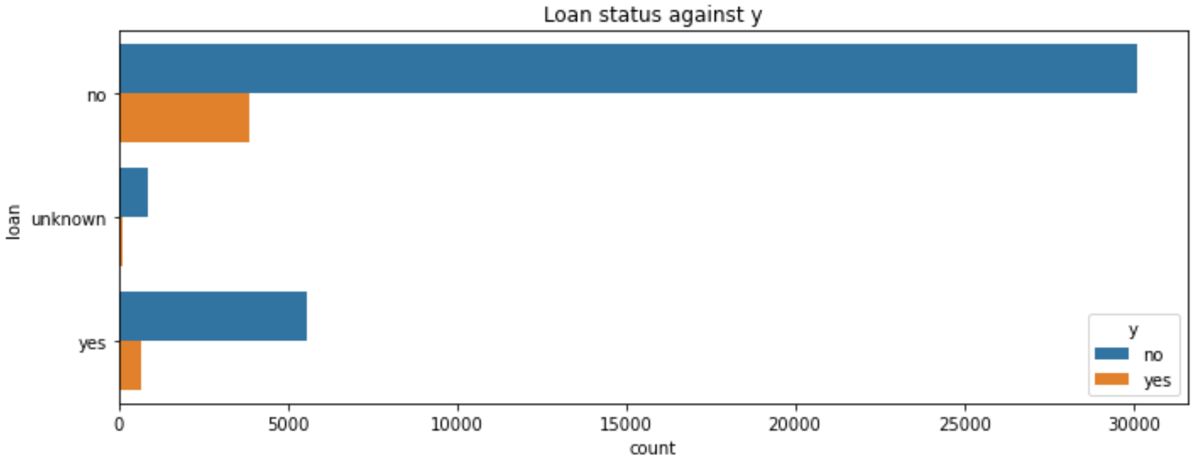


- Analysing housing status and y



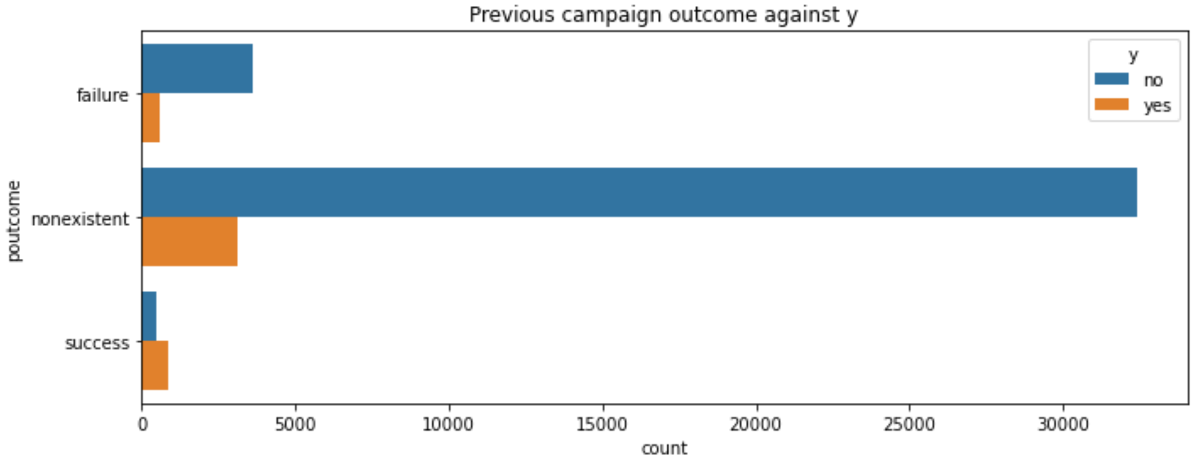
Observation: Number of customers who have subscribed to the term deposit is comparatively more for those with housing loan

- Analysing loan status and y



Observation: Number of customers who have subscribed to the term deposit is comparatively less for those with personal loan

