Name: Mansi Panchal

Email: mansi130698@gmail.com

Contact no.: 8287376653

Linkeden: https://www.linkedin.com/in/mansipanchal98/

Report

Abstract

The customer churn prediction is one of the challenging problems, if customer churn continues to occur; the enterprise will gradually lose its competitive advantage. Hence it is relevant problem to be considered.

The objective of this assignment is to build a predictive model that can predict customer churn for a given company.

Here will use machine learning techniques to build the model and the procedure include feature selection, model evaluation, and performance metrics.

Problem statement

The classification goal is to predict if the client will subscribe a term deposit or not (variable y).

Dataset

The data is related with direct marketing campaigns of a Portuguese banking institution.

The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (or not) subscribed.

Variable names

1. job
2. marital
3. education
4. default (has credit in default)
5. balance (average yearly balance, in euros)
6. housing (has housing loan)
7. loan (has personal loan)

8. contact (contact communication type)

9. day (last contact day of the month)

10. month (last contact month of year)

11. duration (last contact duration, in seconds)

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

13. pdays (number of days that passed by after the client was last contacted from a previous campaign)

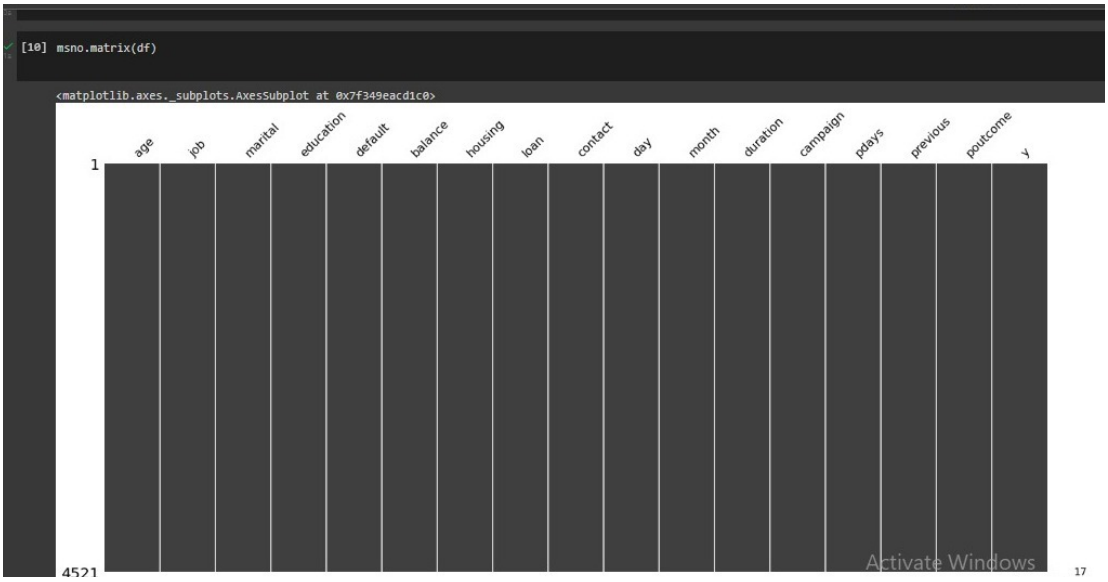
14. previous (number of contacts performed before this campaign and for this client)

15. poutcome (outcome of the previous marketing campaign)

16. age

Data preprocessing

Checking for the missing values



Since there are no missing values we will further proceed with the analysis.

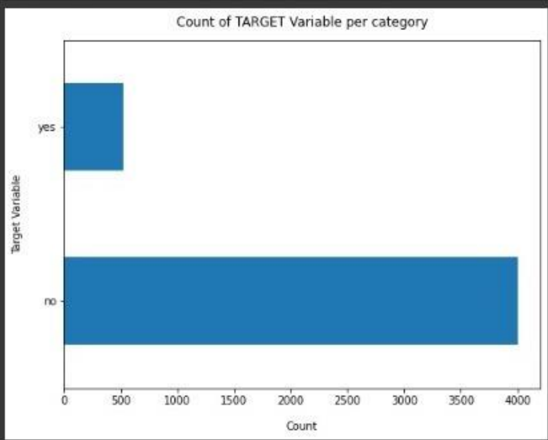
The descriptive statistics for the numerical variable is

```
[11] # Check the descriptive statistics of numeric variables
     df.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

Target variable analysis

```
[12] df['y'].value_counts().plot(kind='barh', figsize=(8, 6))
     plt.xlabel("Count", labelpad=14)
     plt.ylabel("Target Variable", labelpad=14)
     plt.title("Count of TARGET Variable per category", y=1.02);
```



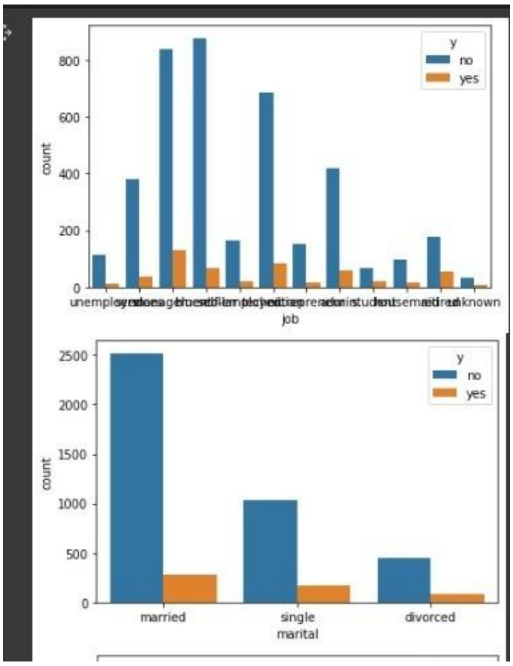

```
[13] 100*df['y'].value_counts()/len(df['y'])

     no     88.476001
     yes    11.523999
     Name: y, dtype: float64
```

Data is highly imbalanced, ratio = 88:11. So we analyse the data with other features while taking the target values separately to get some insights.

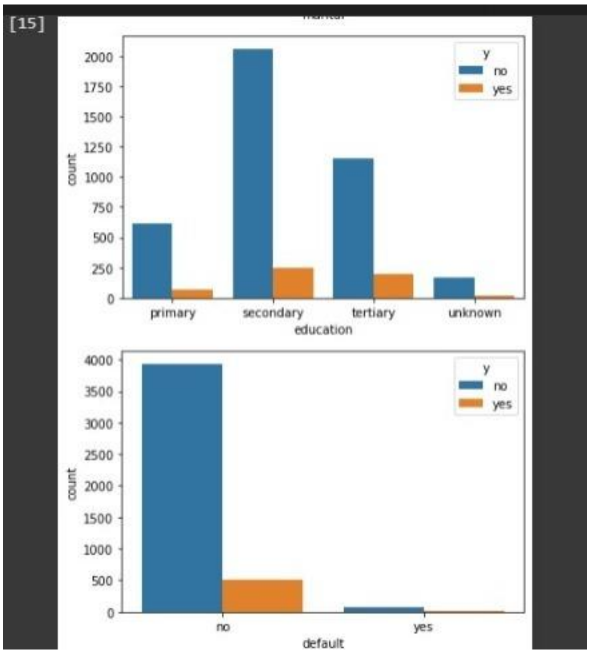
Data exploration

Univariate Analysis

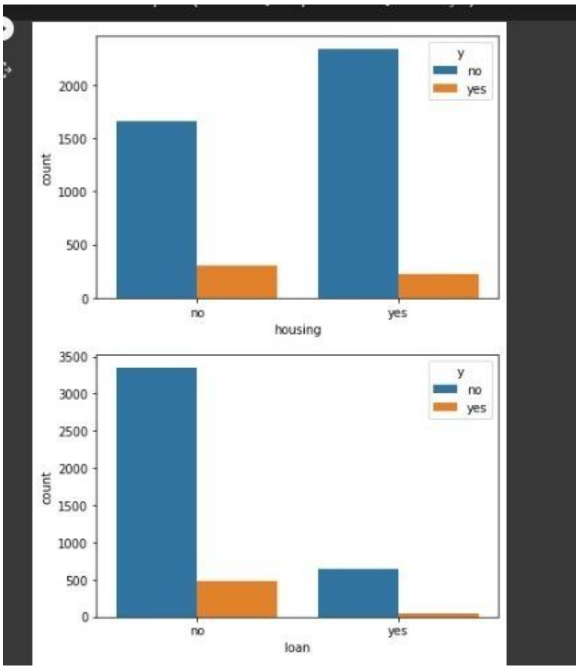
Plot distribution of individual predictors by target variables



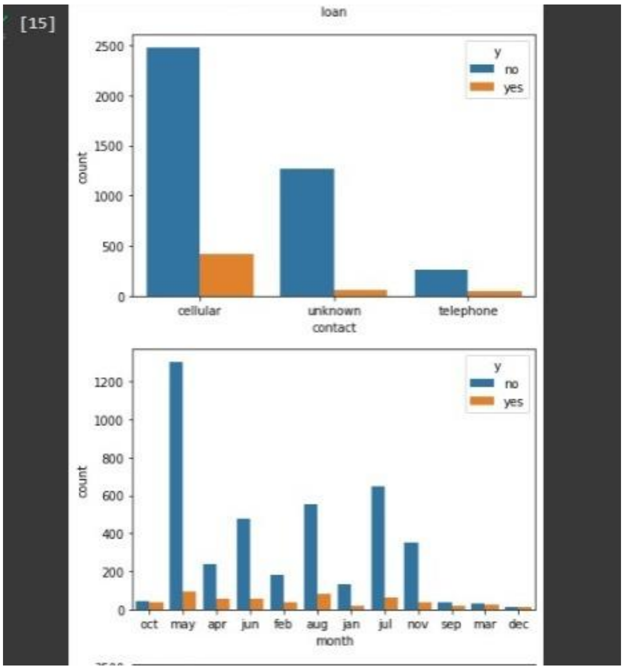
count
job
y
no
yes
count
married
single
divorced
marital



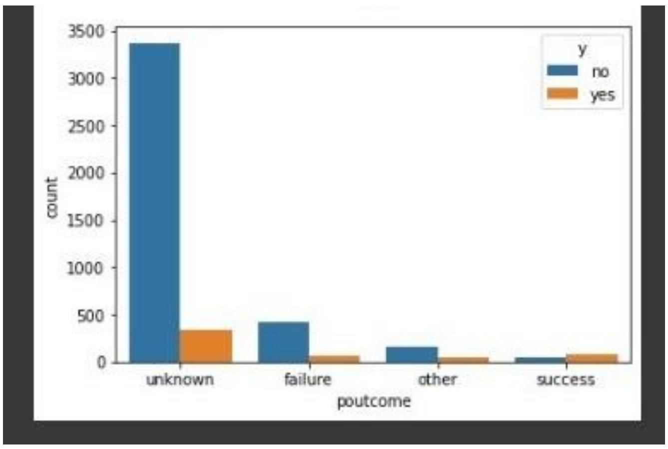
[15]
count
primary
secondary
tertiary
unknown
education
y
no
yes
count
no
yes
default



count
no
yes
housing
y
no
yes
count
no
yes
loan



[15]
loan
count
cellular
unknown
telephone
contact
y
no
yes
count
oct
may
apr
jun
feb
aug
jan
jul
nov
sep
mar
dec
month

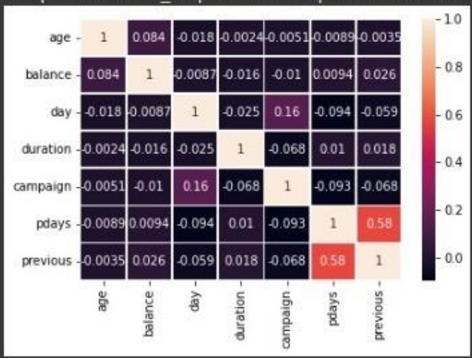


y
no
yes
count
unknown
failure
other
success
poutcome

Heatmap for numeric columns

