Importing sufficient Libararies

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
df=pd.read_csv('/content/Bank Marketing.csv')
df
```

	Age	Job	Marital Status	Education	Credit	Balance (euros)	Housing Loan	Personal Loan	Contact	L Cont
0	58	management	married	tertiary	no	2143	yes	no	unknown	
1	44	technician	single	secondary	no	29	yes	no	unknown	
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	
4	33	unknown	single	unknown	no	1	no	no	unknown	
...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	no	825	no	no	cellular	
45207	71	retired	divorced	primary	no	1729	no	no	cellular	
45208	72	retired	married	secondary	no	5715	no	no	cellular	
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	

45211 rows × 17 columns

```
#Number of Rows and columns
df.shape
```

```
(45211, 17)
```

```
#first 5 observation print
df.head()
```

	Age	Job	Marital Status	Education	Credit	Balance (euros)	Housing Loan	Personal Loan	Contact	Last Contact Day
0	58	management	married	tertiary	no	2143	yes	no	unknown	5
1	44	technician	single	secondary	no	29	yes	no	unknown	5
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5

```
#last 5 observation print
df.tail()
```

	Age	Job	Marital Status	Education	Credit	Balance (euros)	Housing Loan	Personal Loan	Contact	L Cont
45206	51	technician	married	tertiary	no	825	no	no	cellular	
45207	71	retired	divorced	primary	no	1729	no	no	cellular	
45208	72	retired	married	secondary	no	5715	no	no	cellular	
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	

```
#Column Heading print
df.columns
```

```
Index(['Age', 'Job', 'Marital Status', 'Education', 'Credit',
       'Balance (euros)', 'Housing Loan', 'Personal Loan', 'Contact',
       'Last Contact Day', 'Last Contact Month', 'Last Contact Duration',
       'Campaign', 'Pdays', 'Previous', 'Poutcome', 'Subscription'],
      dtype='object')
```

```
#Each column types
df.dtypes
```

```
Age                       int64
Job                      object
Marital Status           object
Education                object
Credit                   object
Balance (euros)           int64
Housing Loan             object
Personal Loan            object
Contact                  object
Last Contact Day          int64
Last Contact Month       object
Last Contact Duration     int64
Campaign                  int64
Pdays                     int64
Previous                  int64
Poutcome                 object
```

```
Subscription                int64
dtype: object
```

```
#Information on features
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    45211 non-null  int64
 1   Job                    45211 non-null  object
 2   Marital Status         45211 non-null  object
 3   Education              45211 non-null  object
 4   Credit                 45211 non-null  object
 5   Balance (euros)        45211 non-null  int64
 6   Housing Loan           45211 non-null  object
 7   Personal Loan          45211 non-null  object
 8   Contact                45211 non-null  object
 9   Last Contact Day       45211 non-null  int64
 10  Last Contact Month     45211 non-null  object
 11  Last Contact Duration  45211 non-null  int64
 12  Campaign               45211 non-null  int64
 13  Pdays                  45211 non-null  int64
 14  Previous               45211 non-null  int64
 15  Poutcome               45211 non-null  object
 16  Subscription           45211 non-null  int64
dtypes: int64(8), object(9)
memory usage: 5.9+ MB
```

```
#Mathematical Correlation
df.describe()
```

	Age	Balance (euros)	Last Contact Day	Last Contact Duration	Campaign	Pdays	Pr
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0

```
#To find Missing values
df.isna().sum()
```

```
Age                      0
Job                      0
Marital Status           0
Education                0
Credit                   0
Balance (euros)          0
```

```
Housing Loan              0
Personal Loan             0
Contact                   0
Last Contact Day          0
Last Contact Month        0
Last Contact Duration     0
Campaign                  0
Pdays                     0
Previous                  0
Poutcome                  0
Subscription              0
dtype: int64
```

Each string column Unique Values

```
df['Job'].unique()
```

```
array(['management', 'technician', 'entrepreneur', 'blue-collar',
       'unknown', 'retired', 'admin.', 'services', 'self-employed',
       'unemployed', 'housemaid', 'student'], dtype=object)
```

```
df['Marital Status'].unique()
```

```
array(['married', 'single', 'divorced'], dtype=object)
```

```
df['Education'].unique()
```

```
array(['tertiary', 'secondary', 'unknown', 'primary'], dtype=object)
```

```
df['Credit'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
df['Housing Loan'].unique()
```

```
array(['yes', 'no'], dtype=object)
```

```
df['Personal Loan'].unique()
```

```
array(['no', 'yes'], dtype=object)
```

```
df['Contact'].unique()
```

```
array(['unknown', 'cellular', 'telephone'], dtype=object)
```

```
df['Last Contact Month'].unique()
```

```
array(['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb',
       'mar', 'apr', 'sep'], dtype=object)
```

```
df['Poutcome'].unique()
```

```
array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

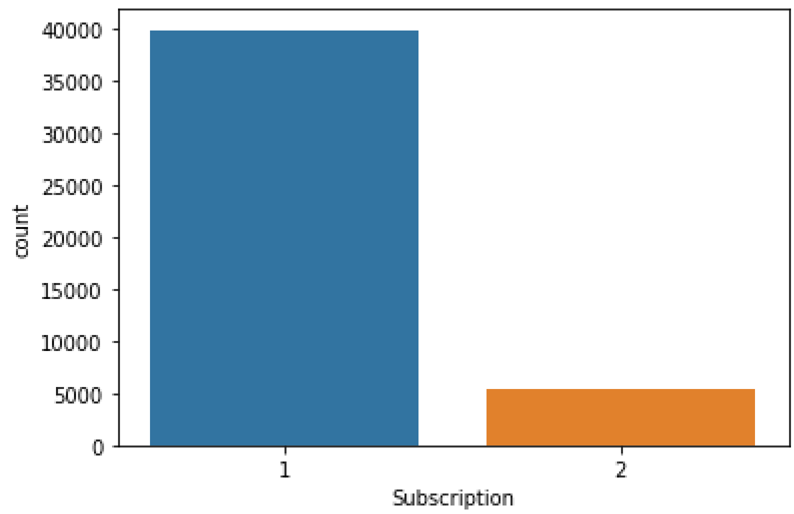
TARGET COLUMN VALUE COUNTS, GRAPH PLOT

```
#Subscription column value counts
df['Subscription'].value_counts()
```

```
1    39922
2     5289
Name: Subscription, dtype: int64
```

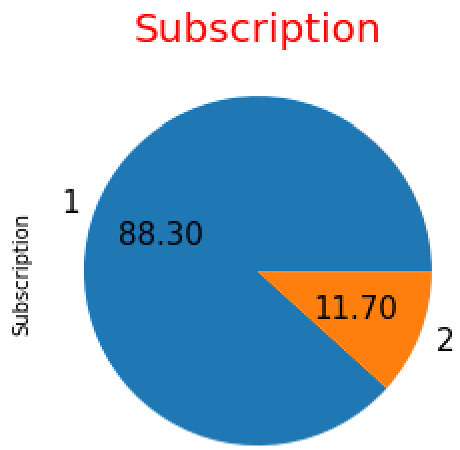
```
#Subscription column value counts graph
sns.countplot(x='Subscription',data=df)
```

```
<Axes: xlabel='Subscription', ylabel='count'>
```




```
#Subscription column Piechart
df['Subscription'].value_counts().plot(kind='pie',fontsize=15,autopct='%.2f')
plt.title('Subscription',fontsize=20,color='red')
```

```
Text(0.5, 1.0, 'Subscription')
```



EACH CATEGORICAL COLUMN VALUE COUNTS

```
#Job column value counts
df['Job'].value_counts()
```

```
blue-collar      9732
management       9458
technician       7597
admin.           5171
services         4154
retired          2264
self-employed    1579
entrepreneur     1487
unemployed       1303
housemaid        1240
student           938
unknown           288
Name: Job, dtype: int64
```

```
#Martial status column value counts
df['Marital Status'].value_counts()
```

```
married     27214
single      12790
divorced     5207
Name: Marital Status, dtype: int64
```

```
#Education column value counts
df['Education'].value_counts()
```

```
secondary    23202
tertiary     13301
primary       6851
unknown       1857
Name: Education, dtype: int64
```

```
#Credit column value counts
df['Credit'].value_counts()
```

```
no     44396
yes      815
Name: Credit, dtype: int64
```

```
#Housing Loan column value counts
df['Housing Loan'].value_counts()
```

```
yes    25130
no     20081
Name: Housing Loan, dtype: int64
```

```
#Personal Loan column value counts
df['Personal Loan'].value_counts()
```

```
no     37967
yes     7244
Name: Personal Loan, dtype: int64
```

```
#Contact column value counts
df['Contact'].value_counts()
```

```
cellular     29285
unknown      13020
telephone     2906
Name: Contact, dtype: int64
```

```
#Last Contact Month column value counts
df['Last Contact Month'].value_counts()
```

```
may    13766
jul     6895
aug     6247
jun     5341
nov     3970
apr     2932
feb     2649
jan     1403
oct      738
sep      579
mar      477
dec      214
Name: Last Contact Month, dtype: int64
```

```
#Poutcome column value counts
df['Poutcome'].value_counts()
```

```
unknown    36959
failure     4901
other       1840
success     1511
Name: Poutcome, dtype: int64
```

