```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
Start coding or generate with AI.
```

```
df=pd.read_csv("/content/Bank Mrketing data.csv")
df
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous
0	59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0
1	56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0
2	41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0
3	55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0
4	54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	20	apr	257	1	-1	0
11158	39	services	married	secondary	no	733	no	no	unknown	16	jun	83	4	-1	0
11159	32	technician	single	secondary	no	29	no	no	cellular	19	aug	156	2	-1	0
11160	43	technician	married	secondary	no	0	no	yes	cellular	8	may	9	2	172	5
11161	34	technician	married	secondary	no	0	no	no	cellular	9	jul	628	1	-1	0

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

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

```
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
deposit      object
dtype: object
```

```
df.describe()
```

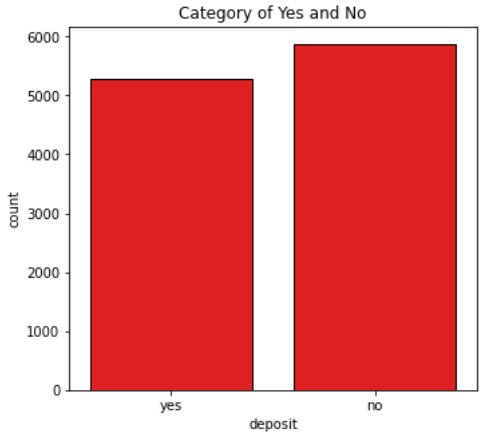
	age	balance	day	duration	campaign	pdays
count	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000	11162.000000
mean	41.231948	1528.538524	15.658036	371.993818	2.508421	51.330407
std	11.913369	3225.413326	8.420740	347.128386	2.722077	108.758282
min	18.000000	-6847.000000	1.000000	2.000000	1.000000	-1.000000
25%	32.000000	122.000000	8.000000	138.000000	1.000000	-1.000000
50%	39.000000	550.000000	15.000000	255.000000	2.000000	-1.000000
75%	49.000000	1708.000000	22.000000	496.000000	3.000000	20.750000
max	95.000000	81204.000000	31.000000	3881.000000	63.000000	854.000000

```
df['deposit'].value_counts()/df.shape[0]
```

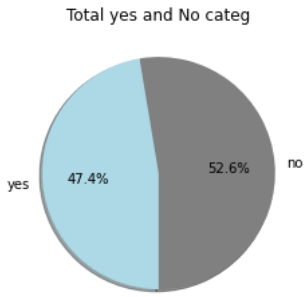
```
no     0.52616
yes    0.47384
Name: deposit, dtype: float64
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
sns.countplot(x='deposit',data=df,color='red',edgecolor="black")
plt.title("Category of Yes and No")
```

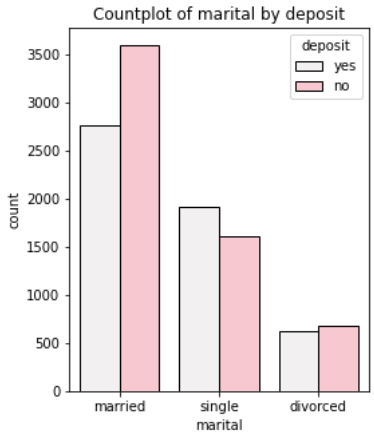
```
Text(0.5, 1.0, 'Category of Yes and No')
```



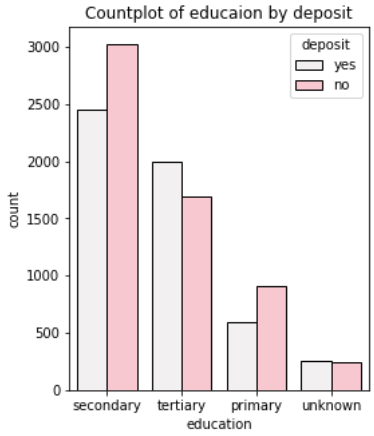

```
from matplotlib import colors
## add colors
colors = ['gray','lightblue']
labels =df['deposit'].value_counts(sort = True).index
sizes = df['deposit'].value_counts(sort = True)
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=270,)
plt.title('Total yes and No categ',size = 12,color='black')
plt.show()
```



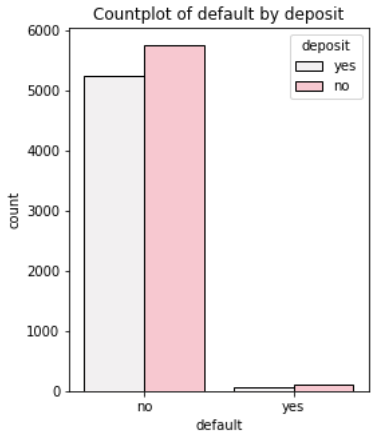

```
plt.figure(figsize=[4,5])
sns.countplot(x='marital', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of marital by deposit")
plt.show()
```



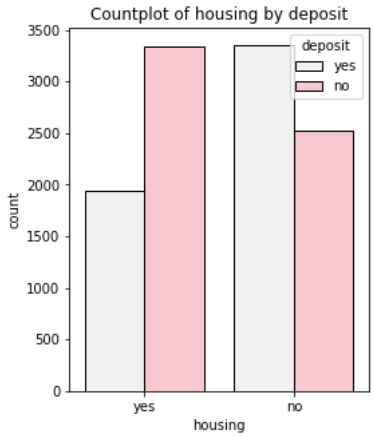

```
plt.figure(figsize=[4,5])
sns.countplot(x='education', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of educaion by deposit")
plt.show()
```



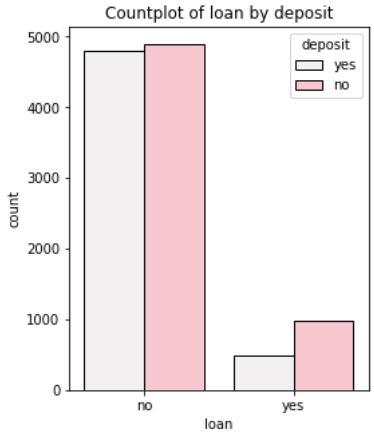

```
plt.figure(figsize=[4,5])
sns.countplot(x='default', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of default by deposit")
plt.show()
```



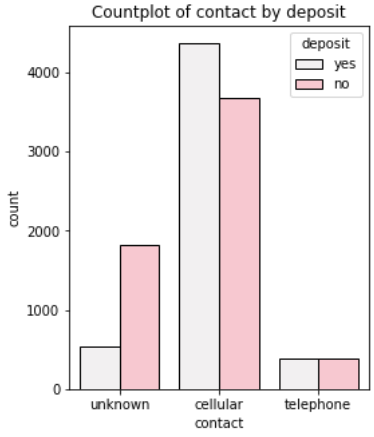

```
plt.figure(figsize=[4,5])
sns.countplot(x='housing', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of housing by deposit")
plt.show()
```



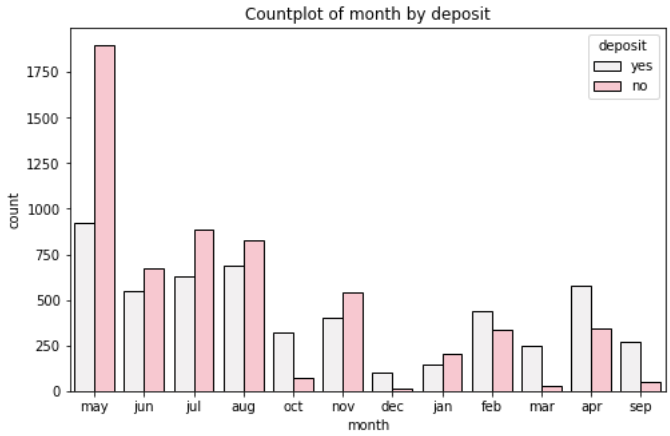

