Customer Churn Analysis

Problem Statement:

Customer churn is when a company's customers stop doing business with that company. Businesses are very keen on measuring churn because keeping an existing customer is far less expensive than acquiring a new customer. New business involves working leads through a sales funnel, using marketing and sales budgets to gain additional customers. Existing customers will often have a higher volume of service consumption and can generate additional customer referrals.

Customer retention can be achieved with good customer service and products. But the most effective way for a company to prevent attrition of customers is to truly know them. The vast volumes of data collected about customers can be used to build churn prediction models. Knowing who is most likely to defect means that a company can prioritise focused marketing efforts on that subset of their customer base.

Preventing customer churn is critically important to the telecommunications sector, as the barriers to entry for switching services are so low.

You will examine customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.

Importing the Dataset:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
df=pd.read_csv("Telecom_customer_churn.csv")

df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Int
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

Checking Null values in dataframe.

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
customerID          0
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 [4]:
```
df.columns
```

Out[4]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

Since there are no null values then we can proceed.

Dataframe Description:

The datset contains the data of the customer.on the basis of the data we have to predict we have to predict the total charges by the total charges by the customer.The dataset contain the data like 'customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'

In the above dataset the target is to predict the "Total charges"paid by the customer.

Churn rate(sometimes called attrition rate).In its broadest sense,is a measure of the number of individuals or items moving out of a collective group over a specific period.It is one of two primary factors that determine the steady-state level of customers a business will support.The term is used in many contexts,but is most widely applied in business with respect to a contractual customer base for example in businesses with a subscriber-based service model such as mobile telephone networks and pay TV operators.The term is also used to refer to participant turnover in peer-to-peer networks.Churn rate is an input into customer lifeline value modelling.and can be part of a simulator used to measure return on marketing investment using marketing mix modeling.

Checking the datatypes of the columns

In [5]:
```
df.dtypes
```

Out[5]:
```
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         object
Churn                object
dtype: object
```

We can observe that the "Total charges"has continuous data but it is an object type.Let us

handle this column.

In [6]:
```
df['TotalCharges'].unique()
```

Out[6]:
```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

In [7]:
```
df.loc[df["TotalCharges"]==" "]
```

Out[7]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
488	4472-LVYGI	Female	0	Yes	Yes	0	No	No phone service
753	3115-CZMZD	Male	0	No	Yes	0	Yes	No
936	5709-LVOEQ	Female	0	Yes	Yes	0	Yes	No
1082	4367-NUYAO	Male	0	Yes	Yes	0	Yes	Yes
1340	1371-DWPAZ	Female	0	Yes	Yes	0	No	No phone service
3331	7644-OMVMY	Male	0	Yes	Yes	0	Yes	No
3826	3213-VVOLG	Male	0	Yes	Yes	0	Yes	Yes
4380	2520-SGTTA	Female	0	Yes	Yes	0	Yes	No
5218	2923-ARZLG	Male	0	Yes	Yes	0	Yes	No
6670	4075-WKNIU	Female	0	Yes	Yes	0	Yes	Yes
6754	2775-SEFEE	Male	0	No	Yes	0	Yes	Yes

11 rows × 21 columns

After some observation,we saw the some columns have space as the value.

In [8]:
```
df['TotalCharges']=df['TotalCharges'].replace(" ",np.nan)
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
customerID           0
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        11
Churn                0
dtype: int64
```

Coverting the column type from object to float

In [10]:
```
df['TotalCharges']=df['TotalCharges'].astype(float)
```

Now we can see total charges column has some space values which is replaced by nan values.Now we will handle the man values.

Handling the Nan values and replacing it with the mean of the column.As the column is continuous in nature

In [11]:
```
df['TotalCharges']=df['TotalCharges'].fillna(np.mean(df["TotalCharges"]))
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
customerID          0
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 [13]:
```
df.dtypes
```

Out[13]:
```
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
```

Making DataFrame for the Nominal Data

In [14]:
```
df_visualization_nominal=df[['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Depe
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn']].copy()
```

In [15]:
```
df_visualization_nominal.columns
```

Out[15]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

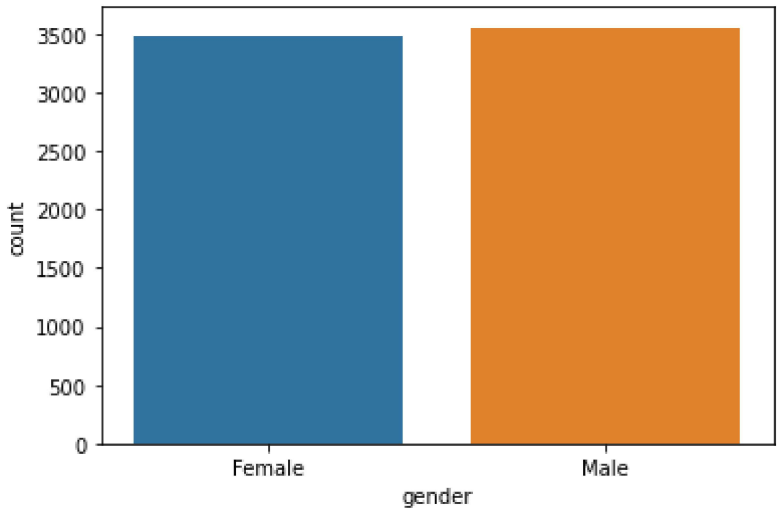
Visualization of the Data

For the nominal categorical data we will use countplot as it will give the frequency of the classes of the columns

In [16]:
```
import seaborn as sns
```

In [17]:
```
ax=sns.countplot(x="gender",data=df_visualization_nominal)
print(df_visualization_nominal["gender"].value_counts())
```

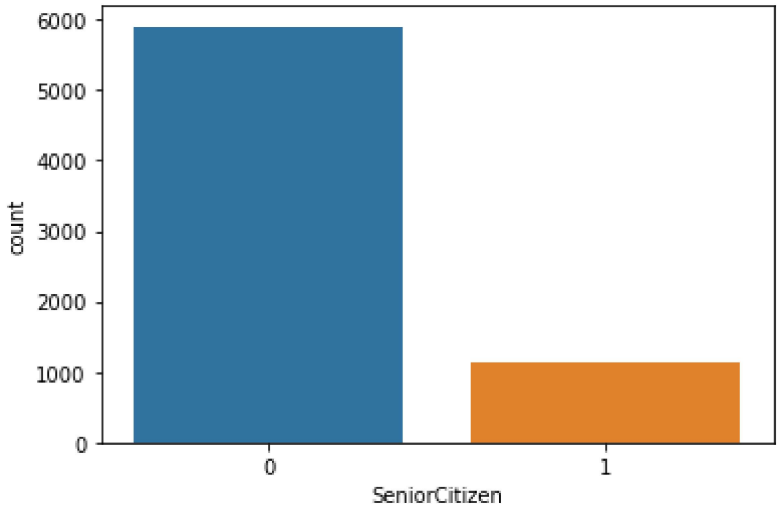
```
Male      3555
Female    3488
Name: gender, dtype: int64
```



From the above observatio the total number of the male is 3555 and total number of female customer is 3488

In [18]:
```
ax=sns.countplot(x="SeniorCitizen",data=df_visualization_nominal)
print(df_visualization_nominal["SeniorCitizen"].value_counts())
```

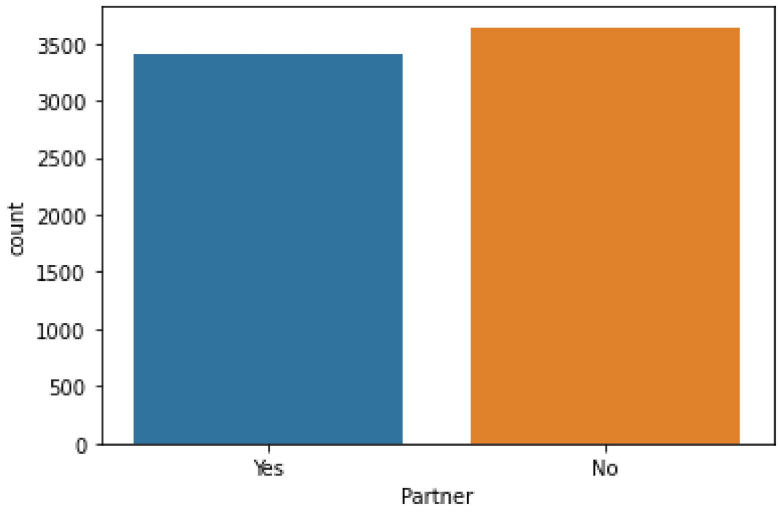
```
0    5901
1    1142
Name: SeniorCitizen, dtype: int64
```



From the above observation the total number of younger customer is 5901 and total number of senior citizen customer is 1142

In [19]:
```
ax=sns.countplot(x="Partner",data=df_visualization_nominal)
print(df_visualization_nominal["Partner"].value_counts())
```

