Marketing Analytics: Personalizations, Campaign, Pricing and Promotions

Rohit Singh
60004170094
TE B Comps

Introduction

The dataset chosen by me is called 'bank.csv'. It is a dataset that contains details about the age, job, education, duration of campaign and outcome of the campaign to name a few. The aim is to look after the marketing campaign of this bank. We have to predict whether the customer will take a term deposit in the bank or not. Proper analysis is made to see the different independent parameters and how the dependent parameter i.e term deposit depends on them.

Marketing campaigns are characterized by focusing on the customer needs and their overall satisfaction. There are several factors that we need to consider while designing any marketing campaign. Some crucial parameters are:

- Population: The target audience. Answering the questions of both how many and who. We should know our potential clients and design our campaign accordingly.
- Pricing: This deals with answering what is the best price to offer to potential clients? In our case i.e bank's marketing campaign that is not necessary since the main interest for the bank is for potential clients to open deposit accounts in order to make the operative activities of the bank to keep on running.
- Promotional Strategy: This is the way the strategy is going to be implemented and how are potential clients going to be addressed. This should be the last part of the marketing campaign analysis since there has to be an in depth analysis of previous campaigns (If possible) in order to learn from previous mistakes and to determine how to make the marketing campaign much more effective.

Now let’s see some information about the attributes:

1 - age: (numeric)

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

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

4 - education: (categorical: primary, secondary, tertiary and unknown)

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

6 - housing: has a housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown')

8 - balance: Balance of the individual.

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

11 - duration: last contact duration, in seconds (numeric).

Aiii. other attributes:

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

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14 - previous: number of contacts performed before this campaign and for this client (numeric)
15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

Output variable (desired target):
21 - y - has the client subscribed to a term deposit? (binary: 'yes','no')

We now have a look at our dataset

Code:

```
import pandas as pd
dataset = pd.read_csv('bank.csv')
dataset.head()
```

```
In [3]: dataset.head()
Out[3]:
   age         job  marital  education  ... pdays  previous poutcome deposit
0   59      admin.  married  secondary  ...    -1         0  unknown     yes
1   56      admin.  married  secondary  ...    -1         0  unknown     yes
2   41  technician  married  secondary  ...    -1         0  unknown     yes
3   55    services  married  secondary  ...    -1         0  unknown     yes
4   54      admin.  married   tertiary  ...    -1         0  unknown     yes
```

dataset.describe()

```
In [4]: dataset.describe()
Out[4]:
                age       balance  ...         pdays      previous
count  11162.000000  11162.000000  ...  11162.000000  11162.000000
mean      41.231948   1528.538524  ...     51.330407      0.832557
std       11.913369   3225.413326  ...    108.758282      2.292007
min       18.000000  -6847.000000  ...     -1.000000      0.000000
25%       32.000000    122.000000  ...     -1.000000      0.000000
50%       39.000000    550.000000  ...     -1.000000      0.000000
75%       49.000000   1708.000000  ...     20.750000      1.000000
max       95.000000  81204.000000  ...    854.000000     58.000000

[8 rows x 7 columns]
```

We now check for any missing values.

```
In [5]: dataset.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
age          11162 non-null int64
job          11162 non-null object
marital      11162 non-null object
education    11162 non-null object
default      11162 non-null object
balance      11162 non-null int64
housing      11162 non-null object
loan         11162 non-null object
contact      11162 non-null object
day          11162 non-null int64
month        11162 non-null object
duration     11162 non-null int64
campaign     11162 non-null int64
pdays        11162 non-null int64
previous     11162 non-null int64
poutcome     11162 non-null object
deposit      11162 non-null object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
```

Fortunately, as shown in the figure above we don't have any missing values. However if we analyse our data thoroughly we see that we have some noise in the dataset. We understand them through the pictures given below:

```
In [12]: dataset['job'].unique()
Out[12]:
array(['admin.', 'technician', 'services', 'management', 'retired',
       'blue-collar', 'unemployed', 'entrepreneur', 'housemaid',
       'unknown', 'self-employed', 'student'], dtype=object)

In [13]: dataset['education'].unique()
Out[13]: array(['secondary', 'tertiary', 'primary', 'unknown'], dtype=object)

In [14]: dataset['contact'].unique()
Out[14]: array(['unknown', 'cellular', 'telephone'], dtype=object)

In [15]: dataset['poutcome'].unique()
Out[15]: array(['unknown', 'other', 'failure', 'success'], dtype=object)
```

When we see the different values in the independent variables given above, we see that there's a value called 'unknown', this is nothing but incomplete data or corrupted data that won't help us in getting any concrete results. So we consider them as noisy values and we need to deal with them.

Code:

```
# Drop the Job,Education,Contact,Poutcome Occupations that are "Unknown"
dataset = dataset.drop(dataset.loc[dataset["job"] == "unknown"].index)
dataset = dataset.drop(dataset.loc[dataset["education"] == "unknown"].index)
dataset = dataset.drop(dataset.loc[dataset["contact"] == "unknown"].index)
dataset = dataset.drop(dataset.loc[dataset["poutcome"] == "unknown"].index)
```

```
In [17]: dataset['job'].unique()
Out[17]:
array(['admin.', 'services', 'retired', 'technician', 'entrepreneur',
       'management', 'unemployed', 'blue-collar', 'self-employed',
       'student', 'housemaid'], dtype=object)

In [18]: dataset['education'].unique()
Out[18]: array(['secondary', 'tertiary', 'primary'], dtype=object)

In [19]: dataset['contact'].unique()
Out[19]: array(['telephone', 'cellular'], dtype=object)

In [20]: dataset['poutcome'].unique()
Out[20]: array(['other', 'failure', 'success'], dtype=object)
```

Hence we have successfully removed all the noisy data from our dataset.