```
plt.figure(figsize=[4,5])
sns.countplot(x='loan', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of loan by deposit")
plt.show()
```



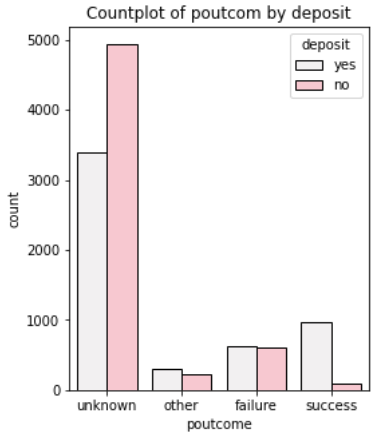

```
plt.figure(figsize=[4,5])
sns.countplot(x='contact', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of contact by deposit")
plt.show()
```



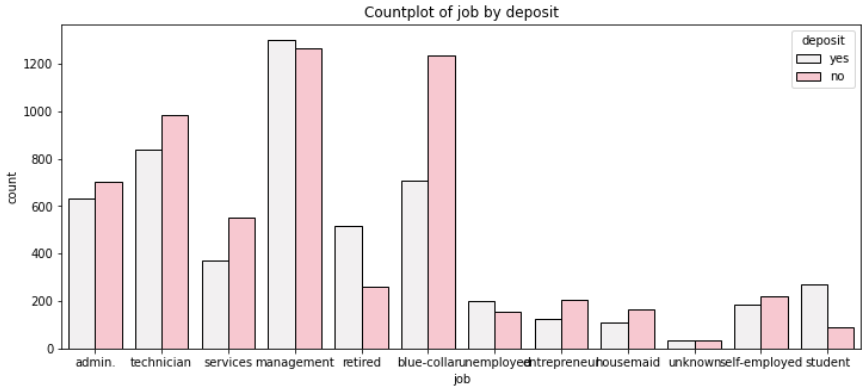

```
plt.figure(figsize=[8,5])
sns.countplot(x='month', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of month by deposit")
plt.show()
```




```
plt.figure(figsize=[4,5])
sns.countplot(x='poutcome', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of poutcom by deposit")
plt.show()
```

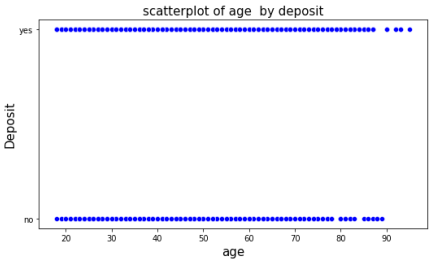


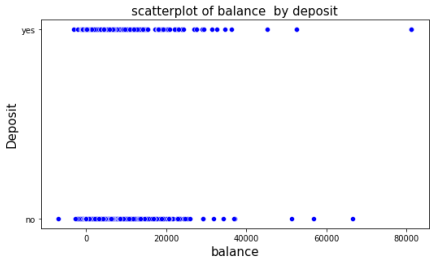

```
plt.figure(figsize=[12,5])
sns.countplot(x='job', hue='deposit',color='pink',edgecolor="black",data=df)
plt.title("Countplot of job by deposit")
plt.show()
```

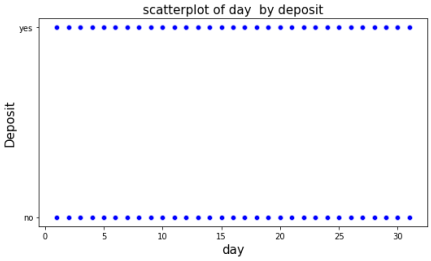


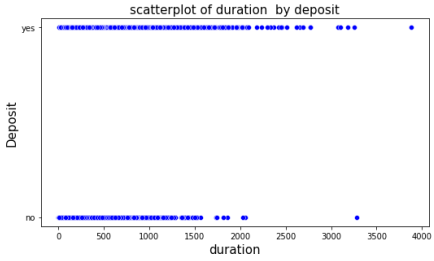

```
df_num = df[['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous','deposit']]
col = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous','deposit']
plt.figure(figsize=(15,18))
for i,v in enumerate(col):
    print(i,v)
    plt.subplot(4,2,i+1)
    sns.scatterplot(x=v,y='deposit' ,data=df_num,color='Blue')
    plt.title("scatterplot of {}  by deposit".format(v),size=15,color="black")
    plt.xlabel("{}".format(v),size=15)
    plt.ylabel("Deposit",size=15)
plt.tight_layout()
plt.show()
```

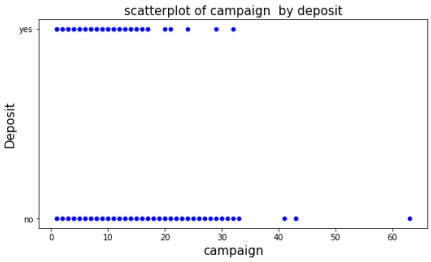
```
0 age
1 balance
2 day
3 duration
4 campaign
5 pdays
6 previous
7 deposit
```

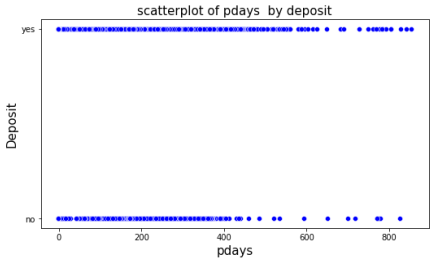


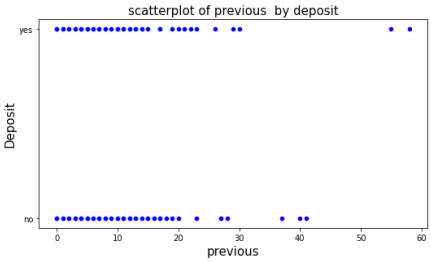


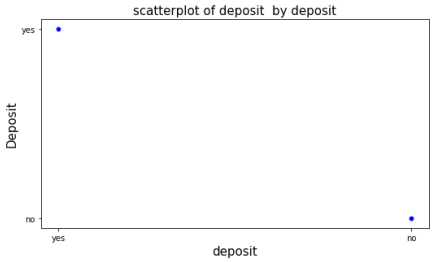






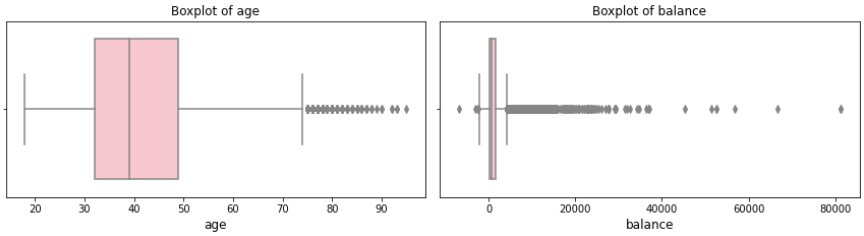






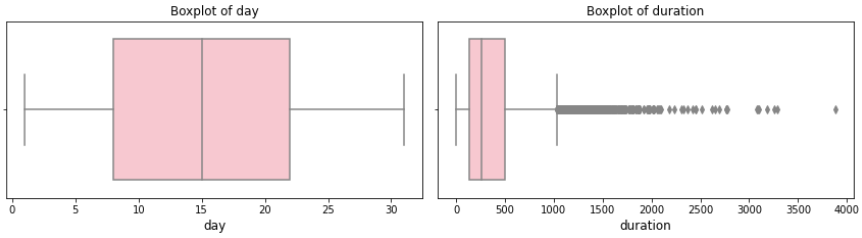
```
df_num = df[['age', 'balance']]
col = ['age', 'balance']
plt.figure(figsize=(12,12))
for i,v in enumerate(col):
    print(i,v)
    plt.subplot(4,2,i+1)
    sns.boxplot(x=v,data=df_num,color='pink')
    plt.title("Boxplot of {}".format(v),size=12,color="Black")
    plt.xlabel("{}".format(v),size=12)
plt.tight_layout()
plt.show()
```

