Bank Customer Churn Analysis

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.

Input variables:

Bank client data: 1 - age (numeric) 2 - job : type of job (categorical: "admin.","unknown","unemployed","management","housemaid","entrepreneur","student", "blue-collar","self-employed","retired","technician","services")   3 - marital : marital status (categorical: "married","divorced","single"; note: "divorced" means divorced or widowed)   4 - education (categorical: "unknown","secondary","primary","tertiary")   5 - default: has credit in default? (binary: "yes","no")   6 - balance: average yearly balance, in euros (numeric)   7 - housing: has housing loan? (binary: "yes","no")   8 - loan: has personal loan? (binary: "yes","no")   ### Related with the last contact of the current campaign: 9 - contact: contact communication type (categorical: "unknown","telephone","cellular")   10 - day: last contact day of the month (numeric)   11 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")   12 - duration: last contact duration, in seconds (numeric)   ### Other attributes: 13 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)   14 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted) 15 - previous: number of contacts performed before this campaign and for this client (numeric)   16 - poutcome: outcome of the previous marketing campaign (categorical: "unknown","other","failure","success")   ## Output variable (desired target): 17 - y - has the client subscribed a term deposit? (binary: "yes","no")

Preproccesing

Converting the data into a pandas dataset

```
import pandas as pd
import numpy as np
df = pd.read_csv("bank.csv")
column_names = np.array(df.columns)
column_names = [item.replace('"', '') for item in
column_names[0].split(';')]

df = pd.read_csv("bank.csv", delimiter=';', header=None,
names=column_names,skiprows=[0])

df.head()
```

```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no
```

```
1   33     services  married  secondary      no     4789     yes  yes

2   35   management   single   tertiary      no     1350     yes   no

3   30   management  married   tertiary      no     1476     yes  yes

4   59  blue-collar  married  secondary      no        0     yes   no


    contact  day month  duration  campaign  pdays  previous poutcome
y
0  cellular   19   oct        79         1     -1         0  unknown
no
1  cellular   11   may       220         1    339         4  failure
no
2  cellular   16   apr       185         1    330         1  failure
no
3   unknown    3   jun       199         4     -1         0  unknown
no
4   unknown    5   may       226         1     -1         0  unknown
no
```

```
df.dtypes
```

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

Data Cleaning

```
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
```

We can see we have no missing values in the dataset

```
#Checking for duplicate entries in the data
df.duplicated().sum()
```

```
0
```

Our data has no duplicate values. Thats great

Exploratory Data Analysis

```
df[['day','month','pdays']].head(5)
```

```
   day month  pdays
0   19   oct     -1
1   11   may    339
2   16   apr    330
3    3   jun     -1
4    5   may     -1
```

pdays has the information for the last contact, rendering the day and month columns useless. So, we do not need these.

```
categorical_features=[feature for feature in df.columns if 
df[feature].dtype=='O' and feature not in['y','month']]

numerical_features = [feature for feature in df.columns if 
df[feature].dtype=='int64' and feature!=['day']]

print(f'Categorical columns : {categorical_features}')
print(f'Numerical columns : {numerical_features}')
```

```
Categorical columns : ['job', 'marital', 'education', 'default',
'housing', 'loan', 'contact', 'poutcome']
Numerical columns : ['age', 'balance', 'day', 'duration', 'campaign',
'pdays', 'previous']
```

Exploring categorical features

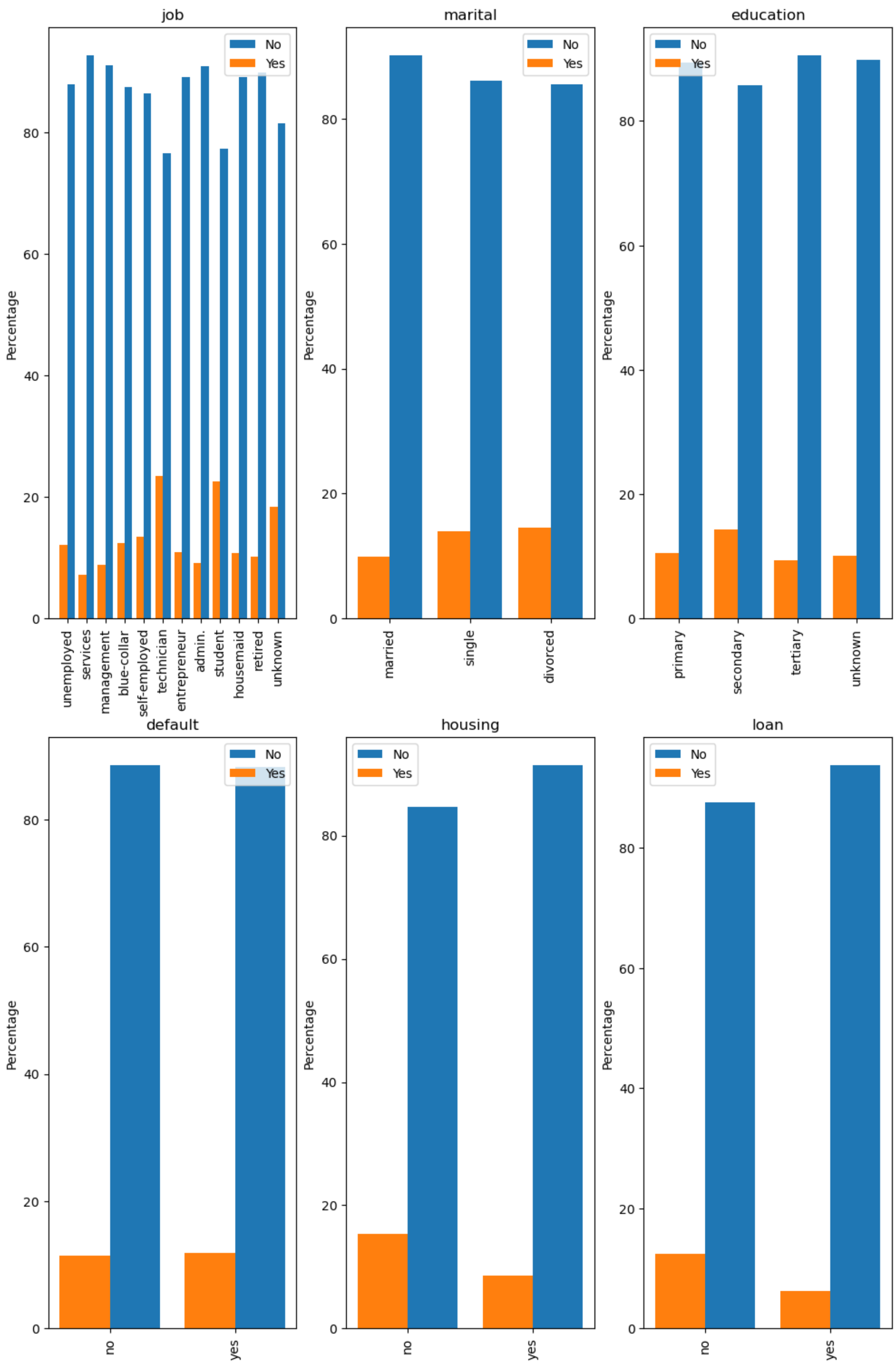
```
for feature in categorical_features:
    print(f'Number of categories in {feature} :
{df[feature].nunique()}')
```

```
Number of categories in job : 12
Number of categories in marital : 3
Number of categories in education : 4
Number of categories in default : 2
Number of categories in housing : 2
Number of categories in loan : 2
Number of categories in contact : 3
Number of categories in poutcome : 4
```