- Analysing poutcome and y

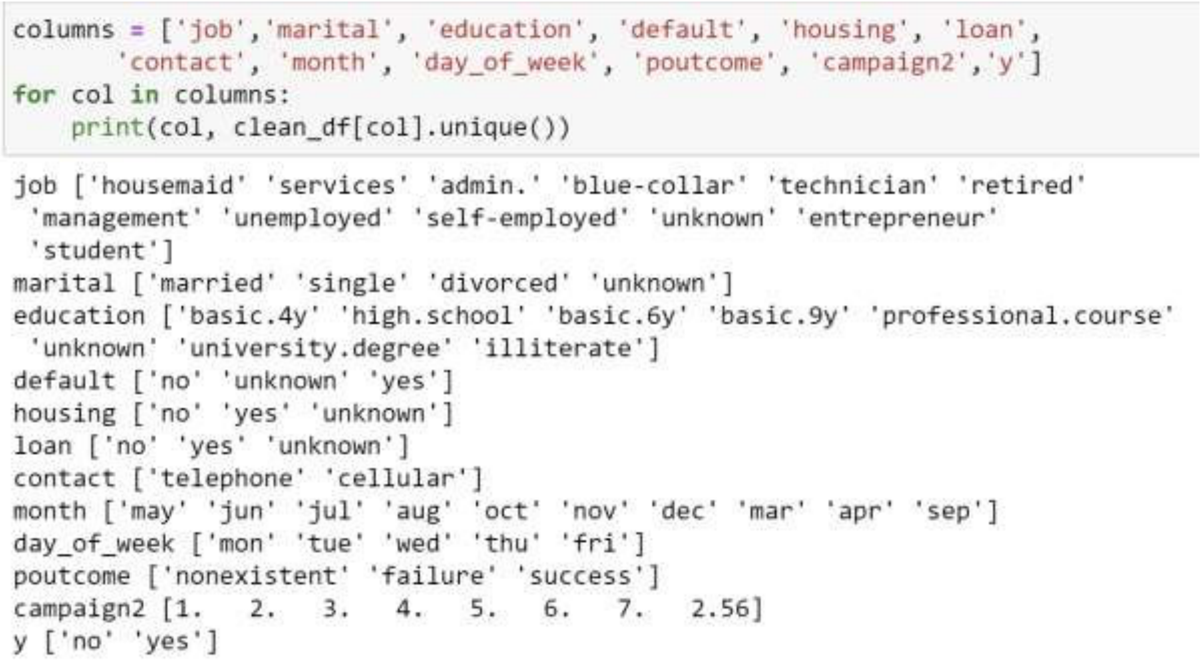


Observation: The success rate of previous marketing campaign has resulted in more number of people subscribing to the term deposit

6. Model Building

Since Machine learning models accepts input features as numbers, all the categorical features are converted to numeric and in the end feature scaling is performed on all features to achieve global minimum fast in gradient descent.

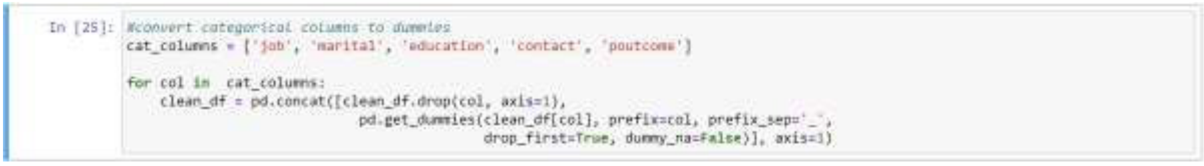
The categorical features include 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'campaign2' and 'y'.



```
columns = ['job','marital', 'education', 'default', 'housing', 'loan',
         'contact', 'month', 'day_of_week', 'poutcome', 'campaign2','y']
for col in columns:
    print(col, clean_df[col].unique())
```

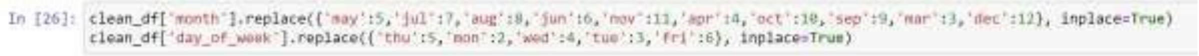
```
job ['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
marital ['married' 'single' 'divorced' 'unknown']
education ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
default ['no' 'unknown' 'yes']
housing ['no' 'yes' 'unknown']
loan ['no' 'yes' 'unknown']
contact ['telephone' 'cellular']
month ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']
day_of_week ['mon' 'tue' 'wed' 'thu' 'fri']
poutcome ['nonexistent' 'failure' 'success']
campaign2 [1.   2.   3.   4.   5.   6.   7.   2.56]
y ['no' 'yes']
```

To perform one hot encoding, dummy variables are created for the features job, marital, education, contact and poutcome.