```
0 age
1 balance
```



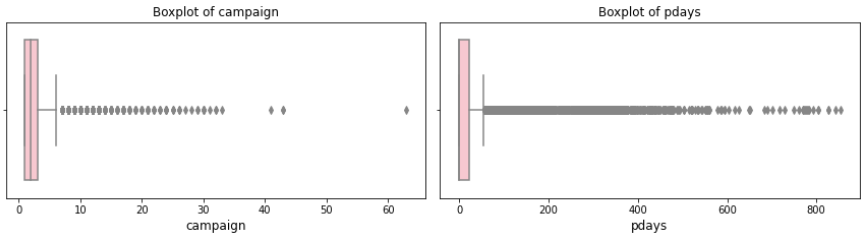

```
df_num = df[['day', 'duration']]
col = ['day', 'duration']
plt.figure(figsize=(12,12))
for i,v in enumerate(col):
    print(i,v)
    plt.subplot(4,2,i+1)
    sns.boxplot(x=v,data=df_num,color='pink')
    plt.title("Boxplot of {}".format(v),size=12,color="Black")
    plt.xlabel("{}".format(v),size=12)
plt.tight_layout()
plt.show()
```

```
0 day
1 duration
```



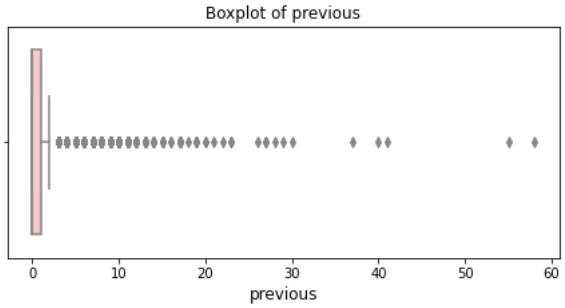

```
df_num = df[['campaign', 'pdays']]
col = ['campaign', 'pdays']
plt.figure(figsize=(12,12))
for i,v in enumerate(col):
    print(i,v)
    plt.subplot(4,2,i+1)
    sns.boxplot(x=v,data=df_num,color='pink')
    plt.title("Boxplot of {}".format(v),size=12,color="Black")
    plt.xlabel("{}".format(v),size=12)
plt.tight_layout()
plt.show()
```

```
0 campaign
1 pdays
```



```
df_num = df[['previous']]
col = ['previous']
plt.figure(figsize=(12,12))
for i,v in enumerate(col):
    print(i,v)
    plt.subplot(4,2,i+1)
    sns.boxplot(x=v,data=df_num,color='pink')
    plt.title("Boxplot of {}".format(v),size=12,color="Black")
    plt.xlabel("{}".format(v),size=12)
plt.tight_layout()
plt.show()
```

```
0 previous
```




```
len(df[df['balance']<0])/len(df)
```

```
0.061637699337036375
```

```
df[(df['balance']>10000)|(df['balance']<0)]
```

	age	job	marital	education	default	balance	housing	loan	contact
17	49	services	married	secondary	no	-8	yes	no	unknown
23	43	blue-collar	married	primary	no	-192	yes	no	unknown
30	32	blue-collar	married	secondary	yes	-1	yes	no	unknown
42	45	entrepreneur	divorced	tertiary	no	-395	yes	no	unknown
59	57	technician	married	tertiary	no	-1	no	no	unknown
...	...	...	...	...	...	...	...	...	...
11119	39	management	married	tertiary	no	-974	no	yes	cellular
11120	50	management	married	tertiary	no	-516	yes	no	unknown
11132	32	blue-collar	married	secondary	no	-325	yes	yes	unknown
11145	60	retired	divorced	tertiary	no	-134	no	no	cellular
11156	34	blue-collar	single	secondary	no	-72	yes	no	cellular

916 rows × 17 columns

```
df.drop(df[(df['balance']>40000)|(df['balance']<0)].index,inplace=True,axis=0)
```

```
df[df['duration']>3000]
```

	age	job	marital	education	default	balance	housing	loan	contact
153	44	services	divorced	secondary	no	51	yes	yes	unknown
271	59	management	married	secondary	no	1321	no	no	unknown
1351	47	blue-collar	married	secondary	no	238	yes	yes	cellular
4364	53	admin.	married	secondary	no	849	yes	no	cellular
7198	30	admin.	married	secondary	no	1310	no	no	telephone

```
df.drop(df[df['duration']>3000].index,inplace=True,axis=0)
```

```
df[df['campaign']>40]
```

	age	job	marital	education	default	balance	housing	loan	contact
6927	51	blue-collar	married	unknown	no	41	yes	no	telephone
7139	42	blue-collar	married	primary	no	170	yes	no	unknown
7240	33	blue-collar	married	secondary	no	0	yes	yes	cellular
7635	45	management	married	unknown	no	9051	yes	no	unknown

```
df.drop(df[df['campaign']>30].index,axis=0,inplace=True)
```

```
df[df['pdays']==-1]
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
11154	52	technician	married	tertiary	no	523	yes	yes	cellular	
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	2
11158	39	services	married	secondary	no	733	no	no	unknown	1
11159	32	technician	single	secondary	no	29	no	no	cellular	1
11161	34	technician	married	secondary	no	0	no	no	cellular	

```
df[df['poutcome']=='unknown']
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
11154	52	technician	married	tertiary	no	523	yes	yes	cellular	
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	2
11158	39	services	married	secondary	no	733	no	no	unknown	1
11159	32	technician	single	secondary	no	29	no	no	cellular	1
11161	34	technician	married	secondary	no	0	no	no	cellular	

```
df[df['previous']==0]
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
11154	52	technician	married	tertiary	no	523	yes	yes	cellular	
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	2
11158	39	services	married	secondary	no	733	no	no	unknown	1
11159	32	technician	single	secondary	no	29	no	no	cellular	1
11161	34	technician	married	secondary	no	0	no	no	cellular	

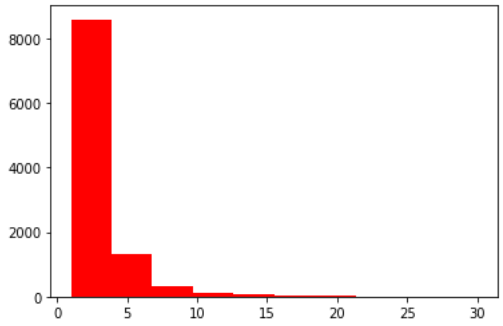
```
df.drop("pdays",inplace=True,axis=1)
```

```
df[df['previous']>30]
```

	age	job	marital	education	default	balance	housing	loan	contact	d
2013	46	blue-collar	married	primary	no	1085	yes	yes	cellular	
3677	37	technician	married	secondary	no	432	yes	no	cellular	
6274	27	blue-collar	married	secondary	no	821	yes	yes	unknown	
8713	35	technician	single	secondary	no	4645	yes	no	cellular	