We see that job has the highest number of categories in it

Categorical feature exploration

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,100))
plotnum = 1

for feature in categorical_features:
    ax1 = plt.subplot(10,3, plotnum)
    x = np.arange(df[feature].nunique())
    feature_yes = df[df['y']=='yes'][feature].value_counts() /
df[feature].value_counts() * 100
    feature_no = 100 - feature_yes
    width = 0.40
    ax1.bar(x + width/2, feature_no, width, label='No')
    ax1.bar(x - width/2, feature_yes, width, label='Yes')
    ax1.set_xticks(x)
    ax1.set_xticklabels(df[feature].unique(), rotation=90)
    ax1.set_ylabel('Percentage')
    ax1.set_title(feature)
    ax1.legend()
    plotnum+=1

plt.show()
```


job
marital
education
default
housing
loan
No
Yes
Percentage
unemployed
services
management
blue-collar
self-employed
technician
entrepreneur
admin.
student
housemaid
retired
unknown
married
single
divorced
primary
secondary
tertiary
no
yes

- Higher percentage of technicians and students tend to opt in
- The default feature's value doesnt seem to affect the outcome much
- People with loans dont often opt in as much as ones without them
- People are much more interested when last contacted in March, April, July and December
- When the previous campaign's outcome is other, people tend to show much more interest
- There is no difference in the outcome when people are eached by cellular or telephone

Numerical feature exploration

Finding discrete numerical features

```
discrete_features = [feature for feature in numerical_features if
df[feature].nunique()<25]
print(f"Discrete features:{discrete_features}")
```

```
Discrete features:['previous']
```

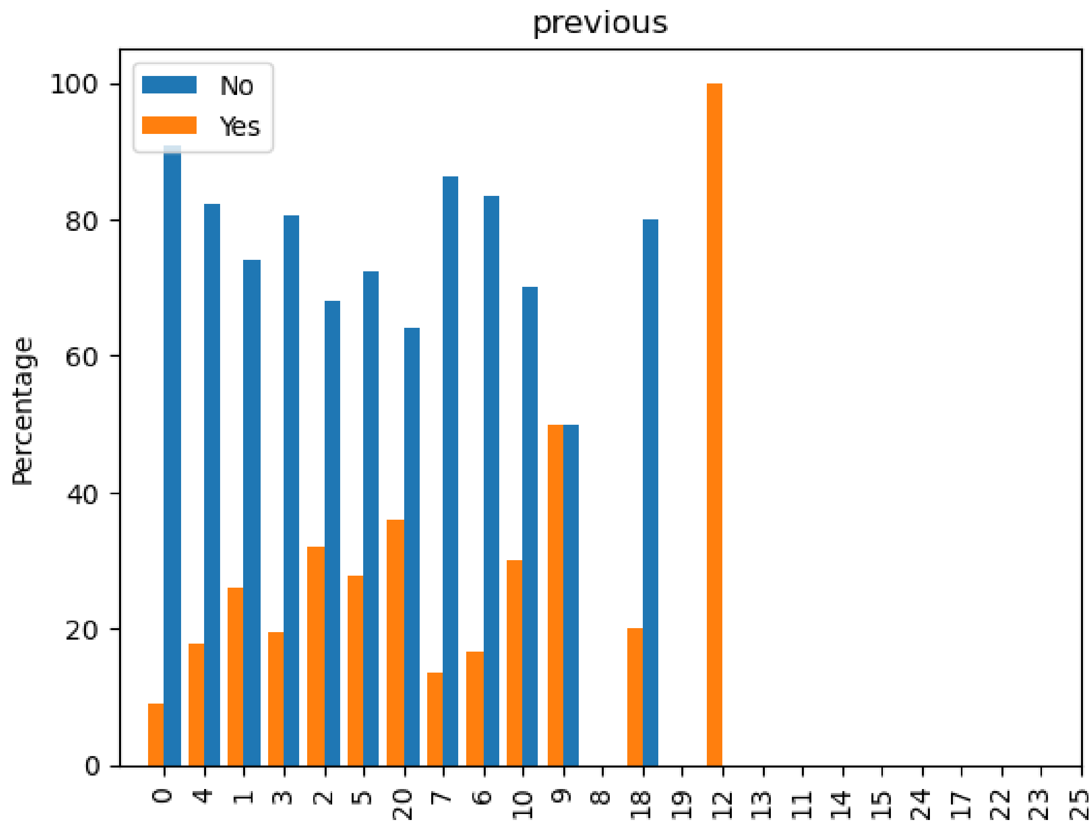
```
df["previous"].unique()
```