```
[17] corr=df[numerical_f].corr()
     sns.heatmap(corr, annot=True,linewidths=.8)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f349b5fae50>



The features 'pdays' and 'previous' are moderately collinear. Since correlation for all variables is < 0.7 hence no 2 columns are multicollinear.

Converting categorical variables to numerical variables

Convert the target variable 'y' in a binary numeric variable i.e. Yes=1 ; No = 0

```
[18] df['y'] = np.where(df.y =='yes',1,0)
```

Convert all the categorical variables into dummy variables

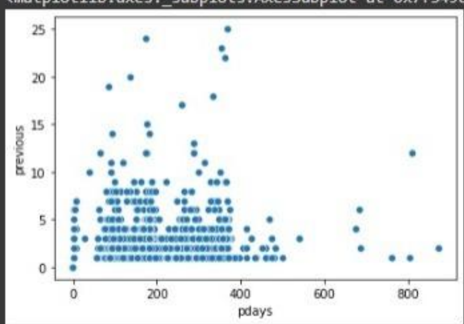
```
[20] data_dummies = pd.get_dummies(df)
     data_dummies.head(10)
```

	age	balance	day	duration	campaign	pdays	previous	y	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	month_oct
0	30	1787	19	79	1	-1	0	0	0	0	...	0	0	0	0	1
1	33	4789	11	220	1	339	4	0	0	0	...	0	0	1	0	0
2	35	1350	16	185	1	330	1	0	0	0	...	0	0	0	0	0
3	30	1476	3	199	4	-1	0	0	0	0	...	1	0	0	0	0
4	59	0	5	226	1	-1	0	0	0	1	...	0	0	1	0	0
5	35	747	23	141	2	176	3	0	0	0	...	0	0	0	0	0
6	36	307	14	341	1	330	2	0	0	0	...	0	0	1	0	0
7	39	147	6	151	2	-1	0	0	0	0	...	0	0	1	0	0
8	41	221	14	57	2	-1	0	0	0	0	...	0	0	1	0	0

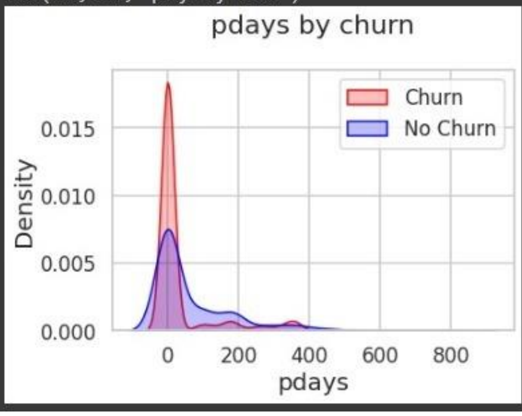
Plots

```
[21] sns.scatterplot(data=data_dummies, x='pdays', y='previous')
```

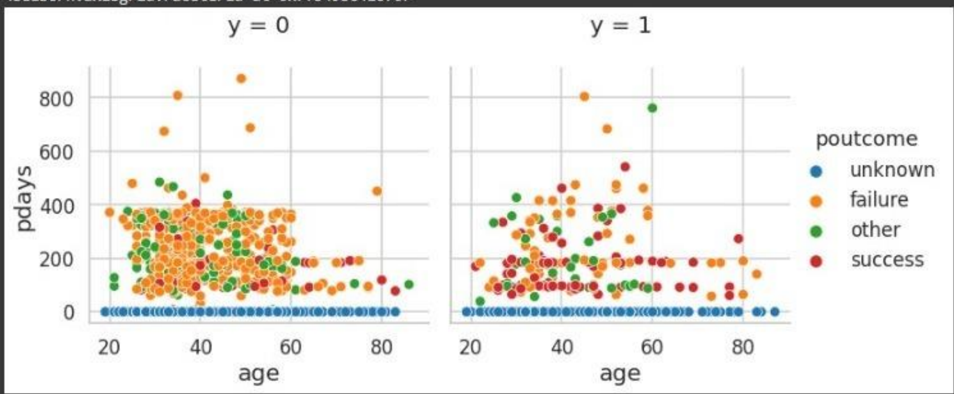
<matplotlib.axes._subplots.AxesSubplot at 0x7f349ea8e880>



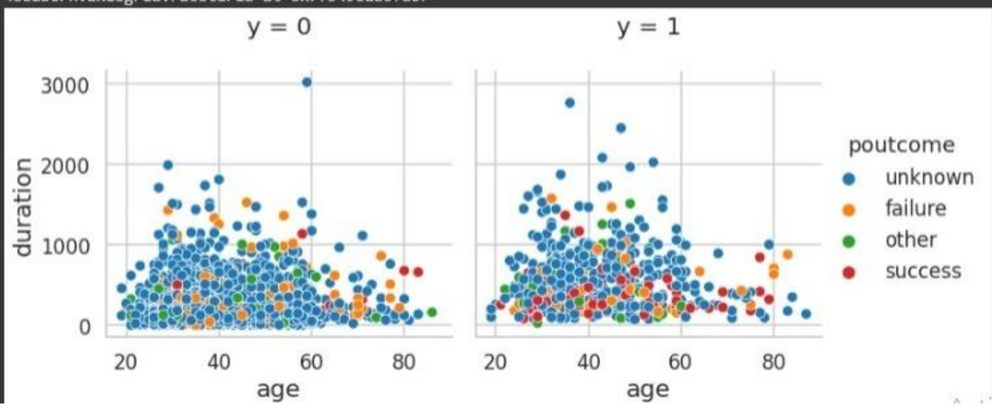
```
[34] Mth = sns.kdeplot(data_dummies.pdays[(data_dummies["y"] == 0) ],
                    color="Red", shade = True)
     Mth = sns.kdeplot(data_dummies.pdays[(data_dummies["y"] == 1) ],
                    ax =Mth, color="Blue", shade= True)
     Mth.legend(["Churn","No Churn"],loc='upper right')
     Mth.set_ylabel('Density')
     Mth.set_xlabel('pdays')
     Mth.set_title('pdays by churn')

     Text(0.5, 1.0, 'pdays by churn')
```



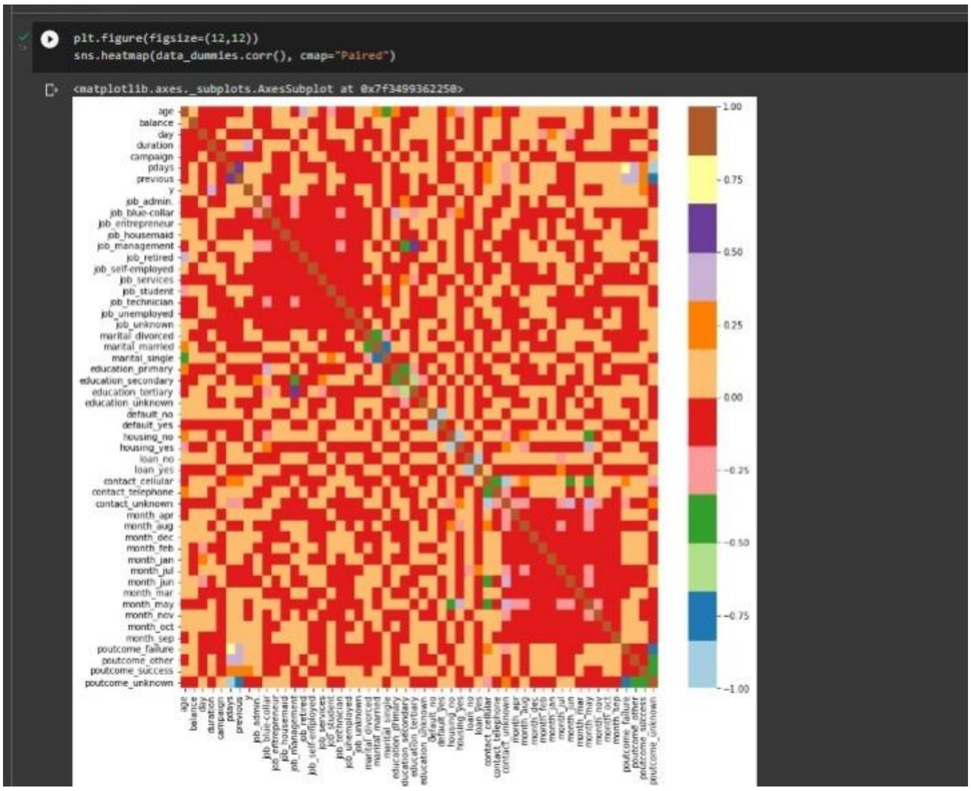

```
[50] sns.relplot(data = df, x="age", y="pdays",hue= 'poutcome', col = 'y')

     <seaborn.axisgrid.FacetGrid at 0x7f349b641070>
```




```
sns.relplot(data = df, x="age", y="duration",hue= 'poutcome', col = 'y')

<seaborn.axisgrid.FacetGrid at 0x7f3496ad0fd0>
```



Clearly the density around zero number of pdays is higher. Hence, pdays is linked to Churn.