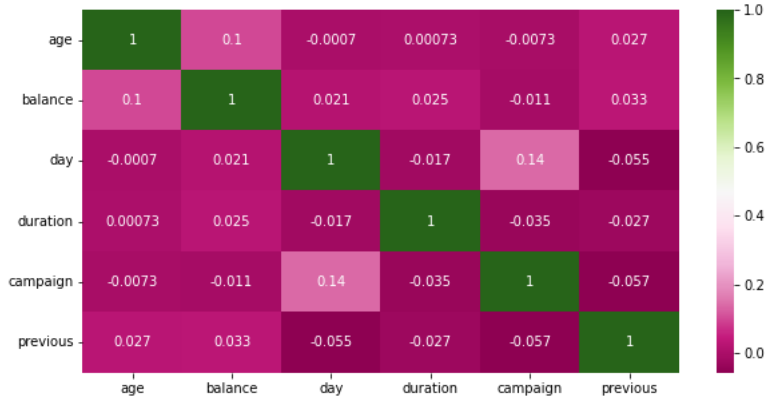
```
df.drop(df[df['previous']>30].index,axis=0,inplace=True)
```

```
#df1 = df[df['balance']>0]
plt.hist(x="campaign",data=df,color='red')
plt.show()
```




```
df1 = df.copy()
```

```
#pearson's Correlation,which measures the strength of a linear relationship
df_num = df[['age', 'balance', 'day', 'duration', 'campaign', 'previous']]
plt.figure(figsize=(10,5))
sns.heatmap(data=df_num.corr(), cmap="PiYG" ,annot=True)
plt.show()
```




```
#create dict for binary encoding
dic = {"yes":1,"no":0}
```

```
from sklearn.preprocessing import LabelEncoder
dic = {"yes":1,"no":0}
lst = ["deposit","loan","default","housing"]
for i in lst:
    df[i] = df[i].map(dic)
```

```
# Ordinal Encoding
l=['month',"contact","poutcome"]
for i in l:
    le=LabelEncoder()
    df[i]=le.fit_transform(df[i].values)
```

```
df = pd.get_dummies(df, columns = ['job','marital','education'])
```

```
df=df.reset_index()
df.drop('index',axis=1,inplace=True)
df
```

	age	default	balance	housing	loan	contact	day	month	duration	campaign
0	59	0	2343	1	0	2	5	8	1042	1
1	56	0	45	0	0	2	5	8	1467	1
2	41	0	1270	1	0	2	5	8	1389	1
3	55	0	2476	1	0	2	5	8	579	1
4	54	0	184	0	0	2	5	8	673	2
...	...	...	...	...	...	...	...	...	...	...
10444	33	0	1	1	0	0	20	0	257	1
10445	39	0	733	0	0	2	16	6	83	4
10446	32	0	29	0	0	0	19	1	156	2
10447	43	0	0	0	1	0	8	8	9	2
10448	34	0	0	0	0	0	9	5	628	1

10449 rows × 32 columns

```
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import StratifiedShuffleSplit, StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

```
X = df.drop('deposit',axis=1)
Y = df['deposit']
Y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
10444    0
```

```
10445    0
10446    0
10447    0
10448    0
Name: deposit, Length: 10449, dtype: int64
```

```
sss = StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=1)
for train_index,test_index in sss.split(X,Y):
    train_df = df.loc[train_index]
    test_df = df.loc[test_index]
```

```
print("Ratio for train dataset")
print(train_df['deposit'].value_counts()/train_df.shape[0])
print()
print("ratio for test dataset")
print(test_df['deposit'].value_counts()/test_df.shape[0])
```

```
Ratio for train dataset
0    0.51504
1    0.48496
Name: deposit, dtype: float64

ratio for test dataset
0    0.515152
1    0.484848
Name: deposit, dtype: float64
```

```
X_train = train_df.drop("deposit",axis=1)
Y_train = train_df['deposit']

X_test = test_df.drop("deposit",axis=1)
Y_test = test_df['deposit']
```

```
X_train
```

	age	default	balance	housing	loan	contact	day	month	duration	campaign	.
5461	28	0	674	1	0	1	14	8	921	4	
4220	36	0	324	1	1	0	16	5	830	1	
5530	56	0	1480	1	1	0	5	3	576	1	
4249	31	0	26965	0	0	0	21	0	654	2	
9514	30	0	177	1	0	0	9	0	62	2	
...	...	...	...	...	...	...	...	...	...	...	
8100	34	0	425	1	0	0	16	5	1389	7	
4223	27	0	11862	0	0	0	25	9	285	2	
343	26	0	551	0	0	0	8	5	531	1	
4449	41	0	5517	1	0	0	10	5	584	1	
2983	76	0	2223	0	0	1	4	3	429	1	

7314 rows × 31 columns