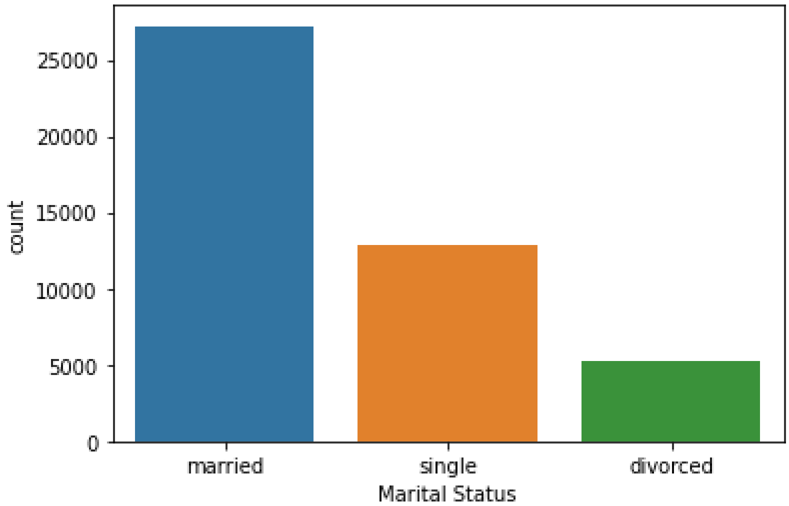
COUNTPLOT EACH CATEGORICAL COLUMN

```
#Job column value counts graph
sns.countplot(x='Job',data=df)
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'management'),
  Text(1, 0, 'technician'),
  Text(2, 0, 'entrepreneur'),
  Text(3, 0, 'blue-collar'),
  Text(4, 0, 'unknown'),
  Text(5, 0, 'retired'),
  Text(6, 0, 'admin.'),
  Text(7, 0, 'services'),
  Text(8, 0, 'self-employed'),
  Text(9, 0, 'unemployed'),
```

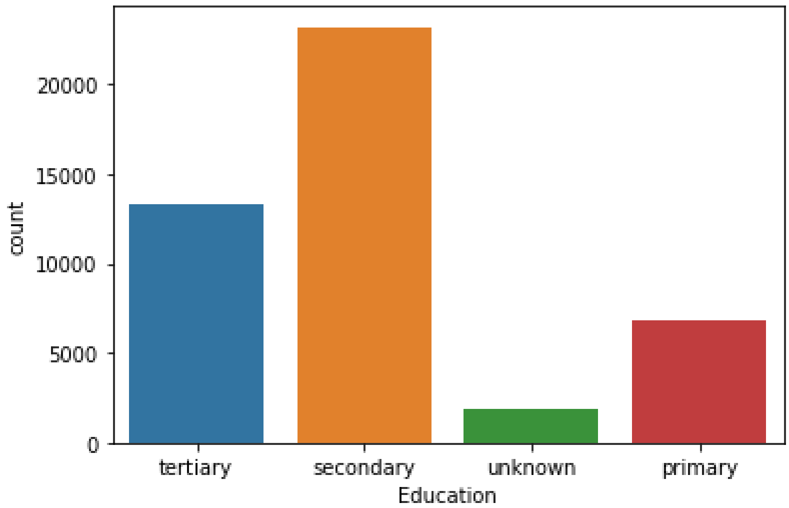
```
#Marital Status column value counts graph
sns.countplot(x='Marital Status',data=df)
```

```
<Axes: xlabel='Marital Status', ylabel='count'>
```




```
#Education column value counts graph
sns.countplot(x='Education',data=df)
```

```
<Axes: xlabel='Education', ylabel='count'>
```

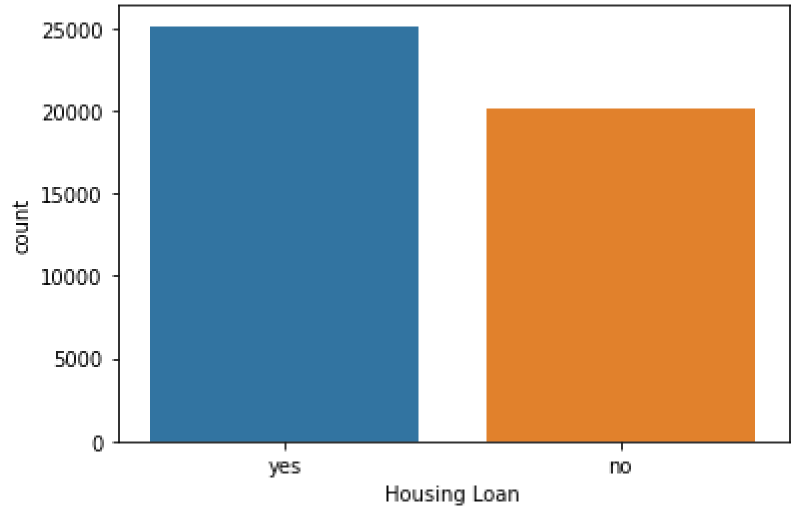



```
#Credit column value counts graph
sns.countplot(x='Credit',data=df)
```

<Axes: xlabel='Credit', ylabel='count'>

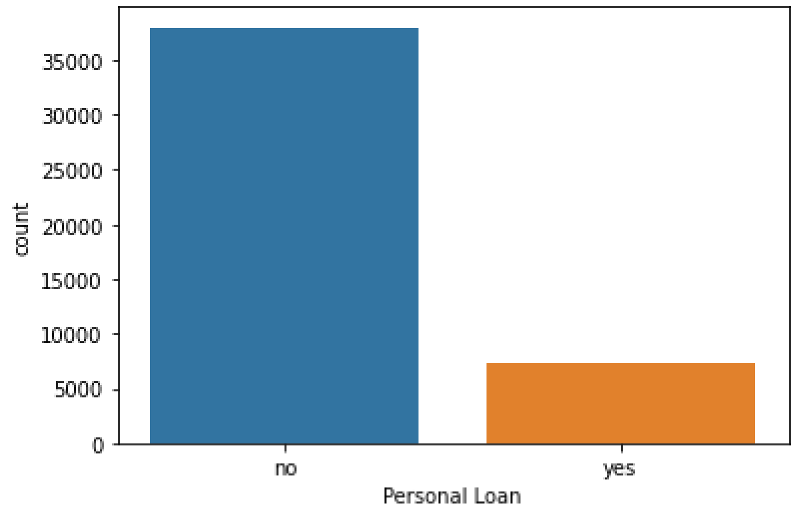

```
#Housing Loan column value counts graph
sns.countplot(x='Housing Loan',data=df)
```

<Axes: xlabel='Housing Loan', ylabel='count'>

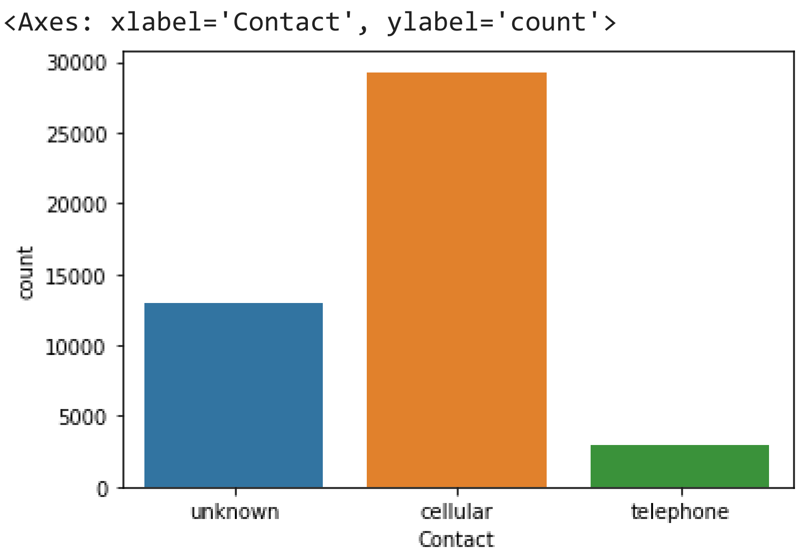



```
#Personal Loan column value counts graph
sns.countplot(x='Personal Loan',data=df)
```

