In [1]:

```
#Loading libraries and data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv('/home/sandhan/Downloads/WA_Fn-UseC_-Telco-Customer-Churn.csv')  #Lo
df
```

Out[1]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleL
0	7590-VHVEG	Female	0	Yes	No	1	No	No pl se
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	No pl se
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No pl se
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

In [2]:

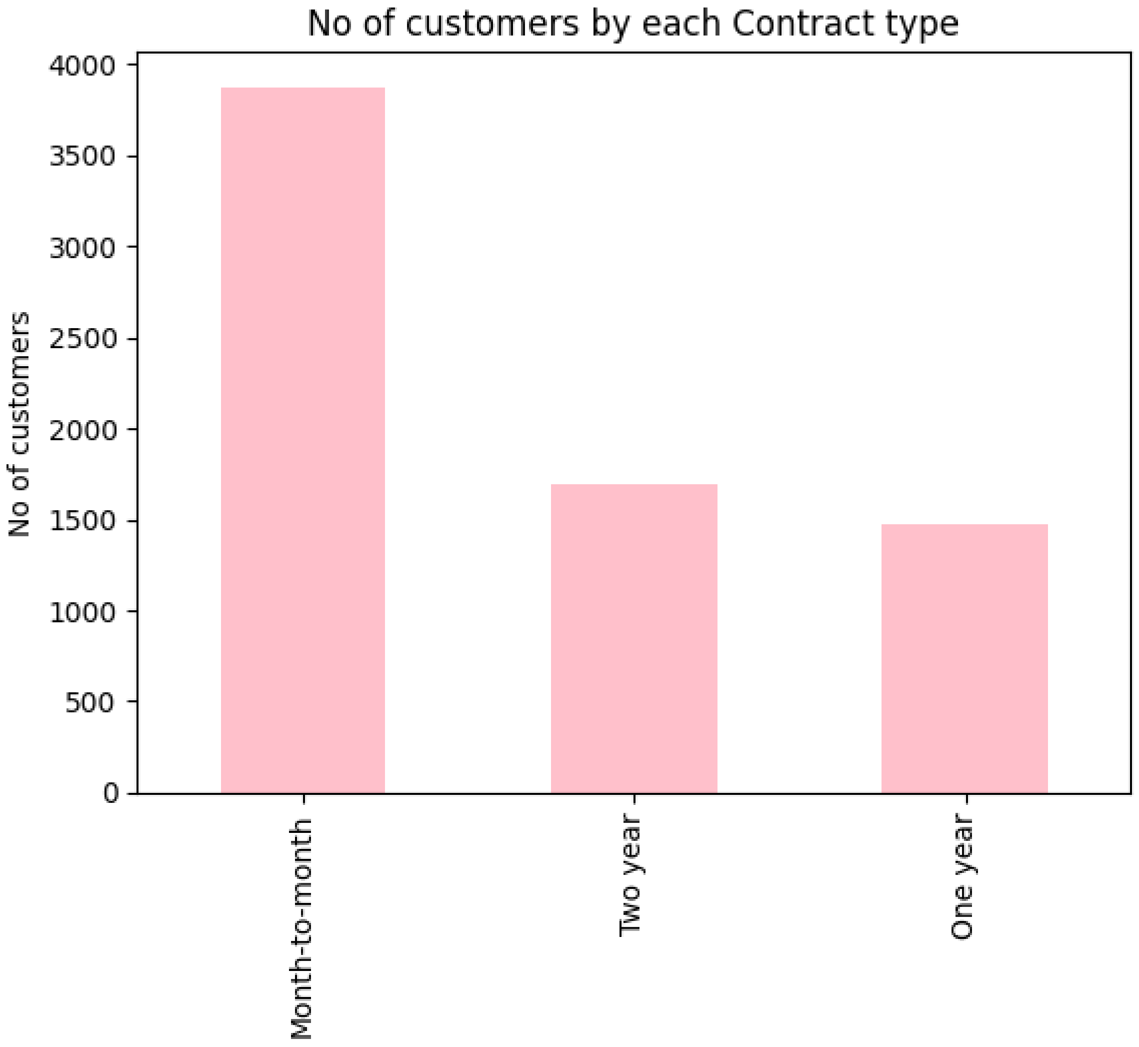
```
#DATA VISUALISATION
```

In [3]:

```
#Number of customers with each contract type
plot1=df["Contract"].value_counts().plot(kind='bar',color='pink')
plot1.set_ylabel("No of customers")
plot1.set_title("No of customers by each Contract type")
```