```
array([ 0,  4,  1,  3,  2,  5, 20,  7,  6, 10,  9,  8, 18, 19, 12, 13,
11,
       14, 15, 24, 17, 22, 23, 25], dtype=int64)
```

We can add this to the list of categorical features. Lets compare this feature's relationship with the outcome

```
feature = discrete_features[0]
ax1 = plt.subplot()
x = np.arange(df[feature].nunique())
feature_yes = df[df['y']=='yes'][feature].value_counts() /
df[feature].value_counts() * 100
feature_no = 100 - feature_yes
width = 0.40
ax1.bar(x + width/2, feature_no, width, label='No')
ax1.bar(x - width/2, feature_yes, width, label='Yes')
ax1.set_xticks(x)
ax1.set_xticklabels(df[feature].unique(), rotation=90)
ax1.set_ylabel('Percentage')
ax1.set_title(feature)
ax1.legend()
```

```
<matplotlib.legend.Legend at 0x159a1190c70>
```




```
df.groupby(['y','previous']).size()

y     previous
no    0           3368
      1            235
      2            143
      3             91
      4             53
      5             34
      6             16
      7             19
      8             15
      9              7
      10             2
      11             3
      12             4
      13             1
      15             1
      17             1
      18             1
      19             1
      20             1
      22             1
      23             1
      24             1
```

```
     25             1
yes  0            337
     1             51
     2             50
     3             22
     4             25
     5             13
     6              9
     7              3
     8              3
     9              3
     10             2
     12             1
     14             2
dtype: int64
```

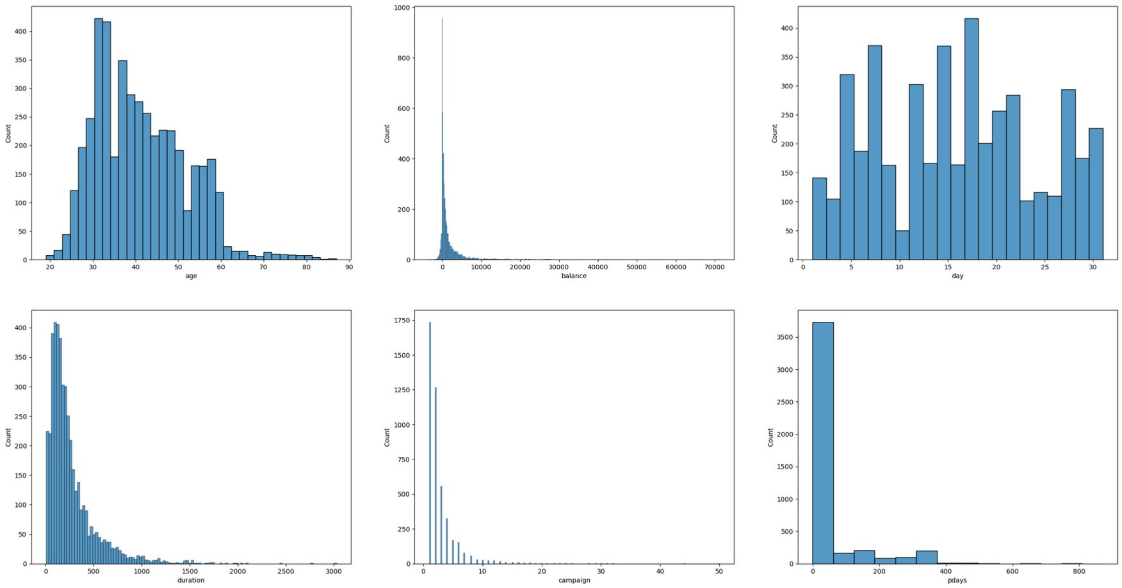
We have very less data for previous contacts greater than 6

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

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

Distribution of numerical features

```
plt.figure(figsize=(30,100), facecolor='white')
plotnumber =1
numerical_features.remove('previous')
for feature in numerical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.histplot(df[feature])
    #sns.displot(df[feature],height=10)
    plt.xlabel(feature)
    plotnumber+=1
plt.show()
```



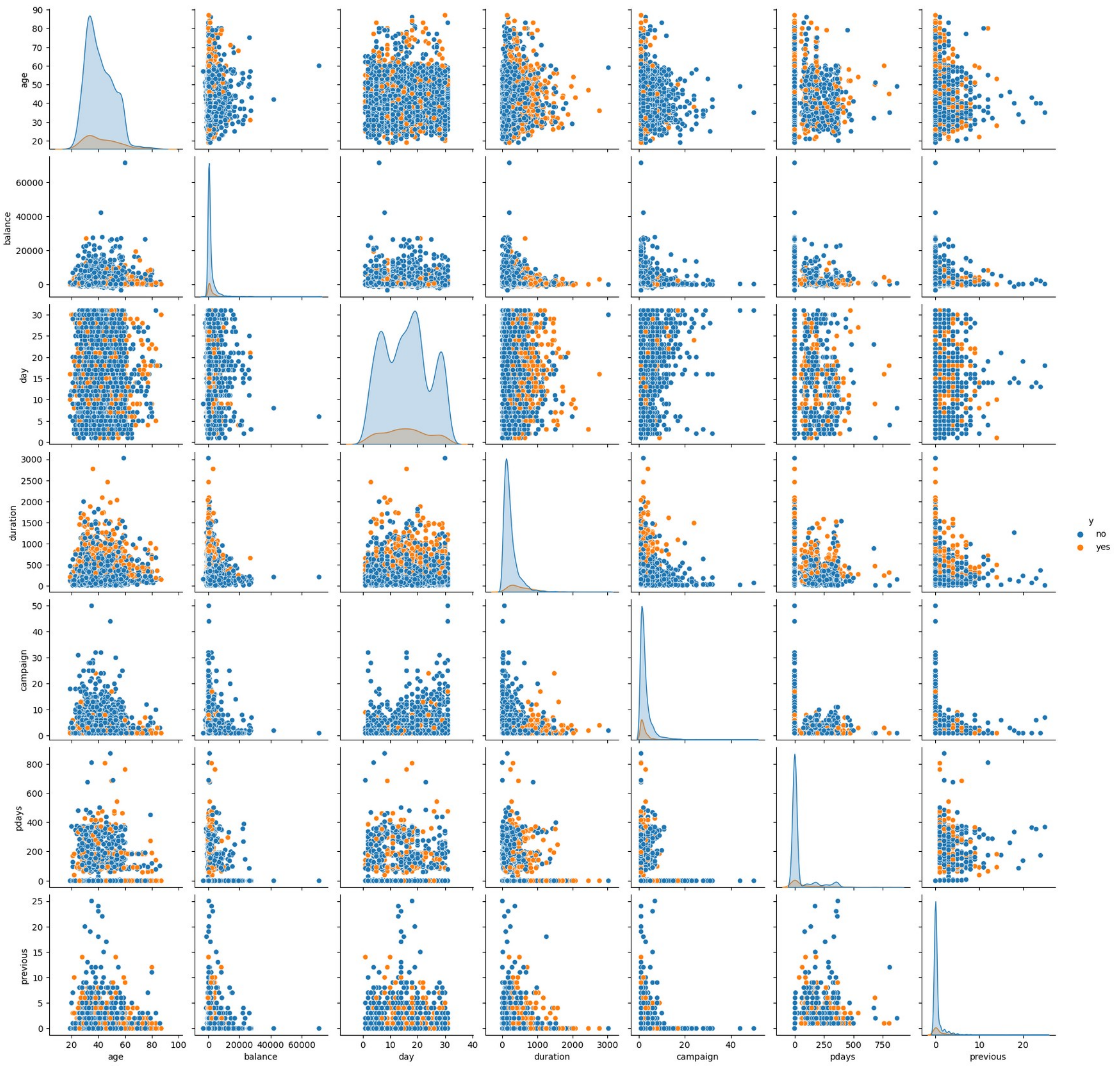
We have very less data for:

- Duration more than 1300 seconds
- Balance > 9000
- Number of contacts performed during this campaign > 9

Multivariate analysis of numerical features