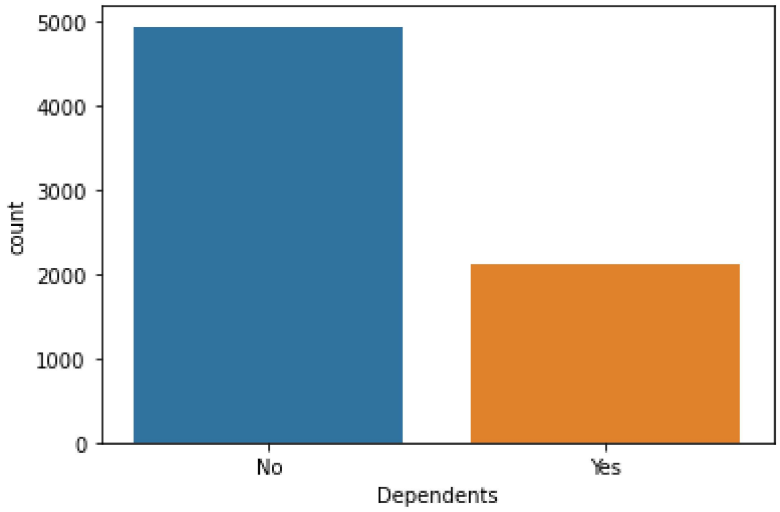
```
No     3641
Yes    3402
Name: Partner, dtype: int64
```



From the above observation the total number of married customer is 3402 and total number of unmarrid customer is 3641

In [20]:
```
ax=sns.countplot(x="Dependents",data=df_visualization_nominal)
print(df_visualization_nominal["Dependents"].value_counts())
```

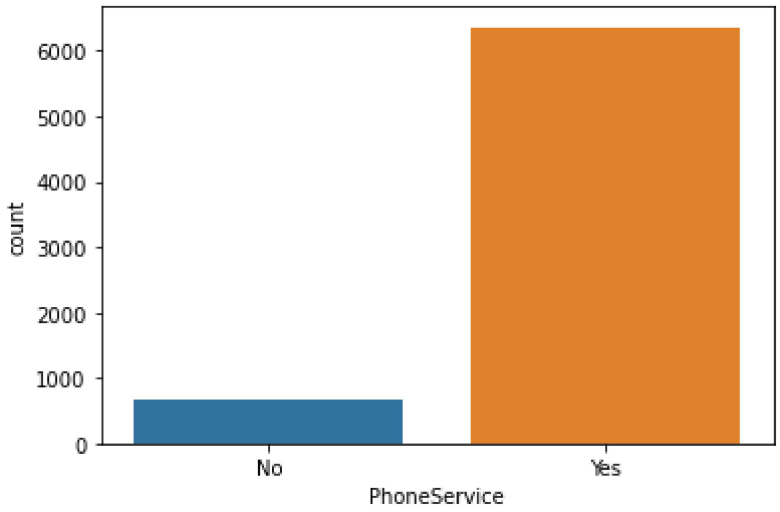
```
No     4933
Yes    2110
Name: Dependents, dtype: int64
```



From the above observation the total number dependent customer is 2110 and number of independent customer is 4933

In [21]:
```
ax=sns.countplot(x="PhoneService",data=df_visualization_nominal)
print(df_visualization_nominal["PhoneService"].value_counts())
```

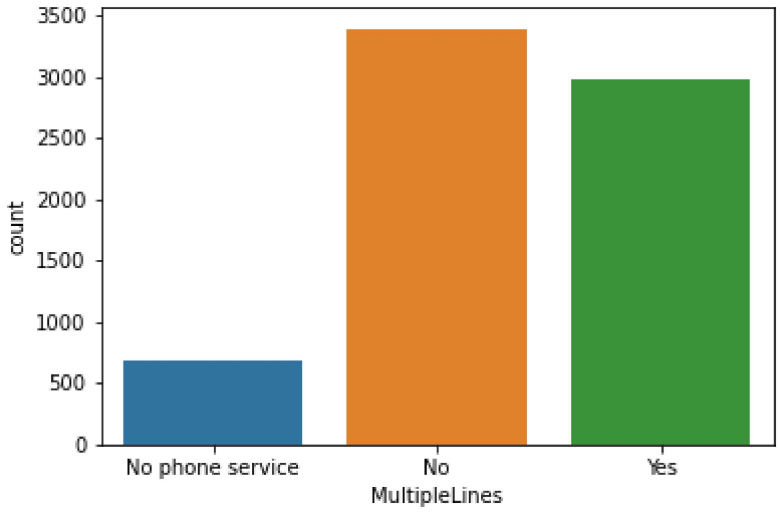
```
Yes    6361
No      682
Name: PhoneService, dtype: int64
```



From the above observation the total number of customer using phone service is 6361 and customer not using phone is 682

In [22]:
```
ax=sns.countplot(x="MultipleLines",data=df_visualization_nominal)
print(df_visualization_nominal["MultipleLines"].value_counts())
```

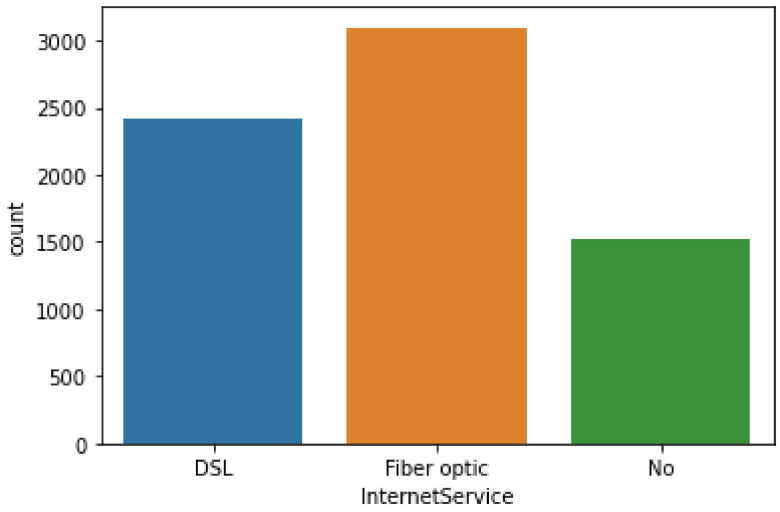
```
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
```



From the above observations total number of customer not using any phone services is 682,using no multiple service is 3390 and using multiple service is 2971

In [23]:
```
ax=sns.countplot(x="InternetService",data=df_visualization_nominal)
print(df_visualization_nominal["InternetService"].value_counts())
```

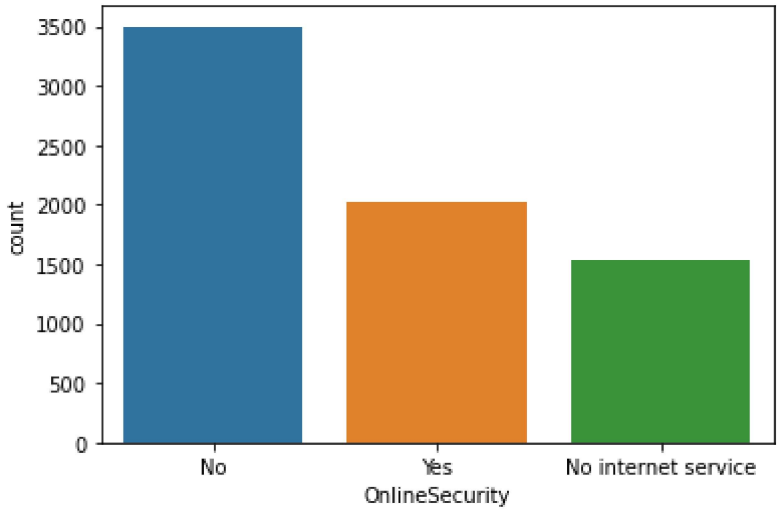
```
Fiber optic    3096
DSL            2421
No             1526
Name: InternetService, dtype: int64
```



From the above observations total number of customer not using any internet services is 1526,fibre optics internet services is 3096 and using DSL internet service is 2421

In [24]:
```
ax=sns.countplot(x="OnlineSecurity",data=df_visualization_nominal)
print(df_visualization_nominal["OnlineSecurity"].value_counts())
```

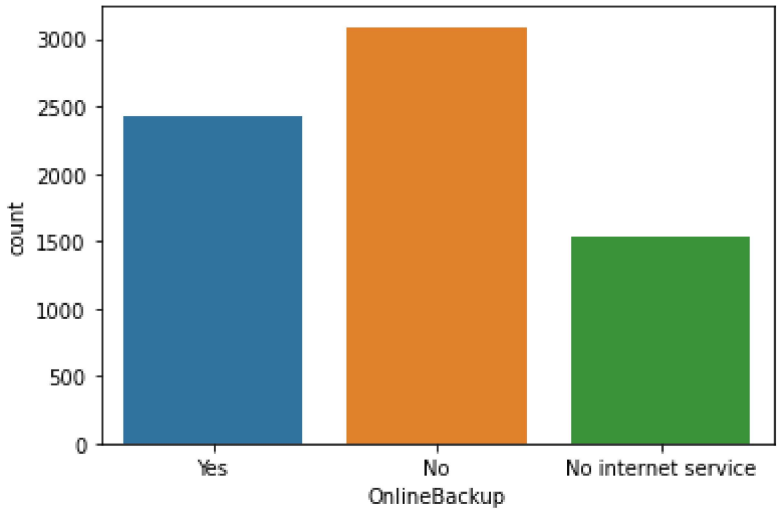
```
No                     3498
Yes                    2019
No internet service    1526
Name: OnlineSecurity, dtype: int64
```



From the above observations the total number of customer not using internet service is 1526,using online security is 2019 and not using online security service is 3498

In [25]:
```
ax=sns.countplot(x="OnlineBackup",data=df_visualization_nominal)
print(df_visualization_nominal["OnlineBackup"].value_counts())
```