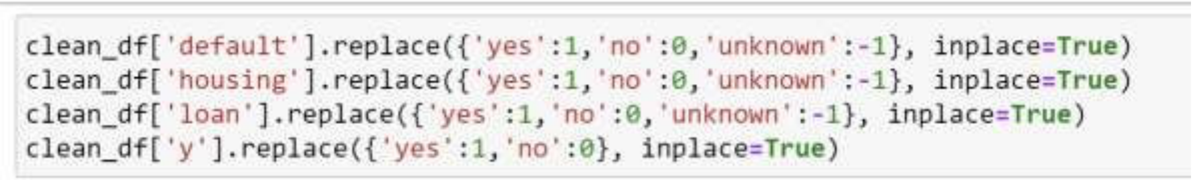
```
In [25]: #convert categorical columns to dummies
         cat_columns = ['job', 'marital', 'education', 'contact', 'poutcome']

         for col in  cat_columns:
             clean_df = pd.concat([clean_df.drop(col, axis=1),
                                  pd.get_dummies(clean_df[col], prefix=col, prefix_sep='_',
                                                 drop_first=True, dummy_na=False)], axis=1)
```

Encoding month and day of the week using the equivalent numbers.



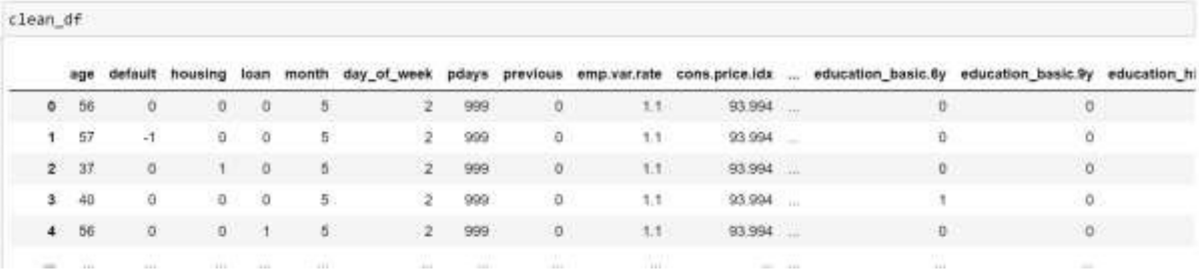
```
In [26]: clean_df['month'].replace({'may':5,'jul':7,'aug':8,'jun':6,'nov':11,'apr':4,'oct':10,'sep':9,'mar':3,'dec':12}, inplace=True)
         clean_df['day_of_week'].replace({'thu':5,'mon':2,'wed':4,'tue':3,'fri':6}, inplace=True)
```

Encoding default, housing, loan and y with 1, 0 and -1 for yes, no and unknown.



```
clean_df['default'].replace({'yes':1,'no':0,'unknown':-1}, inplace=True)
clean_df['housing'].replace({'yes':1,'no':0,'unknown':-1}, inplace=True)
clean_df['loan'].replace({'yes':1,'no':0,'unknown':-1}, inplace=True)
clean_df['y'].replace({'yes':1,'no':0}, inplace=True)
```

Now the features are all encoded into numbers.