Out[3]:

```
Text(0.5, 1.0, 'No of customers by each Contract type')
```

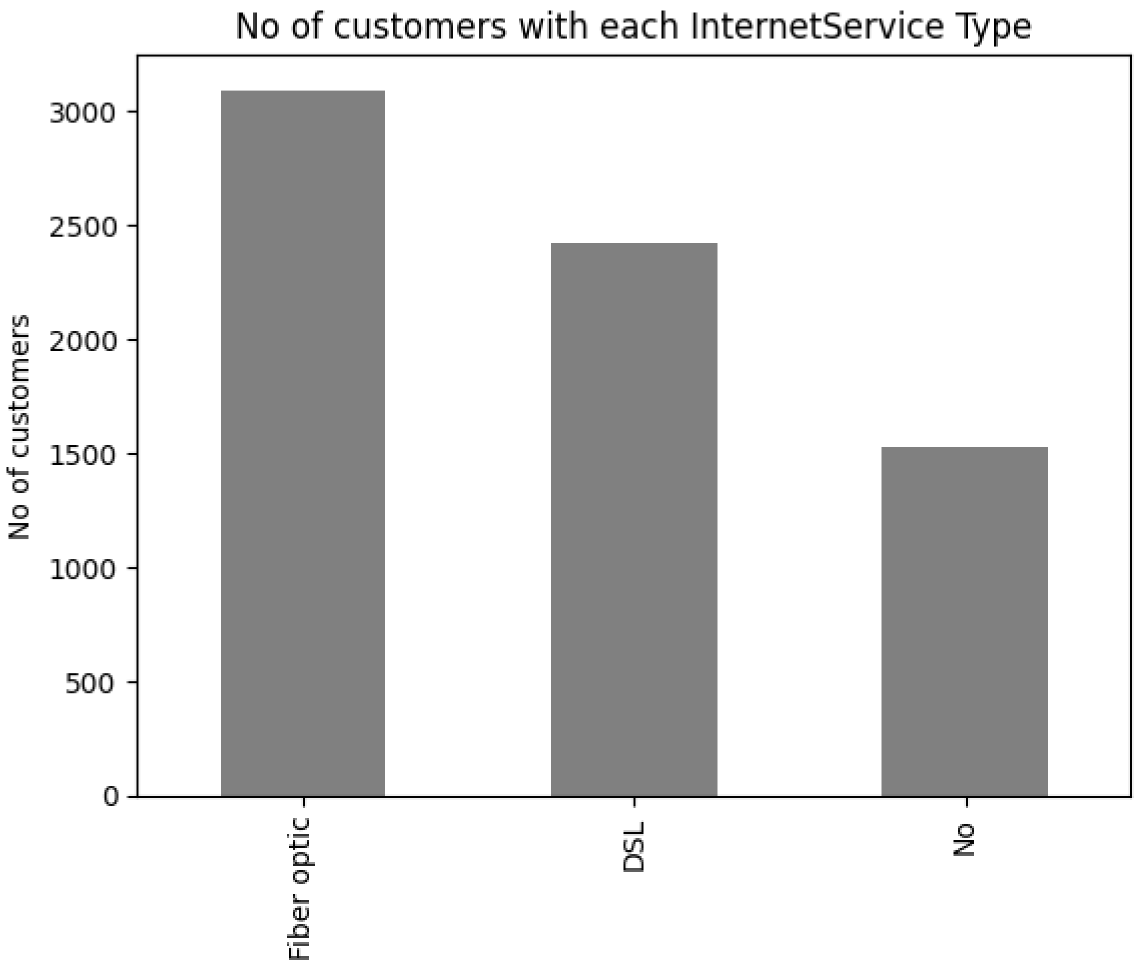


In [4]:

```
#Number of customers with each InternetService Type
plot2=df["InternetService"].value_counts().plot(kind='bar',color='grey')
plot2.set_ylabel("No of customers")
plot2.set_title("No of customers with each InternetService Type")
```

Out[4]:

```
Text(0.5, 1.0, 'No of customers with each InternetService Type')
```



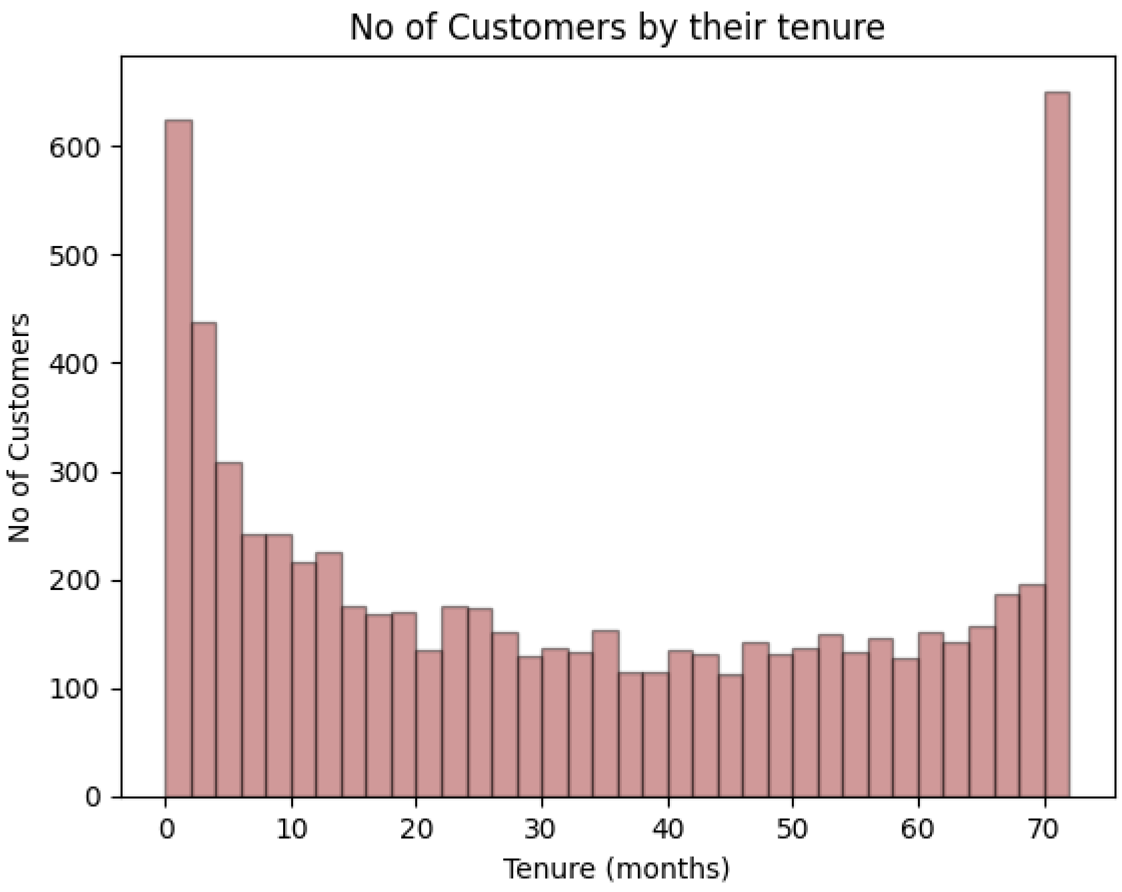
In [5]:

```
#No of customers by their tenure
plot3=sns.distplot(df['tenure'], hist=True, kde=False,bins=int(180/5),\
                   color='darkred',hist_kws={'edgecolor':'black'})
plot3.set_ylabel('No of Customers')
plot3.set_xlabel('Tenure (months)')
plot3.set_title('No of Customers by their tenure')
```

```
/tmp/ipykernel_7786/2452378685.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.
14.0.

Please adapt your code to use either `displot` (a figure-level functio
n with
similar flexibility) or `histplot` (an axes-level function for histogr
ams).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (http
s://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  plot3=sns.distplot(df['tenure'], hist=True, kde=False,bins=int(180/
5),\
```