```
sns.pairplot(df ,hue='y')
```

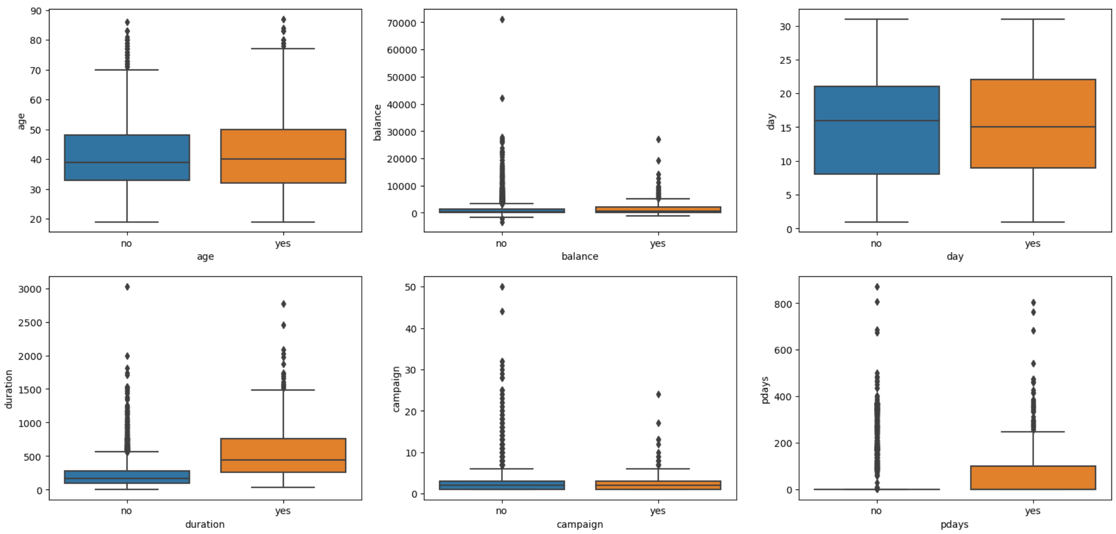
```
<seaborn.axisgrid.PairGrid at 0x29176bbe020>
```



There isnt linear correlation among any of the features

Relation between numerical features and outcome y

```
#Boxplot to show target distribution with respect numerical features
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for feature in numerical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(x="y", y= df[feature], data=df)
    plt.xlabel(feature)
    plotnumber+=1
plt.show()
```



- People tend to show more interest in making a deposit when the duration is higher, also when the days before previous contact were higher.
- We can see some outliers as well

According to our EDA:

- Default feature doesnt play a role in the outcome. So, it can be removed.
- We have about 10 categorical features.

Feature engineering

In this section, lets:

- Handle categorical features
- Scale featues appropriately
- Remove outliers

As previously deduced, we can drop the day and month features

```
df.drop(['day','month'],axis=1,inplace=True)
```

```
df.head()
```

```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no

1   33     services  married  secondary      no     4789     yes  yes

2   35   management   single   tertiary      no     1350     yes   no

3   30   management  married   tertiary      no     1476     yes  yes

4   59  blue-collar  married  secondary      no        0     yes   no
```

```
    contact  duration  campaign  pdays  previous poutcome   y
0  cellular        79         1     -1         0  unknown  no
1  cellular       220         1    339         4  failure  no
2  cellular       185         1    330         1  failure  no
3   unknown       199         4     -1         0  unknown  no
4   unknown       226         1     -1         0  unknown  no
```

```
df2 = df.copy()
```

```
df2.head()
```

```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no   

1   33     services  married  secondary      no     4789     yes  yes   

2   35   management   single   tertiary      no     1350     yes   no   

3   30   management  married   tertiary      no     1476     yes  yes   

4   59  blue-collar  married  secondary      no        0     yes   no   


    contact  duration  campaign  pdays  previous poutcome   y
0  cellular        79         1     -1         0  unknown  no
1  cellular       220         1    339         4  failure  no
2  cellular       185         1    330         1  failure  no
3   unknown       199         4     -1         0  unknown  no
4   unknown       226         1     -1         0  unknown  no
```

Lets handle the unknown values. We replace them **with** null

```
for i in df2.columns:
    df2[i] = np.where(df2[i] == "unknown", np.nan, df2[i])
df2.isna().sum()
```

```
age             0
job            38
marital         0
education     187
default         0
balance         0
housing         0
loan            0
contact      1324
duration        0
campaign        0
pdays           0
```

```
previous        0
poutcome     3705
y               0
dtype: int64
```

poutcome has more than 80% null values. Lets just drop that feature.