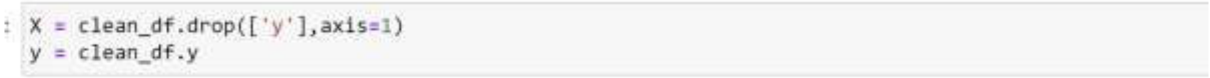


```
clean_df
```

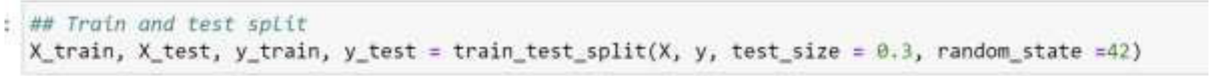
	age	default	housing	loan	month	day_of_week	pdays	previous	emp.var.rate	cons.price.idx	...	education_basic.6y	education_basic.9y	education_hi
0	56	0	0	0	5	2	999	0	1.1	93.994	...	0	0	
1	57	-1	0	0	5	2	999	0	1.1	93.994	...	0	0	
2	37	0	1	0	5	2	999	0	1.1	93.994	...	0	0	
3	40	0	0	0	5	2	999	0	1.1	93.994	...	1	0	
4	56	0	0	1	5	2	999	0	1.1	93.994	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	

Training using Machine Learning classifiers

The dataset is split into Train and Test

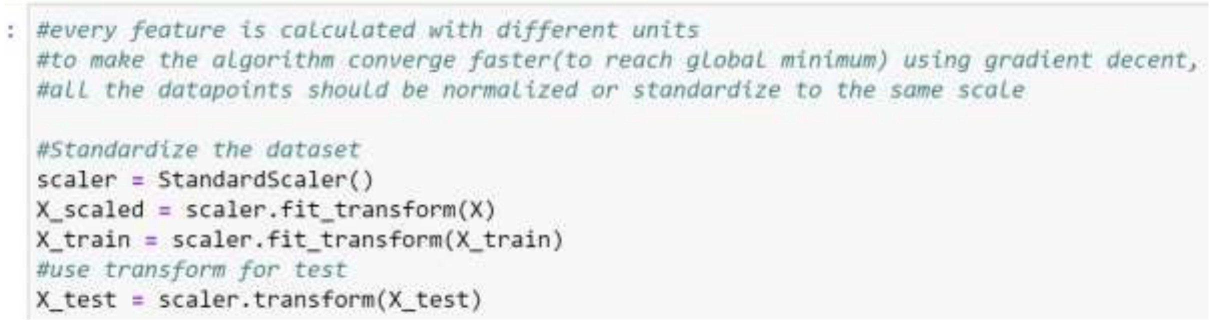


```
X = clean_df.drop(['y'],axis=1)
y = clean_df.y
```



```
## Train and test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state =42)
```

To perform feature scaling, StandardScaler() is used.

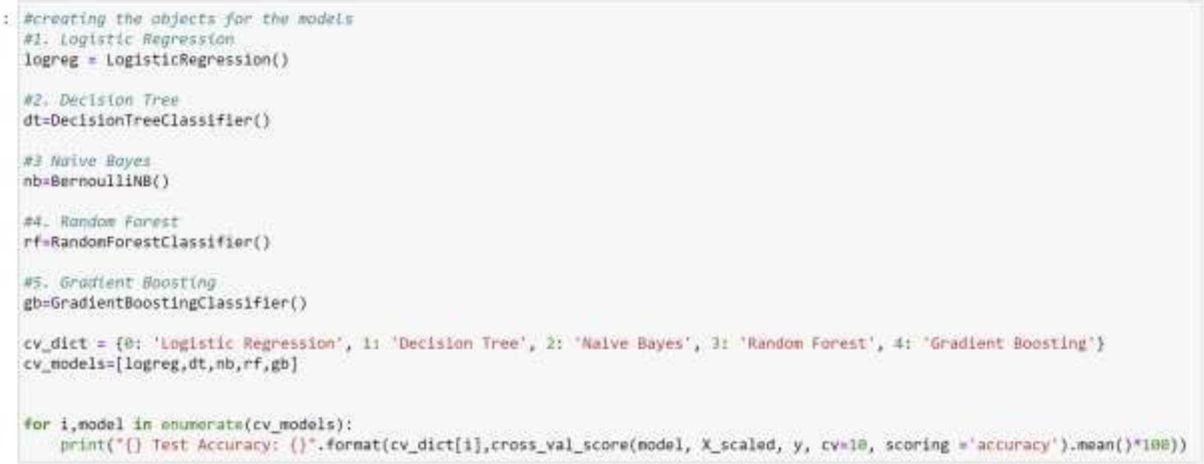


```
#every feature is calculated with different units
#to make the algorithm converge faster(to reach global minimum) using gradient decent,
#all the datapoints should be normalized or standardize to the same scale

#Standardize the dataset
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train = scaler.fit_transform(X_train)
#use transform for test
X_test = scaler.transform(X_test)
```

I have chosen the following models for testing.