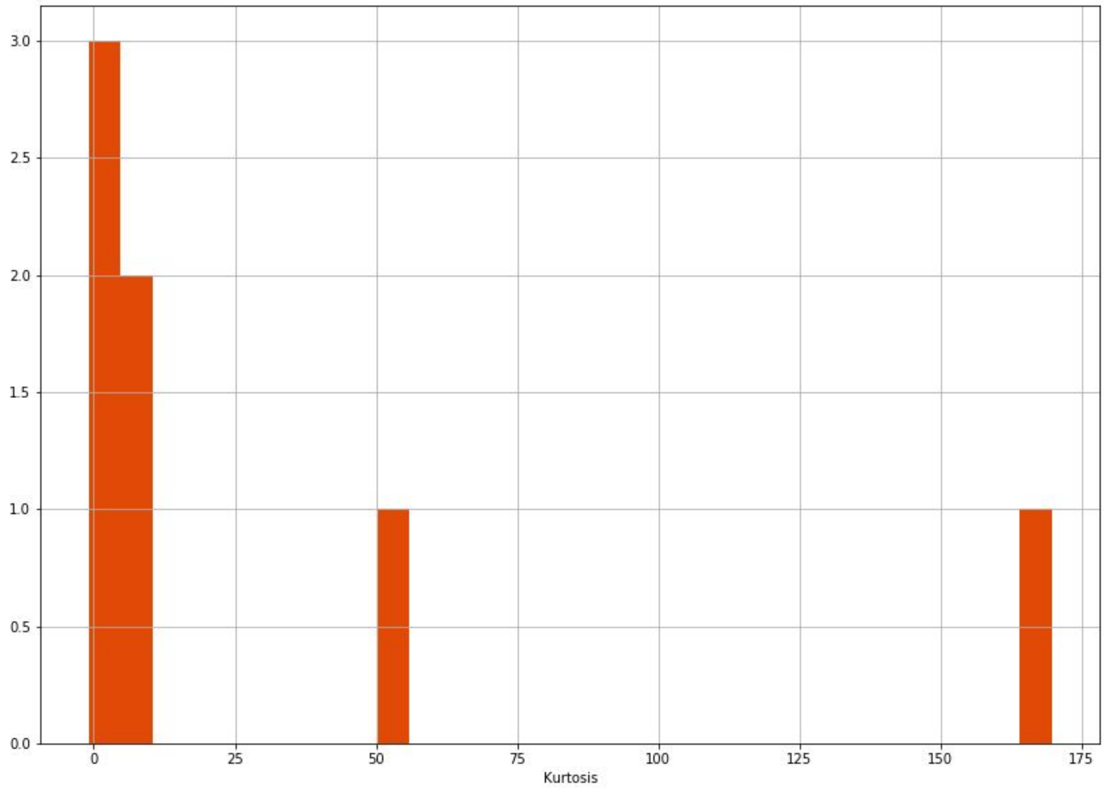
Now we see the skewness and kurtosis

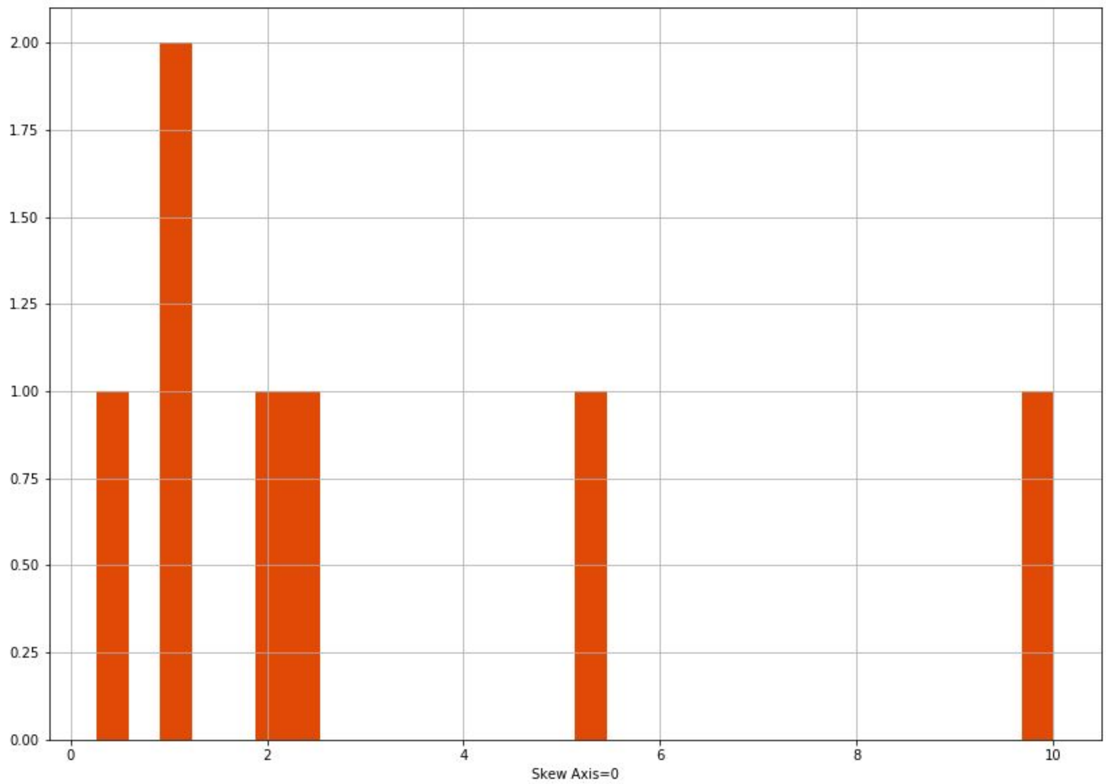
Code:

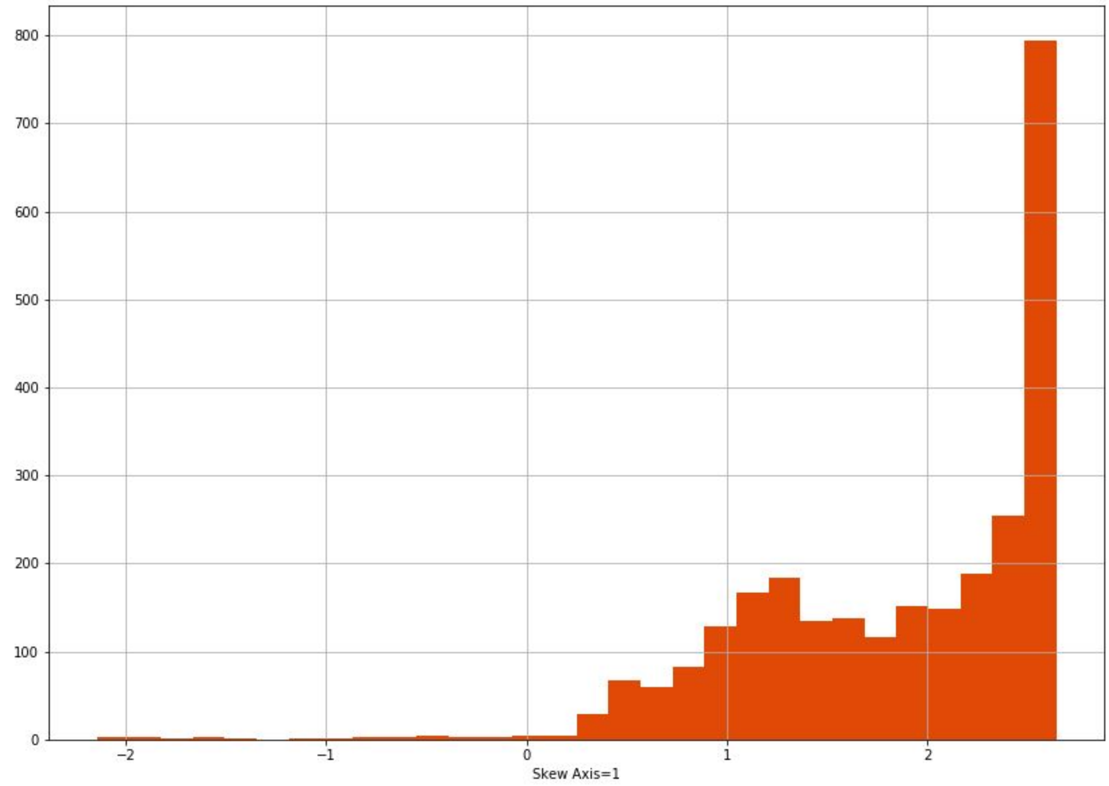
```
dataset.skew(axis=0)

dataset.skew(axis=1)

dataset.skew(axis=0).hist(bins=30,figsize=(14,10),color="#E14906")

plt.xlabel("Skew Axis=0")

dataset.skew(axis=1).hist(bins=30,figsize=(14,10),color="#E14906")

plt.xlabel("Skew Axis=1")

dataset.kurtosis().hist(bins=30,figsize=(14,10),color="#E14906")

plt.xlabel("Kurtosis")
```

```
In [27]: dataset.kurtosis()
Out[27]:
age           0.467404
balance     169.713702
day          -0.900661
duration      5.585274
campaign      7.189481
pdays         2.596477
previous     52.499026
dtype: float64
```

```
In [25]: dataset.skew(axis=0)
Out[25]:
age         0.947625
balance     9.998481
day         0.267819
duration    2.012216
campaign    2.291079
pdays       1.202832
previous    5.267334
dtype: float64

In [26]: dataset.skew(axis=1)
Out[26]:
890      1.058956
891      2.636761
951      1.809701
952      2.383979
953      1.217036
           ...
11125    2.529314
11133    2.515869
11145    0.450051
11155    1.719700
11160    2.377483
Length: 2675, dtype: float64
```