```
df2.drop('poutcome',axis=1,inplace=True)
```

Also, contact has no differnce whether it was from cellular or telephone. Hence, we can drop that as well

```
df2.drop('contact',axis=1,inplace=True)
```

As observed during EDA, the deafult feature doesn't play any role. Hence, lets remove it.

```
df2.drop('default',axis=1,inplace=True)
```

```
df2.head()
```

```
    age          job  marital  education  balance housing loan
duration  \
0  30.0   unemployed  married    primary   1787.0      no   no
79.0
1  33.0     services  married  secondary   4789.0     yes  yes
220.0
2  35.0   management   single   tertiary   1350.0     yes   no
185.0
3  30.0   management  married   tertiary   1476.0     yes  yes
199.0
4  59.0  blue-collar  married  secondary      0.0     yes   no
226.0

   campaign  pdays  previous poutcome   y
0       1.0   -1.0       0.0      NaN  no
1       1.0  339.0       4.0  failure  no
2       1.0  330.0       1.0  failure  no
3       4.0   -1.0       0.0      NaN  no
4       1.0   -1.0       0.0      NaN  no
```

```
df2.isna().sum()
```

```
age             0
job            38
marital         0
education     187
balance         0
housing         0
loan            0
duration        0
campaign        0
pdays           0
```

```
previous        0
poutcome     3705
y               0
dtype: int64
```

Lets fill the NaN values using the forward fill method,ie, we fill them with the last valid observed value

```
df2["job"].fillna(method = "ffill",inplace=True)
df2["education"].fillna(method = "ffill",inplace= True)
```

```
df1 = df2.copy()
```

```
df2 = df1.copy()
```

Lets label encode the categorical features besides job, as it has lots of values in it

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
features = ['marital', 'education', 'housing', 'loan','job']
for feature in features:
    df2[feature]=encoder.fit_transform(df2[feature])

df2['y'] = encoder.fit_transform(df['y'])
```

We can encode jobs using one hot encoding

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
df2[list(df2["job"].unique())] = ohe.fit_transform(df2[["job"]]).A
df2.drop("job",axis = 1, inplace = True)
```

```
df2.columns
```

```
Index(['age', 'job', 'marital', 'education', 'balance', 'housing',
'loan',
       'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'y'],
      dtype='object')
```

```
df2.shape
```

```
(4521, 13)
```

```
df4 = df2.copy()
```

Feature selection

```
df4.head()
```

```
    age  job  marital  education  balance  housing  loan  duration
campaign  \
0  30.0   10        1          0   1787.0        0     0      79.0
1.0
```

```
1  33.0    7        1          1   4789.0        1     1     220.0
1.0
2  35.0    4        2          2   1350.0        1     0     185.0
1.0
3  30.0    4        1          2   1476.0        1     1     199.0
4.0
4  59.0    1        1          1      0.0        1     0     226.0
1.0

   pdays  previous poutcome  y
0   -1.0       0.0      NaN  0
1  339.0       4.0  failure  0
2  330.0       1.0  failure  0
3   -1.0       0.0      NaN  0
4   -1.0       0.0      NaN  0
```

```
df4['poutcome'].isna().sum()
```

```
3705
```

We can see that almost 80% of the values in poutcome are NaN, hence lets drop it

```
df4.drop('poutcome',axis=1,inplace=True)
```

```
from sklearn.feature_selection import chi2
```

Chi2 test between categorical features

```
features = df4[['previous','marital', 'education', 'housing',
'loan','job',
        'campaign','campaign']]
chi, p_val = chi2(features,df4["y"])
res = pd.DataFrame({"Chi2":np.around(chi,2), "P_val":
np.around(p_val,2)}, index = features.columns)
res
```

```
             Chi2  P_val
previous   325.48   0.00
marital      0.32   0.57
education    5.14   0.02
housing     21.50   0.00
loan        19.05   0.00
job          7.57   0.01
campaign    58.50   0.00
campaign    58.50   0.00
```

The p_val for marital is quite high and chi2 is low, this indicates that marital is independent from the target y. So, lets drop it

```
df4.drop('marital',axis=1, inplace=True)
```

```
df4.head()
```

```
    age  job  education  balance  housing  loan  duration  campaign
pdays  \
0  30.0   10          0   1787.0        0     0      79.0       1.0
-1.0
1  33.0    7          1   4789.0        1     1     220.0       1.0
339.0
2  35.0    4          2   1350.0        1     0     185.0       1.0
330.0
3  30.0    4          2   1476.0        1     1     199.0       4.0
-1.0
4  59.0    1          1      0.0        1     0     226.0       1.0
-1.0

   previous  y
0       0.0  0
1       4.0  0
2       1.0  0
3       0.0  0
4       0.0  0
```

Splitting the data