Out[5]:

```
Text(0.5, 1.0, 'No of Customers by their tenure')
```

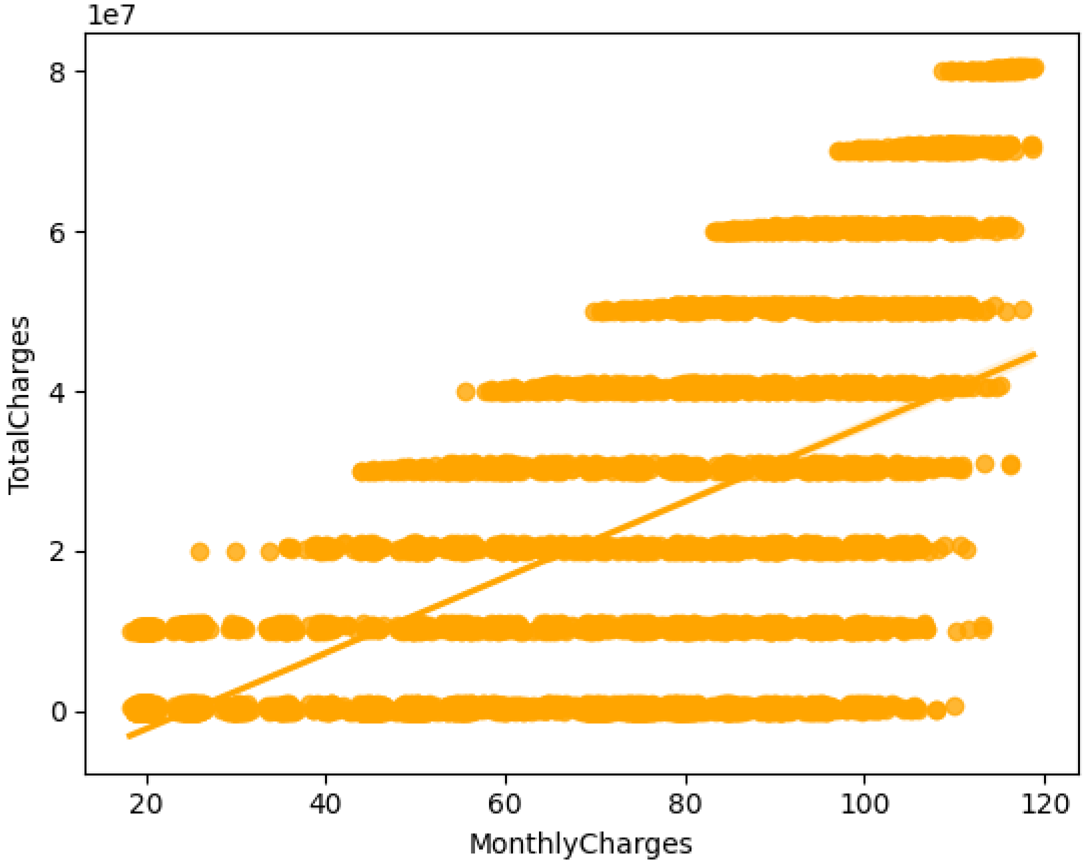


In [6]:

```
#Association between TotalCharges and MonthlyCharges
df['TotalCharges']=df['TotalCharges'].str.replace('','0')
df['TotalCharges']=df['TotalCharges'].str.replace('0 0','0')
df['TotalCharges']=df['TotalCharges'].astype(float)
df['TotalCharges']=df['TotalCharges'].replace('0',df['TotalCharges'].mean())
sns.regplot(x=df['MonthlyCharges'],y=df['TotalCharges'],color='orange')
```