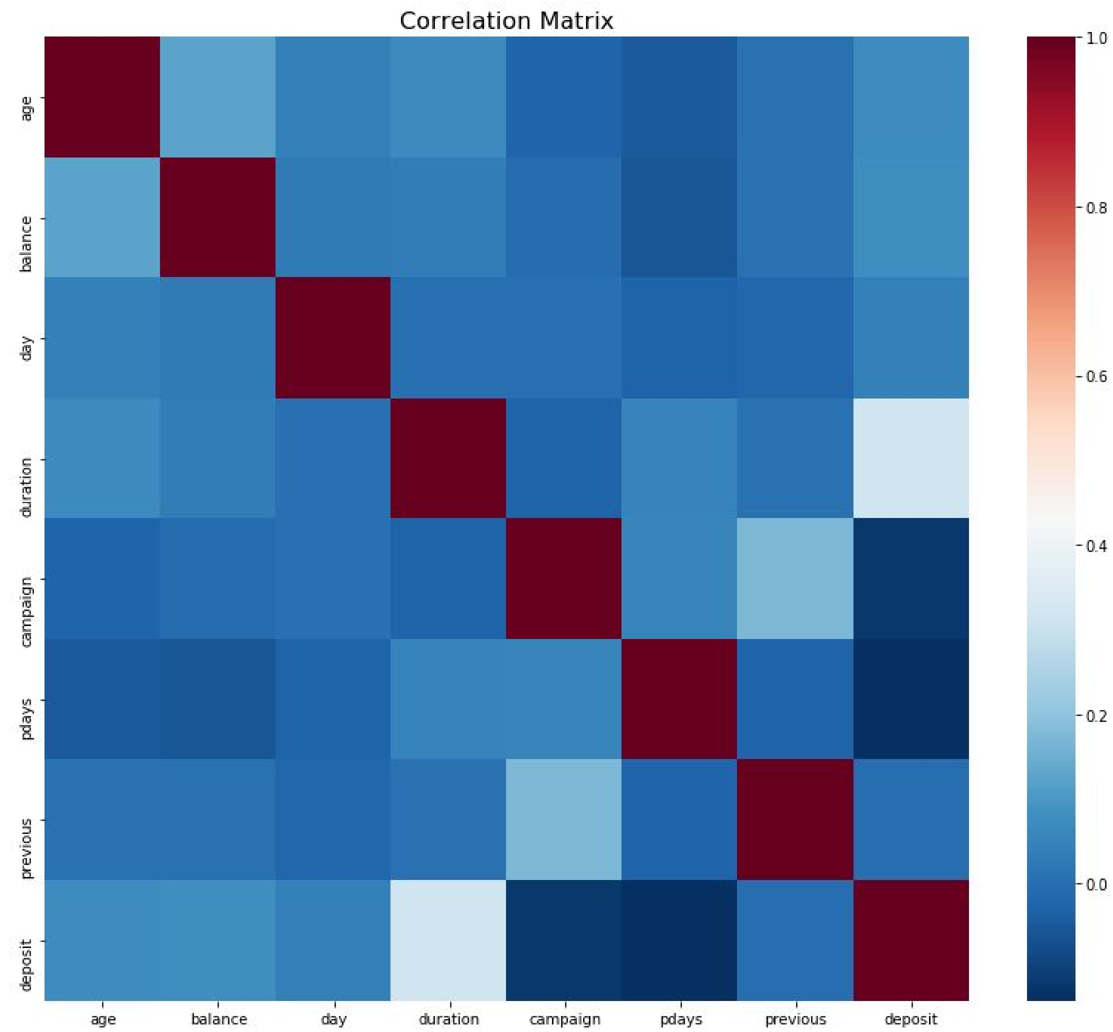

3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
Kurtosis


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
Skew Axis=0



Now let’s see the correlation matrix of our numeric input variables and the output variable:

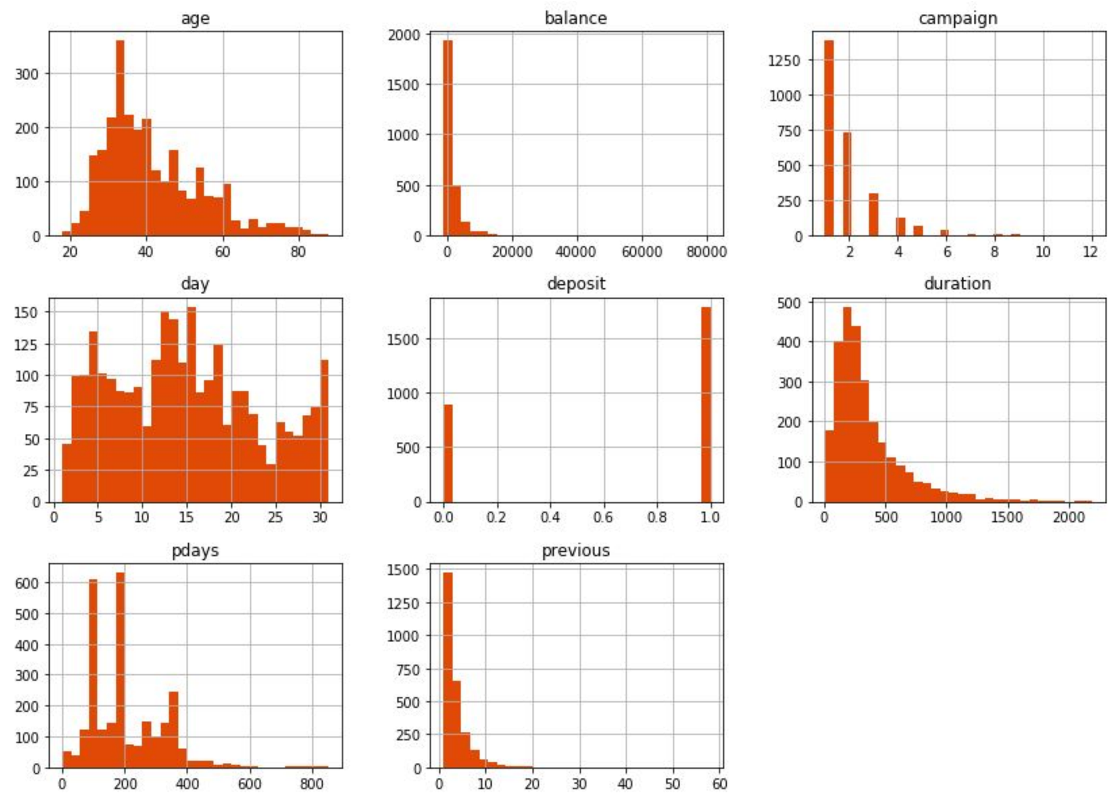
```
from sklearn.preprocessing import LabelEncoder
fig = plt.figure(figsize=(14,12))
dataset['deposit'] = LabelEncoder().fit_transform(dataset['deposit'])
# Separate both dataframes into
numeric_df = dataset.select_dtypes(exclude="object")
# categorical_df = df.select_dtypes(include="object")
corr_numeric = numeric_df.corr()
ax = sns.heatmap(corr_numeric, cbar=True, cmap="RdBu_r")
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.title("Correlation Matrix", fontsize=16)
plt.show()
```



As seen above we conclude that duration of the call is significantly correlated with the deposit attribute. Thus the duration serves as the bias in our model because if the duration is long the customer is more likely to say Yes to subscribing to a term deposit. Shorter duration would result in negative results most of the time.

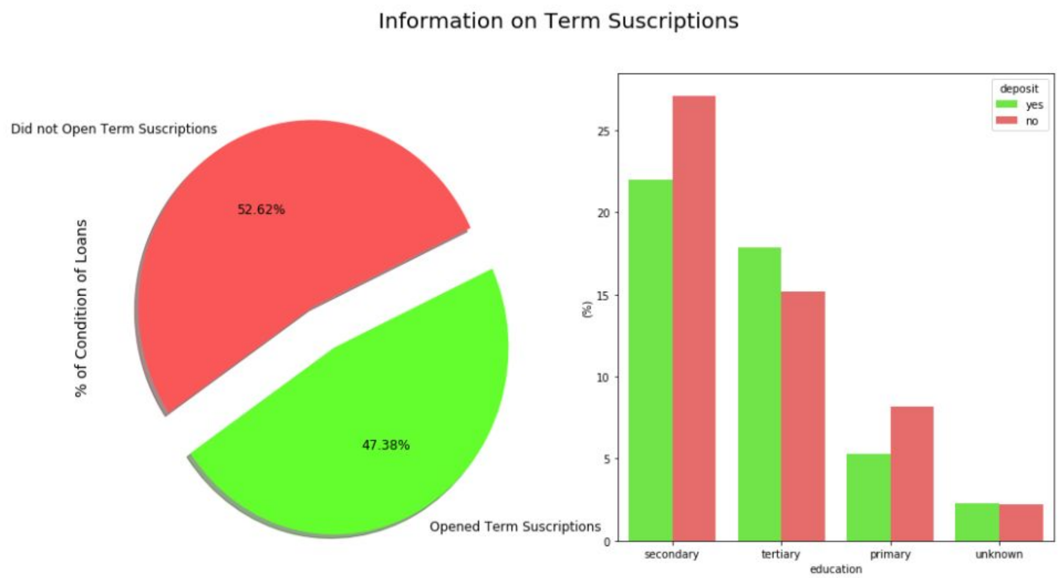
We now see some other statistical distribution in our dataset to analyze it more precisely:

dataset.hist(bins=30,figsize=(14,10),color="#E14906")