By looking at the given heatmat we can say that poutcome_failure and pdays are highly correlated we will drop the poutcome_failure feature from the dummy variable dataset.

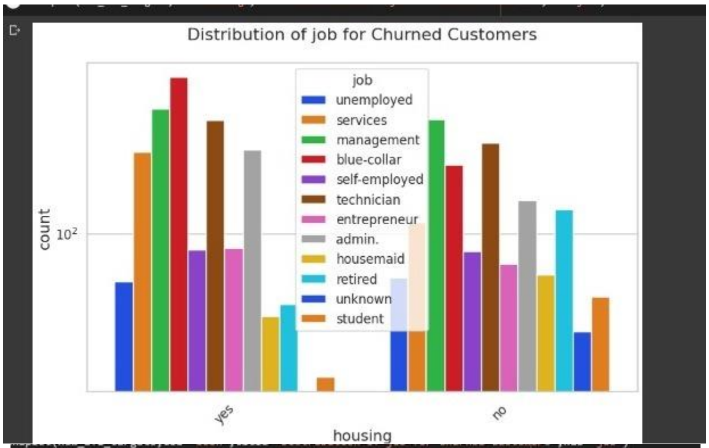
Feature selection

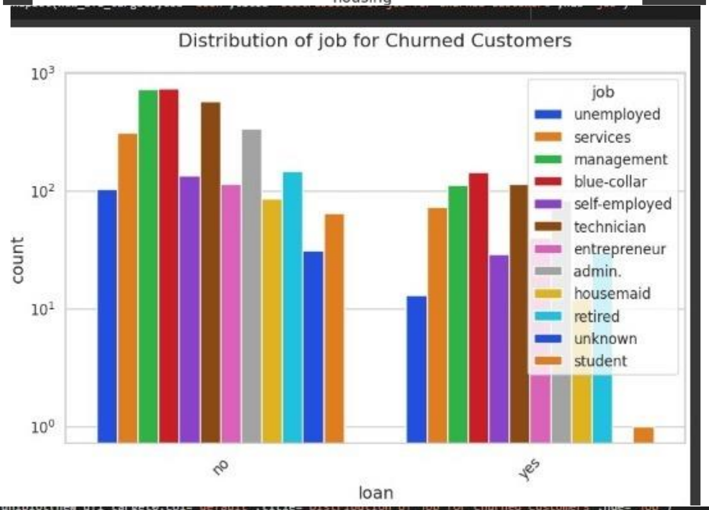
The features/columns dropped are ‘days’, ‘poutcome_failure’

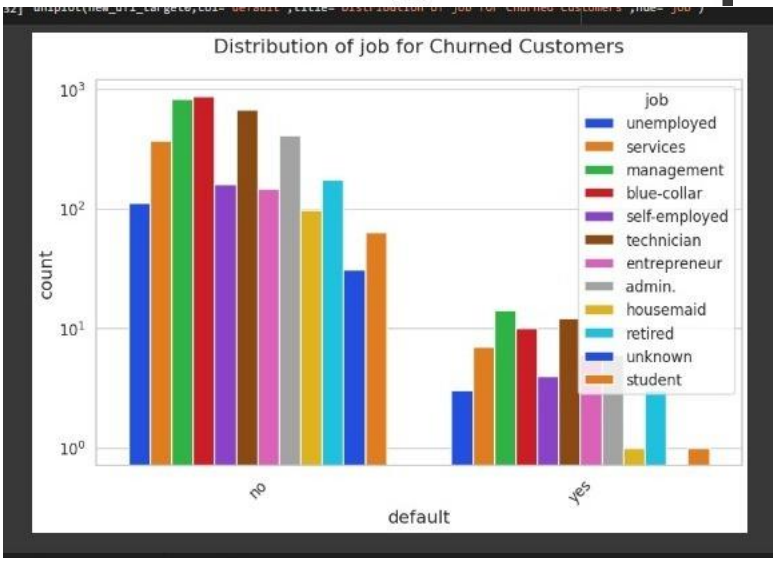
‘days’ does not provide any relevant information for prediction of churn customer

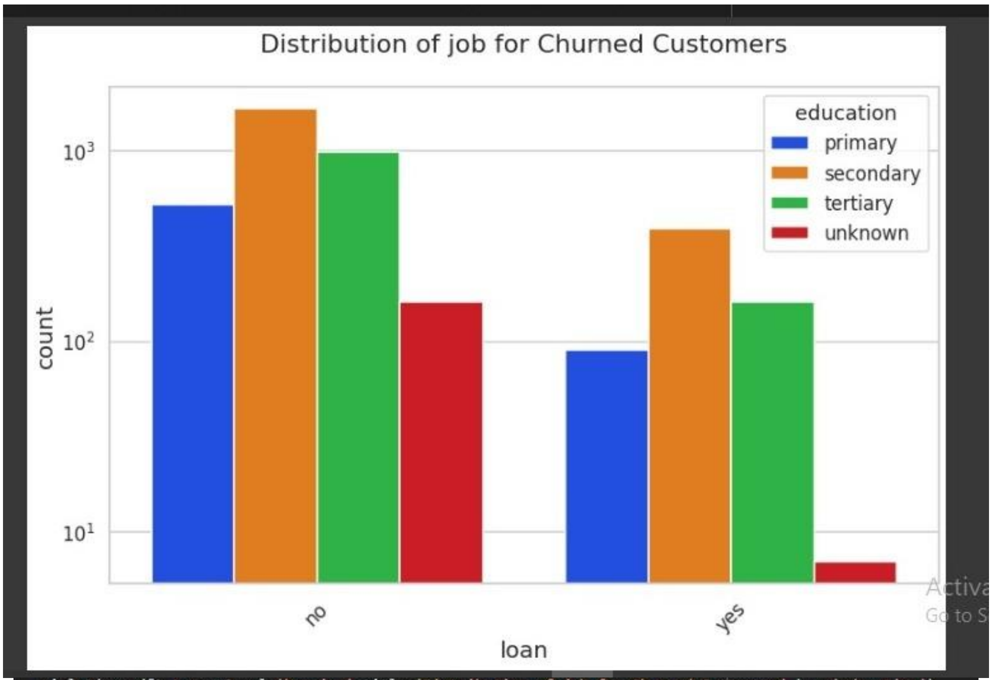
‘poutcome_failure’ is multicollinear with ‘pdays’

Bivariate analysis

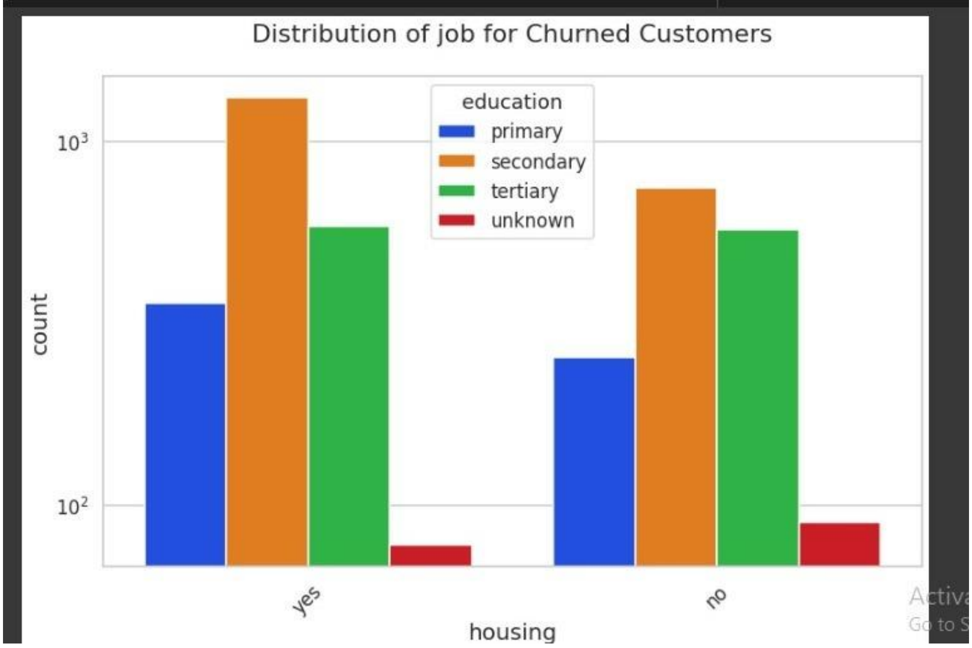

Distribution of job for Churned Customers
job
unemployed
services
management
blue-collar
self-employed
technician
entrepreneur
admin.
housemaid
retired
unknown
student
count
10^2
yes
no
housing

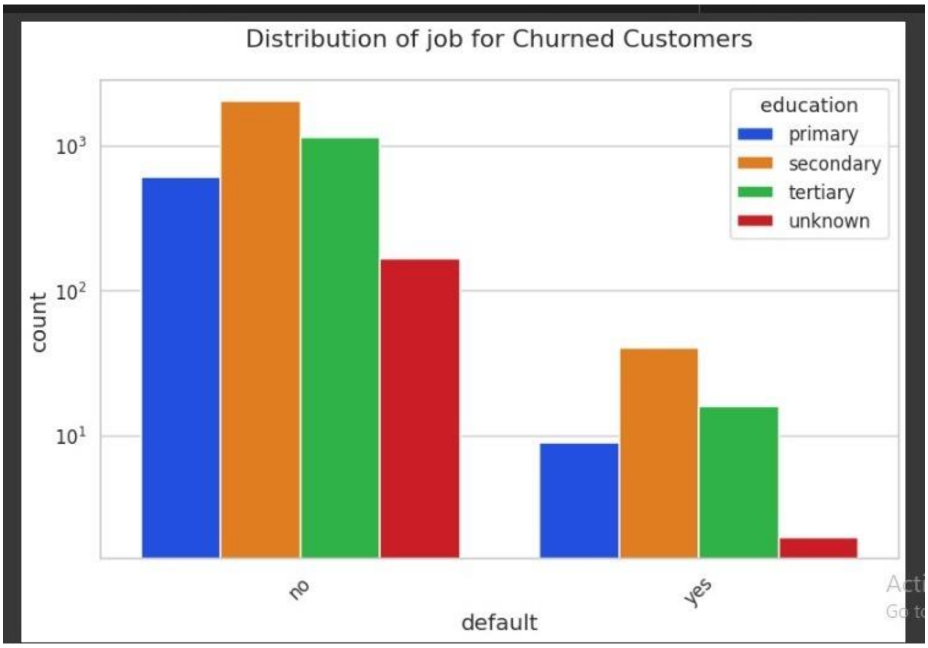

Distribution of job for Churned Customers
job
unemployed
services
management
blue-collar
self-employed
technician
entrepreneur
admin.
housemaid
retired
unknown
student
count
10^3
10^2
10^1
10^0
no
yes
loan