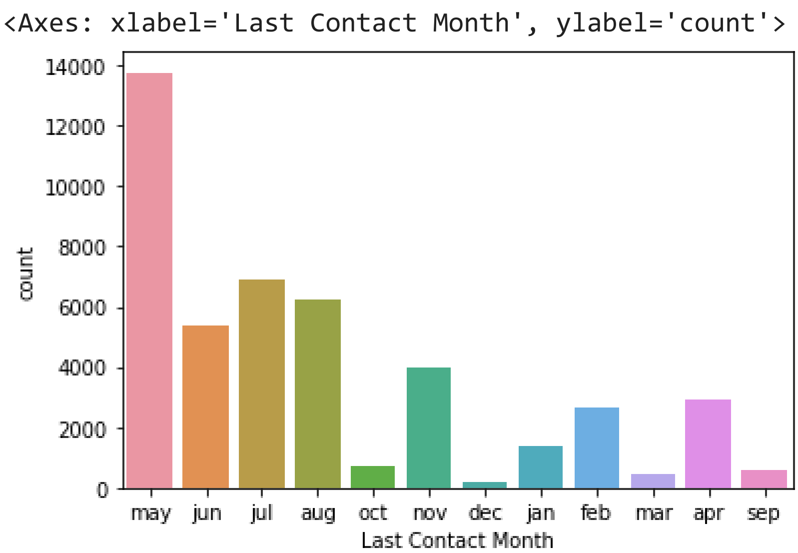
<Axes: xlabel='Personal Loan', ylabel='count'>



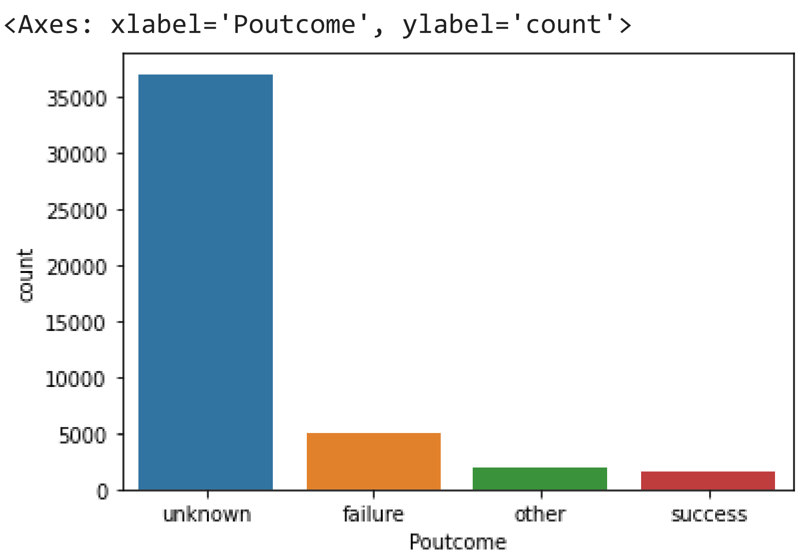

```
#Contact column value counts graph
sns.countplot(x='Contact',data=df)
```




```
#Last Contact Month column value counts graph
sns.countplot(x='Last Contact Month',data=df)
```



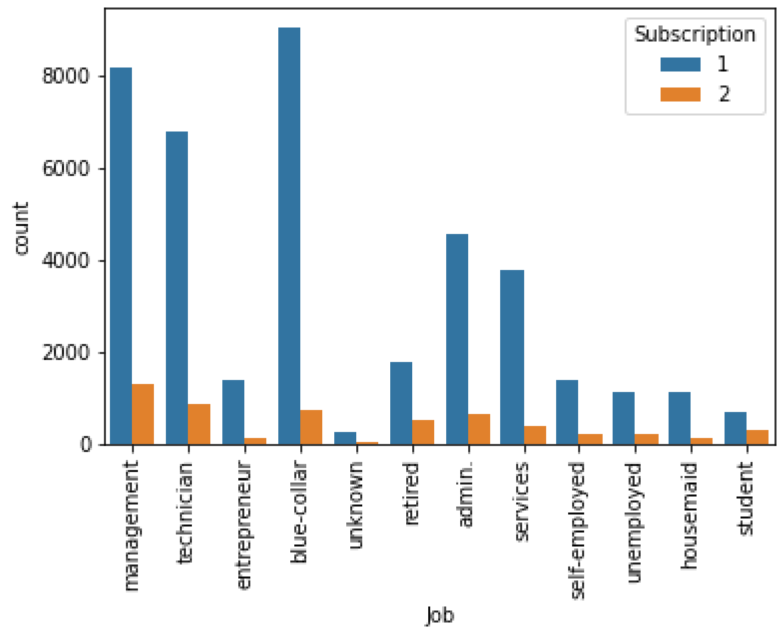

```
#Poutcome column value counts graph
sns.countplot(x='Poutcome',data=df)
```



Checking the relationship between categorical column and target column

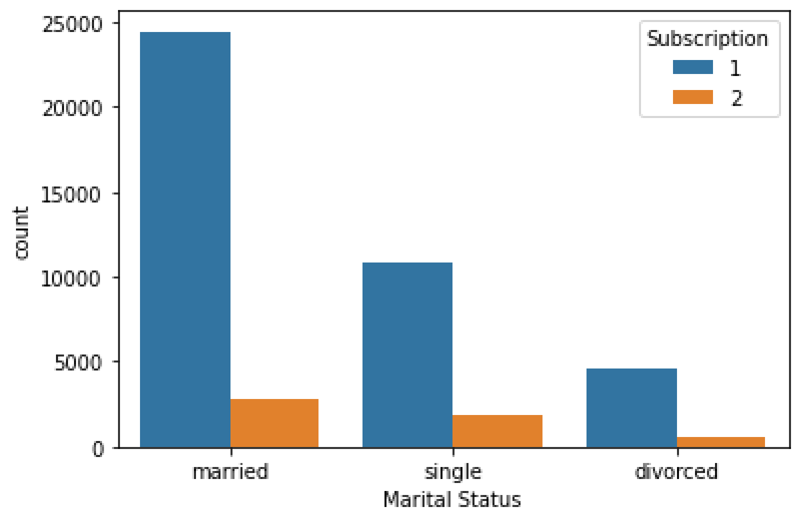
```
#Job column vs target column realtionship
sns.countplot(x='Job',data=df,hue='Subscription')
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'management'),
  Text(1, 0, 'technician'),
  Text(2, 0, 'entrepreneur'),
  Text(3, 0, 'blue-collar'),
  Text(4, 0, 'unknown'),
  Text(5, 0, 'retired'),
  Text(6, 0, 'admin.'),
  Text(7, 0, 'services'),
  Text(8, 0, 'self-employed'),
  Text(9, 0, 'unemployed'),
  Text(10, 0, 'housemaid'),
  Text(11, 0, 'student')])
```



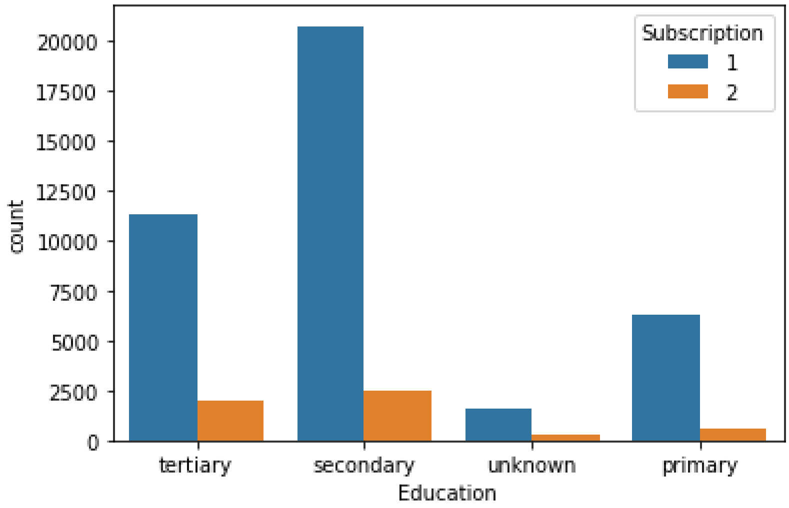

```
#Marital Status column vs target column realtionship
sns.countplot(x='Marital Status',data=df,hue='Subscription')
```

```
<Axes: xlabel='Marital Status', ylabel='count'>
```



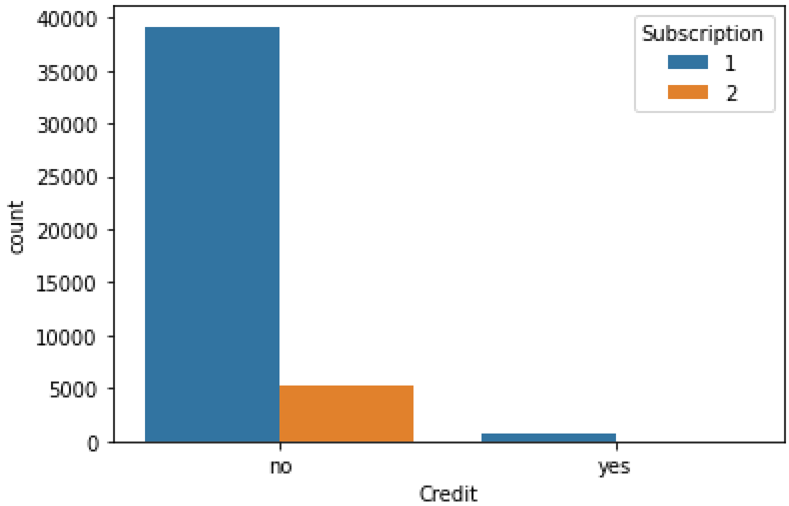
```
#Education column vs target column realtionship
sns.countplot(x='Education',data=df,hue='Subscription')
```

```
<Axes: xlabel='Education', ylabel='count'>
```




```
#Credit column vs target column realtionship
sns.countplot(x='Credit',data=df,hue='Subscription')
```

```
<Axes: xlabel='Credit', ylabel='count'>
```