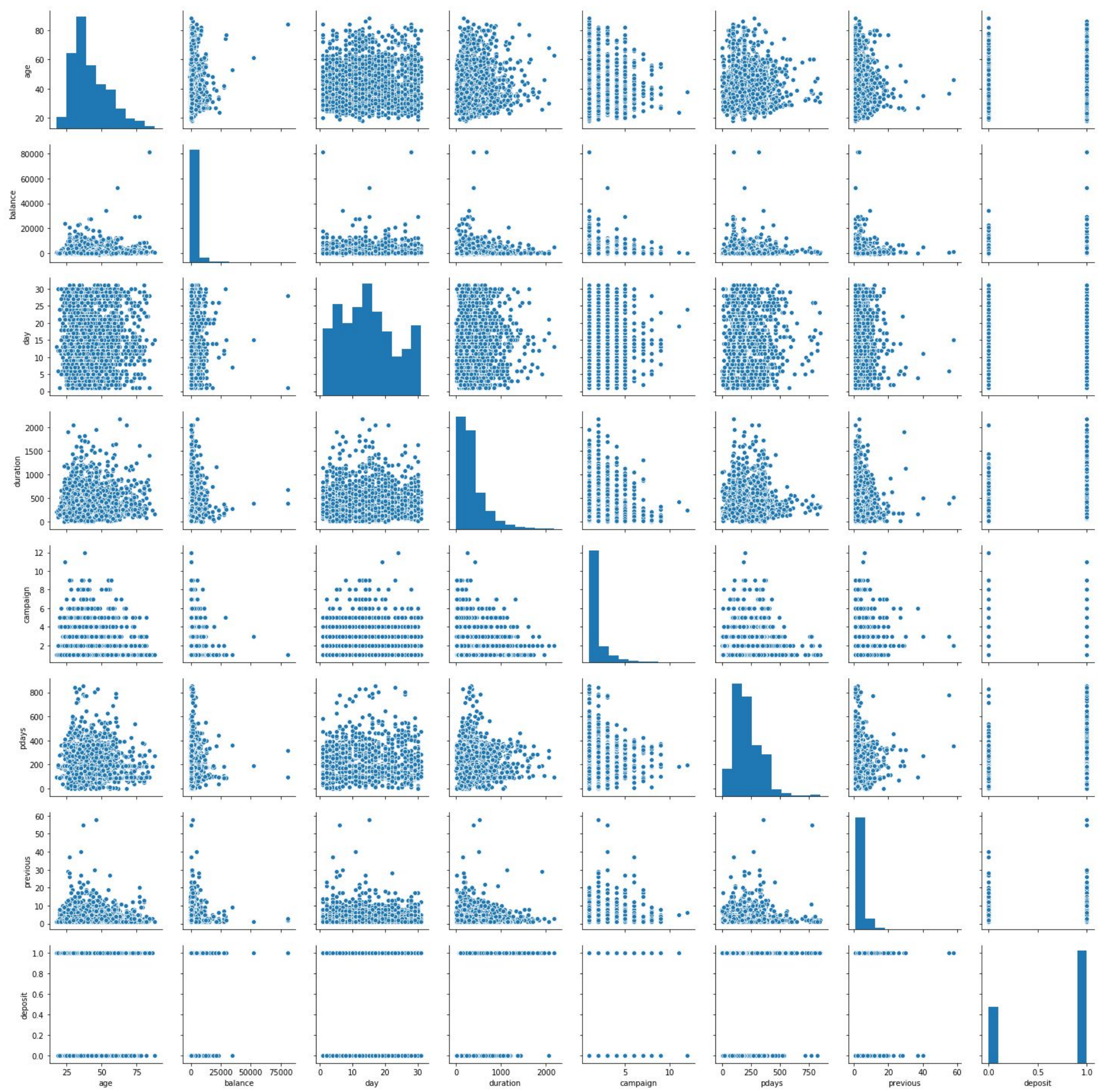



```
f, ax = plt.subplots(1,2, figsize=(16,8))

colors = ["#FA5858", "#64FE2E"]
labels ="Did not Open Term Subscriptions", "Opened Term
Subscriptions"

plt.suptitle('Information on Term Subscriptions', fontsize=20)

dataset["deposit"].value_counts().plot.pie(explode=[0,0.25],
autopct='%1.2f%%', ax=ax[0], shadow=True, colors=colors,
                                    labels=labels, fontsize=12, startangle=25)
ax[0].set_ylabel('% of Condition of Loans', fontsize=14)
palette = ["#64FE2E", "#FA5858"]
```

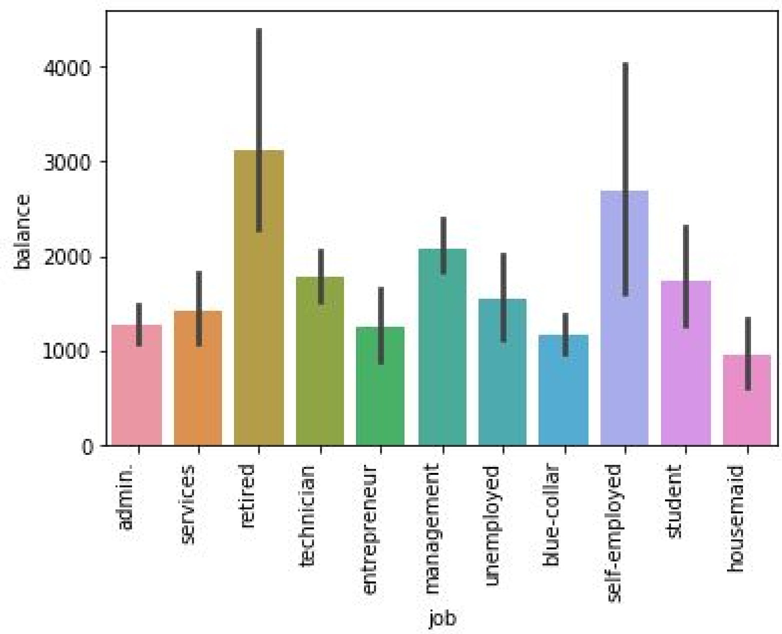
```
sns.barplot(x="education", y="balance", hue="deposit", data=dataset, palette=palette, estimator=lambda x: len(x) / len(dataset) * 100)
ax[1].set(ylabel="(%)")
ax[1].set_xticklabels(dataset["education"].unique(), rotation=0, rotation_mode="anchor")
plt.show()
```




```
sns.pairplot(data=dataset)
```

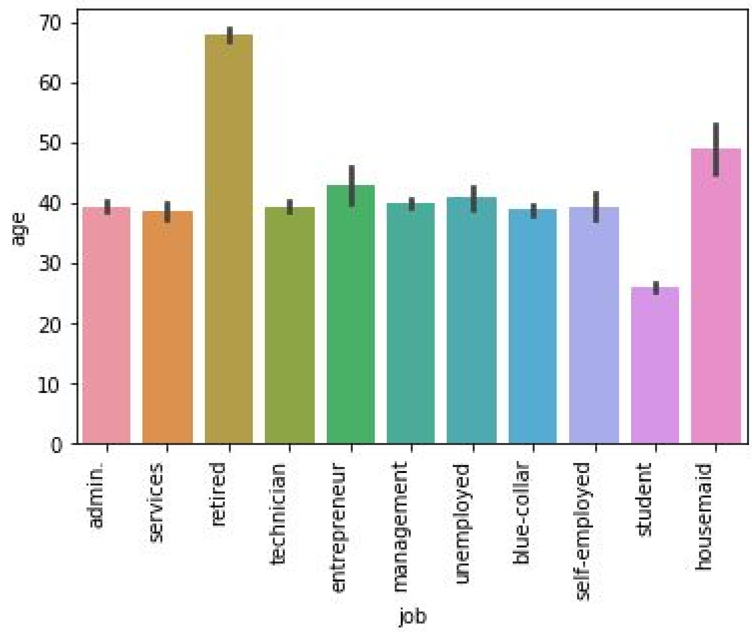



```
ax=sns.barplot(x='job',y='balance',data=dataset)
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="right")
```



ax=sns.barplot(x='job',y='age',data=dataset)