Distribution of job for Churned Customers
job
unemployed
services
management
blue-collar
self-employed
technician
entrepreneur
admin.
housemaid
retired
unknown
student
count
10^3
10^2
10^1
10^0
no
yes
default




```
uniplot(new_df1_target0,col='housing',title='Distribution of job for Churned Customers',hue='education')
```

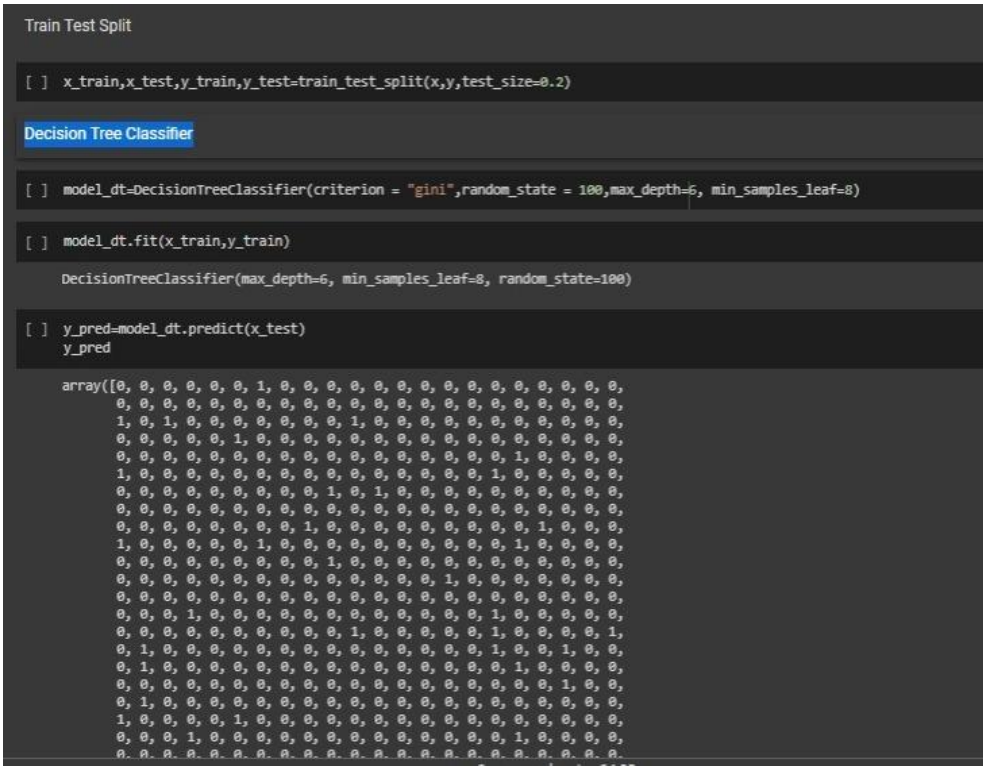




Now we will train a machine learning model over 49 column features ans 4521 data points

Model training

- **Decision Tree Classifier**



Train Test Split

```
[ ] x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

Decision Tree Classifier

```
[ ] model_dt=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
[ ] model_dt.fit(x_train,y_train)

DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

```
[ ] y_pred=model_dt.predict(x_test)
    y_pred

array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
```

```
model_dt.score(x_test,y_test)

0.9060773480662984
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       812
           1       0.58      0.32      0.41        93

    accuracy                           0.91       905
   macro avg       0.75      0.65      0.68       905
weighted avg       0.89      0.91      0.89       905
```

As it's an imbalanced dataset, we shouldn't consider Accuracy as our metrics to measure the model, as Accuracy is cursed in imbalanced datasets.

Hence, we need to check recall, precision & f1 score for the minority class, and it's quite evident that the precision, recall & f1 score is too low for Class 1, i.e. churned customers.

Hence, moving ahead to call SMOTEENN (UpSampling + ENN)

```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(x,y)
```

```
xr_train,xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_size=0.2)
```

```
model_dt_smote=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
model_dt_smote.fit(xr_train,yr_train)
yr_predict = model_dt_smote.predict(xr_test)
model_score_r = model_dt_smote.score(xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))

0.9226240538267452
              precision    recall  f1-score   support

           0       0.92      0.92      0.92       561
           1       0.93      0.93      0.93       628

    accuracy                           0.92      1189
   macro avg       0.92      0.92      0.92      1189
weighted avg       0.92      0.92      0.92      1189
```

Now we can see quite better results, i.e. Accuracy: 92.2 %, and a very good recall, precision & f1 score for minority class.

Let's try with some other classifier.

- **Random Forest Classifier**

Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier

[ ] model_rf=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)

[ ] model_rf.fit(x_train,y_train)

    RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)

[ ] y_pred=model_rf.predict(x_test)

[ ] model_rf.score(x_test,y_test)

    0.9082872928176795

[ ] print(classification_report(y_test, y_pred, labels=[0,1]))

                  precision    recall  f1-score   support

               0       0.91      1.00      0.95       812
               1       0.81      0.14      0.24        93

        accuracy                           0.91       905
       macro avg       0.86      0.57      0.59       905
    weighted avg       0.90      0.91      0.88       905
```



```
[ ] sm = SMOTEENN()
    X_resampled1, y_resampled1 = sm.fit_resample(x,y)

[ ] xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test_size=0.2)

[ ] model_rf_smote=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)

[ ] model_rf_smote.fit(xr_train1,yr_train1)

    RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)

[ ] yr_predict1 = model_rf_smote.predict(xr_test1)

[ ] model_score_r1 = model_rf_smote.score(xr_test1, yr_test1)

[ ] print(model_score_r1)
    print(metrics.classification_report(yr_test1, yr_predict1))

    0.9451013513513513
                  precision    recall  f1-score   support

               0       0.94      0.94      0.94       557
               1       0.95      0.95      0.95       627

        accuracy                           0.95      1184
       macro avg       0.94      0.94      0.94      1184
    weighted avg       0.95      0.95      0.95      1184
```


With RF Classifier, also we are able to get quite good results, infact better than Decision Tree ad after up sampling.

- **Support Vector Classifier**

Support Vector Classifier

```
[ ] from sklearn.svm import SVC

[ ] model_sv=SVC()

[ ] model_sv.fit(x_train,y_train)

    SVC()

[ ] y_pred=model_sv.predict(x_test)

[ ] model_sv.score(x_test,y_test)

    0.8972375690607735

[ ] print(classification_report(y_test, y_pred, labels=[0,1]))

                  precision    recall  f1-score   support

               0       0.90      1.00      0.95       812
               1       0.00      0.00      0.00        93

        accuracy                           0.90       905
       macro avg       0.45      0.50      0.47       905
    weighted avg       0.81      0.90      0.85       905
```

```
[ ] sm = SMOTEENN()
    X_resampled2, y_resampled2 = sm.fit_resample(x,y)

[ ] xr_train2,xr_test2,yr_train2,yr_test2=train_test_split(X_resampled2, y_resampled2,test_size=0.2)

[ ] model_sv_smote=SVC()

[ ] model_sv_smote.fit(xr_train2,yr_train2)

    SVC()

[ ] yr_predict2 = model_sv_smote.predict(xr_test2)

[ ] model_score_r2 = model_sv_smote.score(xr_test2, yr_test2)

[ ] print(model_score_r2)
    print(metrics.classification_report(yr_test2, yr_predict2))

    0.8540084388185654
                  precision    recall  f1-score   support

               0       0.82      0.88      0.85       550
               1       0.89      0.83      0.86       635

        accuracy                           0.85      1185
       macro avg       0.85      0.86      0.85      1185
    weighted avg       0.86      0.85      0.85      1185
```

With SVM, we couldn't see any better results, hence finalise the model which was created by Random Forest Classifier

Model	Model accuracy score	Model accuracy score After upsampling
Decision Tree Classifier	0.906	0.922
Random Forest Classifier	0.908	0.945
Support Vector Classifier	0.897	0.854

Result

The approach used here as follows:

Firstly, we did data preprocessing followed by EDA (exploratory data analysis) where we did univariate and bivariate analysis.

The main insights we got are as

- The customer with secondary education are high churners.
- Those who were contacted last in May were high Churners
- Marital status as married are more among churners.
- customer not having credit in default are high churner

Finally, headed towards experimenting with different machine learning models to predict for the testing data whether client subscribed or churn.

After testing multiple classifiers with and without upsampling. Since the data was highly imbalanced we applied upsampling to improve the accuracy rate. Clearly we can see from the above table Random Forest Classifier performed best with an accuracy score of 0.945 and hence the most suitable model among the other classifiers we experimented with.