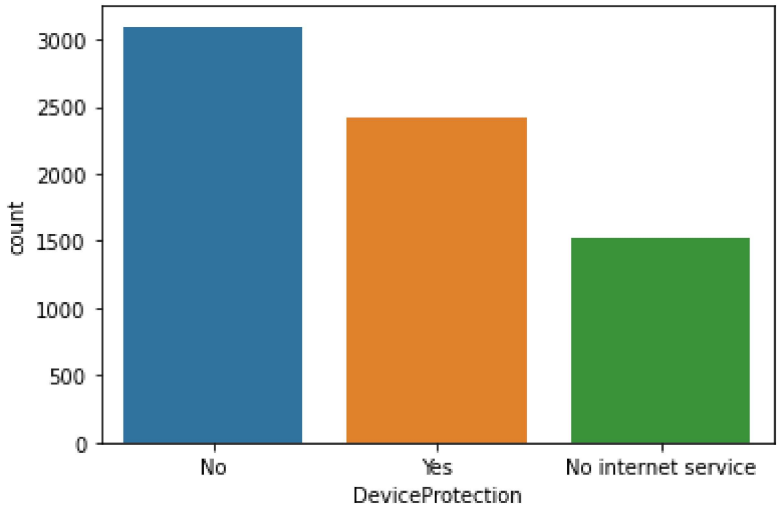
```
No                  3088
Yes                 2429
No internet service    1526
Name: OnlineBackup, dtype: int64
```



From the above observations the total number of customer not using internet service is 1526,using onlineBackup security is 2429 and not using onlineBackup security service is 3088

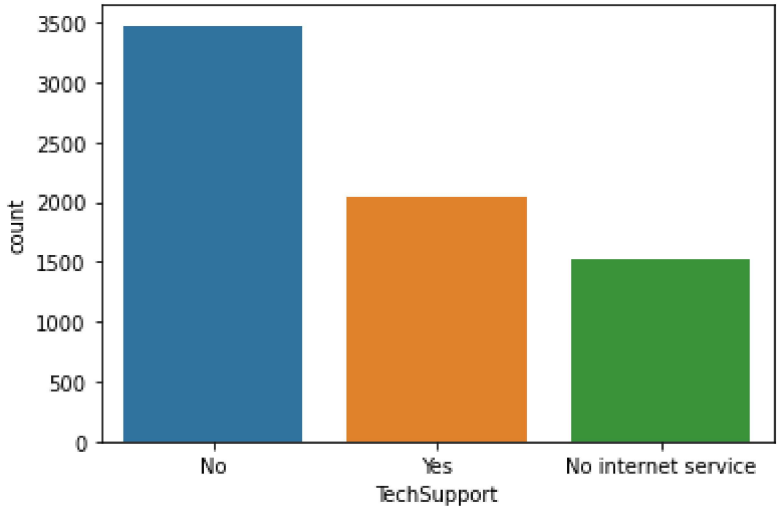
In [26]:
```
ax=sns.countplot(x="DeviceProtection",data=df_visualization_nominal)
print(df_visualization_nominal["DeviceProtection"].value_counts())
```

```
No                  3095
Yes                 2422
No internet service    1526
Name: DeviceProtection, dtype: int64
```



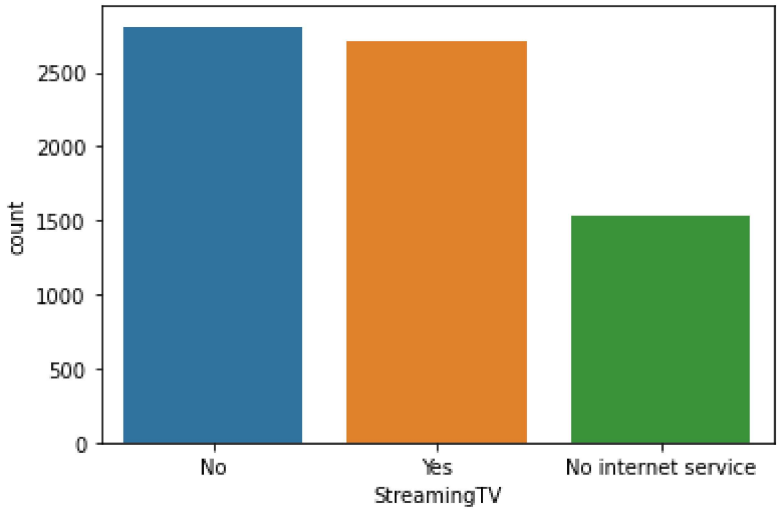
In [27]:
```
ax=sns.countplot(x="TechSupport",data=df_visualization_nominal)
print(df_visualization_nominal["TechSupport"].value_counts())
```

```
No                     3473
Yes                    2044
No internet service    1526
Name: TechSupport, dtype: int64
```



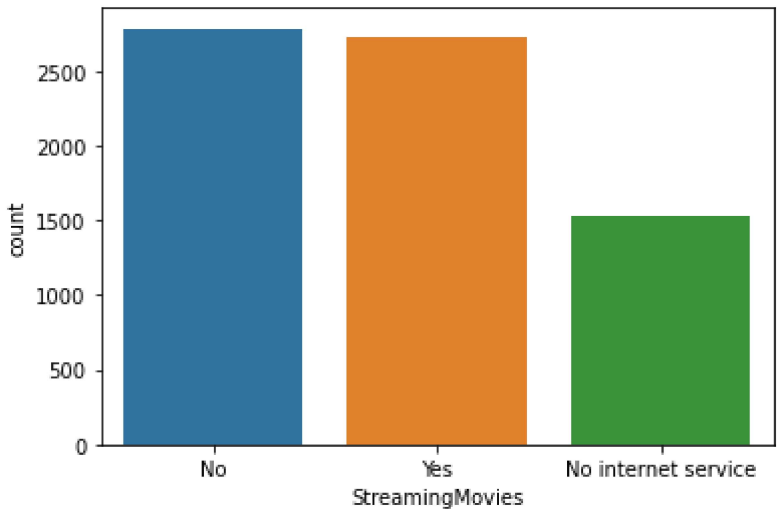
In [28]:
```
ax=sns.countplot(x="StreamingTV",data=df_visualization_nominal)
print(df_visualization_nominal["StreamingTV"].value_counts())
```

```
No                     2810
Yes                    2707
No internet service    1526
Name: StreamingTV, dtype: int64
```



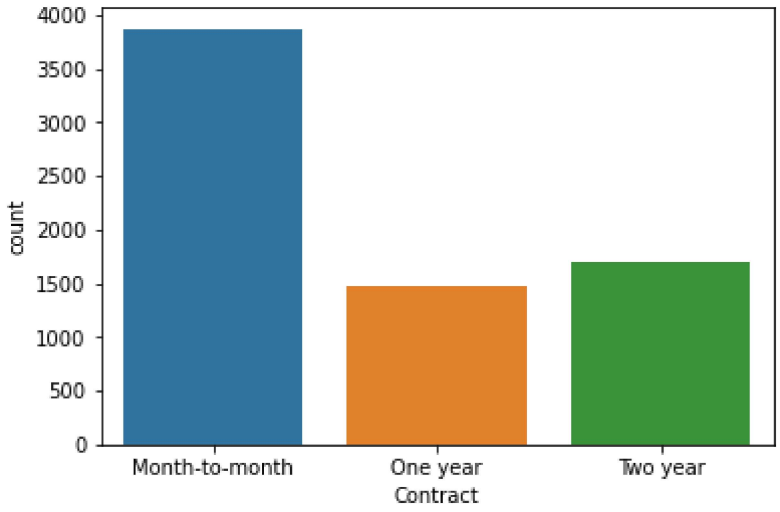
In [29]:
```
ax=sns.countplot(x="StreamingMovies",data=df_visualization_nominal)
print(df_visualization_nominal["StreamingMovies"].value_counts())
```

```
No                     2785
Yes                    2732
No internet service    1526
Name: StreamingMovies, dtype: int64
```



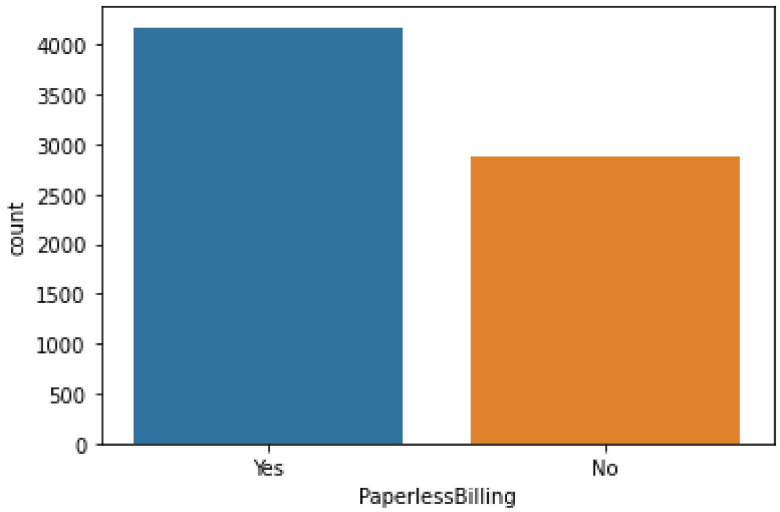
In [30]:
```
ax=sns.countplot(x="Contract",data=df_visualization_nominal)
print(df_visualization_nominal["Contract"].value_counts())
```

```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```



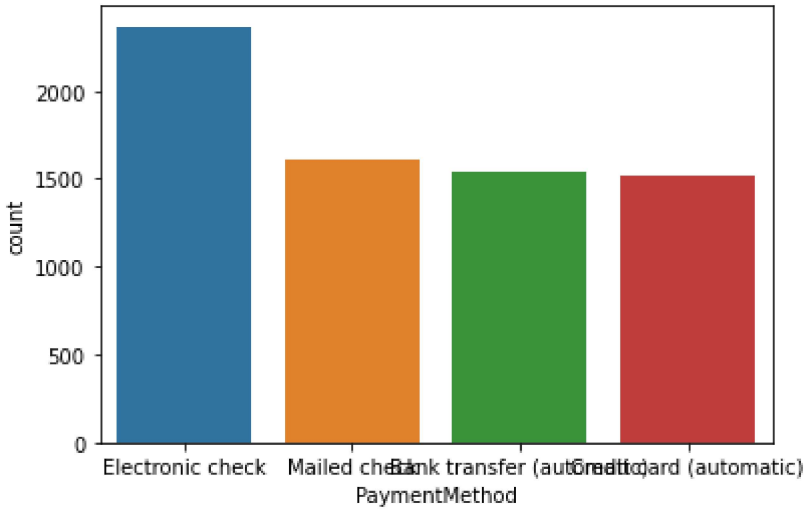
In [31]:
```
ax=sns.countplot(x="PaperlessBilling",data=df_visualization_nominal)
print(df_visualization_nominal["PaperlessBilling"].value_counts())
```