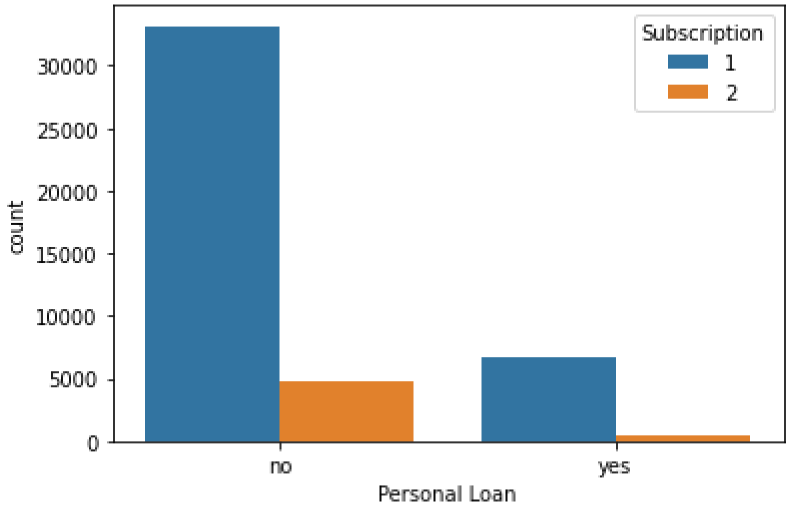



```
#Housing Loan column vs target column realtionship
sns.countplot(x='Housing Loan',data=df,hue='Subscription')
```

```
<Axes: xlabel='Housing Loan', ylabel='count'>
```

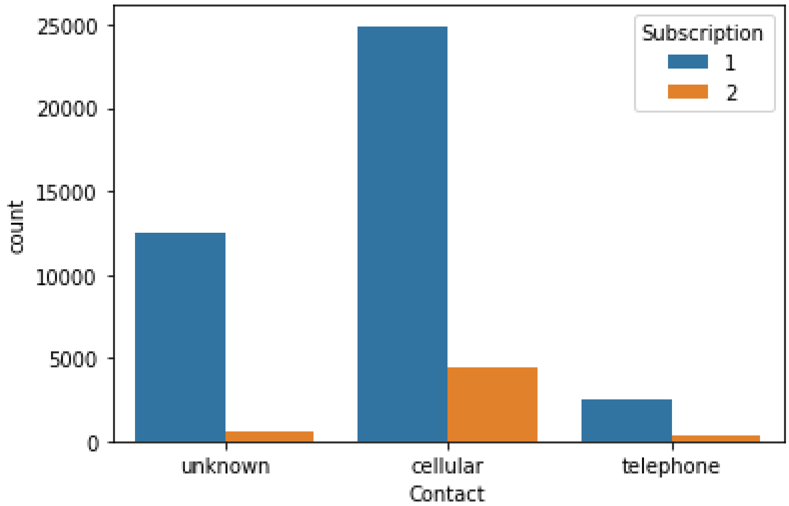
```
#Personal Loan column vs target column realtionship
sns.countplot(x='Personal Loan',data=df,hue='Subscription')
```

```
<Axes: xlabel='Personal Loan', ylabel='count'>
```




```
#Contact column vs target column realtionship
sns.countplot(x='Contact',data=df,hue='Subscription')
```

```
<Axes: xlabel='Contact', ylabel='count'>
```

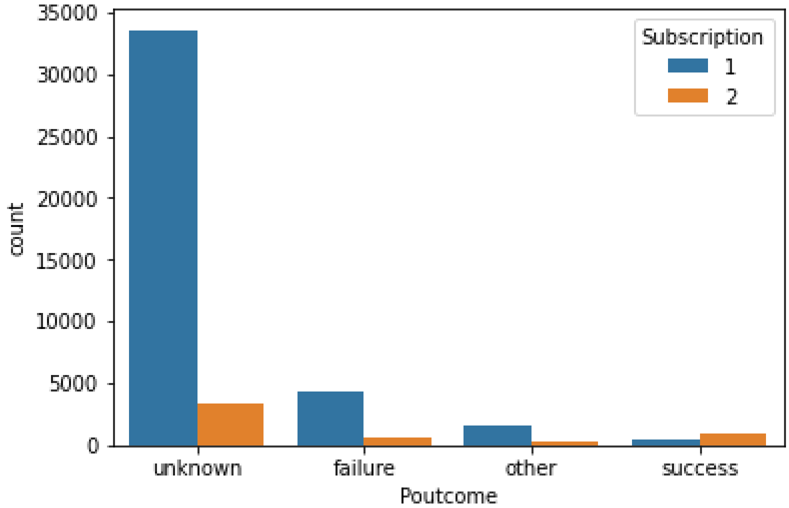



```
#Last Contact Month vs target column realtionship
sns.countplot(x='Last Contact Month',data=df,hue='Subscription')
```

```
<Axes: xlabel='Last Contact Month', ylabel='count'>
```


```
#Poutcome column vs target column realtionship
sns.countplot(x='Poutcome',data=df,hue='Subscription')
```

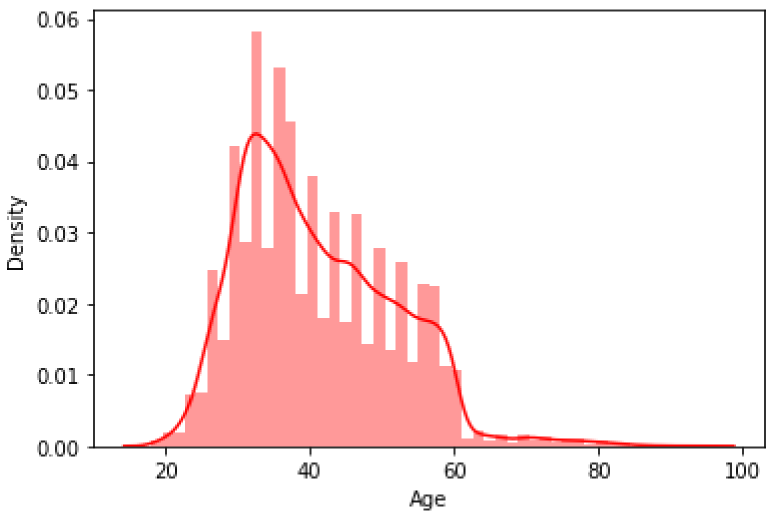
```
<Axes: xlabel='Poutcome', ylabel='count'>
```



DISTPLOT-depicts the variation in data distribution

```
#Age Column distribution Plot
sns.distplot(df['Age'],color='red')
```

```
<Axes: xlabel='Age', ylabel='Density'>
```

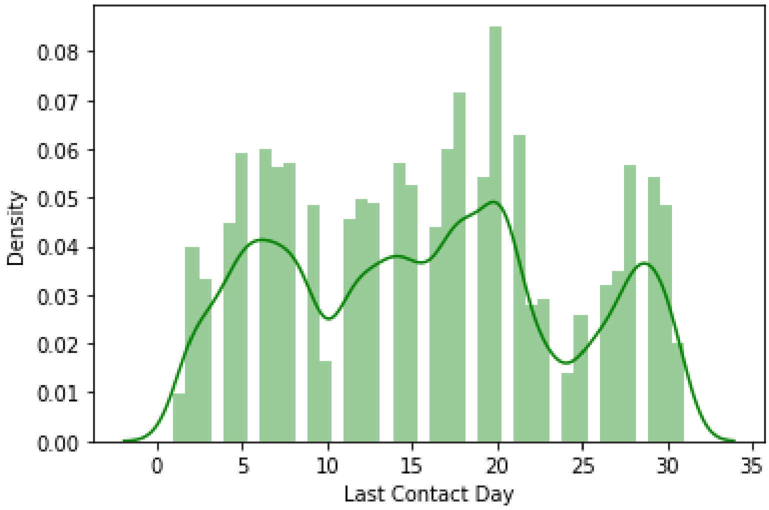



```
#Balance (euros) Column distribution Plot
sns.distplot(df['Balance (euros)'],color='blue')
```