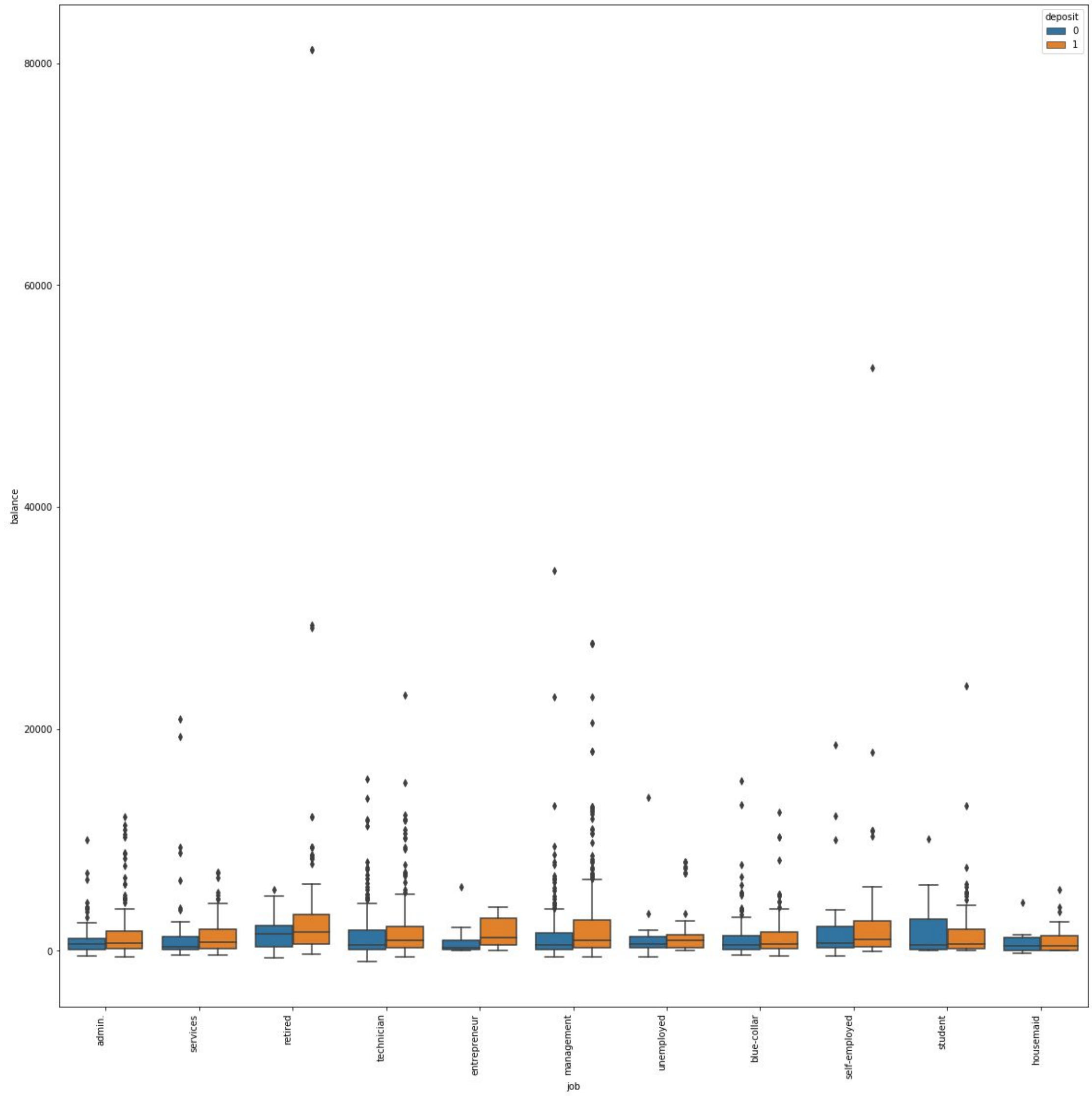
ax.set_xticklabels(ax.get_xticklabels(), rotation=90, ha="right")



figJ = plt.figure(figsize=(20,20))

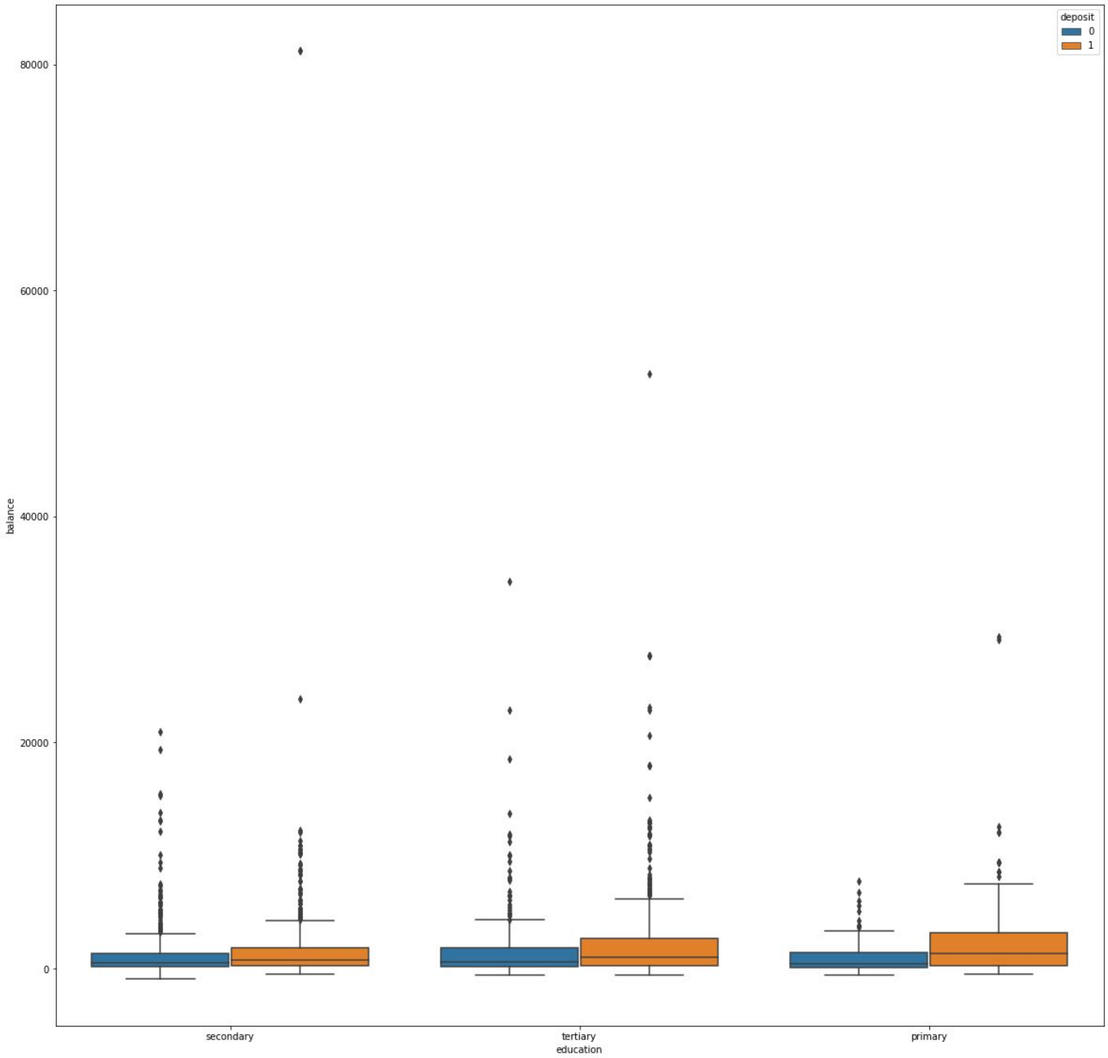
g=sns.boxplot(x='job',y='balance',hue='deposit',data=dataset)

g.set_xticklabels(g.get_xticklabels(),rotation=90,ha="right")



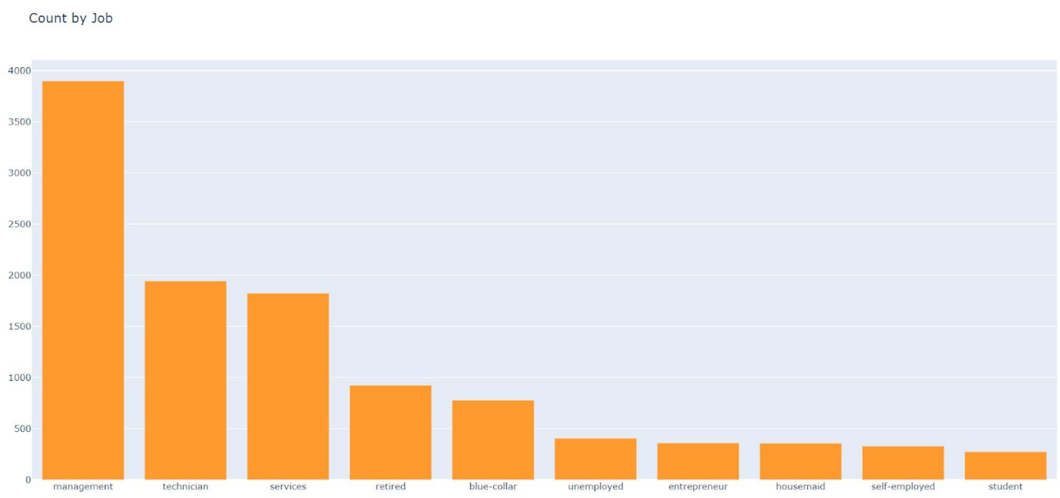
figE = plt.figure(figsize=(20,20))

e=sns.boxplot(x='education',y='balance',hue='deposit',data=dataset)



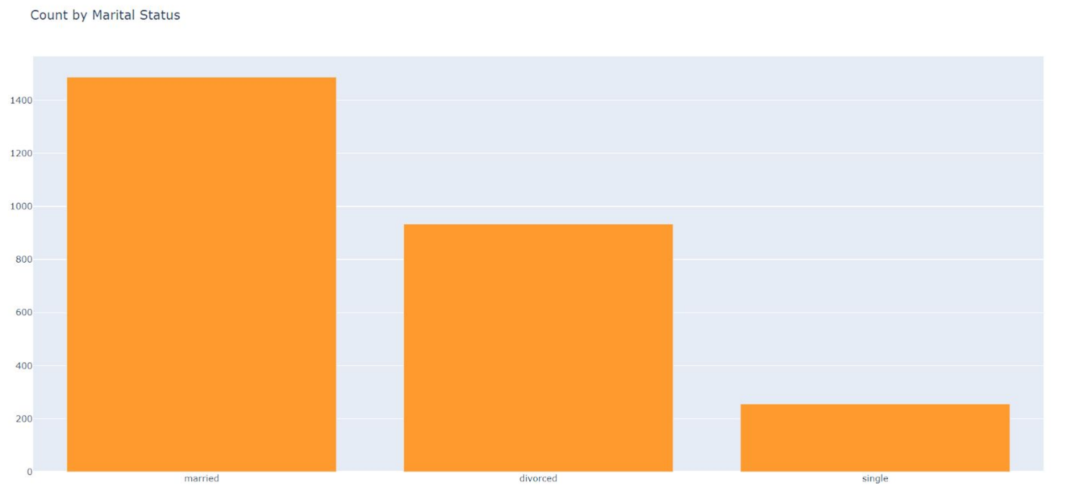

```
# Admin and management are basically the same let's put it under the same categorical value
lst = [dataset]
for col in lst:
    col.loc[col["job"] == "admin.", "job"] = "management"
jobs=dataset['job'].unique()
print(jobs)
vals = dataset['job'].value_counts().tolist()
labels = jobs
data = [go.Bar(
            x=labels,
```

```
        y=vals,
    marker=dict(
    color="#FE9A2E")
    )]
layout = go.Layout(
    title="Count by Job",
)
fig = go.Figure(data=data, layout=layout)
plot(fig, filename='basic-bar')
```

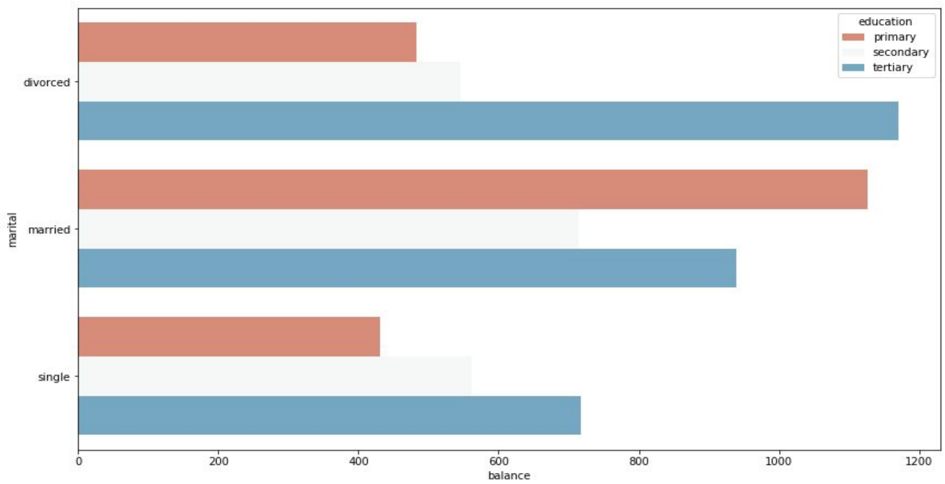



```
#Married
valsMar = dataset['marital'].value_counts().tolist()
labelsMar = ['married', 'divorced', 'single']
dataMar = [go.Bar(
        x=labelsMar,
        y=valsMar,
    marker=dict(
    color="#FE9A2E")
    )]
layout = go.Layout(
```

```
    title="Count by Marital Status",
)
fig = go.Figure(data=dataMar, layout=layout)
plot(fig, filename='basic-bar')
figMar = plt.figure(figsize=(7,3))
mar = sns.barplot(x='marital',y='balance',hue='deposit',data=dataset)
```




```
education_groups = dataset.groupby(['marital','education'],
as_index=False)['balance'].median()
figEdu = plt.figure(figsize=(14,8))
sns.barplot(x="balance", y='marital',hue='education',
data=education_groups,palette="RdBu")
```

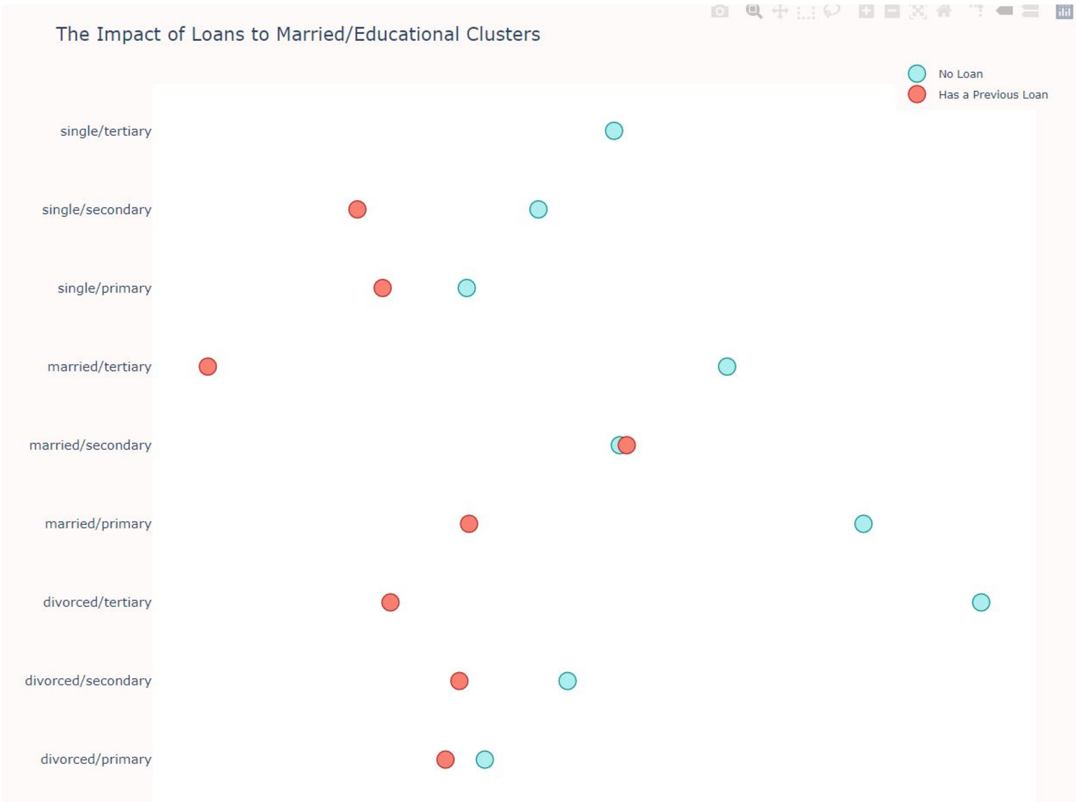


```
dataset['marital/education'] = np.nan
lst = [dataset]
for col in lst:
    col.loc[(col['marital'] == 'single') & (dataset['education'] == 'primary'),
'marital/education'] = 'single/primary'
    col.loc[(col['marital'] == 'married') & (dataset['education'] ==
'primary'), 'marital/education'] = 'married/primary'
    col.loc[(col['marital'] == 'divorced') & (dataset['education'] ==
'primary'), 'marital/education'] = 'divorced/primary'
    col.loc[(col['marital'] == 'single') & (dataset['education'] ==
'secondary'), 'marital/education'] = 'single/secondary'
    col.loc[(col['marital'] == 'married') & (dataset['education'] ==
'secondary'), 'marital/education'] = 'married/secondary'
    col.loc[(col['marital'] == 'divorced') & (dataset['education'] ==
'secondary'), 'marital/education'] = 'divorced/secondary'
    col.loc[(col['marital'] == 'single') & (dataset['education'] == 'tertiary'),
'marital/education'] = 'single/tertiary'
    col.loc[(col['marital'] == 'married') & (dataset['education'] == 'tertiary'),
'marital/education'] = 'married/tertiary'
    col.loc[(col['marital'] == 'divorced') & (dataset['education'] ==
'tertiary'), 'marital/education'] = 'divorced/tertiary'

# Let's see the group who had loans from the marital/education group

loan_balance = dataset.groupby(['marital/education', 'loan'],
as_index=False)['balance'].median()
```

```
no_loan = loan_balance['balance'].loc[loan_balance['loan'] == 'no'].values
has_loan = loan_balance['balance'].loc[loan_balance['loan'] ==
'yes'].values
labels = loan_balance['marital/education'].unique().tolist()
trace0 = go.Scatter(
    x=no_loan,
    y=labels,
    mode='markers',
    name='No Loan',
    marker=dict(
        color='rgb(175,238,238)',
        line=dict(
            color='rgb(0,139,139)',
            width=1,
        ),
        symbol='circle',
        size=16,
    )
)
trace1 = go.Scatter(
    x=has_loan,
    y=labels,
    mode='markers',
    name='Has a Previous Loan',
    marker=dict(
        color='rgb(250,128,114)',
```

```
            line=dict(
                color='rgb(178,34,34)',
                width=1,
            ),
            symbol='circle',
            size=16,
        )
    )

    data = [trace0, trace1]
    layout = go.Layout(
        title="The Impact of Loans to Married/Educational Clusters",
        xaxis=dict(
            showgrid=False,
            showline=True,
            linecolor='rgb(102, 102, 102)',
            titlefont=dict(
                color='rgb(204, 204, 204)'
            ),
            tickfont=dict(
                color='rgb(102, 102, 102)',
            ),
            showticklabels=False,
            dtick=10,
            ticks='outside',
            tickcolor='rgb(102, 102, 102)',
        ),
```

```
    margin=dict(
        l=140,
        r=40,
        b=50,
        t=80
    ),
    legend=dict(
        font=dict(
            size=10,
        ),
        yanchor='middle',
        xanchor='right',
    ),
    width=1000,
    height=800,
    paper_bgcolor='rgb(255,250,250)',
    plot_bgcolor='rgb(255,255,255)',
    hovermode='closest',
)
fig = go.Figure(data=data, layout=layout)
plot(fig, filename='lowest-oecd-votes-cast')
```