```
Yes    4171
No     2872
Name: PaperlessBilling, dtype: int64
```



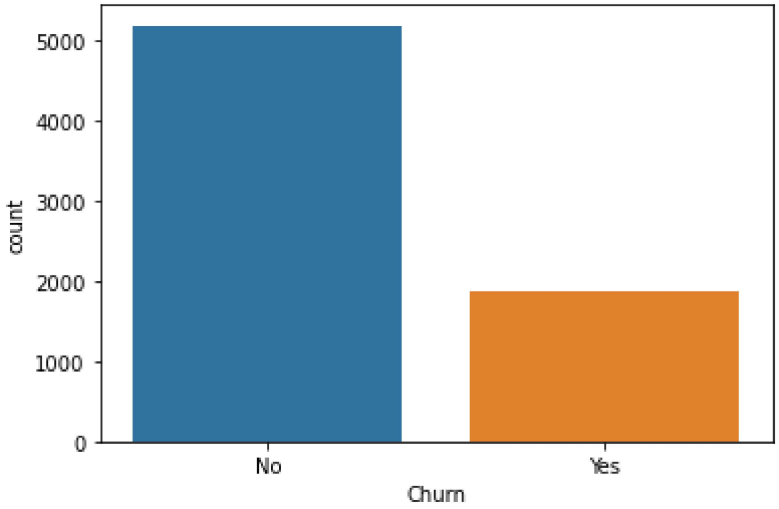
In [32]:
```
ax=sns.countplot(x="PaymentMethod",data=df_visualization_nominal)
print(df_visualization_nominal["PaymentMethod"].value_counts())
```

```
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: PaymentMethod, dtype: int64
```



In [33]:
```
ax=sns.countplot(x="Churn",data=df_visualization_nominal)
print(df_visualization_nominal["Churn"].value_counts())
```

```
No     5174
Yes    1869
Name: Churn, dtype: int64
```



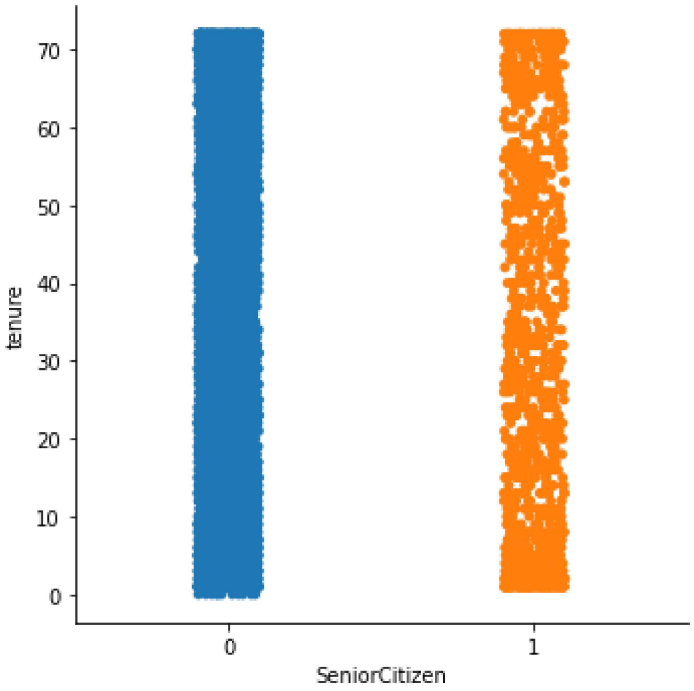
For the ordinal categorical data we will use lineplot as it will give the flow of the line of the classes of the columns:

Making dataframe of the ordinal data.

In [34]:
```
df_visualization_ordinal=df[["customerID","tenure"]].copy()
```

In [35]:
```
sns.catplot(x="SeniorCitizen",y="tenure",data=df)
```

Out[35]:
```
<seaborn.axisgrid.FacetGrid at 0x211e1e5bdc0>
```

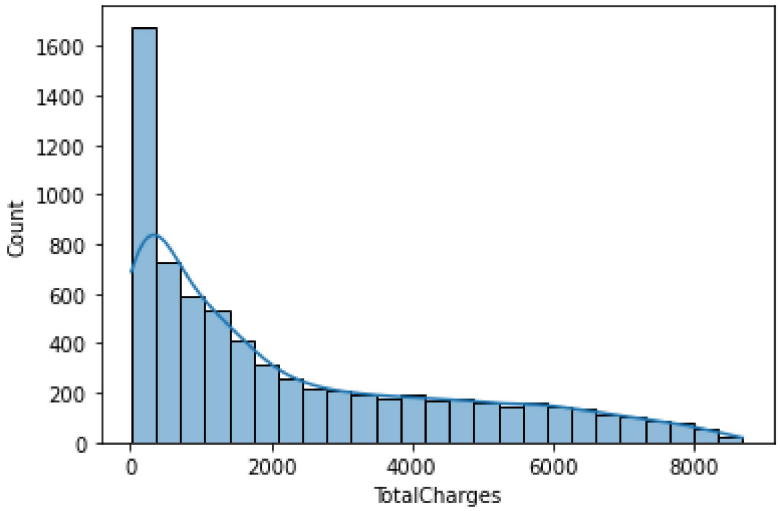


Checking the distribution of the continuous value of the flot type columns.

In [36]:
```
df_visualization_continuous=df[["MonthlyCharges","TotalCharges"]].copy()
```

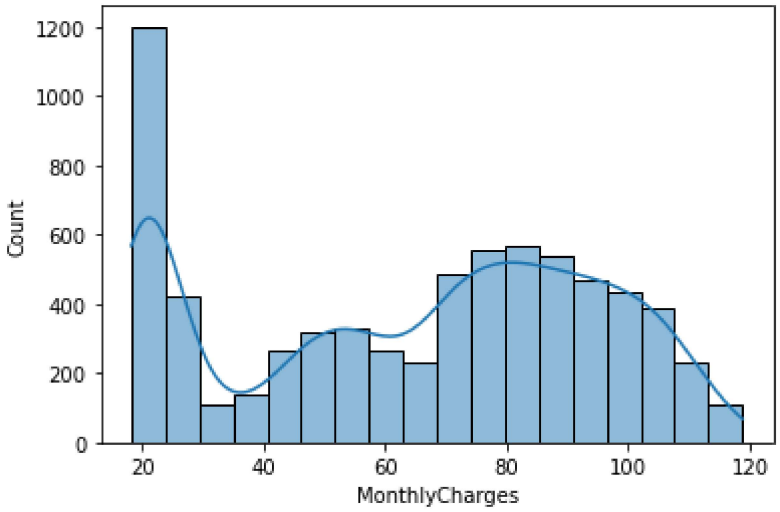
In [37]:
```
import seaborn as sns
sns.histplot(df_visualization_continuous['TotalCharges'],kde=True)
```

Out[37]:
```
<AxesSubplot:xlabel='TotalCharges', ylabel='Count'>
```



In [38]:
```
sns.histplot(df_visualization_continuous['MonthlyCharges'],kde=True)
```

Out[38]:
```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='Count'>
```



Visualization part is completed.

Encoding of DataFrame

In [39]:
```
from sklearn.preprocessing import OrdinalEncoder
enc=OrdinalEncoder()
```

In [40]:
```
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=enc.fit_transform(df[i].values.reshape(-1,1))
```

In [41]:
```
df
```

Out[41]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	5375.0	0.0	0	1.0	0.0	1	0.0	1.0
1	3962.0	1.0	0	0.0	0.0	34	1.0	0.0
2	2564.0	1.0	0	0.0	0.0	2	1.0	0.0
3	5535.0	1.0	0	0.0	0.0	45	0.0	1.0
4	6511.0	0.0	0	0.0	0.0	2	1.0	0.0
...	...	...	...	...	...	...	...	...
7038	4853.0	1.0	0	1.0	1.0	24	1.0	2.0
7039	1525.0	0.0	0	1.0	1.0	72	1.0	2.0
7040	3367.0	0.0	0	1.0	1.0	11	0.0	1.0
7041	5934.0	1.0	1	1.0	0.0	4	1.0	2.0
7042	2226.0	1.0	0	0.0	0.0	66	1.0	0.0

7043 rows × 21 columns

Describe the dataset:

In [42]:
```
df.describe()
```

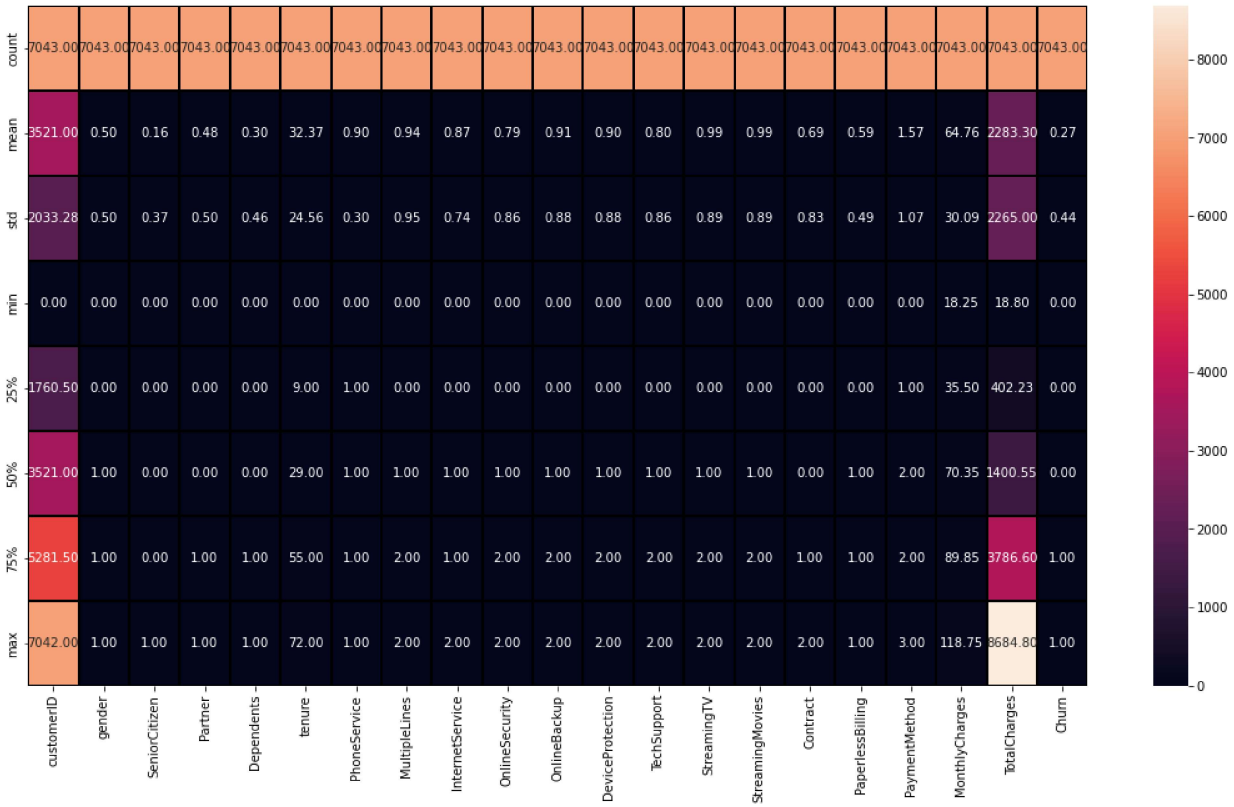
Out[42]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService
count	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000	7043.000000
mean	3521.000000	0.504756	0.162147	0.483033	0.299588	32.371149	0.903166
std	2033.283305	0.500013	0.368612	0.499748	0.458110	24.559481	0.295752
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1760.500000	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000
50%	3521.000000	1.000000	0.000000	0.000000	0.000000	29.000000	1.000000
75%	5281.500000	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000
max	7042.000000	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000

8 rows × 21 columns

In [43]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(19,10))
sns.heatmap(df.describe(),annot=True, linewidths=0.1,linecolor="black",fmt="0.2f")
```