Out[6]:

```
<AxesSubplot: xlabel='MonthlyCharges', ylabel='TotalCharges'>
```

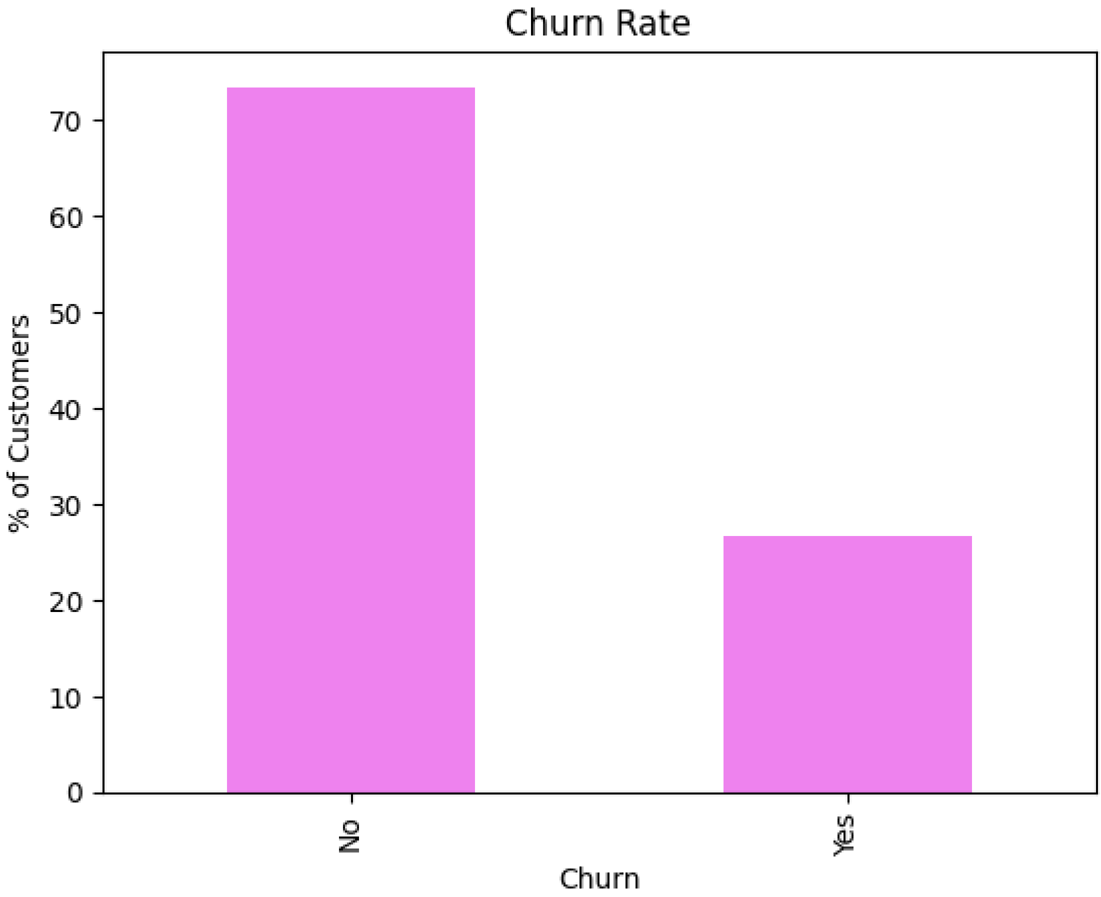


In [7]:

```
#Churn rate of customers with percentage
plot4=(df['Churn'].value_counts()*100/len(df)).plot(kind='bar',color='violet')
plot4.set_ylabel('% of Customers')
plot4.set_xlabel('Churn')
plot4.set_title('Churn Rate')
```

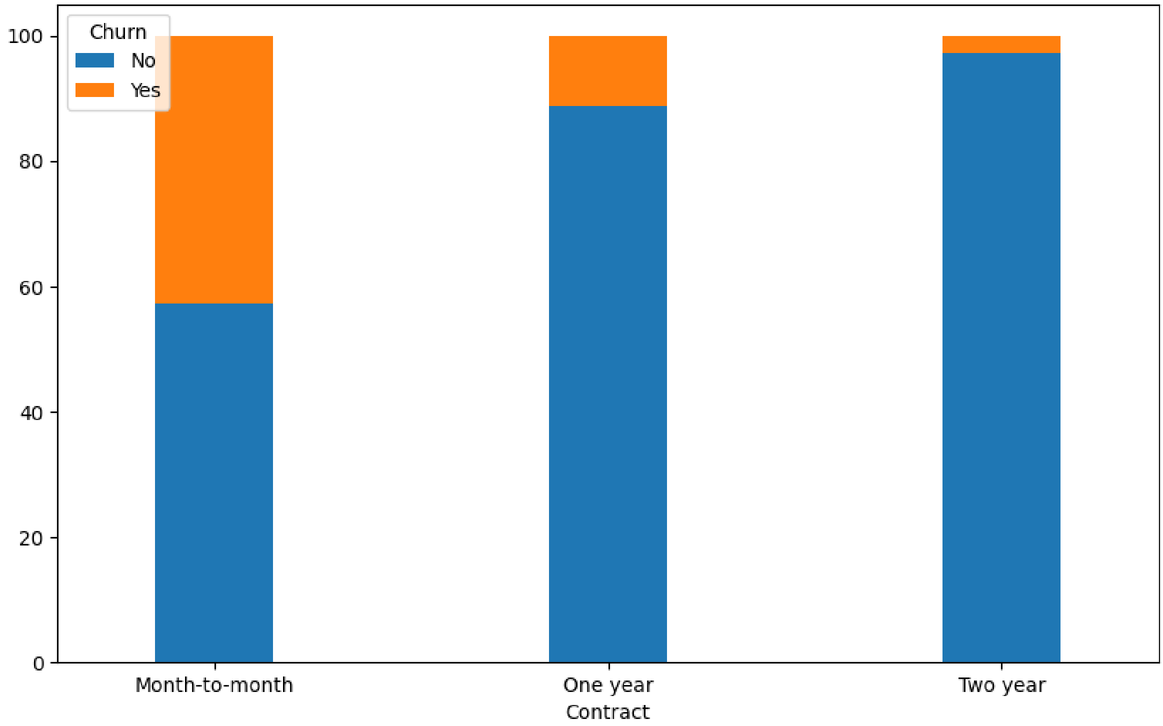
Out[7]:

```
Text(0.5, 1.0, 'Churn Rate')
```