<Axes: xlabel='Balance (euros)', ylabel='Density'>

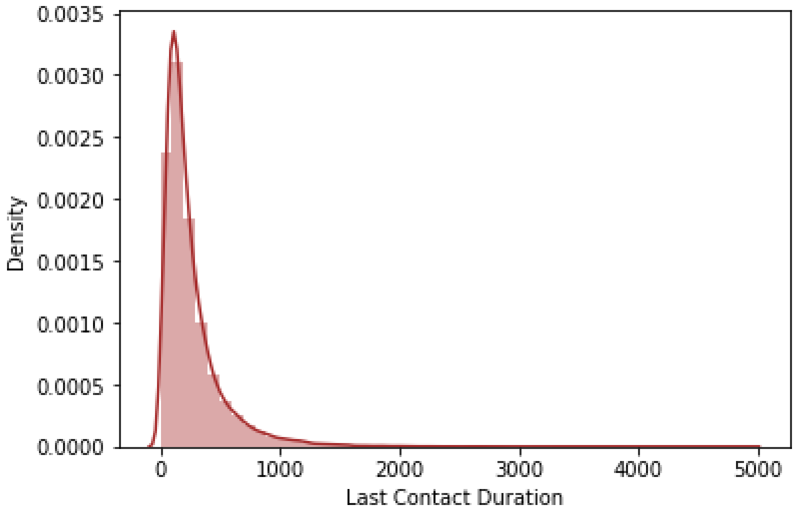
```
#Last Contact Day Column distribution Plot
sns.distplot(df['Last Contact Day'],color='green')
```

<Axes: xlabel='Last Contact Day', ylabel='Density'>




```
#Last Contact Duration Column distribution Plot
sns.distplot(df['Last Contact Duration'],color='brown')
```

<Axes: xlabel='Last Contact Duration', ylabel='Density'>

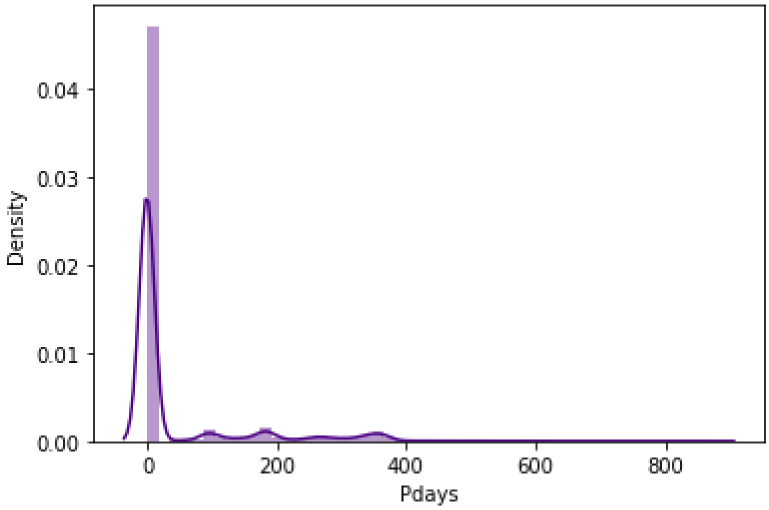



```
#Campaign Column distribution Plot
sns.distplot(df['Campaign'],color='black')
```

<Axes: xlabel='Campaign', ylabel='Density'>

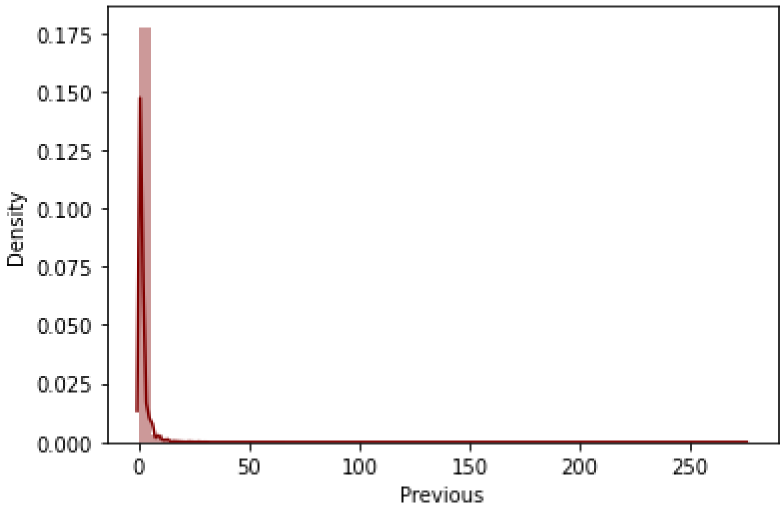

```
#Pdays Column distribution Plot
sns.distplot(df['Pdays'],color='indigo')
```

<Axes: xlabel='Pdays', ylabel='Density'>




```
#Previous Column distribution Plot
sns.distplot(df['Previous'],color='maroon')
```

<Axes: xlabel='Previous', ylabel='Density'>



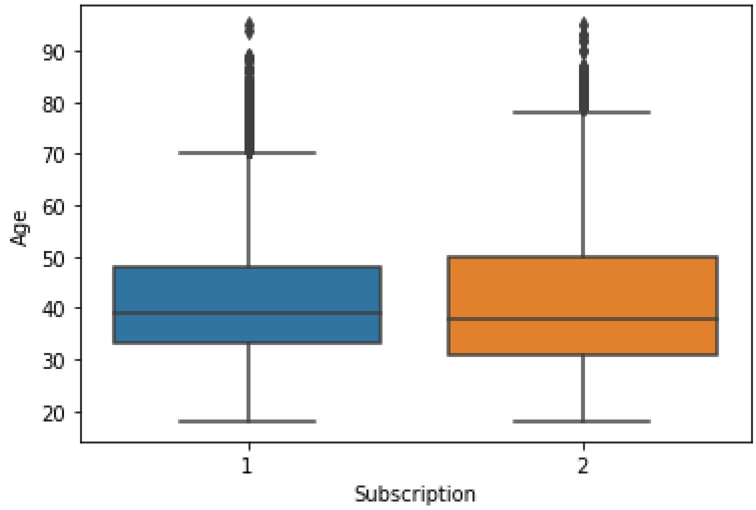

```
#Subscription Column distribution Plot
sns.distplot(df['Subscription'],color='grey')
```

```
<Axes: xlabel='Subscription', ylabel='Density'>
```

BOXPLOT

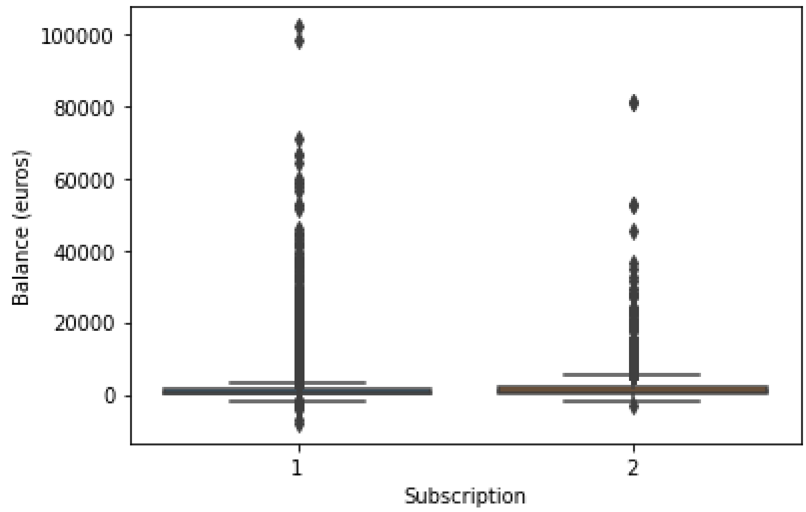
```
#How Age Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Age',data=df)
```

```
<Axes: xlabel='Subscription', ylabel='Age'>
```



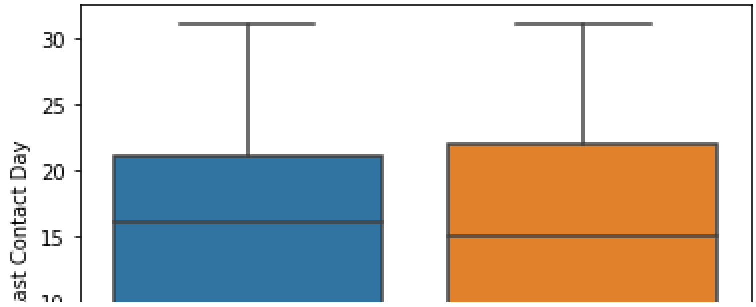

```
#How Balance (euros) Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Balance (euros)',data=df)
```

```
<Axes: xlabel='Subscription', ylabel='Balance (euros)'>
```



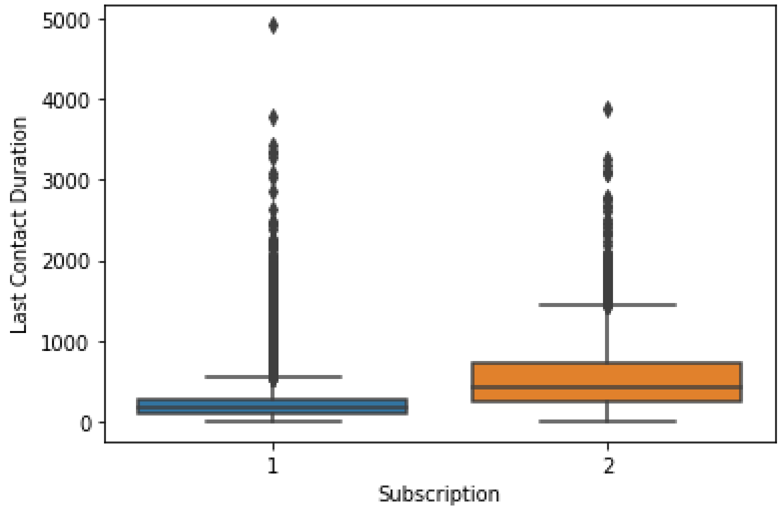

```
#How Last Contact Day Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Last Contact Day',data=df)
```