Out[43]: <AxesSubplot:>



Correlation of the columns with the target columns

In [44]:
```
df.corr()
```

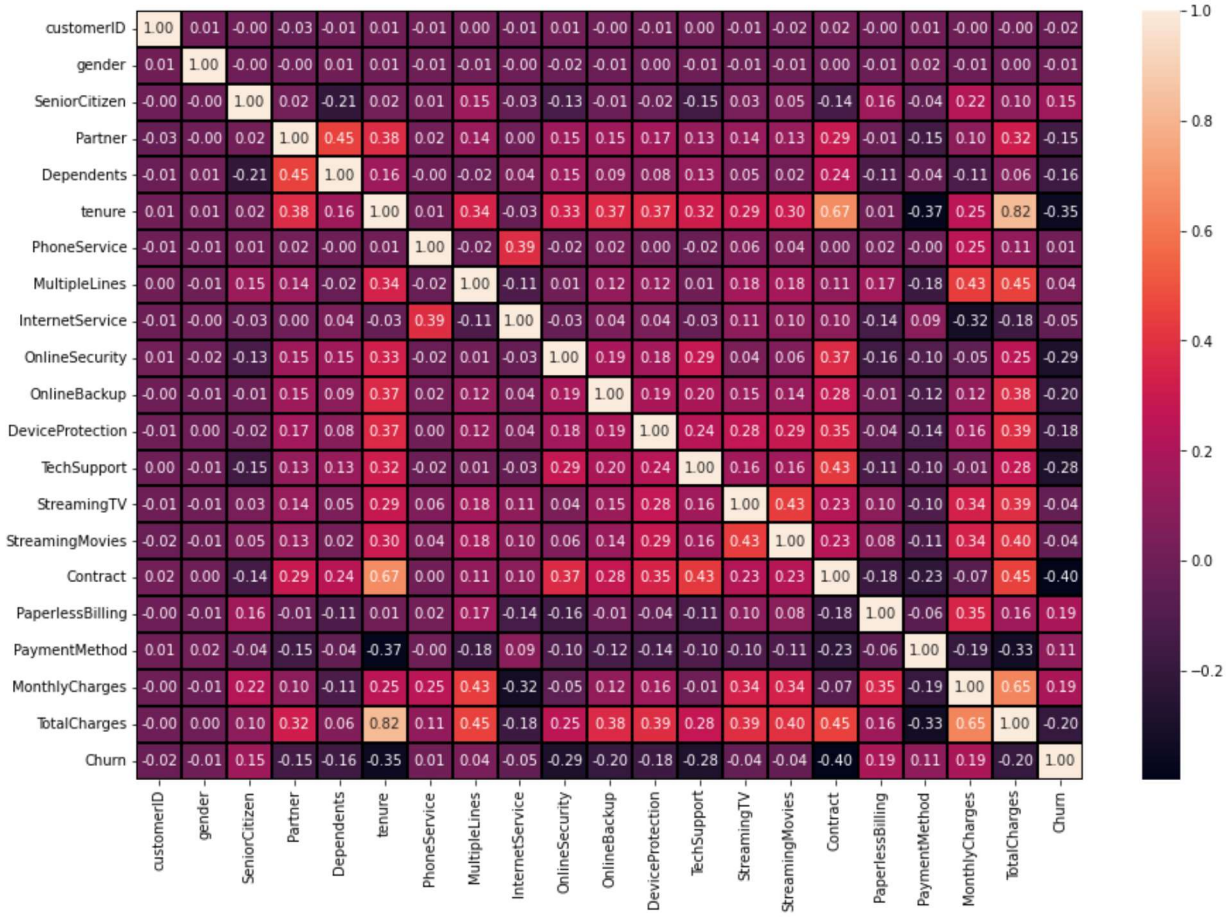
Out[44]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneServ
customerID	1.000000	0.006288	-0.002074	-0.026729	-0.012823	0.008035	-0.006
gender	0.006288	1.000000	-0.001874	-0.001808	0.010517	0.005106	-0.006
SeniorCitizen	-0.002074	-0.001874	1.000000	0.016479	-0.211185	0.016567	0.008
Partner	-0.026729	-0.001808	0.016479	1.000000	0.452676	0.379697	0.017
Dependents	-0.012823	0.010517	-0.211185	0.452676	1.000000	0.159712	-0.001
tenure	0.008035	0.005106	0.016567	0.379697	0.159712	1.000000	0.008
PhoneService	-0.006483	-0.006488	0.008576	0.017706	-0.001762	0.008448	1.000
MultipleLines	0.004316	-0.006739	0.146185	0.142410	-0.024991	0.343032	-0.020
InternetService	-0.012407	-0.000863	-0.032310	0.000891	0.044590	-0.030359	0.387
OnlineSecurity	0.013292	-0.015017	-0.128221	0.150828	0.152166	0.325468	-0.015
OnlineBackup	-0.003334	-0.012057	-0.013632	0.153130	0.091015	0.370876	0.024
DeviceProtection	-0.006918	0.000549	-0.021398	0.166330	0.080537	0.371105	0.003
TechSupport	0.001140	-0.006825	-0.151268	0.126733	0.133524	0.322942	-0.019
StreamingTV	-0.007777	-0.006421	0.030776	0.137341	0.046885	0.289373	0.055
StreamingMovies	-0.016746	-0.008743	0.047266	0.129574	0.021321	0.296866	0.043
Contract	0.015028	0.000126	-0.142554	0.294806	0.243187	0.671607	0.002
PaperlessBilling	-0.001945	-0.011754	0.156530	-0.014877	-0.111377	0.006152	0.016
PaymentMethod	0.011604	0.017352	-0.038551	-0.154798	-0.040292	-0.370436	-0.004
MonthlyCharges	-0.003916	-0.014569	0.220173	0.096848	-0.113890	0.247900	0.247
TotalCharges	-0.000270	0.000048	0.102395	0.318812	0.064535	0.824757	0.112
Churn	-0.017447	-0.008612	0.150889	-0.150448	-0.164221	-0.352229	0.011