```
X_test
```

	age	default	balance	housing	loan	contact	day	month	duration	campaign
1249	36	0	3407	0	0	0	16	7	392	4
8740	39	0	2887	1	0	2	7	8	42	2
9830	34	0	371	0	0	2	20	8	77	1
3279	63	0	115	0	0	1	27	0	325	1
7479	59	0	1525	0	0	1	29	5	90	2
...	...	...	...	...	...	...	...	...	...	...
1409	35	0	293	1	0	0	17	0	1017	1
2579	73	0	7111	1	0	0	12	10	146	1
10313	41	0	141	1	1	2	8	8	146	3
1331	23	0	314	0	1	0	8	0	142	4
1176	44	0	558	0	0	0	19	3	268	5

3135 rows × 31 columns

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_s = ss.fit_transform(X_train)
X_test_s = ss.transform(X_test)


#importing all the required ML packages
#logistic regression
from sklearn.linear_model import LogisticRegression
#support vector Machine
from sklearn import svm
#Random Forest
from sklearn.ensemble import RandomForestClassifier
 #KNN
from sklearn.neighbors import KNeighborsClassifier
#Naive bayes
from sklearn.naive_bayes import GaussianNB
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
#training and testing data split
from sklearn.model_selection import train_test_split
#accuracy measure
from sklearn import metrics
#for confusion matrix
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler


lr= LogisticRegression()

lr.fit(X_train_s,Y_train)
Y_pred_lr = lr.predict(X_test_s)

print("Testing Accuracy of LogisticRegression : ",metrics.accuracy_score(Y_test,Y_pred_lr))
#print("Accuracy of LogisticRegression",pipe_lr.score(X_test,Y_test))

print("Training Accuracy of LogisticRegression : ",lr.score(X_train_s,Y_train))

     Testing Accuracy of LogisticRegression :  0.7894736842105263
     Training Accuracy of LogisticRegression :  0.7950505879135904


svm=svm.SVC(kernel='linear')

svm.fit(X_train_s,Y_train)
Y_pred_svm = svm.predict(X_test_s)

print("Testing Accuracy of SVM : ",metrics.accuracy_score(Y_test,Y_pred_svm))
#print("Accuracy of SVM",pipe_svm.score(X_test,Y_test))

print("Training Accuracy of SVM : ",svm.score(X_train_s,Y_train))

     Testing Accuracy of SVM :  0.7933014354066986
     Training Accuracy of SVM :  0.7955974842767296
```

```
dt= DecisionTreeClassifier()

dt.fit(X_train_s,Y_train)
Y_pred_dt = dt.predict(X_test_s)

print("Testing Accuracy of Decision Tree : ",metrics.accuracy_score(Y_test,Y_pred_dt))
#print("Accuracy of Decision Tree",pipe_dt.score(X_test,Y_test))

print("Training Accuracy of Decision Tree : ",dt.score(X_train_s,Y_train))
```

```
     Testing Accuracy of Decision Tree :  0.7862838915470495
     Training Accuracy of Decision Tree :  1.0
```

```
knn= KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train_s,Y_train)
Y_pred_knn = knn.predict(X_test_s)

print("Testing Accuracy of KNN : ",metrics.accuracy_score(Y_test,Y_pred_knn))
#print("Accuracy of KNN",pipe_knn.score(X_test,Y_test))

print("Training Accuracy of KNN : ",knn.score(X_train_s,Y_train))
```

```
     Testing Accuracy of KNN :  0.7282296650717703
     Training Accuracy of KNN :  0.8185671315285753
```

```
l = list(range(1,25,2))
for i in l:
    knn1= KNeighborsClassifier(n_neighbors=i)
    knn1.fit(X_train_s,Y_train)
    Y_pred = knn1.predict(X_test_s)
    accuracy = metrics.accuracy_score(Y_test,Y_pred)
    train_acc = knn1.score(X_train_s,Y_train)
    print(f"For K={i} test accuracy score is {accuracy} :".format(i,accuracy))
    print(f"For K={i} train accuracy score is {accuracy} :".format(i,train_acc))
    print()
```

```
     For K=1 test accuracy score is 0.703030303030303 :
     For K=1 train accuracy score is 0.703030303030303 :

     For K=3 test accuracy score is 0.7250398724082935 :
     For K=3 train accuracy score is 0.7250398724082935 :

     For K=5 test accuracy score is 0.7282296650717703 :
     For K=5 train accuracy score is 0.7282296650717703 :

     For K=7 test accuracy score is 0.727591706539075 :
     For K=7 train accuracy score is 0.727591706539075 :

     For K=9 test accuracy score is 0.7355661881977671 :
     For K=9 train accuracy score is 0.7355661881977671 :

     For K=11 test accuracy score is 0.7349282296650718 :
     For K=11 train accuracy score is 0.7349282296650718 :

     For K=13 test accuracy score is 0.7317384370015949 :
     For K=13 train accuracy score is 0.7317384370015949 :

     For K=15 test accuracy score is 0.733652312599681 :
     For K=15 train accuracy score is 0.733652312599681 :

     For K=17 test accuracy score is 0.7317384370015949 :
     For K=17 train accuracy score is 0.7317384370015949 :

     For K=19 test accuracy score is 0.7288676236044657 :
     For K=19 train accuracy score is 0.7288676236044657 :

     For K=21 test accuracy score is 0.7311004784688995 :
     For K=21 train accuracy score is 0.7311004784688995 :

     For K=23 test accuracy score is 0.7301435406698564 :
     For K=23 train accuracy score is 0.7301435406698564 :
```

```
nb= GaussianNB()

nb.fit(X_train_s,Y_train)
Y_pred_nb = nb.predict(X_test_s)

print("Testing Accuracy of Naive Bayes : ",metrics.accuracy_score(Y_test,Y_pred_knn))
#print("Accuracy of Naive Bayes",pipe_nb.score(X_test,Y_test))

print("Training Accuracy of Naive Bayes : ",nb.score(X_train_s,Y_train))
```

```
     Testing Accuracy of Naive Bayes :  0.7282296650717703
     Training Accuracy of Naive Bayes :  0.7108285479901558
```

```
rf = RandomForestClassifier(n_estimators=200)

rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)

print("Testing Accuracy of RF : ",metrics.accuracy_score(Y_test,Y_pred_rf))
#print("Accuracy of of RF",pipe_rf.score(X_test,Y_test))

print("Testing Accuracy of RF : ",rf.score(X_train,Y_train))
```

```
     Testing Accuracy of RF :  0.8392344497607656
     Testing Accuracy of RF :  1.0
```

```
from sklearn import svm


svm1=svm.SVC()
C=[0.05,0.2,0.4,0.6,0.8,1,2,5]

hyper={'C':C}
model=GridSearchCV(estimator=svm1, param_grid=hyper,cv=10)
model.fit(X_train_s,Y_train)
print(model.best_score_)
print(model.best_estimator_)
```

```
     0.8024380854133494
     SVC(C=1)
```

```
gd_best=model.best_estimator_
gd_best.fit(X_train_s, Y_train)
```

```
     SVC(C=1)
```

```
print("Test Accuracy :",gd_best.score(X_test_s,Y_test))
print("Train Accuracy :",gd_best.score(X_train_s,Y_train))
```

```
     Test Accuracy : 0.7977671451355662
     Train Accuracy : 0.8424938474159147
```

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(random_state=5)

gb.fit(X_train_s,Y_train.squeeze().values)
y_train_preds = gb.predict(X_train_s)
y_test_preds = gb.predict(X_test_s)

print('The accuracy of the GB is' ,metrics.accuracy_score(Y_test,y_test_preds))
print('The accuracy of the GB is' , metrics.accuracy_score(y_train_preds,Y_train))
#result=cross_val_score(pipe_nb,X,Y,cv=10,scoring='accuracy')
#print('The cross validated score for Gradient Boosting is:',result.mean())
```

```
     The accuracy of the GB is 0.8373205741626795
     The accuracy of the GB is 0.8565764287667487
```

```
import xgboost


xgb = xgboost.XGBClassifier(n_estimators=80, learning_rate=0.1, gamma=0, subsample=0.75,
                            colsample_bytree=1, max_depth=5)
xgb.fit(X_train_s,Y_train.squeeze().values)

#calculate and print scores for the model
y_train_preds = xgb.predict(X_train_s)
y_test_preds = xgb.predict(X_test_s)
```

```
print(xgb.score(X_test_s,Y_test))
print(xgb.score(X_train_s,Y_train))
```

```
     0.85103668261563
     0.8833743505605688
```

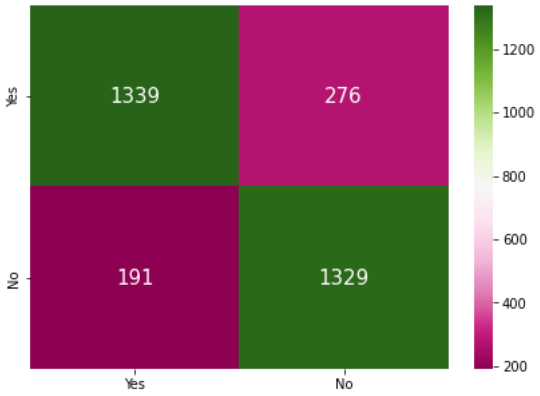
```
print('The Test accuracy of the XGB is',metrics.accuracy_score(Y_test,y_test_preds))
print('The Train accuracy of the XGB is',metrics.accuracy_score(y_train_preds,Y_train))
```

```
The Test accuracy of the XGB is 0.85103668261563
The Train accuracy of the XGB is 0.8833743505605688
```

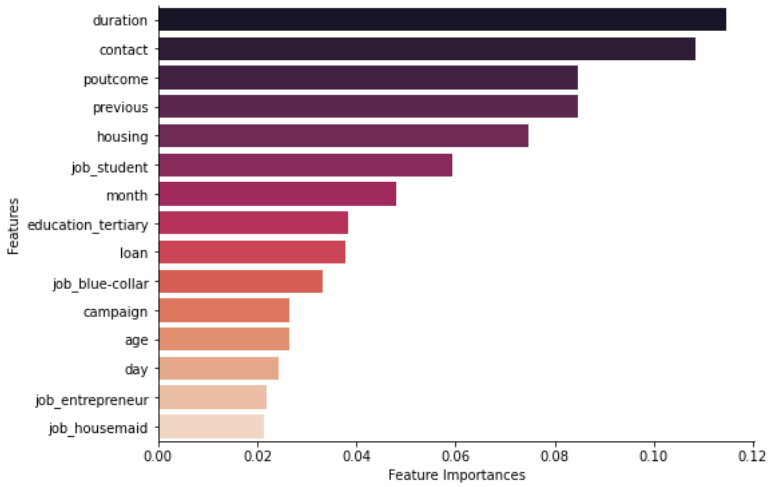
```
print(classification_report(Y_test,y_test_preds))
print(confusion_matrix(Y_test,y_test_preds))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.83      0.85      1615
               1       0.83      0.87      0.85      1520

        accuracy                           0.85      3135
       macro avg       0.85      0.85      0.85      3135
    weighted avg       0.85      0.85      0.85      3135