In [8]:

```
#Churn rate of customers with each contract type
contract_churn=df.groupby(['Contract','Churn']).size().unstack()
plot6=(contract_churn.T*100.0 / contract_churn.T.sum()).T.plot(kind='bar',width=0.3
```



In [9]:

```
#UNDERSTANDING DATA
```

In [10]:

```
df.shape
```

Out[10]:

```
(7043, 21)
```

In [11]:

```
df.columns
```

Out[11]:

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependent
s',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupp
ort',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBillin
g',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [12]:

```
df.dtypes
```

Out[12]:

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                object
dtype: object
```

In [13]:

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

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   OnlineBackup      7043 non-null   object 
 10  DeviceProtection  7043 non-null   object 
 11  TechSupport       7043 non-null   object 
 12  StreamingTV       7043 non-null   object 
 13  StreamingMovies   7043 non-null   object 
 14  Contract          7043 non-null   object 
 15  PaperlessBilling  7043 non-null   object 
 16  PaymentMethod     7043 non-null   object 
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7043 non-null   float64
 19  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

In [15]:

```
#DATA MANIPULATION
```

In [16]:

```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
cols=['gender','Partner','Dependents','PhoneService','MultipleLines','InternetServi
for i in cols:
    df[i]=encoder.fit_transform(df[i])
df
```

Out[16]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Internet
0	0	0	1	0	1	0	1	
1	1	0	0	0	34	1	0	
2	1	0	0	0	2	1	0	
3	1	0	0	0	45	0	1	
4	0	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	1	0	1	1	24	1	2	
7039	0	0	1	1	72	1	2	
7040	0	0	1	1	11	0	1	
7041	1	1	1	0	4	1	2	
7042	1	0	0	0	66	1	0	

7043 rows × 20 columns

In [17]:

```
df.dtypes
```

Out[17]:

```
gender                int64
SeniorCitizen         int64
Partner               int64
Dependents            int64
tenure                int64
PhoneService          int64
MultipleLines         int64
InternetService       int64
OnlineSecurity        int64
OnlineBackup          int64
DeviceProtection      int64
TechSupport           int64
StreamingTV           int64
StreamingMovies       int64
Contract              int64
PaperlessBilling      int64
PaymentMethod         int64
MonthlyCharges      float64
TotalCharges          int64
Churn                 int64
dtype: object
```

In [18]:

```
df.describe()
```

Out[18]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multip
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043
mean	0.504756	0.162147	0.483033	0.299588	32.371149	0.903166	0
std	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0
50%	1.000000	0.000000	0.000000	0.000000	29.000000	1.000000	1
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	2
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	2

In [19]:

```
df.isna().sum()
```

Out[19]:

```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [20]:

```
x=df.iloc[:,0:-1]
y=df.iloc[:,-1]
```

In [21]:

```
#Feature Selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
test=SelectKBest(score_func=chi2)
fi=test.fit(x,y)
fi.scores_
```

Out[21]:

```
array([2.58698618e-01, 1.34351545e+02, 8.24120826e+01, 1.33036443e+02,
       1.62789237e+04, 9.72606249e-02, 9.74692078e+00, 9.82102823e+00,
       5.51611529e+02, 2.30086520e+02, 1.91303140e+02, 5.23303866e+02,
       7.49020319e+00, 8.23539949e+00, 1.11578017e+03, 1.05680863e+02,
       5.84922505e+01, 3.68078770e+03, 4.50421670e+05])
```

In [22]:

```
col=x.columns
score=pd.DataFrame({'features':col,'score_chi2':fi.scores_})
score
```

Out[22]:

	features	score_chi2
0	gender	0.258699
1	SeniorCitizen	134.351545
2	Partner	82.412083
3	Dependents	133.036443
4	tenure	16278.923685
5	PhoneService	0.097261
6	MultipleLines	9.746921
7	InternetService	9.821028
8	OnlineSecurity	551.611529
9	OnlineBackup	230.086520
10	DeviceProtection	191.303140
11	TechSupport	523.303866
12	StreamingTV	7.490203
13	StreamingMovies	8.235399
14	Contract	1115.780167
15	PaperlessBilling	105.680863
16	PaymentMethod	58.492250
17	MonthlyCharges	3680.787699
18	TotalCharges	450421.669826