21 rows × 21 columns

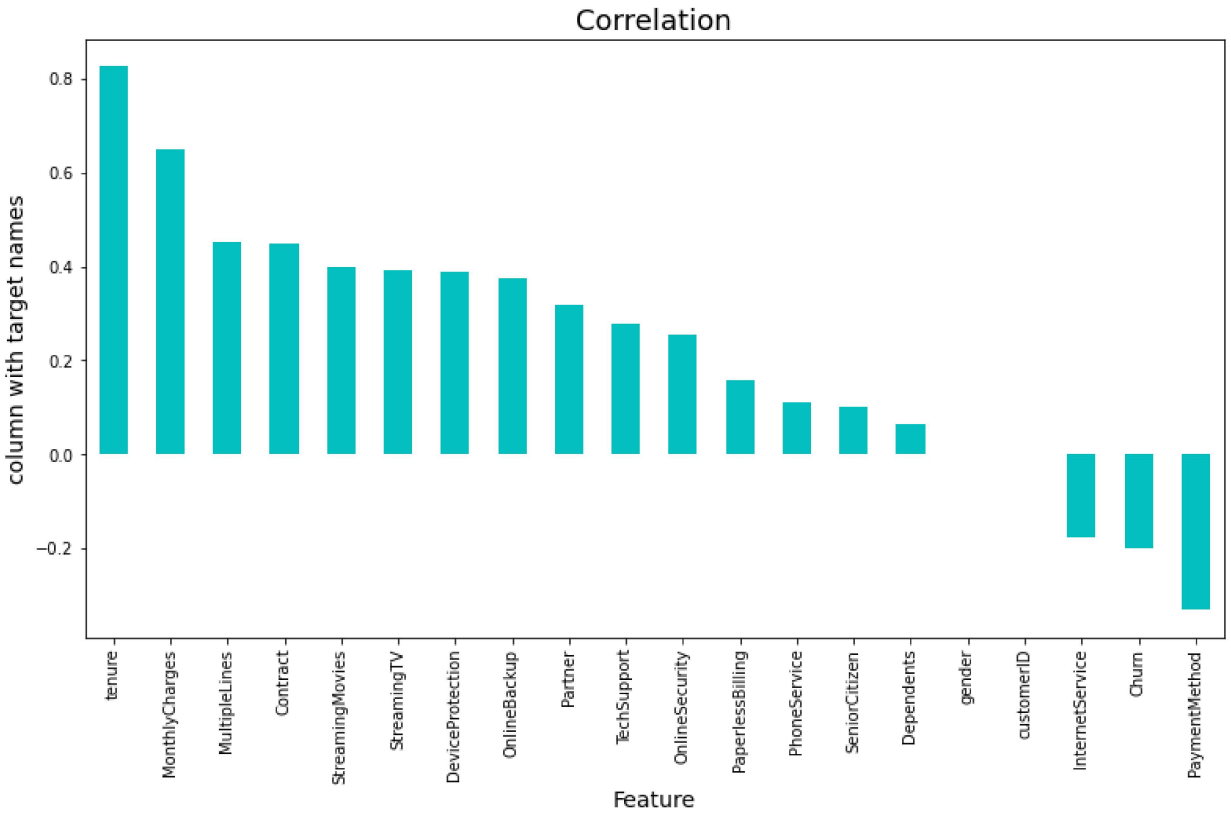
In [45]:
```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True, linewidths=0.1,linecolor="black",fmt="0.2f")
```

Out[45]:
```
<AxesSubplot:>
```



Cheking the columns which are positively and negative correlated with the target columns:

In [46]:
```
plt.figure(figsize=(13,7))
df.corr()['TotalCharges'].sort_values(ascending=False).drop(["TotalCharges"]).plot(kir
plt.xlabel('Feature',fontsize=14)
plt.ylabel('column with target names',fontsize=14)
plt.title('Correlation',fontsize=18)
plt.show()
```



Skewness:

In [47]:
```
df.skew()
```

Out[47]:
```
customerID          0.000000
gender             -0.019031
SeniorCitizen       1.833633
Partner             0.067922
Dependents          0.875199
tenure              0.239540
PhoneService       -2.727153
MultipleLines       0.118719
InternetService     0.205423
OnlineSecurity      0.416985
OnlineBackup        0.182930
DeviceProtection    0.186847
TechSupport         0.402365
StreamingTV         0.028486
StreamingMovies     0.014657
Contract            0.630959
PaperlessBilling   -0.375396
PaymentMethod      -0.170129
MonthlyCharges     -0.220524
TotalCharges        0.962394
Churn               1.063031
dtype: float64
```

Keeping+/-0.5 as the range for skewness,here are the columns which does not lie within this range

- Senior Citizen-categorical.

- Dependents-categorical,
- Phone Service-categorical
- Contract-categorical
- Total Charges- target variable.
- Churn-categorical

Since, no column has skewness,we will not treat that.

Outliers Check:

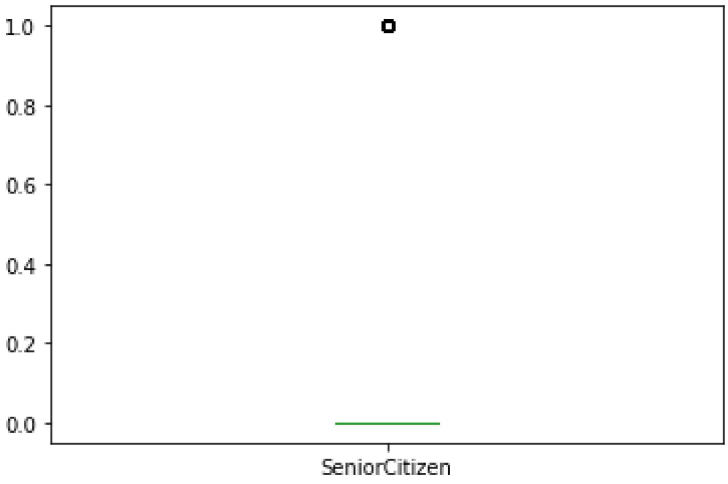
In [48]:
```
df.dtypes
```

Out[48]:
```
customerID          float64
gender              float64
SeniorCitizen         int64
Partner             float64
Dependents          float64
tenure                int64
PhoneService        float64
MultipleLines       float64
InternetService     float64
OnlineSecurity      float64
OnlineBackup        float64
DeviceProtection    float64
TechSupport         float64
StreamingTV         float64
StreamingMovies     float64
Contract            float64
PaperlessBilling    float64
PaymentMethod       float64
MonthlyCharges      float64
TotalCharges        float64
Churn               float64
dtype: object
```

Checking outliers on the int and float type of columns.

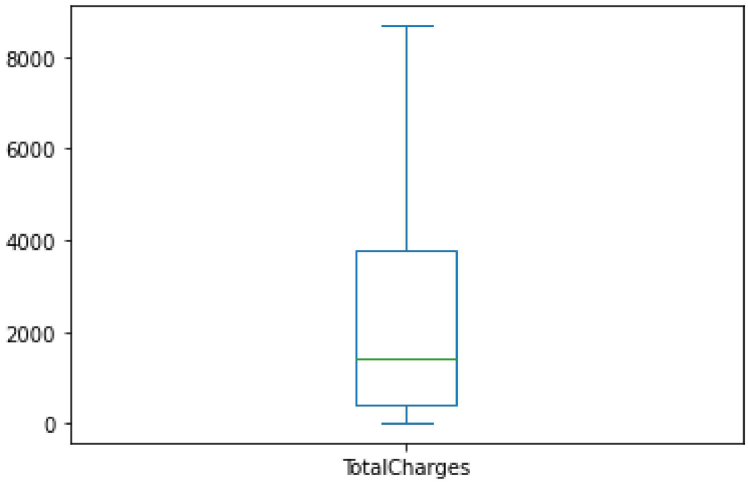
In [49]:
```
df['SeniorCitizen'].plot.box()
```

Out[49]:
```
<AxesSubplot:>
```



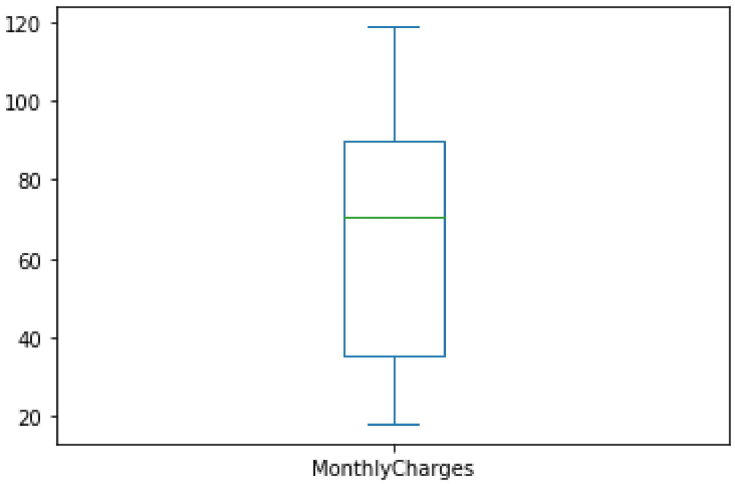
In [50]:
```
df['TotalCharges'].plot.box()
```