1. Logistic Regression (Linear)
2. Decision Tree (Linear)

For evaluating the model, cross validation testing is used with the number of folds as 10 and accuracy as the metric.



```
#creating the objects for the models
#1. Logistic Regression
logreg = LogisticRegression()

#2. Decision Tree
dt=DecisionTreeClassifier()

#3 Naive Bayes
nb=BernoulliNB()

#4. Random Forest
rf=RandomForestClassifier()

#5. Gradient Boosting
gb=GradientBoostingClassifier()

cv_dict = {0: 'Logistic Regression', 1: 'Decision Tree', 2: 'Naive Bayes', 3: 'Random Forest', 4: 'Gradient Boosting'}
cv_models=[logreg,dt,nb,rf,gb]


for i,model in enumerate(cv_models):
    print("{} Test Accuracy: {}".format(cv_dict[i],cross_val_score(model, X_scaled, y, cv=10, scoring ='accuracy').mean()*100))
```

From all the above Models, **Logistic Regression** performed the best with an accuracy of **84%.**

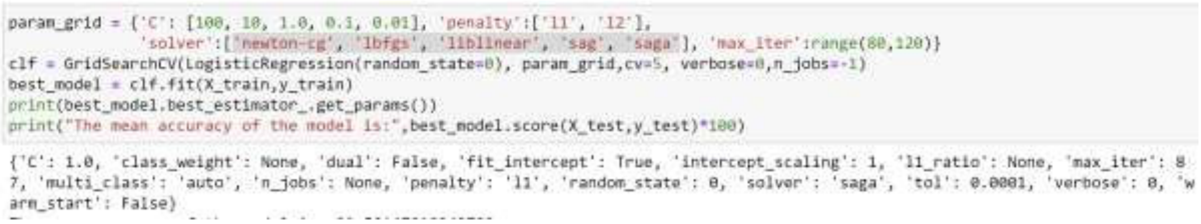
Model	**Accuracy**
Logistic Regression	**84%**

Decision Tree	28%

Hyperparameter Tuning

Since Logistic Regression gave better performance, hyper parameter tuning was performed on it to identify best features.

Parameter	Values
C	100, 10, 1.0, 0.1, 0.01
penalty	'l1', 'l2'
Solver	'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'
max_iter	Between 80 and 120



```
param_grid = {'C': [100, 10, 1.0, 0.1, 0.01], 'penalty':['l1', 'l2'],
              'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'], 'max_iter':range(80,120)}
clf = GridSearchCV(LogisticRegression(random_state=0), param_grid,cv=5, verbose=0,n_jobs=-1)
best_model = clf.fit(X_train,y_train)
print(best_model.best_estimator_.get_params())
print("The mean accuracy of the model is:",best_model.score(X_test,y_test)*100)
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 8
7, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l1', 'random_state': 0, 'solver': 'saga', 'tol': 0.0001, 'verbose': 0, 'w
arm_start': False}
```

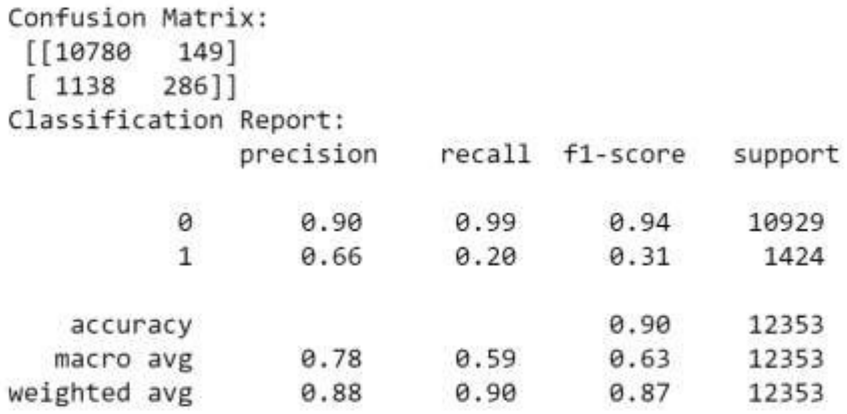
The best parameters values were identified as