```
X = df4.drop('y',axis=1)
y =df4['y']
```

```
X.head()
```

```
    age  job  education  balance  housing  loan  duration  campaign
pdays  \
0  30.0   10          0   1787.0        0     0      79.0       1.0
-1.0
1  33.0    7          1   4789.0        1     1     220.0       1.0
339.0
2  35.0    4          2   1350.0        1     0     185.0       1.0
330.0
3  30.0    4          2   1476.0        1     1     199.0       4.0
-1.0
4  59.0    1          1      0.0        1     0     226.0       1.0
-1.0

   previous
0       0.0
1       4.0
2       1.0
3       0.0
4       0.0
```

```
y.head()
```

```
0    0
1    0
2    0
```

```
3    0
4    0
Name: y, dtype: int32
```

```
y.value_counts()
```

```
0    4000
1     521
Name: y, dtype: int64
```

Scaling the data

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

#Scaling training data

arr = scaler.fit_transform(X)
X = pd.DataFrame(arr,columns = X.columns)
X.head()
```

```
        age  job  education   balance  housing  loan  duration  
campaign  \
0  0.161765  1.0        0.0  0.068455      0.0   0.0  0.024826  
0.000000   
1  0.205882  0.7        0.5  0.108750      1.0   1.0  0.071500  
0.000000   
2  0.235294  0.4        1.0  0.062590      1.0   0.0  0.059914  
0.000000   
3  0.161765  0.4        1.0  0.064281      1.0   1.0  0.064548  
0.061224   
4  0.588235  0.1        0.5  0.044469      1.0   0.0  0.073486  
0.000000   

      pdays  previous  
0  0.000000      0.00  
1  0.389908      0.16  
2  0.379587      0.04  
3  0.000000      0.00  
4  0.000000      0.00  
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size=0.2,stratify=y)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(3616, 10) (3616,)
(905, 10) (905,)
```

Balancing the data

The dataset is quite imbalanced, lets balance it out by oversampling. We use Synthetic Minority Oversampling Technique (SMOTE) to do so. We oversample the training datastet only so as to prevent overfitting.

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(sampling_strategy=0.75,k_neighbors= 3)
sm_x,sm_y = sm.fit_resample(X_train,y_train)
X_train = sm_x
y_train = sm_y
```