Out[50]: <AxesSubplot:>



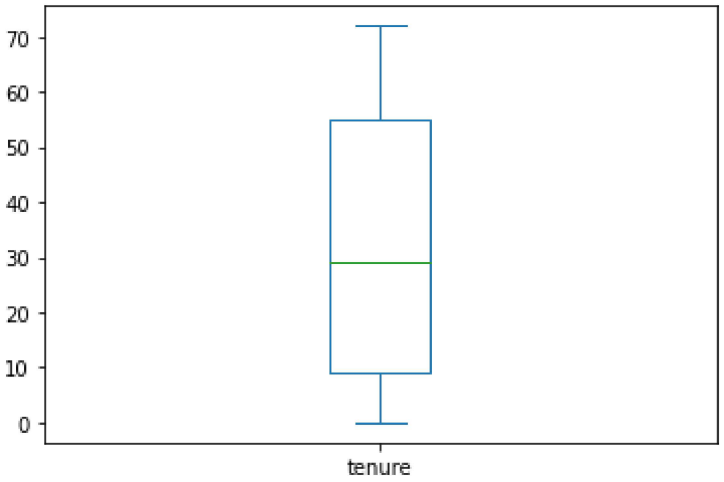
In [51]:
```
df['MonthlyCharges'].plot.box()
```

Out[51]: <AxesSubplot:>



In [52]:
```
df['tenure'].plot.box()
```

Out[52]: <AxesSubplot:>



In [53]:
```
df['SeniorCitizen'].unique()
```

Out[53]:
```
array([0, 1], dtype=int64)
```

Since the column Senior Citizen is nominal type,we can not consider its data point as outlier.

Considering the outlier removal

In [54]:
```
df.shape
```

Out[54]:
```
(7043, 21)
```

In [55]:
```
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(df))
threshold=3
np.where(z>3)
```

Out[55]:
```
(array([   0,    3,    7,   20,   27,   62,   81,   89,  103,  105,  107,
         114,  116,  129,  131,  133,  168,  180,  185,  187,  206,  211,
         215,  216,  217,  225,  236,  252,  255,  259,  260,  263,  272,
         278,  303,  321,  324,  328,  348,  354,  358,  372,  376,  382,
         387,  398,  424,  431,  435,  452,  465,  481,  488,  495,  498,
         544,  569,  596,  610,  616,  620,  634,  660,  667,  669,  674,
         677,  688,  716,  718,  735,  765,  776,  784,  790,  794,  813,
         829,  843,  847,  859,  866,  873,  875,  877,  884,  893,  917,
         934,  941,  943,  960,  973, 1011, 1018, 1037, 1050, 1051, 1053,
        1072, 1110, 1119, 1122, 1144, 1146, 1150, 1161, 1169, 1182, 1204,
        1221, 1225, 1242, 1255, 1257, 1271, 1278, 1298, 1311, 1326, 1331,
        1333, 1334, 1340, 1349, 1352, 1365, 1379, 1402, 1407, 1416, 1452,
        1479, 1480, 1481, 1500, 1506, 1513, 1519, 1560, 1562, 1581, 1584,
        1614, 1620, 1634, 1637, 1652, 1689, 1692, 1694, 1703, 1722, 1734,
        1789, 1802, 1803, 1819, 1827, 1832, 1845, 1851, 1854, 1862, 1881,
        1889, 1892, 1894, 1906, 1910, 1944, 1959, 1969, 1985, 1989, 1998,
        2002, 2031, 2046, 2050, 2087, 2089, 2090, 2117, 2124, 2127, 2131,
        2188, 2215, 2225, 2226, 2237, 2239, 2290, 2295, 2310, 2340, 2344,
        2348, 2362, 2382, 2383, 2385, 2398, 2399, 2409, 2412, 2413, 2417,
        2420, 2421, 2426, 2427, 2431, 2433, 2465, 2468, 2492, 2533, 2538,
        2541, 2547, 2562, 2608, 2610, 2626, 2637, 2644, 2661, 2662, 2681,
        2696, 2700, 2709, 2712, 2718, 2725, 2728, 2748, 2751, 2752, 2754,
        2761, 2773, 2781, 2804, 2809, 2814, 2841, 2842, 2889, 2898, 2899,
        2903, 2913, 2915, 2916, 2918, 2919, 2929, 2940, 2944, 2962, 2966,
        2972, 2990, 2992, 2994, 2995, 3020, 3028, 3036, 3039, 3042, 3043,
        3060, 3062, 3070, 3073, 3080, 3092, 3096, 3126, 3127, 3133, 3139,
        3150, 3160, 3174, 3177, 3183, 3185, 3190, 3191, 3194, 3213, 3221,
        3223, 3233, 3235, 3243, 3258, 3290, 3292, 3311, 3316, 3318, 3342,
        3354, 3363, 3370, 3414, 3422, 3444, 3454, 3492, 3502, 3505, 3541,
        3542, 3557, 3575, 3583, 3586, 3594, 3613, 3614, 3617, 3620, 3621,
        3652, 3653, 3660, 3677, 3680, 3685, 3690, 3722, 3733, 3738, 3753,
        3756, 3773, 3819, 3860, 3870, 3873, 3877, 3902, 3905, 3926, 3934,
        3940, 3945, 3946, 3955, 3961, 3973, 3976, 3983, 3989, 4008, 4020,
        4024, 4027, 4029, 4040, 4041, 4043, 4048, 4052, 4054, 4055, 4056,
        4071, 4075, 4085, 4099, 4109, 4128, 4130, 4132, 4141, 4149, 4151,
        4162, 4168, 4174, 4178, 4180, 4183, 4200, 4207, 4208, 4233, 4239,
        4251, 4281, 4290, 4309, 4310, 4311, 4338, 4369, 4396, 4400, 4402,
        4409, 4411, 4424, 4432, 4465, 4474, 4481, 4521, 4537, 4557, 4565,
        4603, 4612, 4641, 4653, 4657, 4665, 4670, 4702, 4710, 4726, 4728,
        4729, 4740, 4750, 4765, 4773, 4821, 4828, 4831, 4840, 4845, 4849,
        4854, 4857, 4860, 4882, 4883, 4897, 4898, 4915, 4919, 4924, 4933,
        4949, 4965, 4968, 4970, 4974, 4976, 4981, 4983, 4989, 4992, 4993,
        5002, 5013, 5014, 5017, 5034, 5060, 5062, 5064, 5066, 5073, 5085,
        5091, 5117, 5130, 5144, 5147, 5163, 5176, 5180, 5186, 5204, 5207,
        5210, 5212, 5216, 5249, 5263, 5264, 5284, 5290, 5292, 5296, 5303,
        5314, 5329, 5331, 5338, 5343, 5348, 5356, 5359, 5382, 5387, 5391,
        5392, 5411, 5456, 5489, 5497, 5501, 5505, 5531, 5536, 5546, 5559,
        5565, 5601, 5607, 5631, 5636, 5648, 5665, 5666, 5674, 5682, 5683,
        5690, 5717, 5740, 5761, 5788, 5790, 5796, 5799, 5829, 5833, 5837,
        5841, 5880, 5884, 5889, 5891, 5900, 5911, 5939, 5941, 5942, 5949,
        5950, 5954, 5961, 5967, 5976, 5983, 6001, 6006, 6007, 6020, 6030,
        6031, 6039, 6043, 6059, 6064, 6067, 6074, 6080, 6087, 6093, 6108,
        6129, 6132, 6133, 6145, 6149, 6162, 6174, 6183, 6204, 6209, 6212,
        6218, 6219, 6220, 6235, 6248, 6252, 6253, 6256, 6260, 6263, 6269,
        6285, 6296, 6310, 6319, 6326, 6331, 6367, 6377, 6383, 6392, 6406,
        6415, 6416, 6424, 6425, 6435, 6455, 6457, 6459, 6493, 6494, 6500,
        6503, 6509, 6514, 6515, 6522, 6523, 6530, 6536, 6547, 6553, 6570,
        6573, 6593, 6600, 6607, 6624, 6640, 6653, 6661, 6662, 6665, 6677,
        6679, 6683, 6684, 6691, 6693, 6703, 6727, 6747, 6750, 6752, 6757,
        6777, 6779, 6783, 6791, 6810, 6811, 6813, 6834, 6864, 6881, 6884,
```

```
        6891, 6895, 6904, 6905, 6937, 6940, 6941, 6943, 6946, 6949, 6964,
        6966, 6979, 6980, 6984, 6985, 6999, 7003, 7007, 7029, 7036, 7040],
       dtype=int64),
 array([6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6],
       dtype=int64))
```

In [56]:
```
df_new_z=df[(z<3).all(axis=1)]
df_new_z
```

Out[56]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
1	3962.0	1.0	0	0.0	0.0	34	1.0	0.0
2	2564.0	1.0	0	0.0	0.0	2	1.0	0.0
4	6511.0	0.0	0	0.0	0.0	2	1.0	0.0
5	6551.0	0.0	0	0.0	0.0	8	1.0	2.0
6	1002.0	1.0	0	0.0	1.0	22	1.0	2.0
...	...	...	...	...	...	...	...	...
7037	1758.0	0.0	0	0.0	0.0	72	1.0	0.0
7038	4853.0	1.0	0	1.0	1.0	24	1.0	2.0
7039	1525.0	0.0	0	1.0	1.0	72	1.0	2.0
7041	5934.0	1.0	1	1.0	0.0	4	1.0	2.0
7042	2226.0	1.0	0	0.0	0.0	66	1.0	0.0

6361 rows × 21 columns

In [57]:
```
df_new_z.shape
```

Out[57]:
```
(6361, 21)
```

In [58]:
```
# Percentange loss of data:
```

In [59]:
```
data_loss=((7043-6361)/7043)*100

data_loss
```

Out[59]:
```
9.683373562402386
```

Our data is becoming biased as it not considering the case of Senior Citizen.Thus,we will not remove outliers.