```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv("/content/bank.csv")
data
```

	age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"prev
0	30;"unemployed";"marrie
1	33;"services";"married";"
2	35;"management";"sing
3	30;"management";"marri
4	59;"blue-collar";"marri
...	
4516	33;"services";"married";
4517	57;"self-employed";"n
4518	57;"technician";"married
4519	28;"blue-collar";"marrie
4520	44;"entrepreneur";"si

4521 rows × 1 columns

```
# creating list of column names in colmns variable
colmns = re.sub(";", " ", data.columns[0])
colmns = re.sub("\"", "", colmns)
colmns = colmns.split()

# cleaning each row of data
def clean_data(row):
    row = re.sub(";", " ", row)
    row = re.sub("\"", "", row)
    row = row.split()
    return row

data1 = data.copy()
data1.iloc[:,0] = data1.iloc[:,0].map(lambda x: clean_data(x))
data1
```

	age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"prev
0	[30, unemployed, marrie

```
# creating columns from "columns" list which we created above and filling values from row lists
idx = 0
for row in data1.iloc[:,0]:
    if len(row) == 17:
        i = 0
        for col in colmns:
            data1.loc[idx,col] = row[i]
            i += 1
    idx += 1

data1.drop(data1.columns[0], axis=1, inplace=True)
df = data1.copy()
df
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	po
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	ui
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	ui
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	ui
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	jul	329	5	-1	0	ui
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	may	153	1	-1	0	ui
4518	57	technician	married	secondary	no	295	no	no	cellular	19	aug	151	11	-1	0	ui
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	feb	129	4	211	3	
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	apr	345	2	249	7	

4521 rows × 17 columns

```
# converting datatypes of numerical varaibles which were object dtypes
convert_dtype = {"age":int, "balance":int, "day":int, "duration":int, "campaign":int, "pdays":int, "previous":int }
df = df.astype(convert_dtype)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        4521 non-null   int64
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64
 12  campaign   4521 non-null   int64
 13  pdays      4521 non-null   int64
 14  previous   4521 non-null   int64
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

```
df.head()
```

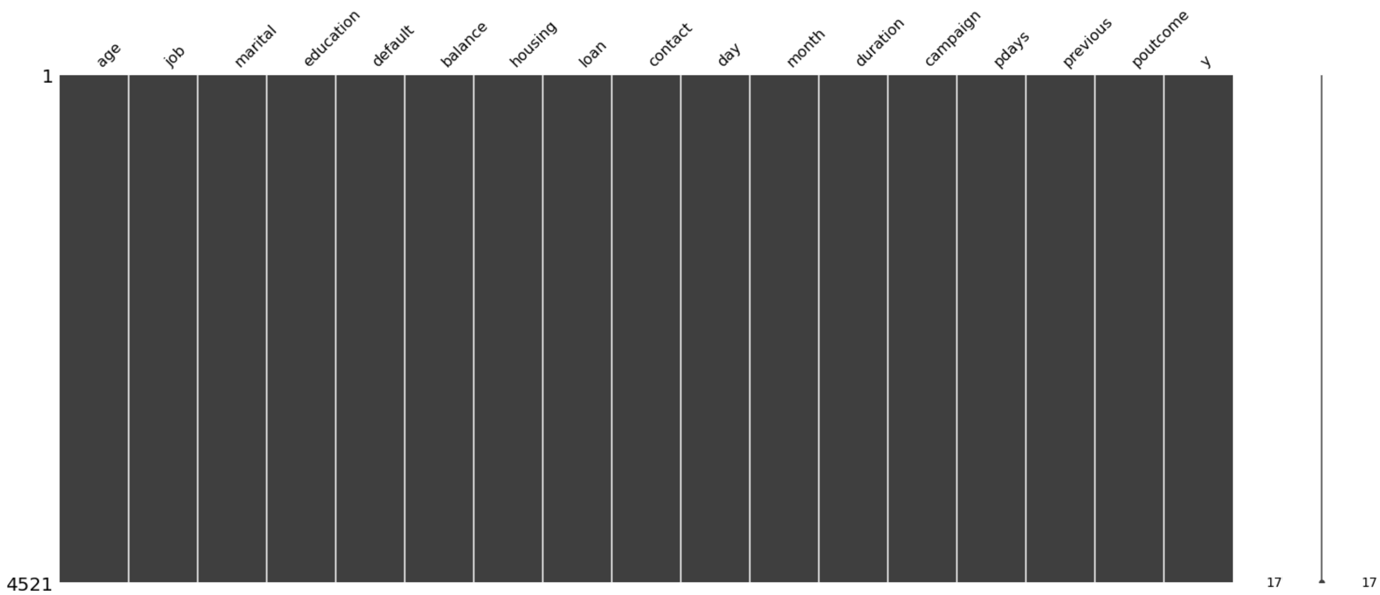
	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutc
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unkn
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	fai
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	fai
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unkn
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unkn

The classification goal is to predict if the client will subscribe a term deposit (variable y).

```
import missingno as msno
```

```
msno.matrix(df)
```

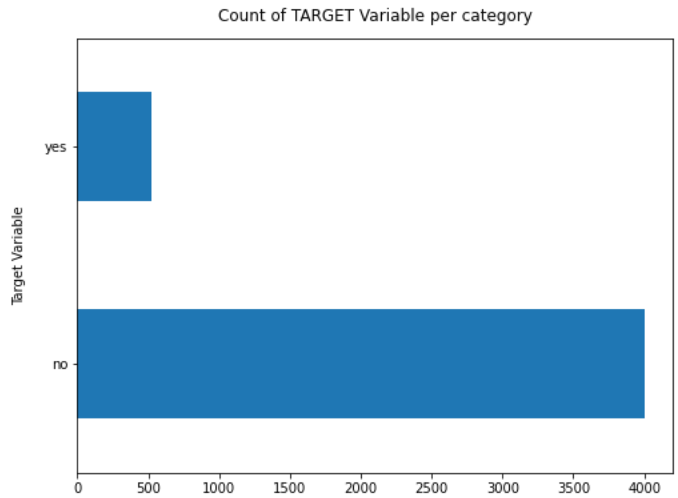
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f349eacd1c0>
```




```
# Check the descriptive statistics of numeric variables
df.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000

```
df['y'].value_counts().plot(kind='barh', figsize=(8, 6))
plt.xlabel("Count", labelpad=14)
plt.ylabel("Target Variable", labelpad=14)
plt.title("Count of TARGET Variable per category", y=1.02);
```




```
100*df['y'].value_counts()/len(df['y'])
```

```
no     88.476001
yes    11.523999
Name: y, dtype: float64
```

```
df['y'].value_counts()
```

```
no     4000
yes     521
Name: y, dtype: int64
```

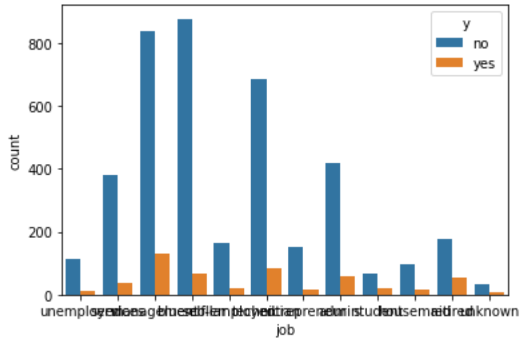
Data is highly imbalanced, ratio = 88:11 So we analyse the data with other features while taking the target values separately to get some insights.

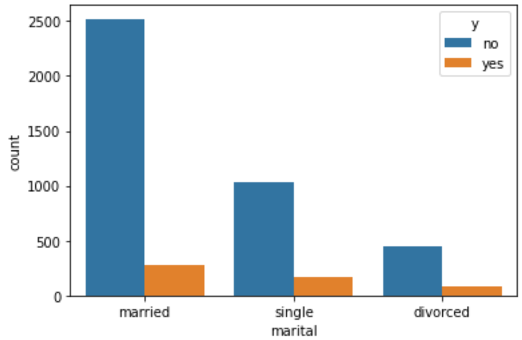
Data Exploration

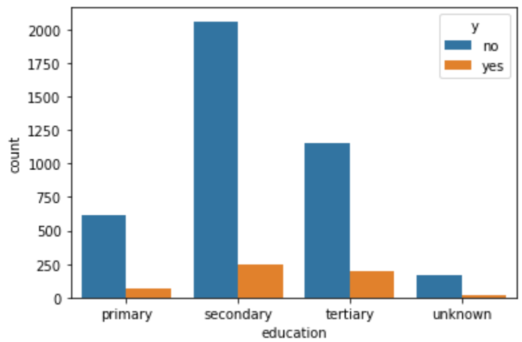
1. Plot distibution of individual predictors by target variables

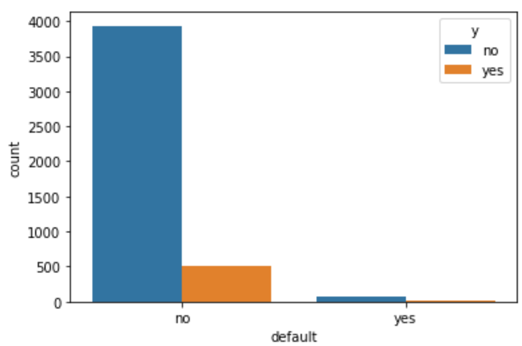
Univariate Analysis

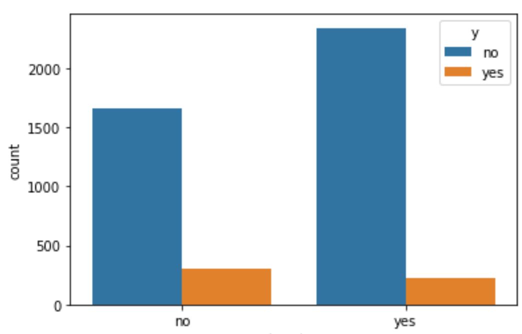
```
for i, predictor in enumerate(df.drop(columns=['y','age', 'balance','day' ,'duration','campaign','pdays','previous'])):
    plt.figure(i)
    sns.countplot(data=df, x=predictor, hue='y')
```


count
job
y
no
yes
0
200
400
600
800
unknown


count
marital
married
single
divorced
y
no
yes
0
500
1000
1500
2000
2500


count
education
primary
secondary
tertiary
unknown
y
no
yes
0
250
500
750
1000
1250
1500
1750
2000


count
default
no
yes
y
no
yes
0
500
1000
1500
2000
2500
3000
3500
4000

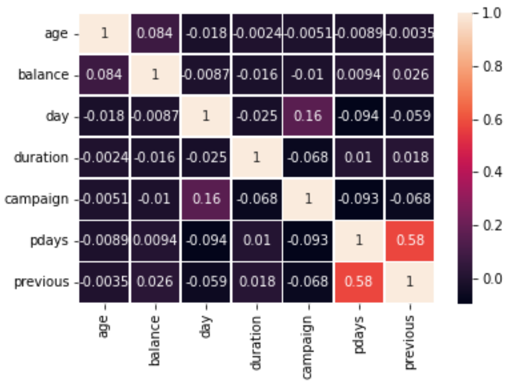

count
housing
no
yes
y
no
yes
0
500
1000
1500
2000


count
y
no
yes
1500
2000
2500
3000
3500

```
#segregating numerical columns
numerical_f=df.drop(['y'],axis=1).select_dtypes(include=np.number).columns
```

```
corr=df[numerical_f].corr()
sns.heatmap(corr, annot=True,linewidths=.8)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f349b5fae50>
```



the featues 'pdays' and 'previous' are moderatily collinear. since correlation for all variables is < 0.7 hence no 2 columns are multicollinear

Convert the target variable 'y' in a binary numeric variable i.e. Yes=1 ; No = 0