    [[1339  276]
     [ 191 1329]]
```

```
plt.figure(figsize=(7,5))
sns.heatmap(confusion_matrix(Y_test,y_test_preds),annot=True,cmap="PiYG",
            fmt="d",cbar=True,xticklabels=['Yes','No'],yticklabels=['Yes','No'],
            annot_kws={"fontsize":15})
plt.show()
```




```
# using random forest here to get feature importances
plt.figure(figsize=(8,6))
importances= xgb.feature_importances_
feature_importances= pd.Series(importances, index=X_train.columns).sort_values(ascending=False)
sns.barplot(x=feature_importances[:15], y=feature_importances.index[:15], palette="rocket")
sns.despine()
plt.xlabel("Feature Importances")
plt.ylabel("Features")
plt.show()
```




```
rf1 =RandomForestClassifier(random_state=0,n_estimators=200,max_features=25,max_depth=10,min_samples_leaf=50)
rf1.fit(X_train_s,Y_train.squeeze().values)

#calculate and print scores for the model for top 15 features
y_train_preds = rf1.predict(X_train_s)
y_test_preds = rf1.predict(X_test_s)
```

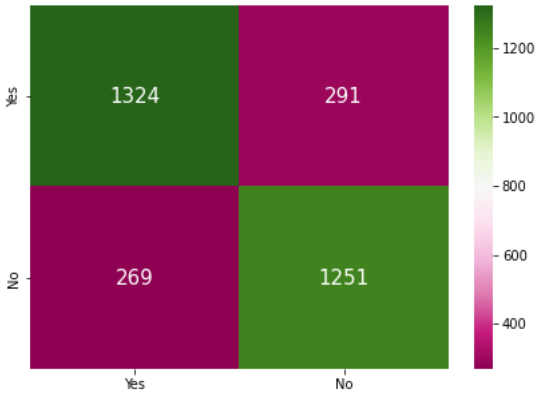
```
print(rf1.score(X_test_s,Y_test))
print(rf1.score(X_train_s,Y_train))
```

```
0.8213716108452951
0.8311457478807766
```

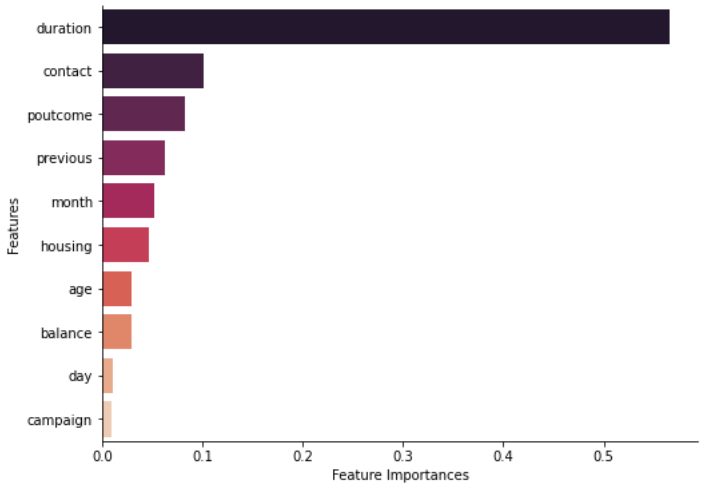
```
print(classification_report(Y_test,y_test_preds))
print(confusion_matrix(Y_test,y_test_preds))
```

```
                  precision    recall  f1-score   support

               0       0.83      0.82      0.83      1615
               1       0.81      0.82      0.82      1520

        accuracy                           0.82      3135
       macro avg       0.82      0.82      0.82      3135
    weighted avg       0.82      0.82      0.82      3135

    [[1324  291]
     [ 269 1251]]
```

```
plt.figure(figsize=(7,5))
sns.heatmap(confusion_matrix(Y_test,y_test_preds),annot=True,cmap="PiYG",
            fmt="d",cbar=True,xticklabels=['Yes','No'],yticklabels=['Yes','No'],
            annot_kws={"fontsize":15})
plt.show()
```




```
plt.figure(figsize=(8,6))
importances= rf1.feature_importances_
feature_importances= pd.Series(importances, index=X_train.columns).sort_values(ascending=False)
sns.barplot(x=feature_importances[0:10], y=feature_importances.index[0:10], palette="rocket")
sns.despine()
plt.xlabel("Feature Importances")
plt.ylabel("Features")
plt.show()
```



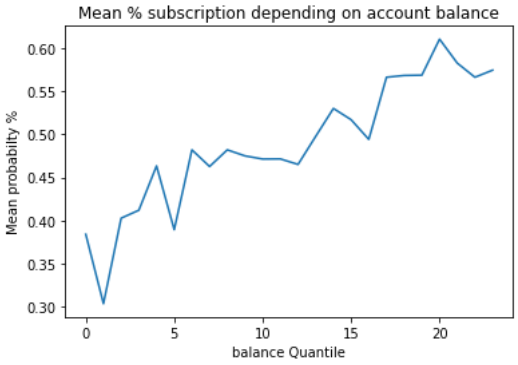

```
## XG Boost
gs_svm_scores = cross_val_score(xgb, X=X_train_s, y=Y_train, cv=5,scoring='accuracy', n_jobs=-1)
print('CV Mean Accuracy: {0:.1f}%'.format(np.mean(gs_svm_scores)*100))
```

```
CV Mean Accuracy: 84.5%
```

```
## Random Forest
gs_svm_scores = cross_val_score(rf1, X=X_train_s, y=Y_train, cv=5,scoring='accuracy', n_jobs=-1)
print('CV Mean Accuracy: {0:.1f}%'.format(np.mean(gs_svm_scores)*100))
```

```
CV Mean Accuracy: 81.7%
```

```
#lets create one dataframe df3
df3 = pd.DataFrame()
df3['balance']=df['balance']

df3['deposit']=df['deposit']
df3['balance_quantile'] = pd.qcut(df3['balance'], q=25, labels=False, duplicates ='drop')

#group by 'balance_buckets' and find average campaign outcome per balance bucket
mean_deposit = df3.groupby(['balance_quantile'])['deposit'].mean()

#plot
plt.plot(mean_deposit.index, mean_deposit.values)
plt.title('Mean % subscription depending on account balance')
plt.xlabel('balance Quantile')
plt.ylabel('Mean probabilty %')
plt.show()
```



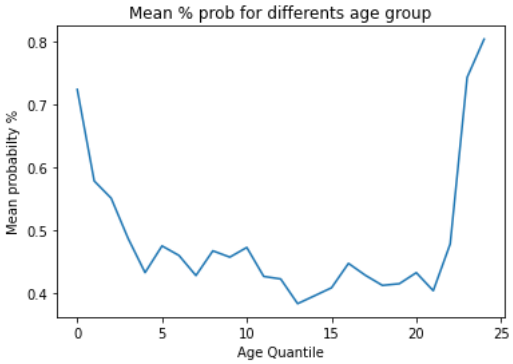

```
df3['balance_group'] = pd.qcut(df3['balance'], q=25, precision=0,duplicates ='drop')
mean_deposit = df3.groupby(['balance_group'])['deposit'].mean()
mean_deposit.sort_values(ascending=False).head(10)
```

```
balance_group
(2879.0, 3586.0]     0.610048
(3586.0, 4721.0]     0.582339
(7102.0, 37127.0]    0.574163
(2360.0, 2879.0]     0.568345
(1938.0, 2360.0]     0.568019
(4721.0, 7102.0]     0.565947
(1610.0, 1938.0]     0.565947
(948.0, 1133.0]      0.529833
(1133.0, 1337.0]     0.516827
(805.0, 948.0]       0.497608
Name: deposit, dtype: float64
```