```
y_train.value_counts()
```

```
0    3199
1    2399
Name: y, dtype: int64
```

```
y_test.value_counts()
```

```
0    801
1    104
Name: y, dtype: int64
```

Model selection and training

Lets try a variety of algorithms to determine which one fits our data best. We should use models which are suitable for non-linearly separable classes

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score,
roc_auc_score,confusion_matrix,classification_report
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate
```

```
svc = SVC(kernel='linear', gamma=0.01)
knc = KNeighborsClassifier()
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=1000, random_state=2,)
abc = AdaBoostClassifier(n_estimators=100, random_state=2)
gbdt = GradientBoostingClassifier(n_estimators=100,random_state=2)
xgb = XGBClassifier(n_estimators=100,random_state=2)
```

```
pipeline = make_pipeline(xgb)
scoring = "accuracy"
```

```
scoresx

{'fit_time': array([0.50122738, 0.47577286, 0.52945256, 0.51323652,
0.46604681]),
 'score_time': array([0.00201416, 0.01004171, 0.0020175 , 0.00698209,
0.01003361]),
 'estimator': [Pipeline(steps=[('xgbclassifier',
                   XGBClassifier(base_score=None, booster=None,
callbacks=None,
                                 colsample_bylevel=None,
colsample_bynode=None,
                                 colsample_bytree=None,
                                 early_stopping_rounds=None,
                                 enable_categorical=False,
eval_metric=None,
                                 feature_types=None, gamma=None,
gpu_id=None,
                                 grow_policy=None,
importance_type=None,
                                 interaction_constraints=None,
learning_rate=None,
                                 max_bin=None, max_cat_threshold=None,
                                 max_cat_to_onehot=None,
max_delta_step=None,
                                 max_depth=None, max_leaves=None,
                                 min_child_weight=None, missing=nan,
                                 monotone_constraints=None,
n_estimators=100,
                                 n_jobs=None, num_parallel_tree=None,
                                 predictor=None,
random_state=2, ...))]),
  Pipeline(steps=[('xgbclassifier',
                   XGBClassifier(base_score=None, booster=None,
callbacks=None,
                                 colsample_bylevel=None,
colsample_bynode=None,
                                 colsample_bytree=None,
                                 early_stopping_rounds=None,
                                 enable_categorical=False,
eval_metric=None,
                                 feature_types=None, gamma=None,
gpu_id=None,
                                 grow_policy=None,
importance_type=None,
                                 interaction_constraints=None,
learning_rate=None,
                                 max_bin=None, max_cat_threshold=None,
                                 max_cat_to_onehot=None,
max_delta_step=None,
                                 max_depth=None, max_leaves=None,
```

```
                                  min_child_weight=None, missing=nan,
                                  monotone_constraints=None,
n_estimators=100,
                                  n_jobs=None, num_parallel_tree=None,
                                  predictor=None,
random_state=2, ...))]),
  Pipeline(steps=[('xgbclassifier',
                   XGBClassifier(base_score=None, booster=None,
callbacks=None,
                                  colsample_bylevel=None,
colsample_bynode=None,
                                  colsample_bytree=None,
                                  early_stopping_rounds=None,
                                  enable_categorical=False,
eval_metric=None,
                                  feature_types=None, gamma=None,
gpu_id=None,
                                  grow_policy=None,
importance_type=None,
                                  interaction_constraints=None,
learning_rate=None,
                                  max_bin=None, max_cat_threshold=None,
                                  max_cat_to_onehot=None,
max_delta_step=None,
                                  max_depth=None, max_leaves=None,
                                  min_child_weight=None, missing=nan,
                                  monotone_constraints=None,
n_estimators=100,
                                  n_jobs=None, num_parallel_tree=None,
                                  predictor=None,
random_state=2, ...))]),
  Pipeline(steps=[('xgbclassifier',
                   XGBClassifier(base_score=None, booster=None,
callbacks=None,
                                  colsample_bylevel=None,
colsample_bynode=None,
                                  colsample_bytree=None,
                                  early_stopping_rounds=None,
                                  enable_categorical=False,
eval_metric=None,
                                  feature_types=None, gamma=None,
gpu_id=None,
                                  grow_policy=None,
importance_type=None,
                                  interaction_constraints=None,
learning_rate=None,
                                  max_bin=None, max_cat_threshold=None,
                                  max_cat_to_onehot=None,
max_delta_step=None,
                                  max_depth=None, max_leaves=None,
```

```
                                 min_child_weight=None, missing=nan,
                                 monotone_constraints=None,
n_estimators=100,
                                 n_jobs=None, num_parallel_tree=None,
                                 predictor=None,
random_state=2, ...))]),
  Pipeline(steps=[('xgbclassifier',
                   XGBClassifier(base_score=None, booster=None,
callbacks=None,
                                 colsample_bylevel=None,
colsample_bynode=None,
                                 colsample_bytree=None,
                                 early_stopping_rounds=None,
                                 enable_categorical=False,
eval_metric=None,
                                 feature_types=None, gamma=None,
gpu_id=None,
                                 grow_policy=None,
importance_type=None,
                                 interaction_constraints=None,
learning_rate=None,
                                 max_bin=None, max_cat_threshold=None,
                                 max_cat_to_onehot=None,
max_delta_step=None,
                                 max_depth=None, max_leaves=None,
                                 min_child_weight=None, missing=nan,
                                 monotone_constraints=None,
n_estimators=100,
                                 n_jobs=None, num_parallel_tree=None,
                                 predictor=None,
random_state=2, ...))])],
 'test_score': array([-0.54116277, -0.26558561, -0.23904572, -
0.23538609, -0.23155842])}

clfs = {
    'SVC' : svc,
    'KN' : knc,
    'LR': lrc,
    'RF': rfc,
    'AdaBoost': abc,
    'GBDT':gbdt,
    'xgb':xgb
}

def train_classifier(clf,X_train,y_train,X_test,y_test):
#     clf.fit(X_train,y_train)
#     y_preds = clf.predict(X_test)
#     acc = clf.score(X_train,y_train)
#     accuracy = accuracy_score(y_test,y_preds)
#     precision = precision_score(y_test,y_preds)
#     return acc,accuracy,precision
```

```
    return cross_validate(clf, X_train, y_train, scoring=scoring,
cv=5, return_estimator=False)

for name,clf in clfs.items():

    #acc,current_accuracy,current_precision = train_classifier(clf,
X_train,y_train,X_test,y_test)
    scores = train_classifier(clf,X_train,y_train,X_test,y_test)
    print("For ",name)
    print("Scores:",scores)
#     print("Training accuracy -",acc)
#     print("Test Accuracy - ",current_accuracy)
#     print("Precision - ",current_precision)
```

```
For  SVC
Scores: {'fit_time': array([0.95300436, 0.99873233, 0.9831624 ,
1.04408145, 1.00076365]), 'score_time': array([0.18348551, 0.17459273,
0.16059756, 0.18035626, 0.17043924]), 'test_score': array([0.79553571,
0.775     , 0.78214286, 0.80160858, 0.79892761])}
For  KN
Scores: {'fit_time': array([0.03012824, 0.03108072, 0.02892303,
0.02209258, 0.03106117]), 'score_time': array([0.12178326, 0.12189698,
0.10094118, 0.11058593, 0.10931993]), 'test_score': array([0.88214286,
0.87053571, 0.86964286, 0.87756926, 0.87756926])}
For  LR
Scores: {'fit_time': array([0.03112912, 0.03017497, 0.03215075,
0.03012204, 0.0301187 ]), 'score_time': array([0.0090394 , 0.00802445,
0.        , 0.01013088, 0.        ]), 'test_score': array([0.8       ,
0.77410714, 0.78214286, 0.80339589, 0.7971403 ])}
For  RF
Scores: {'fit_time': array([15.01381302, 14.78281498, 14.73240829,
14.65531754, 14.8762641 ]), 'score_time': array([0.44979692, 0.4501543
, 0.45408249, 0.4452045 , 0.45192552]), 'test_score':
array([0.86517857, 0.91785714, 0.92142857, 0.9463807 , 0.93476318])}
For  AdaBoost
Scores: {'fit_time': array([1.15545511, 1.04736924, 1.04754329,
1.04530025, 1.0557127 ]), 'score_time': array([0.06030345, 0.07864666,
0.07048726, 0.07061124, 0.07045531]), 'test_score': array([0.73571429,
0.87678571, 0.89107143, 0.9106345 , 0.89365505])}
For  GBDT
Scores: {'fit_time': array([1.58619261, 1.51730895, 1.48796344,
1.47801495, 1.46718788]), 'score_time': array([0.00201869, 0.01006055,
0.01004362, 0.01004171, 0.01015306]), 'test_score': array([0.75625   ,
0.88214286, 0.89107143, 0.91867739, 0.90974084])}
For  xgb
Scores: {'fit_time': array([0.54235244, 0.51422858, 0.52839017,
0.53253222, 0.52253342]), 'score_time': array([0.00201702, 0.01004267,
0.00404692, 0.01004696, 0.01003909]), 'test_score': array([0.70714286,
0.92946429, 0.94285714, 0.94459339, 0.9463807 ])}
```

We observe that the Random Forest Classifier, Extra Tree Classifier and XGBoost Classifier give us the best results. However, XGBoost gives us the best recall. Hence, lets move forward with it and try to improve it

Model tuning

```
from sklearn.model_selection import RandomizedSearchCV

model = XGBClassifier()
parameters = {"n_estimators": [50,100,150,200,250,300,350,400,450],
             "max_depth": np.arange(2,11),
             "learning_rate": np.arange(0.01,0.1,0.02),
             'subsample': np.arange(0.5, 1.0, 0.05),
             'colsample_bytree': np.arange(0.4, 1.0, 0.05),
             'colsample_bylevel': np.arange(0.4, 1.0, 0.05)}
search = RandomizedSearchCV(model, parameters, cv = 5, random_state=
42)
search.fit(X_train,y_train)
search.best_params_
```

```
{'subsample': 0.9000000000000004,
 'n_estimators': 350,
 'max_depth': 10,
 'learning_rate': 0.08999999999999998,
 'colsample_bytree': 0.55,
 'colsample_bylevel': 0.6499999999999999}
```

```
xgb = XGBClassifier(subsample = 0.9000000000000004, n_estimators =
350, max_depth =10, learning_rate=0.08999999999999998,\
             colsample_bytree =  0.55, colsample_bylevel =
0.6499999999999999)

xgb.fit(X_train, y_train)
y_pr_train = xgb.predict(X_train)
acc_train = accuracy_score(y_train,y_pr_train)
class_re = classification_report(y_train,y_pr_train)
con_mat = confusion_matrix(y_train,y_pr_train)
print("Confusion Matrix:\n",con_mat)
print("\n")
print("The accuracy of the model:",(acc_train)*100)
print("\n")
print("The classification report:\n",class_re)
```

```
Confusion Matrix:
 [[3199    0]
 [   0 2399]]


The accuracy of the model: 100.0
```

```
The classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      3199
           1       1.00      1.00      1.00      2399

    accuracy                           1.00      5598
   macro avg       1.00      1.00      1.00      5598
weighted avg       1.00      1.00      1.00      5598
```