```
df['y'] = np.where(df.y =='yes',1,0)
```

Convert all the categorical variables into dummy variables

```
data_dummies = pd.get_dummies(df)
data_dummies.head(10)
```

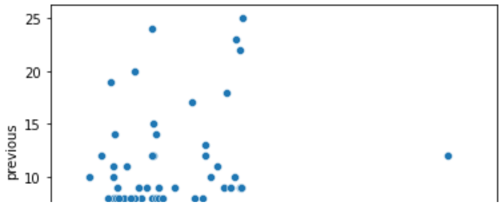
	age	balance	day	duration	campaign	pdays	previous	y	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	mon
0	30	1787	19	79	1	-1	0	0	0	0	...	0	0	0	0	
1	33	4789	11	220	1	339	4	0	0	0	...	0	0	1	0	
2	35	1350	16	185	1	330	1	0	0	0	...	0	0	0	0	
3	30	1476	3	199	4	-1	0	0	0	0	...	1	0	0	0	
4	59	0	5	226	1	-1	0	0	0	1	...	0	0	1	0	
5	35	747	23	141	2	176	3	0	0	0	...	0	0	0	0	
6	36	307	14	341	1	330	2	0	0	0	...	0	0	1	0	
7	39	147	6	151	2	-1	0	0	0	0	...	0	0	1	0	
8	41	221	14	57	2	-1	0	0	0	0	...	0	0	1	0	
9	43	-88	17	313	1	147	2	0	0	0	...	0	0	0	0	

10 rows × 52 columns

Relationship between Pdays and Previous

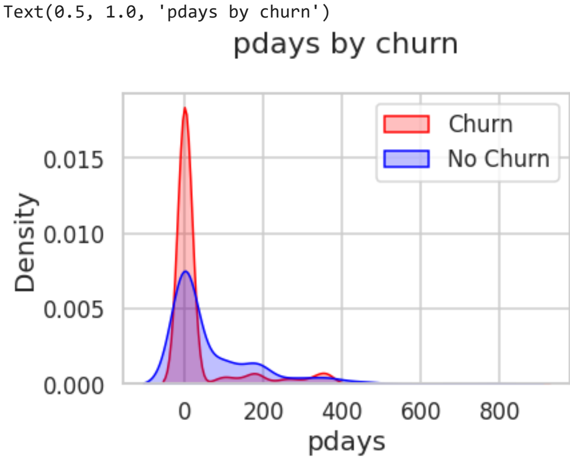
```
sns.scatterplot(data=data_dummies, x='pdays', y='previous')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f349ea8e880>



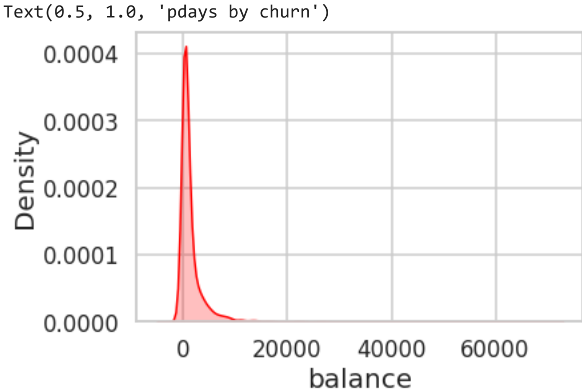

```
Mth = sns.kdeplot(data_dummies.pdays[(data_dummies["y"] == 0) ],
                color="Red", shade = True)
Mth = sns.kdeplot(data_dummies.pdays[(data_dummies["y"] == 1) ],
                ax =Mth, color="Blue", shade= True)
Mth.legend(["Churn","No Churn"],loc='upper right')
Mth.set_ylabel('Density')
Mth.set_xlabel('pdays')
Mth.set_title('pdays by churn')
```

Text(0.5, 1.0, 'pdays by churn')



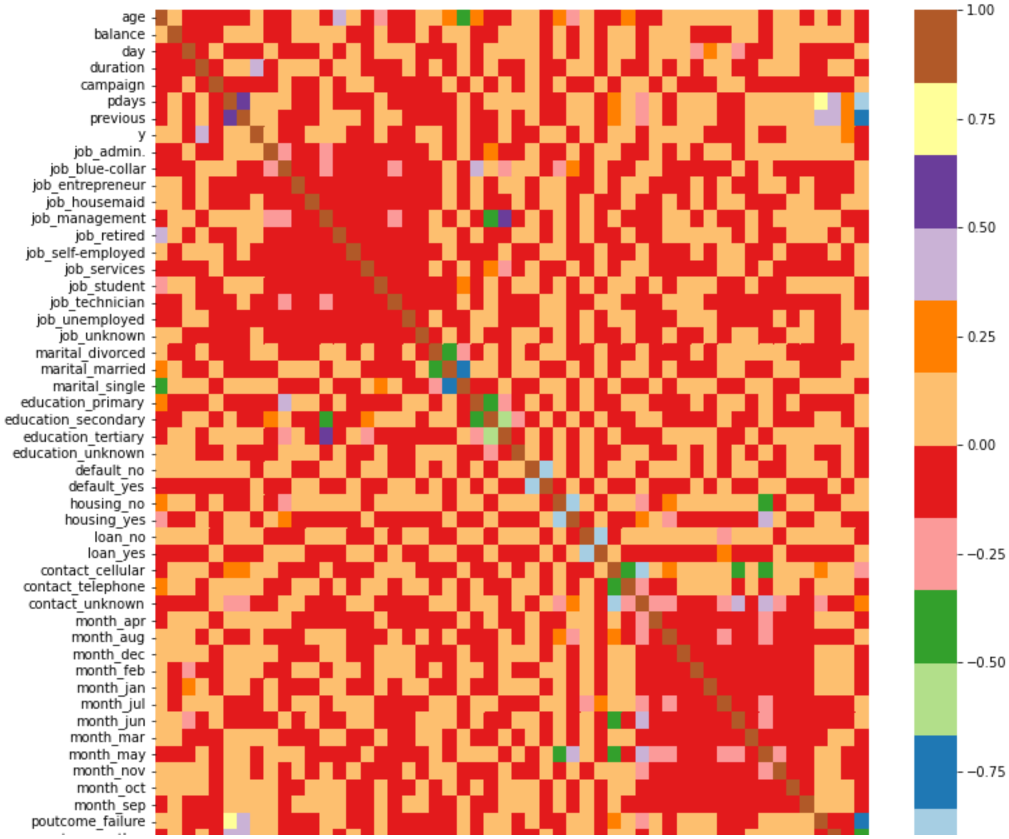

```
tot = sns.kdeplot(data_dummies.balance[(data_dummies["y"] == 0) ],
                color="Red", shade = True)
tot = sns.kdeplot(data_dummies.balance[(data_dummies["y"] == 1) ],
                ax =Mth, color="Blue", shade= True)
tot.legend(["Churn","No Churn"],loc='upper right')
tot.set_ylabel('Density')
tot.set_xlabel('duration')
tot.set_title('pdays by churn')
```

Text(0.5, 1.0, 'pdays by churn')




```
plt.figure(figsize=(12,12))
sns.heatmap(data_dummies.corr(), cmap="Paired")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3499362250>
```



by looking at the given heatmat we can say that poutcome_failure and pdays are highly correlated we will drop the poutcome_failure feature

```
# Remove column name ' poutcome_failure'
data_dummies=data_dummies.drop(['poutcome_failure'], axis=1)
```

```
data_dummies=data_dummies.drop(['day'], axis=1)
```

```
data_dummies.info()
```

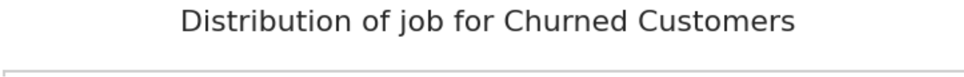
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 50 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  4521 non-null   int64
 1   balance              4521 non-null   int64
 2   duration             4521 non-null   int64
 3   campaign             4521 non-null   int64
 4   pdays                4521 non-null   int64
 5   previous             4521 non-null   int64
 6   y                    4521 non-null   int64
 7   job_admin.           4521 non-null   uint8
 8   job_blue-collar      4521 non-null   uint8
 9   job_entrepreneur     4521 non-null   uint8
 10  job_housemaid        4521 non-null   uint8
 11  job_management       4521 non-null   uint8
 12  job_retired          4521 non-null   uint8
 13  job_self-employed    4521 non-null   uint8
 14  job_services         4521 non-null   uint8
 15  job_student          4521 non-null   uint8
 16  job_technician       4521 non-null   uint8
 17  job_unemployed       4521 non-null   uint8
 18  job_unknown          4521 non-null   uint8
 19  marital_divorced     4521 non-null   uint8
 20  marital_married      4521 non-null   uint8
 21  marital_single       4521 non-null   uint8
 22  education_primary    4521 non-null   uint8
 23  education_secondary  4521 non-null   uint8
 24  education_tertiary   4521 non-null   uint8
 25  education_unknown    4521 non-null   uint8
 26  default_no           4521 non-null   uint8
 27  default_yes          4521 non-null   uint8
```