```
for i in range(0,24):
    mean=df3[df3['balance_quantile']==i]['deposit'].mean()
    print(i,"--" ,df3[df3['balance_quantile']==i]['balance_group'].values[0]," Mean prob", mean)
```

```
0 -- (-1.0, 2.0]  Mean prob 0.38416075650118203
1 -- (2.0, 32.0]  Mean prob 0.30392156862745096
2 -- (32.0, 76.0]  Mean prob 0.4028436018957346
3 -- (76.0, 121.0]  Mean prob 0.41204819277108434
4 -- (121.0, 169.0]  Mean prob 0.46335697399527187
5 -- (169.0, 222.0]  Mean prob 0.38954869358669836
6 -- (222.0, 280.0]  Mean prob 0.4819277108433735
7 -- (280.0, 336.0]  Mean prob 0.46265060240963857
8 -- (336.0, 408.0]  Mean prob 0.4819277108433735
9 -- (408.0, 488.0]  Mean prob 0.47494033412887826
10 -- (488.0, 577.0]  Mean prob 0.47129186602870815
11 -- (577.0, 679.0]  Mean prob 0.4714285714285714
12 -- (679.0, 805.0]  Mean prob 0.4650602409638554
13 -- (805.0, 948.0]  Mean prob 0.49760765550239233
14 -- (948.0, 1133.0]  Mean prob 0.5298329355608592
15 -- (1133.0, 1337.0]  Mean prob 0.5168269230769231
16 -- (1337.0, 1610.0]  Mean prob 0.49403341288782815
17 -- (1610.0, 1938.0]  Mean prob 0.565947242206235
18 -- (1938.0, 2360.0]  Mean prob 0.568019093078759
19 -- (2360.0, 2879.0]  Mean prob 0.5683453237410072
20 -- (2879.0, 3586.0]  Mean prob 0.6100478468899522
21 -- (3586.0, 4721.0]  Mean prob 0.5823389021479713
22 -- (4721.0, 7102.0]  Mean prob 0.565947242206235
23 -- (7102.0, 37127.0]  Mean prob 0.5741626794258373
```

```
df3['age']=df['age']
df3['age_quantile'] = pd.qcut(df3['age'], q=25, labels=False, duplicates ='drop')

#group by 'age_quantile' and find average campaign outcome per balance bucket
mean_deposit = df3.groupby(['age_quantile'])['deposit'].mean()

#plot
plt.plot(mean_deposit.index, mean_deposit.values)
plt.title('Mean % prob for differents age group')
plt.xlabel('Age Quantile')
plt.ylabel('Mean probabilty %')
plt.show()
```



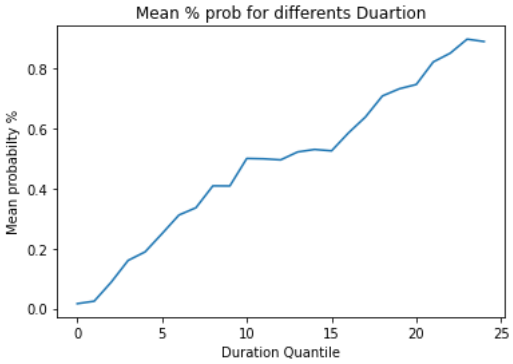

```
df3['age_group'] = pd.qcut(df3['age'], q=25, precision=0,duplicates ='drop')
mean_deposit = df3.groupby(['age_group'])['deposit'].mean()
mean_deposit.sort_values(ascending=False).head(10)
```

```
age_group
(65.0, 95.0]    0.803030
(59.0, 65.0]    0.742547
(17.0, 25.0]    0.723256
(25.0, 27.0]    0.578231
(27.0, 29.0]    0.550877
(29.0, 30.0]    0.487119
(57.0, 59.0]    0.478395
(31.0, 32.0]    0.475225
(36.0, 37.0]    0.472779
(34.0, 35.0]    0.467442
Name: deposit, dtype: float64
```

```
for i in range(0,25):
    mean=df3[df3['age_quantile']==i]['deposit'].mean()
    print(i,"--" ,df3[df3['age_quantile']==i]['age_group'].values[0]," Mean prob", mean)
```

```
0 -- (17.0, 25.0]  Mean prob 0.7232558139534884
1 -- (25.0, 27.0]  Mean prob 0.5782312925170068
2 -- (27.0, 29.0]  Mean prob 0.5508771929824562
3 -- (29.0, 30.0]  Mean prob 0.48711943793911006
4 -- (30.0, 31.0]  Mean prob 0.4329004329004329
5 -- (31.0, 32.0]  Mean prob 0.4752252252252252
6 -- (32.0, 33.0]  Mean prob 0.4601366742596811
7 -- (33.0, 34.0]  Mean prob 0.428246013667426
8 -- (34.0, 35.0]  Mean prob 0.46744186046511627
9 -- (35.0, 36.0]  Mean prob 0.45742092457420924
10 -- (36.0, 37.0]  Mean prob 0.47277936962750716
11 -- (37.0, 38.0]  Mean prob 0.42686567164179107
12 -- (38.0, 39.0]  Mean prob 0.4228395061728395
13 -- (39.0, 41.0]  Mean prob 0.3836805555555556
14 -- (41.0, 42.0]  Mean prob 0.39622641509433965
15 -- (42.0, 44.0]  Mean prob 0.4088888888888889
16 -- (44.0, 46.0]  Mean prob 0.4475374732334047
17 -- (46.0, 48.0]  Mean prob 0.42857142857142855
18 -- (48.0, 50.0]  Mean prob 0.41265822784810124
19 -- (50.0, 52.0]  Mean prob 0.41530054644808745
20 -- (52.0, 54.0]  Mean prob 0.4327956989247312
21 -- (54.0, 57.0]  Mean prob 0.40417457305502846
22 -- (57.0, 59.0]  Mean prob 0.4783950617283951
23 -- (59.0, 65.0]  Mean prob 0.7425474254742548
24 -- (65.0, 95.0]  Mean prob 0.803030303030303
```

```
df3['duration']=df['duration']
df3['duration_quantile'] = pd.qcut(df3['duration'], q=25, labels=False, duplicates ='drop')

#group by 'age_quantile' and find average campaign outcome per balance bucket
mean_deposit = df3.groupby(['duration_quantile'])['deposit'].mean()

#plot
plt.plot(mean_deposit.index, mean_deposit.values)
plt.title('Mean % prob for differents Duartion')
plt.xlabel('Duration Quantile')
plt.ylabel('Mean probabilty %')
plt.show()
```




```
df3['duration_group'] = pd.qcut(df3['duration'], q=25, precision=0,duplicates ='drop')
mean_deposit = df3.groupby(['duration_group'])['deposit'].mean()
print(mean_deposit.sort_values(ascending=False).head(10))
```

```
duration_group
(910.0, 1141.0]     0.897375
(1141.0, 2775.0]    0.889423
(763.0, 910.0]      0.850602
(660.0, 763.0]      0.821429
(580.0, 660.0]      0.746411
(510.0, 580.0]      0.732057
(448.0, 510.0]      0.708134
(397.0, 448.0]      0.638095
(355.0, 397.0]      0.585132
(293.0, 323.0]      0.529976
Name: deposit, dtype: float64
```

```
for i in range(0,25):
    mean=df3[df3['duration_quantile']==i]['deposit'].mean()
    print(i,"--" ,df3[df3['duration_quantile']==i]['duration_group'].values[0]," Mean prob", mean)
```