In [23]:

```
score.sort_values(by='score_chi2',ascending=False)
```

Out[23]:

	features	score_chi2
18	TotalCharges	450421.669826
4	tenure	16278.923685
17	MonthlyCharges	3680.787699
14	Contract	1115.780167
8	OnlineSecurity	551.611529
11	TechSupport	523.303866
9	OnlineBackup	230.086520
10	DeviceProtection	191.303140
1	SeniorCitizen	134.351545
3	Dependents	133.036443
15	PaperlessBilling	105.680863
2	Partner	82.412083
16	PaymentMethod	58.492250
7	InternetService	9.821028
6	MultipleLines	9.746921
13	StreamingMovies	8.235399
12	StreamingTV	7.490203
0	gender	0.258699
5	PhoneService	0.097261

In [24]:

```
df.drop(['PhoneService','gender','StreamingTV','StreamingMovies','MultipleLines','I
df
```

Out[24]:

	SeniorCitizen	Partner	Dependents	tenure	OnlineSecurity	OnlineBackup	DeviceProtecti
0	0	1	0	1	0	2	
1	0	0	0	34	2	0	
2	0	0	0	2	2	2	
3	0	0	0	45	2	0	
4	0	0	0	2	0	0	
...	...	...	...	...	...	...	
7038	0	1	1	24	2	0	
7039	0	1	1	72	0	2	
7040	0	1	1	11	2	0	
7041	1	1	0	4	0	0	
7042	0	0	0	66	2	0	

7043 rows × 14 columns

In [25]:

```
#DATA PREPROCESSING
```

In [26]:

```
#Splitting the dataset into training and testing data
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=1)
```

In [27]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

In [28]:

```
#Machine Learning Model Prediction and Performance Evaluation
```

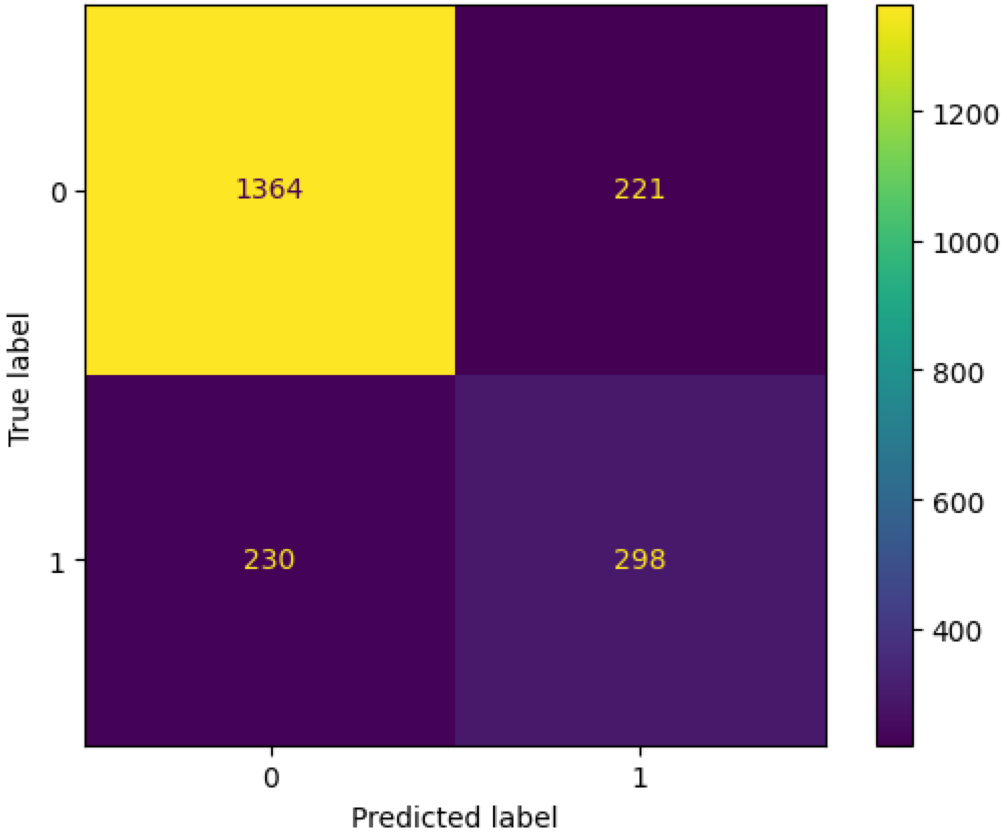
In [29]:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
classifier=KNeighborsClassifier(n_neighbors=9)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
```

In [30]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,C
print(classification_report(y_test,y_pred))
result=confusion_matrix(y_test,y_pred)
print(result)
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      1585
           1       0.57      0.56      0.57       528

    accuracy                           0.79      2113
   macro avg       0.71      0.71      0.71      2113
weighted avg       0.79      0.79      0.79      2113

[[1364  221]
 [ 230  298]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f3cafadf970>
```



In [31]:

```
score=accuracy_score(y_test,y_pred)
print(score)
```

```
0.7865593942262187
```