<Axes: xlabel='Subscription', ylabel='Last Contact Day'>



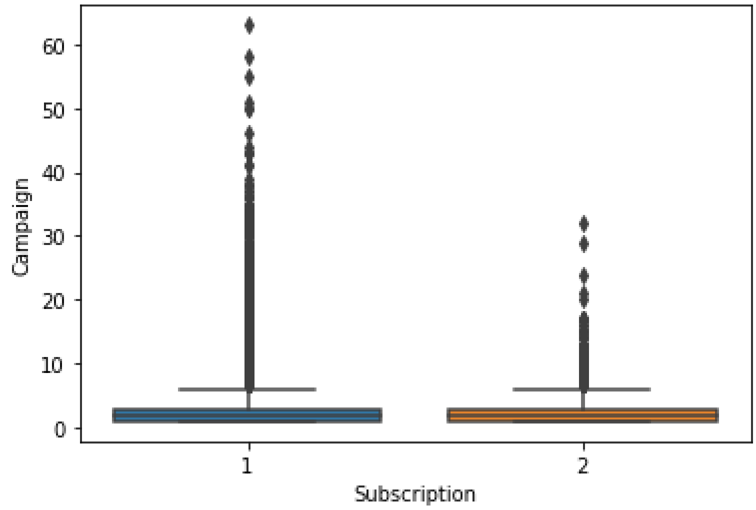

```
#How Last Contact Duration Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Last Contact Duration',data=df)
```

<Axes: xlabel='Subscription', ylabel='Last Contact Duration'>



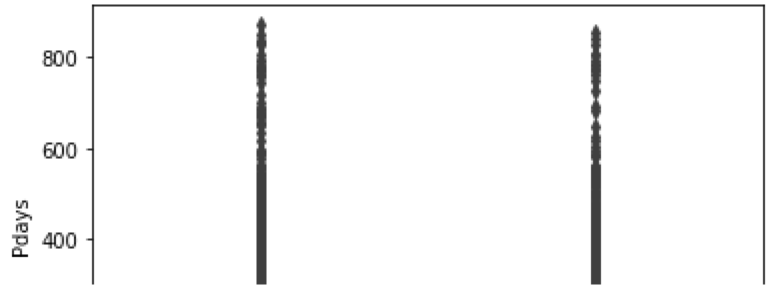

```
#How Campaign Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Campaign',data=df)
```

<Axes: xlabel='Subscription', ylabel='Campaign'>



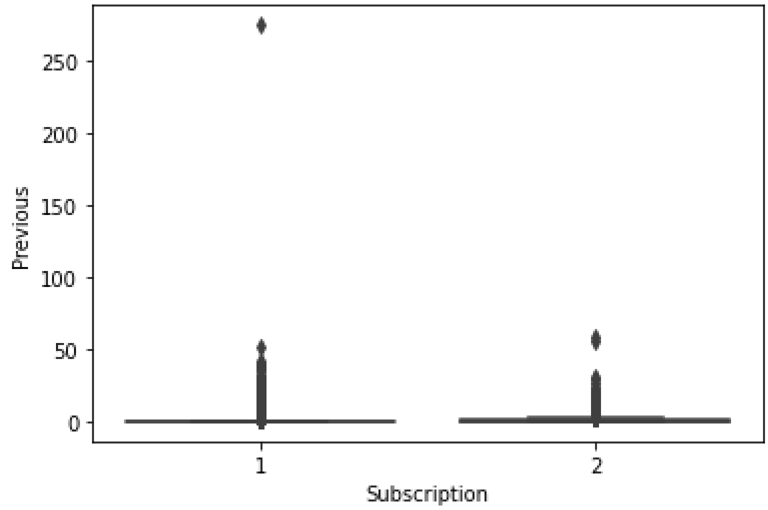

```
#How Pdays Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Pdays',data=df)
```

```
<Axes: xlabel='Subscription', ylabel='Pdays'>
```




```
#How Previous Column Affect Subsription Column
sns.boxplot(x='Subscription',y='Previous',data=df)
```

```
<Axes: xlabel='Subscription', ylabel='Previous'>
```



CORRELATION

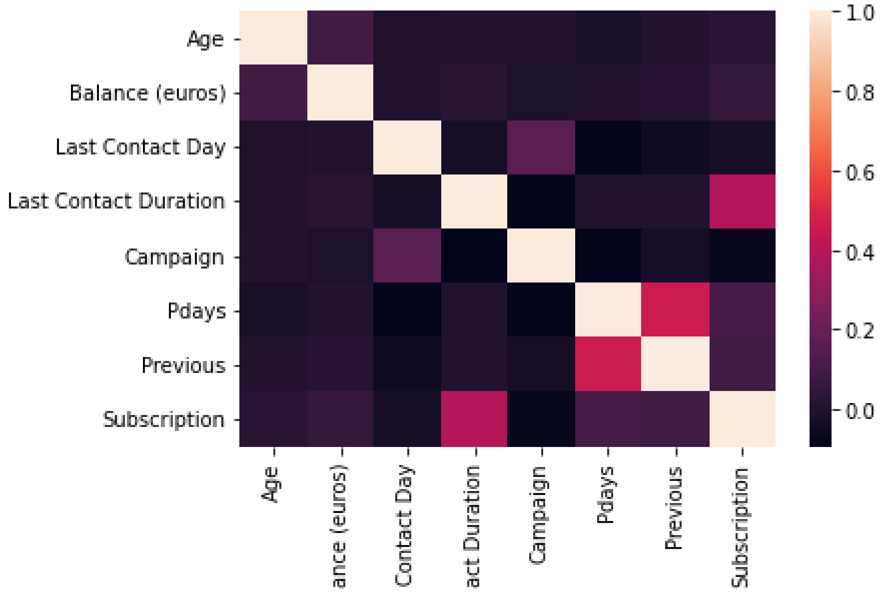
```
df.corr()
```

	Age	Balance (euros)	Last Contact Day	Last Contact Duration	Campaign	Pdays	Previous	Subs
Age	1.000000	0.097783	-0.009120	-0.004648	0.004760	-0.023758	0.001288	
Balance (euros)	0.097783	1.000000	0.004503	0.021560	-0.014578	0.003435	0.016674	
Last Contact Day	-0.009120	0.004503	1.000000	-0.030206	0.162490	-0.093044	-0.051710	-
Last Contact Duration	-0.004648	0.021560	-0.030206	1.000000	-0.084570	-0.001565	0.001203	
Campaign	0.004760	-0.014578	0.162490	-0.084570	1.000000	-0.088628	-0.032855	-
Pdays	-0.023758	0.003435	-0.093044	-0.001565	-0.088628	1.000000	0.454820	

HEATMAP-CORRELATION SHOWING

```
sns.heatmap(df.corr())
```

```
<Axes: >
```



Encoding string to Numeric using LabelEncoding

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df['Housing Loan']=le.fit_transform(df['Housing Loan'])
df['Housing Loan']
```

```
0        1
1        1
2        1
3        1
4        0
        ..
45206    0
45207    0
45208    0
45209    0
45210    0
Name: Housing Loan, Length: 45211, dtype: int64
```

```
df['Personal Loan']=le.fit_transform(df['Personal Loan'])
df['Personal Loan']
```

```
0        0
1        0
2        1
3        0
4        0
        ..
45206    0
45207    0
45208    0
45209    0
45210    0
Name: Personal Loan, Length: 45211, dtype: int64
```

```
df.dtypes
```

```
Age                         int64
Job                        object
Marital Status             object
Education                  object
Credit                     object
Balance (euros)             int64
Housing Loan                int64
Personal Loan               int64
Contact                    object
Last Contact Day            int64
Last Contact Month         object
Last Contact Duration       int64
Campaign                    int64
Pdays                       int64
Previous                    int64
Poutcome                   object
Subscription                int64
dtype: object
```

Encoding string to Numeric using GETDUMMIES

```
df1=pd.get_dummies(df[['Job','Marital Status','Education','Credit','Contact',
        'Last Contact Month','Poutcome']],drop_first=True)
df1
```

	Job_blue-collar	Job_entrepreneur	Job_housemaid	Job_management	Job_retired	Job_self-employed	Job
0	0	0	0	1	0	0	
1	0	0	0	0	0	0	
2	0	1	0	0	0	0	
3	1	0	0	0	0	0	
4	0	0	0	0	0	0	
...	...	...	...	...	...	...	
45206	0	0	0	0	0	0	
45207	0	0	0	0	1	0	
45208	0	0	0	0	1	0	
45209	1	0	0	0	0	0	
45210	0	1	0	0	0	0	

45211 rows × 33 columns

Concatination-combining