Our model is 100% accurate on the training data. This is because it has been oversampled appropriately to make up for the dfifference in the percentages of Yes's and No's in the dataset given to us

```
y_preds = xgb.predict(X_test)
acc = accuracy_score(y_test,y_pr_test)
class_re = classification_report(y_test,y_preds)
con_mat = confusion_matrix(y_test,y_preds)
print("Confusion Matrix:\n",con_mat)
print("\n")
print("The accuracy of the model:",(acc)*100)
print("\n")
print("The classification report:\n",class_re)
```

```
Confusion Matrix:
 [[753  48]
 [ 57  47]]


The accuracy of the model: 88.39779005524862


The classification report:
               precision    recall  f1-score   support

           0       0.93      0.94      0.93       801
           1       0.49      0.45      0.47       104

    accuracy                           0.88       905
   macro avg       0.71      0.70      0.70       905
weighted avg       0.88      0.88      0.88       905
```

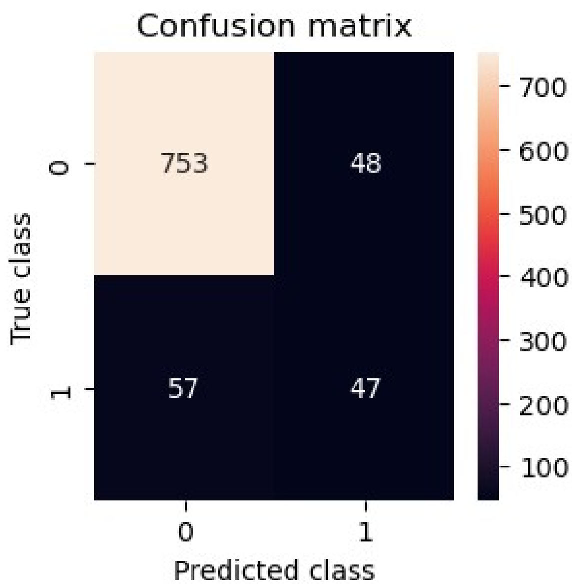
Hence, we have about 88.4% test accuracy with our model.

Confusion matrix :

```
y_pred = xgb.predict(X_test)
```

```
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

# Plot confusion matrix as a heatmap
plt.figure(figsize=(3,3))
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel('Predicted class')
plt.ylabel('True class')
plt.title('Confusion matrix')
plt.show()
```

```
Confusion matrix:
[[753  48]
 [ 57  47]]
```

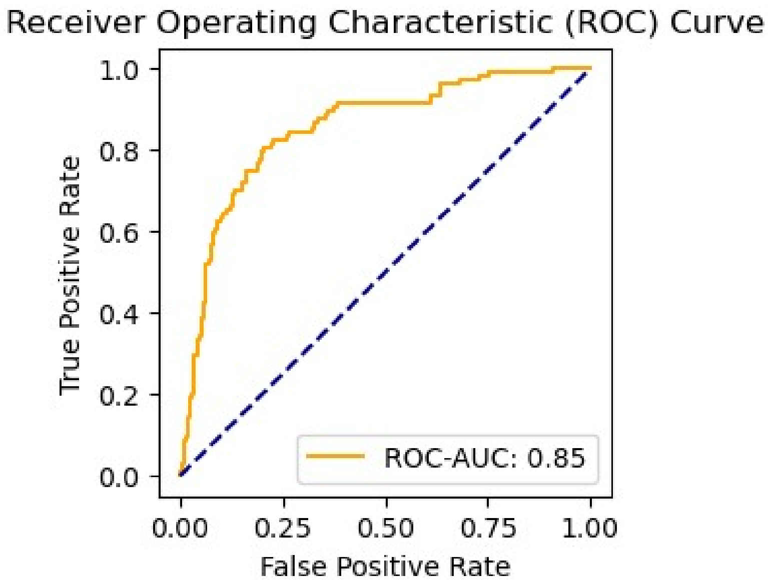


Plotting the ROC AUC curve

```
from sklearn.metrics import roc_curve
y_probs = xgb.predict_proba(X_test)[:,1]

# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, y_probs)
print("ROC-AUC:", roc_auc)

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Plot ROC curve
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, color='orange', label='ROC-AUC: %0.2f' % roc_auc)
```

```
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

```
ROC-AUC: 0.8523600307308172
```



We get a recall of about 47% for the Yes values, and 93% for the No values in our test dataset. We get low recall for yes values as the ratio of Yes to No values in our given dataset is about 11%, which is quite low. If we want higher recall, we should oversample the test set as well, but we never do so coz that would rastically change the prediction for the actual data which is fed to the model. Hence, we stop our model tuning here and conclude that our model predicts with a recall of over 47%.

Hence, we conclude our report with out final choice of model being XGBoost Classifier, which is able to predict with an accuracy of 88.4% and a recall of 47%

Our model has the following limitations:

- The data fed to it was highly imbalanced, hence it has very less recall.
- The data given to it had several missing values, which results in lower accuracy.

Future scope:

- This model can be modified so as to fit a dataset of customers who made purchases, cancelled subsciptions, etc.
- This model can also be modified so as to make predictions for customers who are moer likely to make purchases for further marketing campaings. The dataset can be manipulated by the model so as to produce new data