```
 0 -- (1.0, 45.0]  Mean prob 0.016317016317016316
 1 -- (45.0, 68.0]  Mean prob 0.02457002457002457
 2 -- (68.0, 86.0]  Mean prob 0.08767772511848342
 3 -- (86.0, 102.0]  Mean prob 0.16028708133971292
 4 -- (102.0, 119.0]  Mean prob 0.18867924528301888
 5 -- (119.0, 135.0]  Mean prob 0.24939467312348668
 6 -- (135.0, 151.0]  Mean prob 0.3115942028985507
 7 -- (151.0, 166.0]  Mean prob 0.33573141486810554
 8 -- (166.0, 185.0]  Mean prob 0.408675799086758
 9 -- (185.0, 205.0]  Mean prob 0.4083129584352078
 10 -- (205.0, 225.0]  Mean prob 0.5
 11 -- (225.0, 245.0]  Mean prob 0.49876543209876545
 12 -- (245.0, 266.0]  Mean prob 0.495260663507109
 13 -- (266.0, 293.0]  Mean prob 0.5216346153846154
 14 -- (293.0, 323.0]  Mean prob 0.5299760191846523
 15 -- (323.0, 355.0]  Mean prob 0.5253012048192771
 16 -- (355.0, 397.0]  Mean prob 0.5851318944844125
 17 -- (397.0, 448.0]  Mean prob 0.638095238095238
 18 -- (448.0, 510.0]  Mean prob 0.7081339712918661
 19 -- (510.0, 580.0]  Mean prob 0.7320574162679426
 20 -- (580.0, 660.0]  Mean prob 0.7464114832535885
 21 -- (660.0, 763.0]  Mean prob 0.8214285714285714
 22 -- (763.0, 910.0]  Mean prob 0.8506024096385543
 23 -- (910.0, 1141.0]  Mean prob 0.8973747016706444
 24 -- (1141.0, 2775.0]  Mean prob 0.8894230769230769
```

```
#Import Library
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler,LabelEncoder,OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
import xgboost
```

```
#Dataframe
df1
```

	age	job	marital	education	default	balance	housing	loan	contact	da
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
11157	33	blue-collar	single	primary	no	1	yes	no	cellular	2
11158	39	services	married	secondary	no	733	no	no	unknown	1
11159	32	technician	single	secondary	no	29	no	no	cellular	1
11160	43	technician	married	secondary	no	0	no	yes	cellular	
11161	34	technician	married	secondary	no	0	no	no	cellular	

```
#Correct Index
df1=df1.reset_index()
df1.drop('index',axis=1,inplace=True)
```

```
#dataframe
df1
```

	age	job	marital	education	default	balance	housing	loan	contact	d
0	59	admin.	married	secondary	no	2343	yes	no	unknown	
1	56	admin.	married	secondary	no	45	no	no	unknown	
2	41	technician	married	secondary	no	1270	yes	no	unknown	
3	55	services	married	secondary	no	2476	yes	no	unknown	
4	54	admin.	married	tertiary	no	184	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
10444	33	blue-collar	single	primary	no	1	yes	no	cellular	1
10445	39	services	married	secondary	no	733	no	no	unknown	
10446	32	technician	single	secondary	no	29	no	no	cellular	
10447	43	technician	married	secondary	no	0	no	yes	cellular	
10448	34	technician	married	secondary	no	0	no	no	cellular	

```
#Binary Encoding
dic = {"yes":1,"no":0}
lst = ["loan","default","housing"]
for i in lst:
    df1[i] = df1[i].map(dic)


#Encode of target variable(deposit)
dic = {"yes":1,"no":0}
df1["deposit"] = df1["deposit"].map(dic)


df1
```

	age	job	marital	education	default	balance	housing	loan	contact	d
0	59	admin.	married	secondary	0	2343	1	0	unknown	
1	56	admin.	married	secondary	0	45	0	0	unknown	
2	41	technician	married	secondary	0	1270	1	0	unknown	
3	55	services	married	secondary	0	2476	1	0	unknown	
4	54	admin.	married	tertiary	0	184	0	0	unknown	
...	...	...	...	...	...	...	...	...	...	
10444	33	blue-collar	single	primary	0	1	1	0	cellular	
10445	39	services	married	secondary	0	733	0	0	unknown	
10446	32	technician	single	secondary	0	29	0	0	cellular	
10447	43	technician	married	secondary	0	0	0	1	cellular	
10448	34	technician	married	secondary	0	0	0	0	cellular	

```
#Ordinal Encoding
contact_list = df1['contact'].unique().tolist()
poutcome_list = ['success','unknown', 'other','failure']
month_list = df1['month'].unique().tolist()
month_list = list(reversed(month_list))


print(contact_list)
print(poutcome_list)
print(month_list)
```

```
['unknown', 'cellular', 'telephone']
['success', 'unknown', 'other', 'failure']
['sep', 'apr', 'mar', 'feb', 'jan', 'dec', 'nov', 'oct', 'aug', 'jul', 'jun', 'may']
```

```
contact_label=list(range(0,3,1))
poutcome_label=list(range(0,4,1))
month_label=list(range(0,12,1))


dic_contact=dict(zip(contact_list,contact_label))
dic_poutcome=dict(zip(poutcome_list,poutcome_label))
dic_month=dict(zip(month_list,month_label))


print(dic_contact)
print(dic_poutcome)
print(dic_month)
```

```
{'unknown': 0, 'cellular': 1, 'telephone': 2}
{'success': 0, 'unknown': 1, 'other': 2, 'failure': 3}
{'sep': 0, 'apr': 1, 'mar': 2, 'feb': 3, 'jan': 4, 'dec': 5, 'nov': 6, 'oct': 7, 'aug': 8, 'jul': 9, 'jun': 10, 'may': 11}
```

```
#mapping dict with ordinal categ features
df1["contact"] = df1["contact"].map(dic_contact)
df1["poutcome"] = df1["poutcome"].map(dic_poutcome)
df1["month"] = df1["month"].map(dic_month)


#X and Y split
X = df1.drop('deposit',axis=1)
Y = df1['deposit']


#StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=1)
for train_index,test_index in sss.split(X,Y):
    train_df = df1.loc[train_index]
    test_df = df1.loc[test_index]


#Train and Test dataset
X_train = train_df.drop("deposit",axis=1)
Y_train = train_df['deposit']

X_test = test_df.drop("deposit",axis=1)
Y_test = test_df['deposit']


X_train
```

	age	job	marital	education	default	balance	housing	loan	contact	da
5461	28	blue-collar	married	primary	0	674	1	0	2	1
4220	36	blue-collar	married	secondary	0	324	1	1	1	1
5530	56	technician	divorced	secondary	0	1480	1	1	1	
4249	31	housemaid	single	primary	0	26965	0	0	1	2
9514	30	blue-collar	married	secondary	0	177	1	0	1	
...	...	...	...	...	...	...	...	...	...	...
8100	34	blue-collar	married	primary	0	425	1	0	1	1
4223	27	technician	single	tertiary	0	11862	0	0	1	2
343	26	self-employed	single	secondary	0	551	0	0	1	
4449	41	blue-collar	married	secondary	0	5517	1	0	1	1
2983	76	retired	married	primary	0	2223	0	0	2	

```
## Tranformer


trf1 = ColumnTransformer([
    ('ohe',OneHotEncoder(sparse=False,handle_unknown='ignore'),[1,2,3])
],remainder='passthrough')


trf2 = ColumnTransformer(
    transformers=[('scaler', StandardScaler(), [0,-1])],
    remainder='passthrough'
)


trf3 = xgboost.XGBClassifier(n_estimators=80, learning_rate=0.1, gamma=0, subsample=0.75,
                             colsample_bytree=1, max_depth=5)


pipe = Pipeline([
    ('trf1',trf1),
    ('trf2',trf2),
    ('trf3',trf3)
])


#stpes of pipes
pipe.steps
```

```
     [('trf1', ColumnTransformer(remainder='passthrough',
                         transformers=[('ohe',
```