```
 28  housing_no           4521 non-null   uint8
 29  housing_yes          4521 non-null   uint8
 30  loan_no              4521 non-null   uint8
 31  loan_yes             4521 non-null   uint8
 32  contact_cellular     4521 non-null   uint8
 33  contact_telephone    4521 non-null   uint8
 34  contact_unknown      4521 non-null   uint8
 35  month_apr            4521 non-null   uint8
 36  month_aug            4521 non-null   uint8
 37  month_dec            4521 non-null   uint8
 38  month_feb            4521 non-null   uint8
 39  month_jan            4521 non-null   uint8
 40  month_jul            4521 non-null   uint8
 41  month_jun            4521 non-null   uint8
 42  month_mar            4521 non-null   uint8
 43  month_may            4521 non-null   uint8
 44  month_nov            4521 non-null   uint8
 45  month_oct            4521 non-null   uint8
 46  month_sep            4521 non-null   uint8
 47  poutcome_other       4521 non-null   uint8
 48  poutcome_success     4521 non-null   uint8
 49  poutcome_unknown     4521 non-null   uint8
dtypes: int64(7), uint8(43)
memory usage: 437.2 KB
```

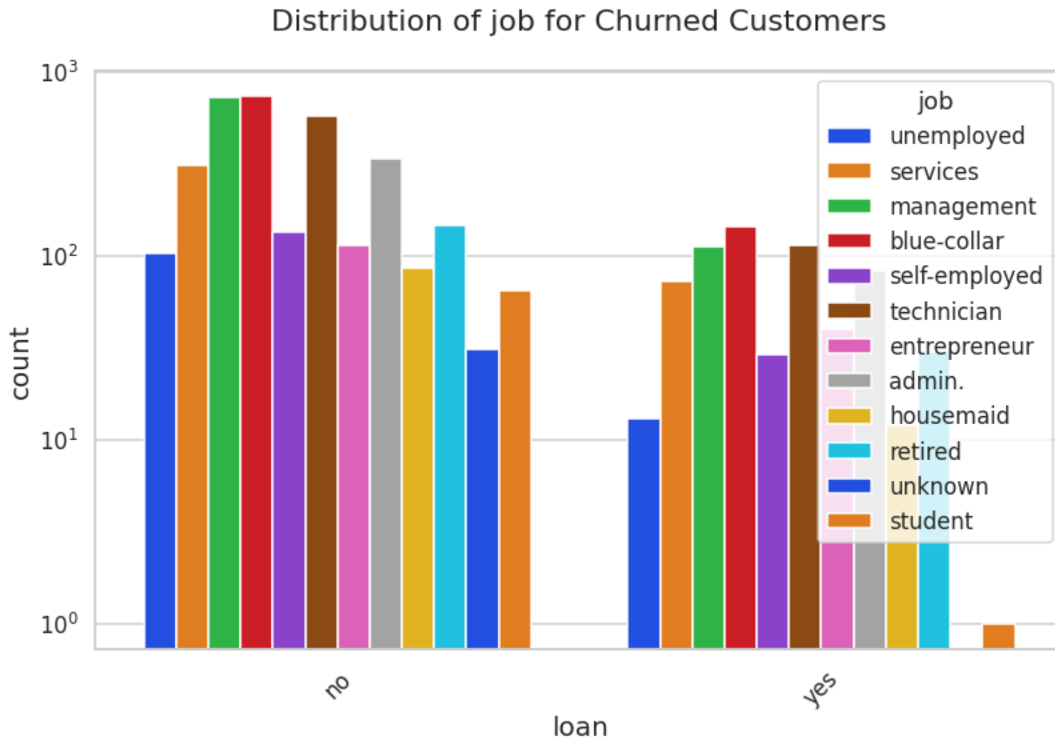
```
new_df1_target0=df.loc[df["y"]==0]
new_df1_target1=df.loc[df["y"]==1]
```

```
def uniplot(df,col,title,hue =None):

    sns.set_style('whitegrid')
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 20
    plt.rcParams['axes.titlesize'] = 22
    plt.rcParams['axes.titlepad'] = 30


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(df[col].unique()) + 7 + 4*len(temp.unique())
    fig.set_size_inches(width , 8)
    plt.xticks(rotation=45)
    plt.yscale('log')
    plt.title(title)
    ax = sns.countplot(data = df, x= col, order=df[col].value_counts().index,hue = hue,palette='bright')

    plt.show()
```

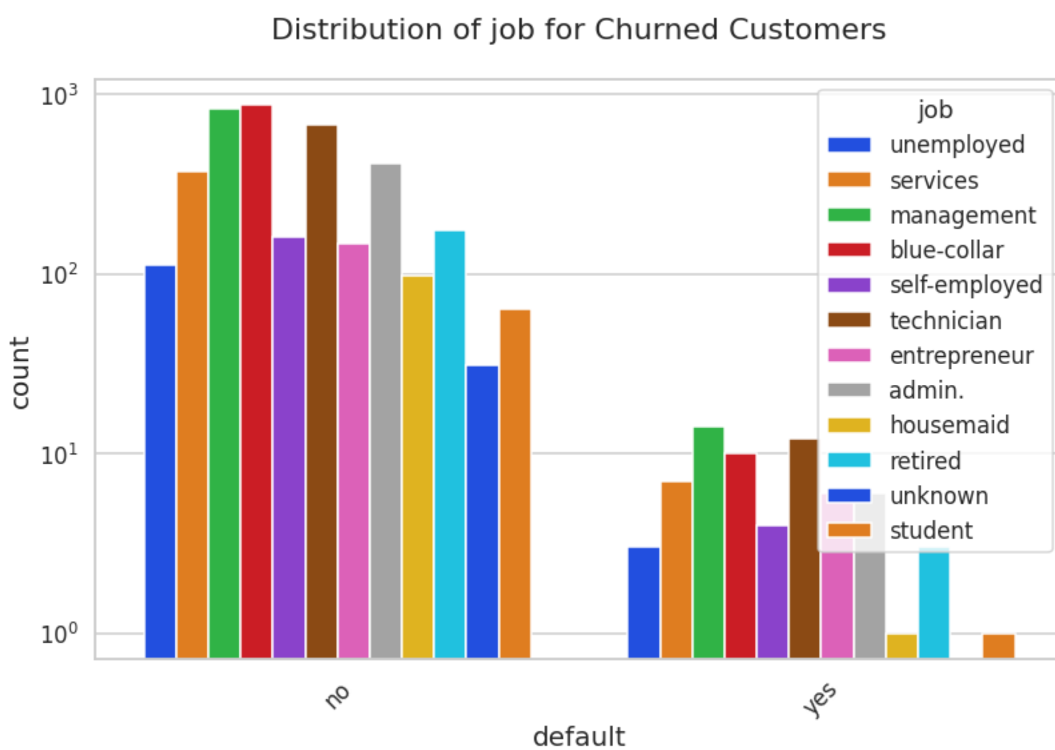
```
uniplot(new_df1_target0,col='housing',title='Distribution of job for Churned Customers',hue='job')
```



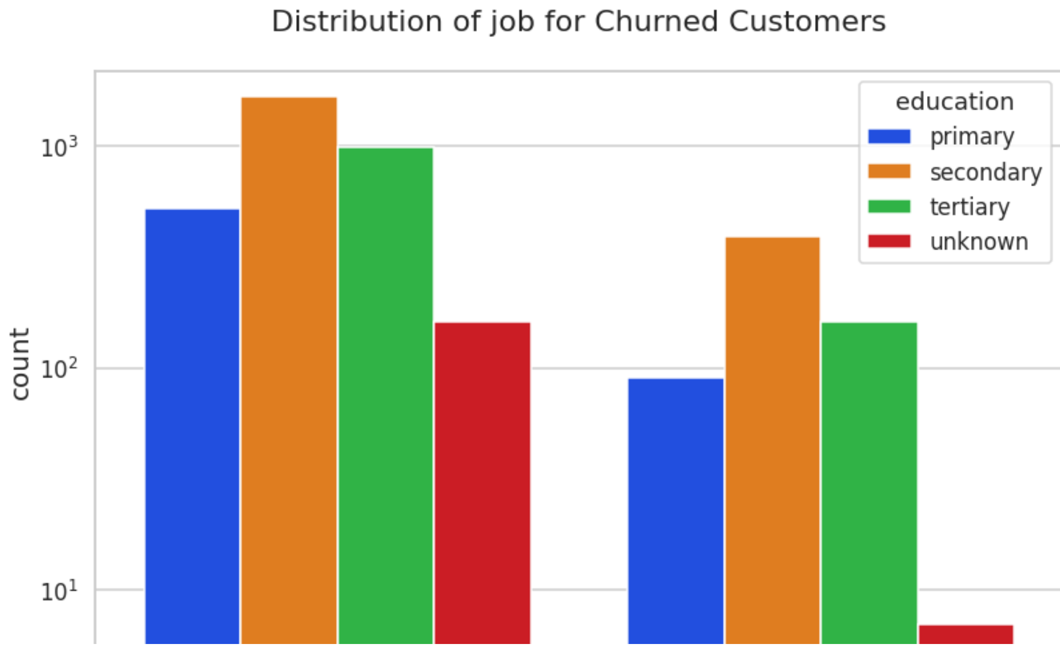

```
uniplot(new_df1_target0,col='loan',title='Distribution of job for Churned Customers',hue='job')
```



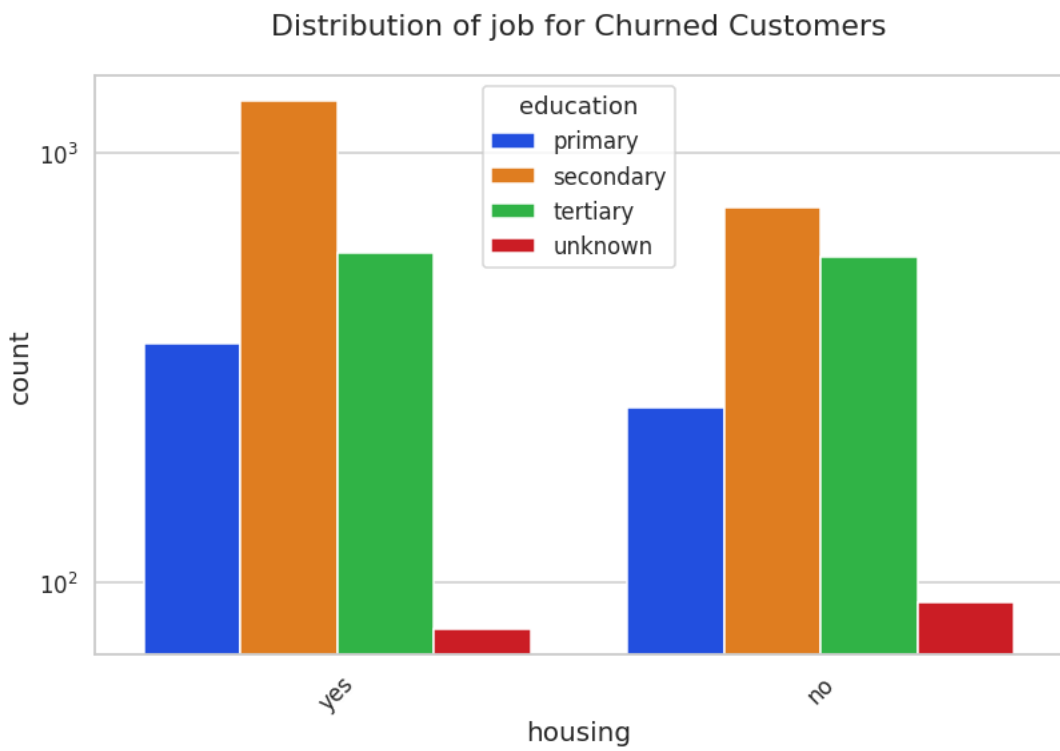