Parameter	Values
C	1.0
penalty	'l1'
Solver	'saga'
max_iter	87

With Hyperparameter tuning the accuracy increased from **84% to 89.58%.**

Evaluation metrices

The classification report and the confusion matrix are reported as



```
Confusion Matrix:
 [[10780   149]
 [ 1138   286]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94     10929
           1       0.66      0.20      0.31      1424

    accuracy                           0.90     12353
   macro avg       0.78      0.59      0.63     12353
weighted avg       0.88      0.90      0.87     12353
```

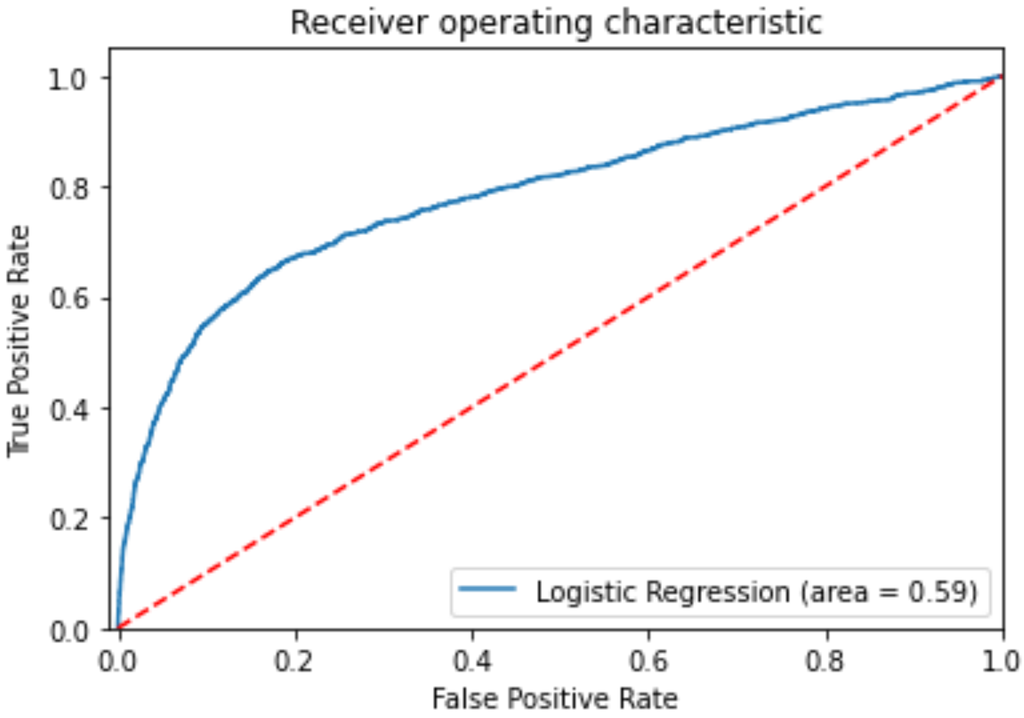
Observations:

1. Confusion matrix results tell us that we have 10780 + 149 Correct predictions and 1138+286 incorrect

2. Classification report shows precision as 90% which is the ability of a classification model to identify only the relevant data points, that is in this case people who would be subscribing to the term deposit is correctly classified.

Evaluating using AUC-ROC curve

The model is also evaluated using AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve which is an evaluation metric for binary classification problems. An ROC curve is a graph showing the performance of a classifier. ROC is a probability curve plotted with True Positive Rate (also called Recall or Sensitivity) on the y-axis against False Positive Rate (also called as Precision) on the x-axis. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



7. Github Repo link https://github.com/Suyog005/Data-Glacier