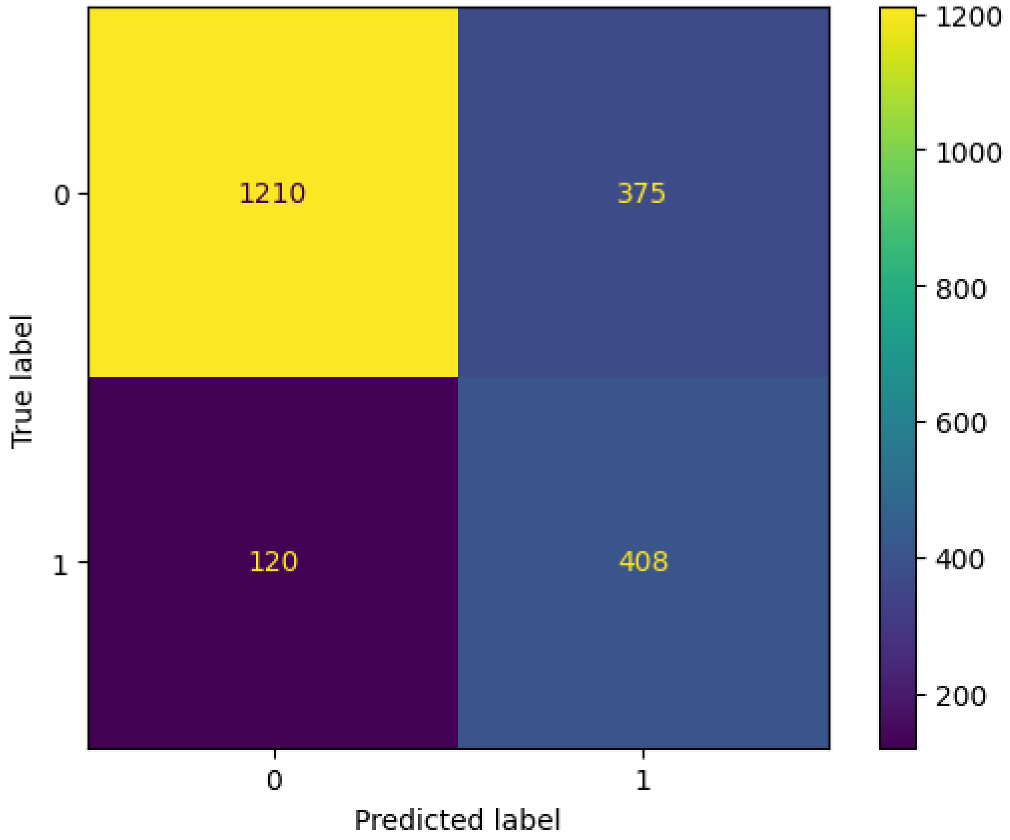
In [32]:

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
```

In [33]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,C
print(classification_report(y_test,y_pred))
result=confusion_matrix(y_test,y_pred)
print(result)
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.76      0.83      1585
           1       0.52      0.77      0.62       528

    accuracy                           0.77      2113
   macro avg       0.72      0.77      0.73      2113
weighted avg       0.81      0.77      0.78      2113

[[1210  375]
 [ 120  408]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f3cabe4cc70>
```



In [34]:

```
score=accuracy_score(y_test,y_pred)
print(score)
```

```
0.7657359204921912
```

In [35]:

```
#SVM
from sklearn.svm import SVC
classifier=SVC()
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
y_pred
```

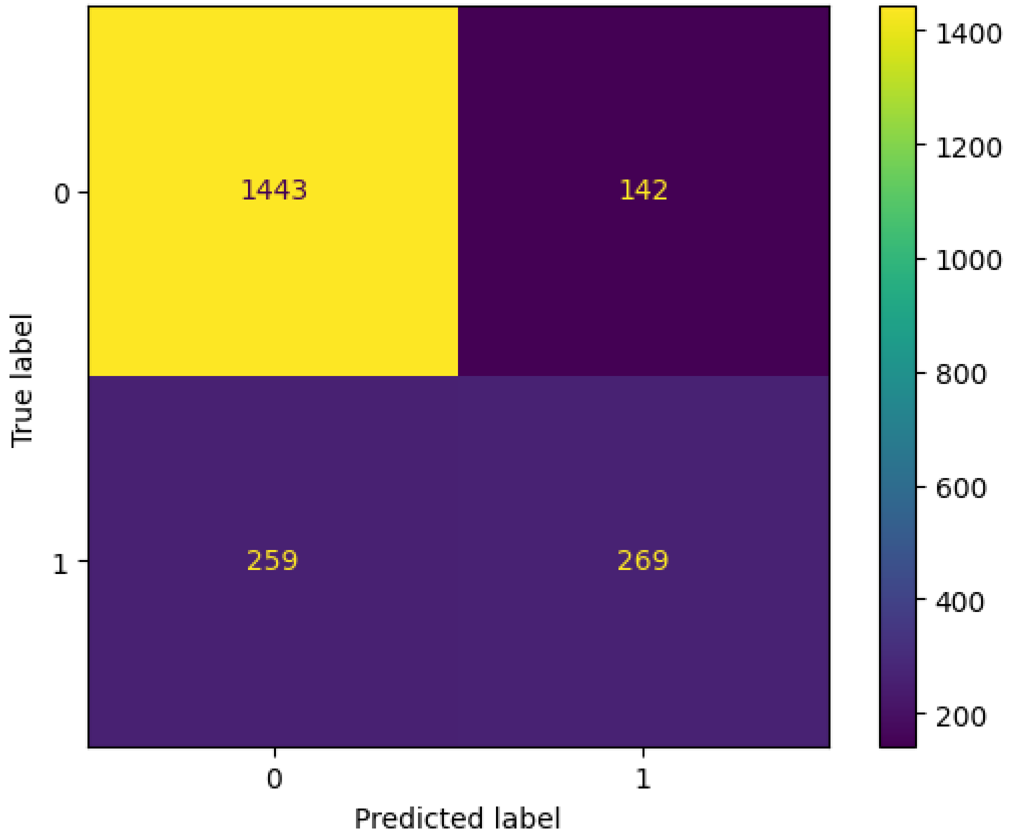
Out[35]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [36]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print(classification_report(y_test,y_pred))
result=confusion_matrix(y_test,y_pred)
print(result)
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1585
           1       0.65      0.51      0.57       528