We have now seen statistical distribution of almost every valuable attribute and its impact on our dependent variable. We now have an idea about each attribute and its role in determining our dependent variable. Such in-depth analysis would help us in giving a conclusive result at the end. We further look at our model now. How to train our model and the feature extraction.

```
X = dataset.iloc[:,0:15]
y = dataset.iloc[:,15]
X.head()
loner = dataset.pop('loan')
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
labelEncoderX = LabelEncoder()
le = LabelEncoder()
y=le.fit_transform(y)
loner = le.fit_transform(loner)
ls=['default','housing']
for i in ls:
    X[i] = labelEncoderX.fit_transform(dataset[i])
ct = ColumnTransformer(transformers
=[('encoder',OneHotEncoder(categories='auto'),[1,2,3,7,9,14])],remainder
='passthrough')
X=np.array(ct.fit_transform(X))
```

#Feature Extraction and Parameter Tuning

In the above code, we have extracted our independent variables in X and our dependent variable in y. Further, we extract the categorical variables by encoding them. OneHotEncoder is used when there isn't a higher importance attached to 1 than 0. And normal encoding is done using LabelEncoder where 1 has higher importance than 0.

We now apply standardisation and normalization to ensure that one variable having higher range of values doesn't dominate the other one having comparatively low range of values.

```
#Standard Scaling
from sklearn.preprocessing import StandardScaler
scX = StandardScaler()
X = scX.fit_transform(X)
```

We now split the data into the training set and test set and stratify the data. We carry out stratified sampling on loan to ensure that it maintains the proportion of yes and no in the training and test set.

```
#Train-Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.25,random_state=0,stratify=loner)
```

Now we have to decide which classification model to use for this problem. In order to decide the most accurate classification model, I tried every classification model possible and analysed them using confusion matrix and k-cross validation. The Logistic Regression Model turned out to perform better than SVM, Naive Bayes, Decision Tree and Random Forest Classification. SVM and Random Forest were very close to Logistic Regression but Logistic Regression just edged them in its prediction. We now look at the code:

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
lrClassifier = LogisticRegression()
```

```
lrClassifier.fit(X_train,y_train)

y_pred=lrClassifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test, y_pred)
cr = classification_report(y_test,y_pred)

#K Cross Validation
# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = lrClassifier, X = X_train, y =
y_train, cv = 10)
```

cm - NumPy array

	0	1
0	170	62
1	45	392

Text editor - cr

	precision	recall	f1-score	support
0	0.79	0.73	0.76	232
1	0.86	0.90	0.88	437
accuracy			0.84	669
macro avg	0.83	0.81	0.82	669
weighted avg	0.84	0.84	0.84	669

In [10]: accuracies.mean()

Out[10]: 0.8255191254781369

#Learning Methodology

I further improved this model by implementing XGBoost Classifier. The learning methodology of XGBoost involves gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning. This algorithm is designed to be used in large datasets like this and it doesn't require feature scaling and is very efficient. Parameter tuning in XGBoost involves selecting the booster type in our case it is gbtree which is used for tree based models. Then comes nthread which allows parallel computing. Then there are many booster parameters that work in enhancing the algorithm however we haven't specified any because by default the best set of parameters are chosen by XGBoost.

Code:

```
xgClassifier = xgb.XGBClassifier()
xgClassifier.fit(X_train,y_train)
# Predicting the Test set results
y_pred = xgClassifier.predict(X_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix,classification_report
cm = confusion_matrix(y_test, y_pred)
cr = classification_report(y_test,y_pred)

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = xgClassifier, X = X_train, y =
y_train, cv = 10)
accuracies.mean()
```

cm - NumPy array

	0	1
0	172	60
1	22	415

Text editor - cr

```
              precision    recall  f1-score   support

           0       0.89      0.74      0.81       232
           1       0.87      0.95      0.91       437

    accuracy                           0.88       669
   macro avg       0.88      0.85      0.86       669
weighted avg       0.88      0.88      0.87       669
```

In [15]: accuracies.mean()

Out[15]: 0.8444570114252856

#Accuracy

evaluate predictions

accuracy = accuracy_score(y_test, y_pred)

print(In [32]: print("Accuracy: %.2f%%" % (accuracy * 100.0))

Accuracy: 87.74%

We clearly see the improvement in accuracy and hence we use this model.

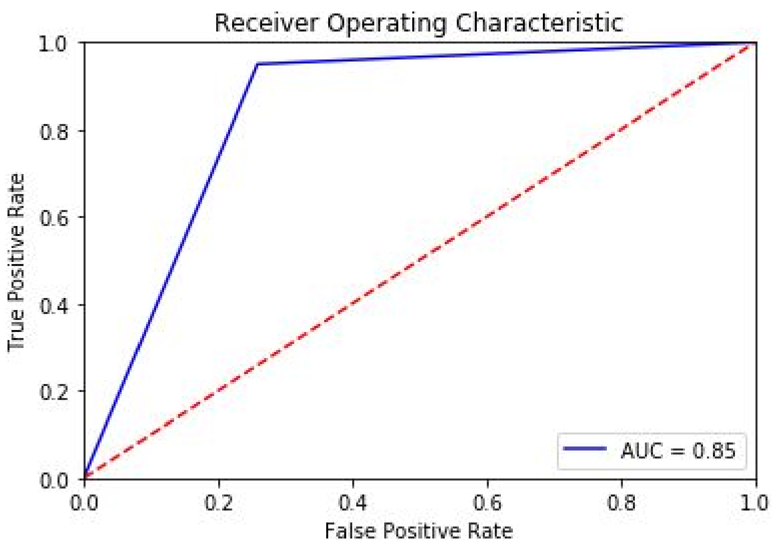
The ROC Curve of our model:

Code:

from sklearn.metrics import

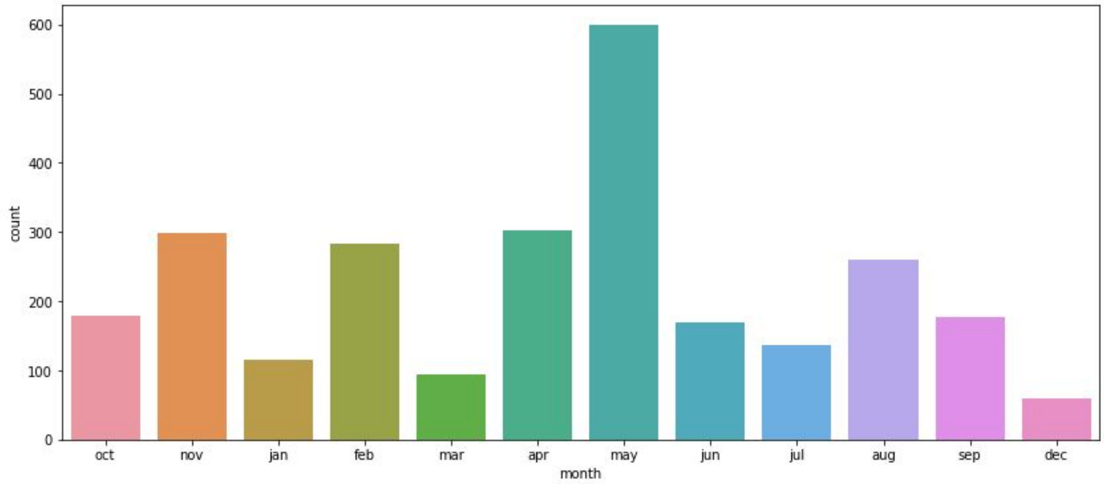
confusion_matrix,classification_report,roc_curve,auc