```
dfe=pd.concat([df,df1],axis=1)
dfe
```

	Age	Job	Marital Status	Education	Credit	Balance (euros)	Housing Loan	Personal Loan	Contact	L Cont
0	58	management	married	tertiary	no	2143	1	0	unknown	
1	44	technician	single	secondary	no	29	1	0	unknown	
2	33	entrepreneur	married	secondary	no	2	1	1	unknown	
3	47	blue-collar	married	unknown	no	1506	1	0	unknown	
4	33	unknown	single	unknown	no	1	0	0	unknown	
...	...	...	...	...	...	...	...	...	...	
45206	51	technician	married	tertiary	no	825	0	0	cellular	
45207	71	retired	divorced	primary	no	1729	0	0	cellular	
45208	72	retired	married	secondary	no	5715	0	0	cellular	
45209	57	blue-collar	married	secondary	no	668	0	0	telephone	
45210	37	entrepreneur	married	secondary	no	2971	0	0	cellular	

45211 rows × 50 columns

```
#DRopping Unwanted columns
dfe.drop(['Job','Marital Status','Education','Credit','Contact',
        'Last Contact Month','Poutcome'],axis=1,inplace=True)
dfe
```

	Age	Balance (euros)	Housing Loan	Personal Loan	Last Contact Day	Last Contact Duration	Campaign	Pdays	Previous	Subscript
0	58	2143	1	0	5	261	1	-1	0	

```
dfe.dtypes
```

```
Age                          int64
Balance (euros)              int64
Housing Loan                 int64
Personal Loan                int64
Last Contact Day             int64
Last Contact Duration        int64
Campaign                     int64
Pdays                        int64
Previous                     int64
Subscription                 int64
Job_blue-collar              uint8
Job_entrepreneur             uint8
Job_housemaid                uint8
Job_management               uint8
Job_retired                  uint8
Job_self-employed            uint8
Job_services                 uint8
Job_student                  uint8
Job_technician               uint8
Job_unemployed               uint8
Job_unknown                  uint8
Marital Status_married       uint8
Marital Status_single        uint8
Education_secondary          uint8
Education_tertiary           uint8
Education_unknown            uint8
Credit_yes                   uint8
Contact_telephone            uint8
Contact_unknown              uint8
Last Contact Month_aug       uint8
Last Contact Month_dec       uint8
Last Contact Month_feb       uint8
Last Contact Month_jan       uint8
Last Contact Month_jul       uint8
Last Contact Month_jun       uint8
Last Contact Month_mar       uint8
Last Contact Month_may       uint8
Last Contact Month_nov       uint8
Last Contact Month_oct       uint8
Last Contact Month_sep       uint8
Poutcome_other               uint8
Poutcome_success             uint8
Poutcome_unknown             uint8
dtype: object
```

```
#Seperate x
x=dfe.drop(['Subscription'],axis=1)
x
```

	Age	Balance (euros)	Housing Loan	Personal Loan	Last Contact Day	Last Contact Duration	Campaign	Pdays	Previous	Job_blue-collar
0	58	2143	1	0	5	261	1	-1	0	0
1	44	29	1	0	5	151	1	-1	0	0
2	33	2	1	1	5	76	1	-1	0	0
3	47	1506	1	0	5	92	1	-1	0	1
4	33	1	0	0	5	198	1	-1	0	0
...	...	...	...	...	...	...	...	...	...	...
45206	51	825	0	0	17	977	3	-1	0	0
45207	71	1729	0	0	17	456	2	-1	0	0
45208	72	5715	0	0	17	1127	5	184	3	0
45209	57	668	0	0	17	508	4	-1	0	1
45210	37	2971	0	0	17	361	2	188	11	0

45211 rows × 42 columns

```
#Seperate y
y=dfe['Subscription']
y
```

```
0        1
1        1
2        1
3        1
4        1
        ..
45206    2
45207    2
45208    2
45209    1
45210    1
Name: Subscription, Length: 45211, dtype: int64
```

Split-Train,Test

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
x_train
```

	Age	Balance (euros)	Housing Loan	Personal Loan	Last Contact Day	Last Contact Duration	Campaign	Pdays	Previous	Job_blue-collar
10747	36	0	0	0	17	153	4	-1	0	0
26054	56	196	0	0	19	312	3	-1	0	0
9125	46	0	1	0	5	83	2	-1	0	1
41659	41	3426	0	0	1	302	1	119	5	0
4443	38	0	1	0	20	90	1	-1	0	1
...	...	...	...	...	...	...	...	...	...	...
11284	44	1059	0	0	18	2093	1	-1	0	0
44732	23	508	0	0	8	210	1	92	1	0

x_test

	Age	Balance (euros)	Housing Loan	Personal Loan	Last Contact Day	Last Contact Duration	Campaign	Pdays	Previous	Job_blue-collar
3776	40	580	1	0	16	192	1	-1	0	1
9928	47	3644	0	0	9	83	2	-1	0	0
33409	25	538	1	0	20	226	1	-1	0	0
31885	42	1773	0	0	9	311	1	336	1	0
15738	56	217	0	1	21	121	2	-1	0	0
...	...	...	...	...	...	...	...	...	...	...
9016	46	2800	0	0	5	47	1	-1	0	0
380	38	757	1	0	6	133	1	-1	0	1
7713	41	4539	0	0	30	298	3	-1	0	0
12188	41	1309	0	0	20	28	4	-1	0	0
28550	57	1016	1	0	29	462	2	234	5	0

13564 rows × 42 columns

y_train

```
10747    1
26054    1
9125     1
41659    1
4443     1
        ..
11284    2
44732    1
38158    1
860      1
```

```
15795    1
Name: Subscription, Length: 31647, dtype: int64
```

```
y_test
```

```
3776     1
9928     1
33409    1
31885    1
15738    1
        ..
9016     1
380      1
7713     1
12188    1
28550    1
Name: Subscription, Length: 13564, dtype: int64
```

Normalization using MinMaxscaler

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(x_train)
x_train=scaler.fit_transform(x_train)
x_train
```

```
array([[0.23376623, 0.07776175, 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.49350649, 0.07998773, 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.36363636, 0.07776175, 1.        , ..., 0.        , 0.        ,
        1.        ],
       ...,
       [0.20779221, 0.09271899, 1.        , ..., 0.        , 0.        ,
        1.        ],
       [0.19480519, 0.07963567, 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.25974026, 0.07729611, 1.        , ..., 0.        , 0.        ,
        1.        ]])
```

```
x_test=scaler.fit_transform(x_test)
x_test
```

```
array([[0.29333333, 0.07806911, 1.        , ..., 0.        , 0.        ,
        1.        ],
       [0.38666667, 0.10588673, 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.09333333, 0.0776878 , 1.        , ..., 0.        , 0.        ,
        1.        ],
       ...,
       [0.30666667, 0.11401231, 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.30666667, 0.0846876 , 0.        , ..., 0.        , 0.        ,
        1.        ],
       [0.52      , 0.08202749, 1.        , ..., 0.        , 0.        ,
        0.        ]])
```

MODEL CREATION KNN

```
from sklearn.neighbors import KNeighborsClassifier
modelkn=KNeighborsClassifier(n_neighbors=3)
modelkn.fit(x_train,y_train)
y_predkn=modelkn.predict(x_test)
y_predkn
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

PERFOMANCE EVALUATION

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
resultkn=confusion_matrix(y_test,y_predkn)
resultkn
```

```
array([[11496,   470],
       [ 1107,   491]])
```

```
scorekn=accuracy_score(y_test,y_predkn)
scorekn
```

```
0.8837363609554704
```

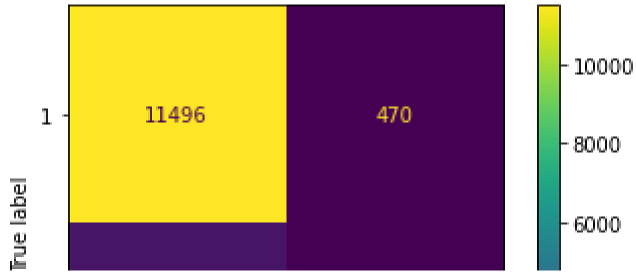
```
print(classification_report(y_test,y_predkn))
```

```
              precision    recall  f1-score   support

           1       0.91      0.96      0.94     11966
           2       0.51      0.31      0.38      1598

    accuracy                           0.88     13564
   macro avg       0.71      0.63      0.66     13564
weighted avg       0.86      0.88      0.87     13564
```

Display confusion Metrics

```
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=['1','2']
cmd=ConfusionMatrixDisplay(resultkn,display_labels=cm)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd1b3f02100>
```



MODEL CREATION DECISIONTREECLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier
modeldt=DecisionTreeClassifier()
modeldt.fit(x_train,y_train)
y_preddt=modeldt.predict(x_test)
y_preddt
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

Perfomance Evaluation

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
resultdt=confusion_matrix(y_test,y_preddt)
resultdt
```

```
array([[10424,  1542],
       [  753,   845]])
```

```
scoredt=accuracy_score(y_test,y_preddt)
scoredt
```

```
0.8308021232674727
```

```
print(classification_report(y_test,y_preddt))
```