    accuracy                           0.81      2113
   macro avg       0.75      0.71      0.73      2113
weighted avg       0.80      0.81      0.80      2113

[[1443  142]
 [ 259  269]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f3cac868f40>
```



In [37]:

```
score=accuracy_score(y_test,y_pred)
print(score)
```

```
0.8102224325603408
```

In [38]:

```
#Decision Tree
from sklearn.tree import DecisionTreeClassifier
dec=DecisionTreeClassifier(criterion='entropy')
dec.fit(x_train,y_train)
y_pred=dec.predict(x_test)
y_pred
```

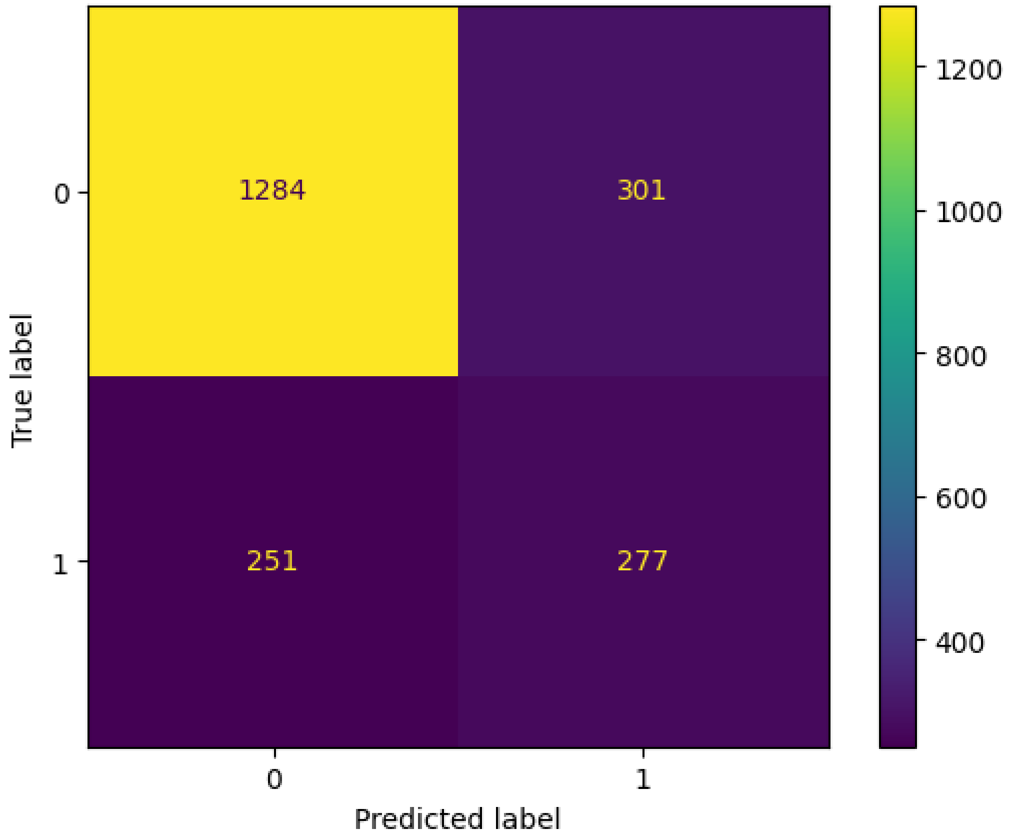
Out[38]:

```
array([0, 0, 0, ..., 1, 0, 0])
```

In [39]:

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
print(classification_report(y_test,y_pred))
result=confusion_matrix(y_test,y_pred)
print(result)
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.81      0.82      1585
           1       0.48      0.52      0.50       528

    accuracy                           0.74      2113
   macro avg       0.66      0.67      0.66      2113
weighted avg       0.75      0.74      0.74      2113

[[1284  301]
 [ 251  277]]
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object
at 0x7f3cda9ec580>
```



In [40]:

```
score=accuracy_score(y_test,y_pred)
print(score)
```

```
0.738760056791292
```

In [ ]:

```
#Comparing the results obtained from each of the above classification algorithms,
#we see that SVM algorthm has the highest accuracy of about 81% for this dataset,
#So we choose SVM classification algorithm for this dataset.
```