```
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
# method I: plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

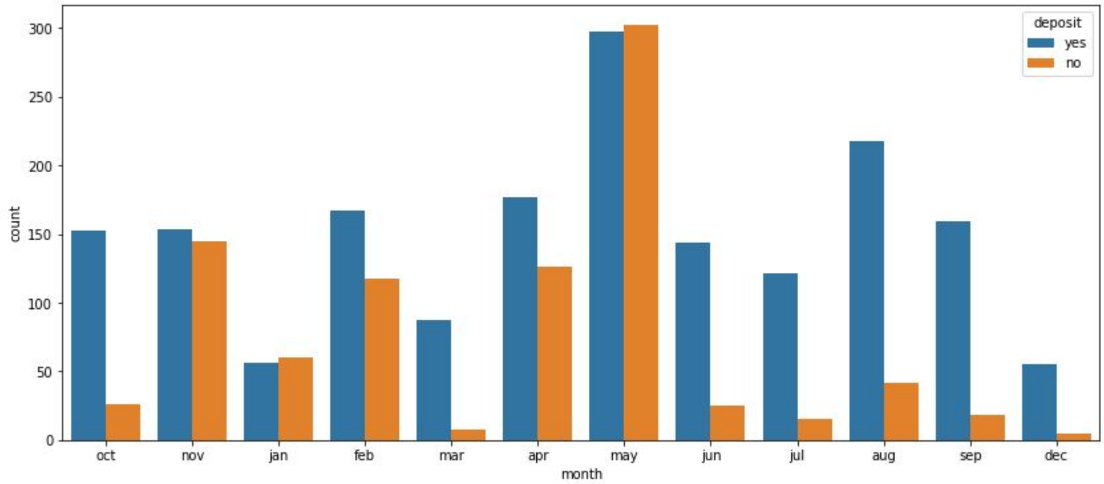


Coming to the conclusive results from all our analysis, we can make improvements in the following way:

- Month: We see that May has the highest number of calls made.

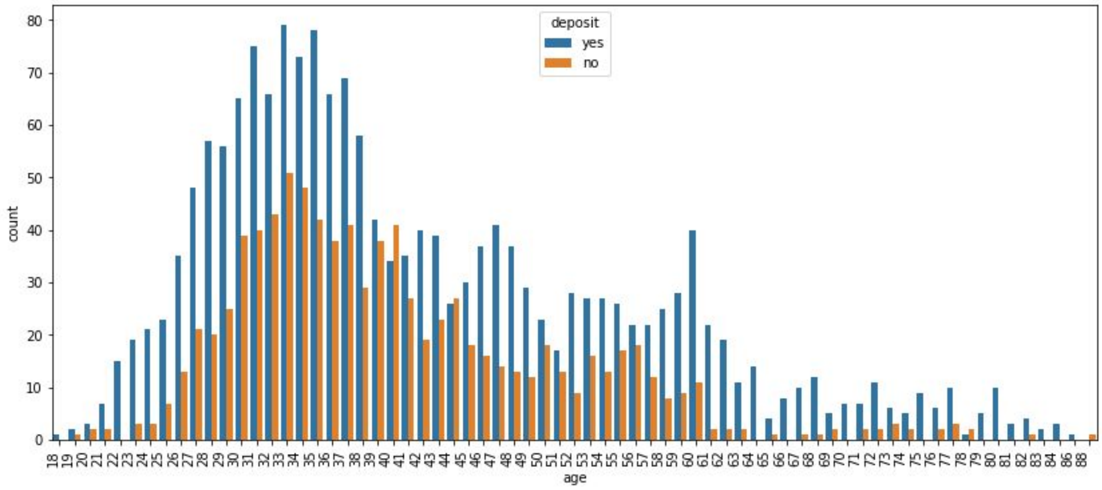


However it wasn't that effective as shown in image below

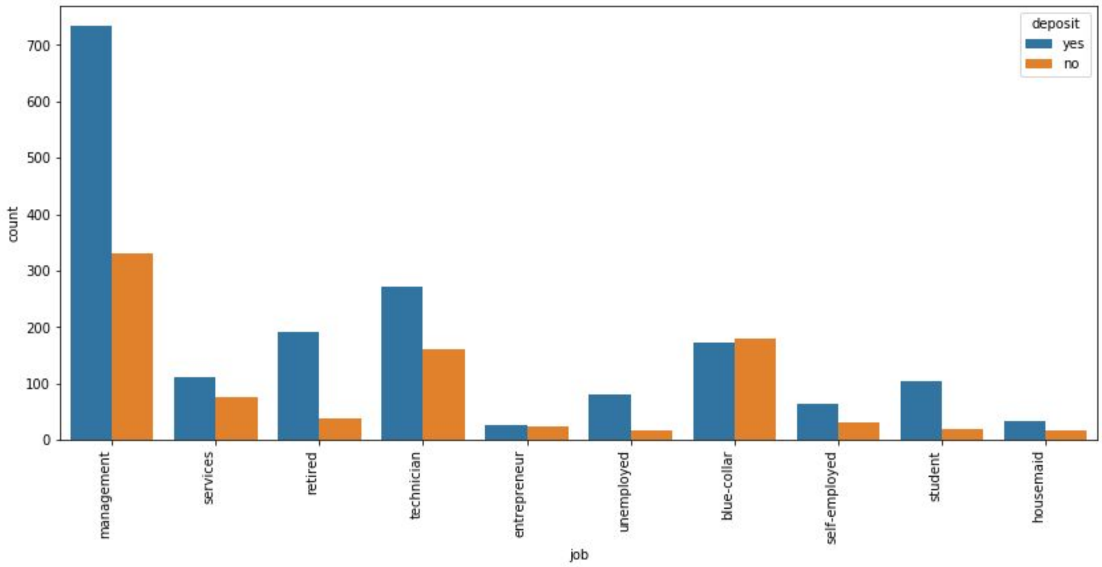


We can see more effectiveness in June,July,August,September and December. So for the next marketing campaign we should target these months to get some positive response.

- Age: We observed age groups of 20-40 and above 60s giving positive results and thus there should be our target audience for the next campaign.



- Job : People in management had the highest count of subscription to term deposit. However if we see the difference between yes and no we observe that students and retired individuals respond more positively in subscribing to a term deposit.



- Loan and Balance: Potential clients in the low balance and no balance category were more likely to have a house loan than people in the average and high balance category. What does it mean to have a house loan? This means that the potential client has financial compromises to pay back its house loan and thus, there is no cash for him or she to subscribe to a term deposit account. However, we see that potential clients in the average and high balances are less likely to have a house loan and therefore, more likely to open a term deposit. Lastly, the next marketing campaign should focus on individuals of average and high balances in order to increase the likelihood of subscribing to a term deposit.
- Call Duration: As seen earlier in the correlation matrix, duration is highly correlated with a person taking a term deposit so in order to turn a customer into a potential client we should develop a questionnaire and prepare a convincing script in order for turning the marketing campaign into a huge success.