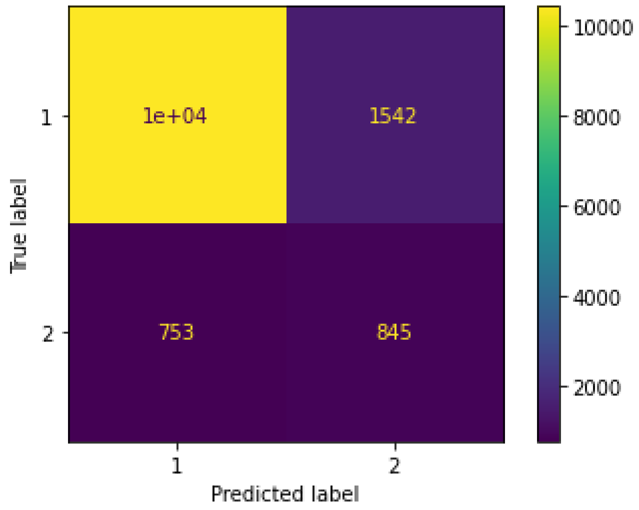
```
              precision    recall  f1-score   support

           1       0.93      0.87      0.90     11966
           2       0.35      0.53      0.42      1598

    accuracy                           0.83     13564
   macro avg       0.64      0.70      0.66     13564
weighted avg       0.86      0.83      0.84     13564
```

Display confusion metrics

```
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=['1','2']
```

```
cmd=ConfusionMatrixDisplay(resultdt,display_labels=cm)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd1b3e98e80>
```



MODEL CREATION RANDOMFORESTCLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier
modelrf=RandomForestClassifier(n_estimators=4,criterion='entropy')
modelrf.fit(x_train,y_train)
y_predrf=modelrf.predict(x_test)
y_predrf
```

```
array([1, 1, 2, ..., 1, 1, 2])
```

Perfomance Evaluation

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
resultrf=confusion_matrix(y_test,y_predrf)
resultrf
```

```
array([[11433,   533],
       [ 1046,   552]])
```

```
scorerf=accuracy_score(y_test,y_predrf)
scorerf
```

```
0.8835889118254202
```

```
print(classification_report(y_test,y_predrf))
```

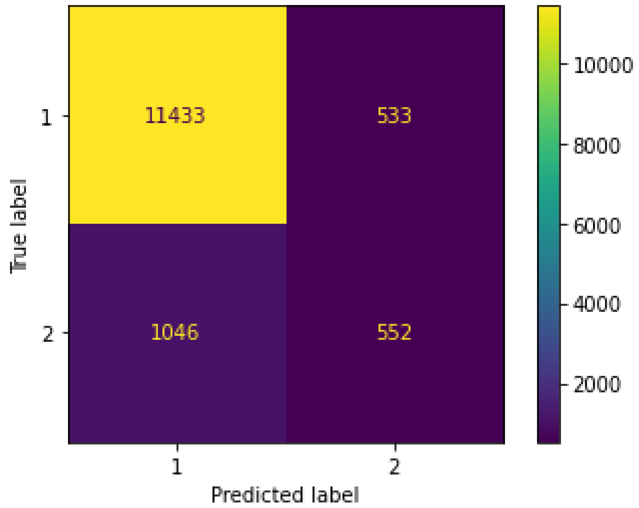
```
              precision    recall  f1-score   support

           1       0.92      0.96      0.94     11966
           2       0.51      0.35      0.41      1598

    accuracy                           0.88     13564
   macro avg       0.71      0.65      0.67     13564
```

```
weighted avg       0.87      0.88      0.87     13564
```

Display Confusion Metrics

```
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=['1','2']
cmd=ConfusionMatrixDisplay(resultrf,display_labels=cm)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd1b3dcbd00>
```



MODEL CREATION NAIVEBAYES

```
from sklearn.naive_bayes import MultinomialNB
modelnb=MultinomialNB()
modelnb.fit(x_train,y_train)
y_prednb=modelnb.predict(x_test)
y_prednb
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

Perfomance Evaluation

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
resultnb=confusion_matrix(y_test,y_prednb)
resultnb
```

```
array([[11603,   363],
       [ 1150,   448]])
```

```
scorenb=accuracy_score(y_test,y_prednb)
scorenb
```

```
0.8884547331170746
```

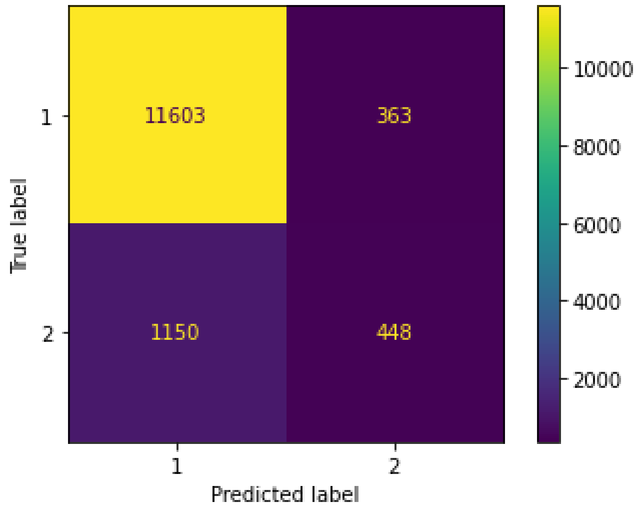
```
print(classification_report(y_test,y_prednb))
```

```
              precision    recall  f1-score   support

           1       0.91      0.97      0.94     11966
           2       0.55      0.28      0.37      1598

    accuracy                           0.89     13564
   macro avg       0.73      0.63      0.66     13564
weighted avg       0.87      0.89      0.87     13564
```

Display Confusion matrix

```
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=['1','2']
cmd=ConfusionMatrixDisplay(resultnb,display_labels=cm)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd1b3d76e20>
```



MODEL CREATION SVM

```
from sklearn.svm import SVC
svmodel=SVC()
svmodel.fit(x_train,y_train)
y_predsv=svmodel.predict(x_test)
y_predsv
```

```
array([1, 1, 1, ..., 1, 1, 1])
```

Perfomance Evaluation

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
resultsv=confusion_matrix(y_test,y_predsv)
resultsv
```

```
array([[11761,   205],
       [ 1173,   425]])
```

```
scoresv=accuracy_score(y_test,y_predsv)
scoresv
```

```
0.8984075493954585
```

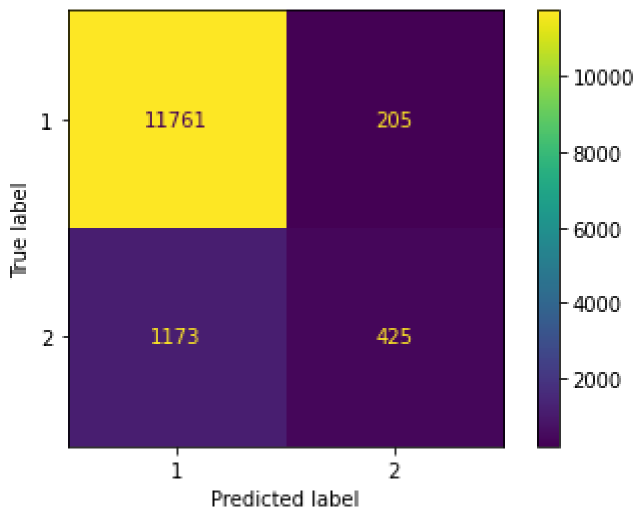
```
print(classification_report(y_test,y_predsv))
```

```
              precision    recall  f1-score   support

           1       0.91      0.98      0.94     11966
           2       0.67      0.27      0.38      1598

    accuracy                           0.90     13564
   macro avg       0.79      0.62      0.66     13564
weighted avg       0.88      0.90      0.88     13564
```

Display Confusion matrix

```
from sklearn.metrics._plot.confusion_matrix import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm=['1','2']
cmd=ConfusionMatrixDisplay(resultsv,display_labels=cm)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fd1b543ad60>
```



ACCURACY SCORE EACH ALGORITHMS

```
print("accuracy score in KNN algorithm is",accuracy_score(y_test,y_predkn))
print("accuracy score in DECISON TREE algorithm is",accuracy_score(y_test,y_preddt))
print("accuracy score in RANDOMFORESTCLASSIFIER algorithm is",accuracy_score(y_test,y_p
print("accuracy score in SVM algorithm is",accuracy_score(y_test,y_predsv))
print("accuracy score in NAIVE BAYES algorithm is",accuracy_score(y_test,y_prednb))
```

```
print("accuracy score in NAIVE BAYES algorithm is",accuracy_score(y_test,y_prednb))
```

```
accuracy score in KNN algorithm is 0.8837363609554704
accuracy score in DECISON TREE algorithm is 0.8326452373930994
accuracy score in RANDOMFORESTCLASSIFIER algorithm is 0.8796815098790917
accuracy score in SVM algorithm is 0.8984075493954585
accuracy score in NAIVE BAYES algorithm is 0.8884547331170746
```

✓ 0s completed at 8:42 PM