```
uniplot(new_df1_target0,col='default',title='Distribution of job for Churned Customers',hue='job')
```



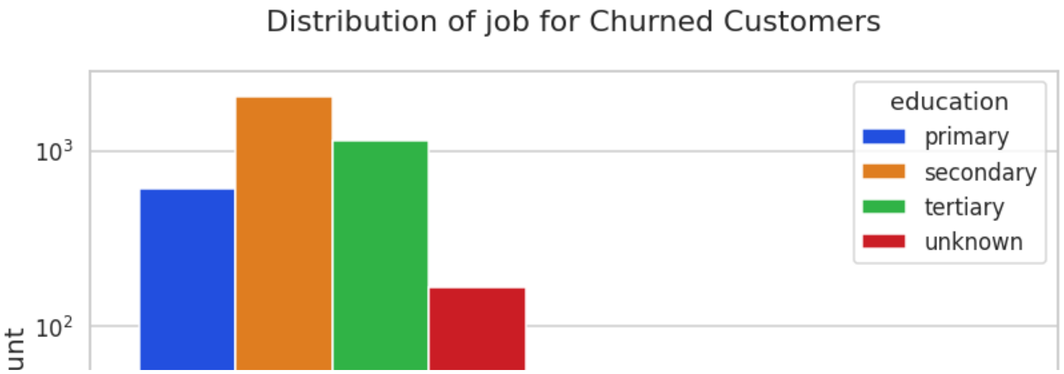

```
uniplot(new_df1_target0,col='loan',title='Distribution of job for Churned Customers',hue='education')
```




```
uniplot(new_df1_target0,col='housing',title='Distribution of job for Churned Customers',hue='education')
```

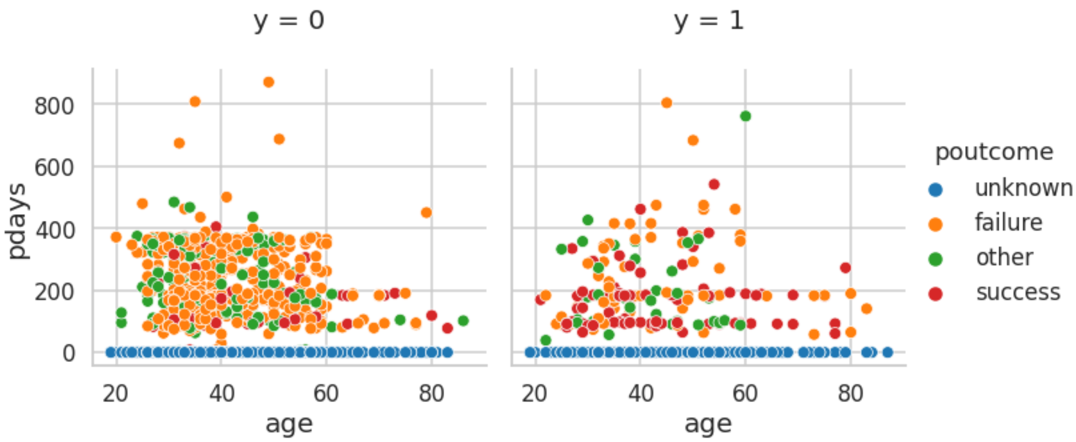



```
uniplot(new_df1_target0,col='default',title='Distribution of job for Churned Customers',hue='education')
```



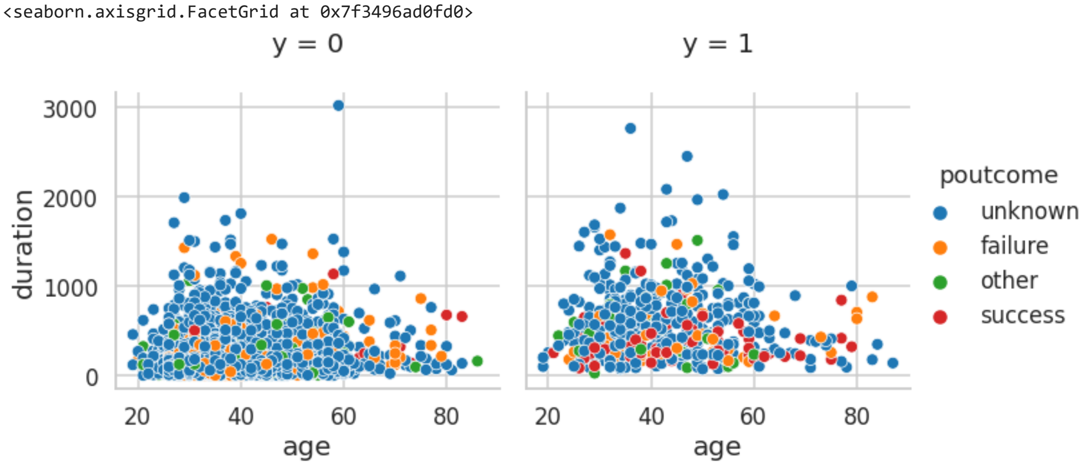

```
sns.relplot(data = df, x="age", y="pdays",hue= 'poutcome', col = 'y')
```

<seaborn.axisgrid.FacetGrid at 0x7f349b641070>




```
sns.relplot(data = df, x="age", y="duration",hue= 'poutcome', col = 'y')
```

<seaborn.axisgrid.FacetGrid at 0x7f3496ad0fd0>




```
data_dummies.to_csv('data.csv')
```

```
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
```

```
df=pd.read_csv("/content/data (1).csv")
df.head()
```

	Unnamed: 0	age	balance	duration	campaign	pdays	previous	y	job_adm
0	0	30	1787	79	1	-1	0	0	
1	1	33	4789	220	1	339	4	0	
2	2	35	1350	185	1	330	1	0	
3	3	30	1476	199	4	-1	0	0	
4	4	59	0	226	1	-1	0	0	

5 rows × 51 columns

```
df=df.drop(['Unnamed: 0'], axis=1)
```

```
df.head()
```

	age	balance	duration	campaign	pdays	previous	y	job_admin.	job_b co
0	30	1787	79	1	-1	0	0	0	
1	33	4789	220	1	339	4	0	0	
2	35	1350	185	1	330	1	0	0	
3	30	1476	199	4	-1	0	0	0	
4	59	0	226	1	-1	0	0	0	

5 rows × 50 columns

```
x=df.drop('y',axis=1)
x
```

	age	balance	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	...	month_jul	month_ju
0	30	1787	79	1	-1	0	0	0	0	0	...	0	
1	33	4789	220	1	339	4	0	0	0	0	...	0	

```
y=df['y']
y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
4516    0
4517    0
4518    0
4519    0
4520    0
Name: y, Length: 4521, dtype: int64
```

4521 rows × 49 columns

Train Test Split

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

Decision Tree Classifier

```
model_dt=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
model_dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

```
y_pred=model_dt.predict(x_test)
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0])
```

```
model_dt.score(x_test,y_test)
```

```
0.9060773480662984
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       812
           1       0.58      0.32      0.41        93

    accuracy                           0.91       905
   macro avg       0.75      0.65      0.68       905
weighted avg       0.89      0.91      0.89       905
```

```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(x,y)
```

```
xr_train,xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_size=0.2)
```

```
model_dt_smote=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
model_dt_smote.fit(xr_train,yr_train)
yr_predict = model_dt_smote.predict(xr_test)
model_score_r = model_dt_smote.score(xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
0.9226240538267452
              precision    recall  f1-score   support

           0       0.92      0.92      0.92       561
           1       0.93      0.93      0.93       628

    accuracy                           0.92      1189
   macro avg       0.92      0.92      0.92      1189
weighted avg       0.92      0.92      0.92      1189
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
model_rf=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
model_rf.fit(x_train,y_train)
```

```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

```
y_pred=model_rf.predict(x_test)
```

```
model_rf.score(x_test,y_test)
```

```
0.9082872928176795
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95       812
           1       0.81      0.14      0.24        93

    accuracy                           0.91       905
   macro avg       0.86      0.57      0.59       905
```

```
weighted avg       0.90      0.91      0.88       905
```

```
sm = SMOTEENN()
X_resampled1, y_resampled1 = sm.fit_resample(x,y)
```

```
xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test_size=0.2)
```

```
model_rf_smote=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6, min_samples_leaf=8)
```

```
model_rf_smote.fit(xr_train1,yr_train1)
```

```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

```
yr_predict1 = model_rf_smote.predict(xr_test1)
```

```
model_score_r1 = model_rf_smote.score(xr_test1, yr_test1)
```

```
print(model_score_r1)
print(metrics.classification_report(yr_test1, yr_predict1))
```

```
0.9451013513513513
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       557
           1       0.95      0.95      0.95       627

    accuracy                           0.95      1184
   macro avg       0.94      0.94      0.94      1184
weighted avg       0.95      0.95      0.95      1184
```

Support Vector Classifier

```
from sklearn.svm import SVC
```

```
model_sv=SVC()
```

```
model_sv.fit(x_train,y_train)
```

```
SVC()
```

```
y_pred=model_sv.predict(x_test)
```

```
model_sv.score(x_test,y_test)
```

```
0.8972375690607735
```

```
print(classification_report(y_test, y_pred, labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       812
           1       0.00      0.00      0.00        93

    accuracy                           0.90       905
   macro avg       0.45      0.50      0.47       905
weighted avg       0.81      0.90      0.85       905

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-d
  _warn_prf(average, modifier, msg_start, len(result))
```

```
sm = SMOTEENN()
X_resampled2, y_resampled2 = sm.fit_resample(x,y)
```

```
xr_train2,xr_test2,yr_train2,yr_test2=train_test_split(X_resampled2, y_resampled2,test_size=0.2)
```

```
model_sv_smote=SVC()
```

```
model_sv_smote.fit(xr_train2,yr_train2)
```

```
SVC()
```

```
yr_predict2 = model_sv_smote.predict(xr_test2)
```

```
model_score_r2 = model_sv_smote.score(xr_test2, yr_test2)
```

```
print(model_score_r2)
print(metrics.classification_report(yr_test2, yr_predict2))
```

```
0.8540084388185654
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       550
           1       0.89      0.83      0.86       635

    accuracy                           0.85      1185
   macro avg       0.85      0.86      0.85      1185
weighted avg       0.86      0.85      0.85      1185
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.