Separating the columns into features and target:

In [60]:
```
x=df.drop("TotalCharges",axis=1)
y=df["TotalCharges"]
```

Scaling the data using Min-Max Scaler:

In [61]:
```
from sklearn.preprocessing import MinMaxScaler
mns=MinMaxScaler()
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [62]:
```
for i in range(0,20):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=i)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    print(f"At the random state{i},the training accuracy is -: {r2_score(y_train,pred_
    print(f"At the random state{i},the training accuracy is = {r2_score(y_test,pred_te

    print("\n")
```

```
At the random state0,the training accuracy is -: 0.9058706722803964
At the random state0,the training accuracy is = 0.8977317673785854


At the random state1,the training accuracy is -: 0.9052277082152647
At the random state1,the training accuracy is = 0.9003132716690803


At the random state2,the training accuracy is -: 0.9056858398935238
At the random state2,the training accuracy is = 0.8984735750686544


At the random state3,the training accuracy is -: 0.9044956724241888
At the random state3,the training accuracy is = 0.9032713403489679


At the random state4,the training accuracy is -: 0.9031354749784841
At the random state4,the training accuracy is = 0.9085937883962479


At the random state5,the training accuracy is -: 0.9026338340566921
At the random state5,the training accuracy is = 0.9101290251842342


At the random state6,the training accuracy is -: 0.9028087364691222
At the random state6,the training accuracy is = 0.9097783155230829


At the random state7,the training accuracy is -: 0.9030731852162728
At the random state7,the training accuracy is = 0.9090152662295625


At the random state8,the training accuracy is -: 0.9031695399147395
At the random state8,the training accuracy is = 0.9089840252888701


At the random state9,the training accuracy is -: 0.9034746951288706
At the random state9,the training accuracy is = 0.9074646730902043


At the random state10,the training accuracy is -: 0.9039273530214657
At the random state10,the training accuracy is = 0.9057313444009284


At the random state11,the training accuracy is -: 0.90416749132796
At the random state11,the training accuracy is = 0.904768680713289


At the random state12,the training accuracy is -: 0.904295058479142
At the random state12,the training accuracy is = 0.9042173834952026


At the random state13,the training accuracy is -: 0.9061128342153906
At the random state13,the training accuracy is = 0.897031636227697


At the random state14,the training accuracy is -: 0.9032235219356887
At the random state14,the training accuracy is = 0.9085611847383555
```

```
At the random state15,the training accuracy is -: 0.9036417240520397
At the random state15,the training accuracy is = 0.9065354335769835


At the random state16,the training accuracy is -: 0.902513180979179
At the random state16,the training accuracy is = 0.910871863840758


At the random state17,the training accuracy is -: 0.9029763238324856
At the random state17,the training accuracy is = 0.9090481298163142


At the random state18,the training accuracy is -: 0.9040046108882651
At the random state18,the training accuracy is = 0.9053965803704586


At the random state19,the training accuracy is -: 0.9034412311305086
At the random state19,the training accuracy is = 0.9074326529099017
```

In [63]:
```
lr.fit(x_train,y_train)
```

Out[63]:
```
▾ LinearRegression

LinearRegression()
```

In [64]:
```
pred_test=lr.predict(x_test)
```

In [65]:
```
print(r2_score(y_test,pred_test))
```

```
0.9074326529099017
```

Cross-validation of the model:

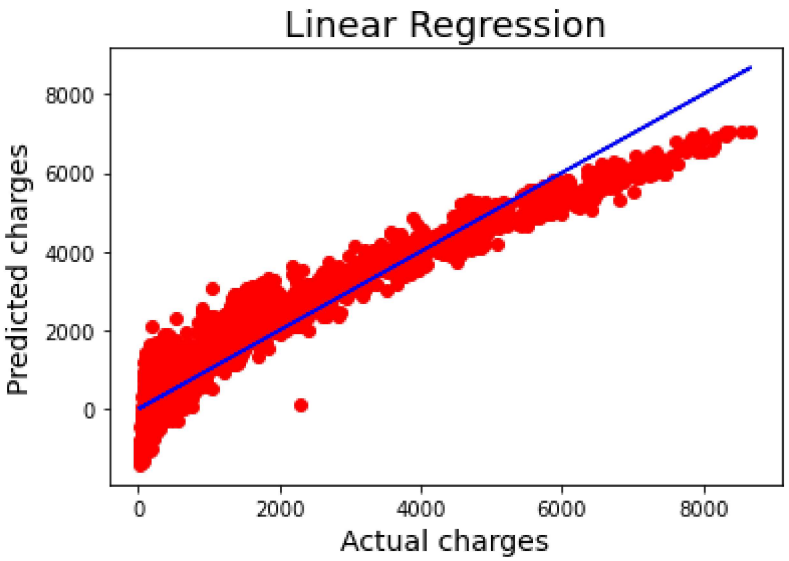
In [66]:
```
Train_accuracy=r2_score(y_train,pred_train)
Test_accuracy=r2_score(y_test,pred_test)

from sklearn.model_selection import cross_val_score
for j in range(2,6):
    cv_score=cross_val_score(lr,x,y,cv=j)
    cv_mean=cv_score.mean()
    print(f"At cross fold{j}, the cv score is {cv_mean} and accuracy score for trainir
    print("\n")
```

```
At cross fold2, the cv score is 0.9033382537767014 and accuracy score for training is
0.9034412311305086 and accuracy for the testing is 0.9074326529099017


At cross fold3, the cv score is 0.9037012261875313 and accuracy score for training is
0.9034412311305086 and accuracy for the testing is 0.9074326529099017


At cross fold4, the cv score is 0.9037044584134893 and accuracy score for training is
0.9034412311305086 and accuracy for the testing is 0.9074326529099017


At cross fold5, the cv score is 0.9036665825623267 and accuracy score for training is
0.9034412311305086 and accuracy for the testing is 0.9074326529099017
```

Since the number of folds don't have such impact on the accuracy and cv_score.So cv=5 is selected

Here we have handled the problem of the overfiting and the underfiting by checking the training and testing score.

In [67]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
plt.scatter(x=y_test, y=pred_test,color='r')
plt.plot(y_test,y_test,color='b')
plt.xlabel('Actual charges',fontsize=14)
plt.ylabel('Predicted charges',fontsize=14 )
plt.title('Linear Regression',fontsize=18)
plt.show()
```



Best fot line is covering most of the datapoints which shows good fit of our model.

Regularization:

In [68]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```
import warnings
warnings.filterwarnings('ignore')
```

In [69]:
```
from sklearn.linear_model import Lasso

parameters={'alpha':[.0001,.001, .01, .1, 1, 10],'random_state':list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(x_train,y_train)

print(clf.best_params_)
```

```
{'alpha': 0.1, 'random_state': 0}
```

In [70]:
```
ls=Lasso(alpha=1,random_state=0)
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
pred_ls=ls.predict(x_test)
lss=r2_score(y_test,pred_ls)
lss
```

Out[70]:
```
0.9075609764751776
```

In [71]:
```
cv_score=cross_val_score(ls,x,y,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

Out[71]:
```
0.9036629382160928
```

Ensemble technique:

In [72]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

parameters={'criterion':['mse','mae'],'max_features':["auto","sqrt","log2"]}
rf=RandomForestRegressor()
clf=GridSearchCV(rf,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'criterion': 'mae', 'max_features': 'auto'}
```

In [73]:
```
rf=RandomForestRegressor(criterion='mse',max_features='auto')
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision=rf.predict(x_test)

rfs=r2_score(y_test,pred_decision)
print('R2 Score :',rfs*100)

rfscore=cross_val_score(rf,x,y,cv=5)
rfc=rfscore.mean()
print('Cross val Score:',rfc*100)
```

```
R2 Score : 99.87648420286278
Cross val Score: 99.87186457504545
```

We are getting model accuracy and cross validation both as

99.8% which show our model is performing extermely well

In [74]:
```
import pickle
filename='churn.pkl'
pickle.dump(rf,open(filename,'wb'))
```

Conclusion:

In [75]:
```
loaded_model=pickle.load(open('churn.pkl','rb'))
result=loaded_model.score(x_test,y_test)
print(result)
```

```
0.9987648420286278
```

conclusion=pd.DataFrame([loaded_model.predict(features_test)[:],pred_decision[:]],index=["predicted","Original"])

In [76]:
```
conclusion=pd.DataFrame([loaded_model.predict(x_test)[:],pred_decision[:]],index=["pre
```

In [77]:
```
conclusion
```

Out[77]:

	0	1	2	3	4	5	6	7	8	9
predicted	35.439	80.0755	158.741	760.8425	3083.3585	2444.288	197.074	893.2915	851.8795	70.90
Original	35.439	80.0755	158.741	760.8425	3083.3585	2444.288	197.074	893.2915	851.8795	70.90

2 rows × 1409 columns

In [ ]:

In [ ]: