In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

In [3]:
```
df=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/DSData/master/Tele
```

In [4]:
```
print(" Data has\n Rows = ",df.shape[0],'\n Columns = ',df.shape[1])
```

```
 Data has
 Rows =  7043
 Columns =  21
```

In [5]:
```
df = df.replace(r'^\s*$', np.nan, regex=True)
```

In [6]:
```
df.head()
```

Out[6]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Ir
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

In [7]:
```
pd.set_option('display.max_columns',None)
```

In [8]:
```
df.head()
```

Out[8]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Ir
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	

In [9]:
```
df.tail()
```

Out[9]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes
7042	3186-AJIEK	Male	0	No	No	66	Yes	No

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7032 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [11]:
```
df.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
SeniorCitizen	7043.0	0.162147	0.368612	0.00	0.0	0.00	0.00	1.00
tenure	7043.0	32.371149	24.559481	0.00	9.0	29.00	55.00	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.5	70.35	89.85	118.75

Missing Values

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

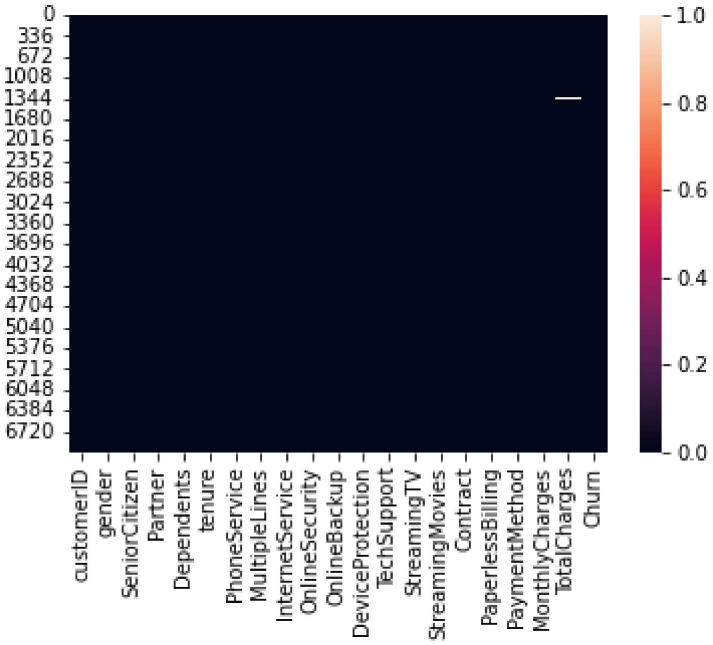
Out[12]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

In [13]:
```
(df.isnull().sum()/df.shape[0])*100
```

Out[13]:
```
customerID          0.000000
gender              0.000000
SeniorCitizen       0.000000
Partner             0.000000
Dependents          0.000000
tenure              0.000000
PhoneService        0.000000
MultipleLines       0.000000
InternetService     0.000000
OnlineSecurity      0.000000
OnlineBackup        0.000000
DeviceProtection    0.000000
TechSupport         0.000000
StreamingTV         0.000000
StreamingMovies     0.000000
Contract            0.000000
PaperlessBilling    0.000000
PaymentMethod       0.000000
MonthlyCharges      0.000000
TotalCharges        0.156183
Churn               0.000000
dtype: float64
```

In [14]:
```
sns.heatmap(df.isnull())
```

Out[14]: <AxesSubplot:>



Univarient Exploratory Data Analysis

In [15]:
```
df.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')
```

In [16]:
```
df['customerID'].nunique()
```

Out[16]: 7043

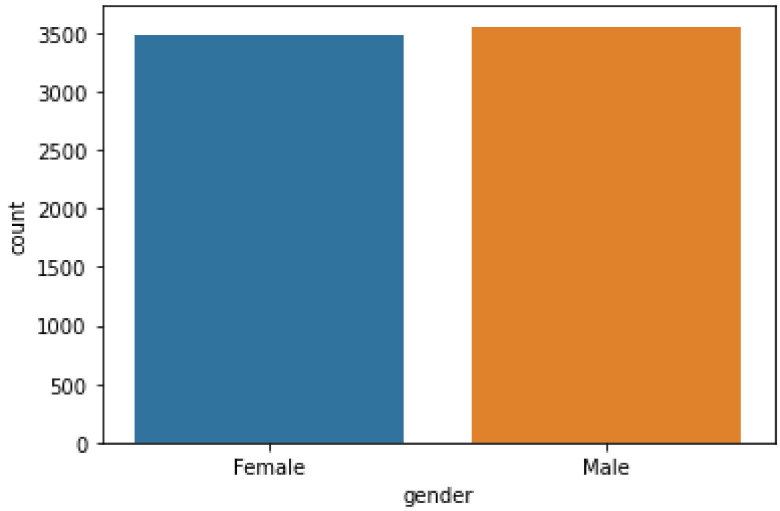
In [17]:
```
# Customer ID is unique, It will be irrelevant to predict customer churn, We can
```

In [18]:
```
df['gender'].unique()
```

Out[18]:
```
array(['Female', 'Male'], dtype=object)
```

In [19]:
```
sns.countplot(df['gender'])
```

Out[19]:
```
<AxesSubplot:xlabel='gender', ylabel='count'>
```



In [20]:
```
df['gender'].value_counts(normalize=True)*100
```

Out[20]:
```
Male      50.47565
Female    49.52435
Name: gender, dtype: float64
```

In [21]:
```
df['SeniorCitizen'].unique()
```

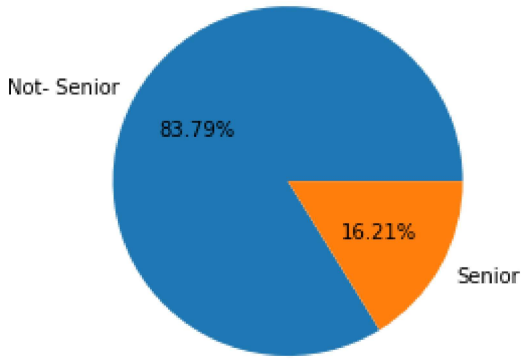
Out[21]:
```
array([0, 1], dtype=int64)
```

In [22]:
```
df['SeniorCitizen'].value_counts(normalize=True)*100
```

Out[22]:
```
0    83.785319
1    16.214681
Name: SeniorCitizen, dtype: float64
```

In [23]:
```
plt.pie((df['SeniorCitizen'].value_counts(normalize=True)*100).values,labels=['No
```

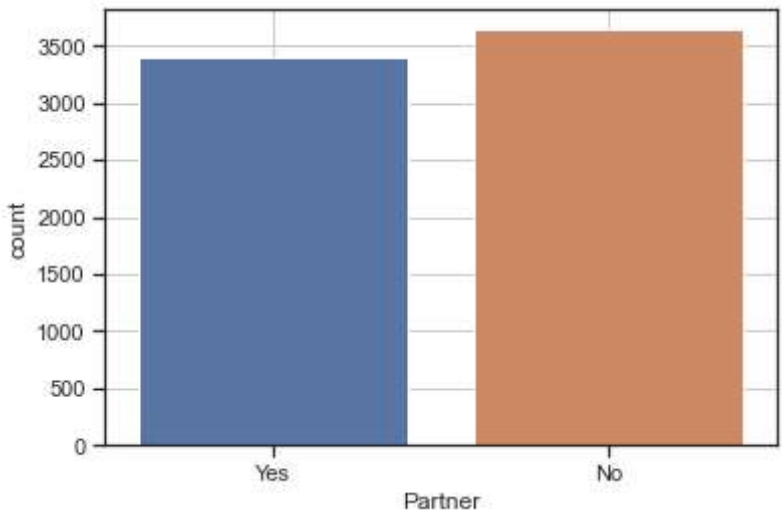
Out[23]:
```
([<matplotlib.patches.Wedge at 0x14ef7ea3f10>,
  <matplotlib.patches.Wedge at 0x14ef7eb16d0>],
 [Text(-0.9603414027015803, 0.5364181114179138, 'Not- Senior'),
  Text(0.9603414278131233, -0.5364180664610878, 'Senior')],
 [Text(-0.523822583291771, 0.29259169713704386, '83.79%'),
  Text(0.5238225969889763, -0.29259167261513874, '16.21%')])
```



In [24]:
```
df['Partner'].unique()
```

Out[24]:
```
array(['Yes', 'No'], dtype=object)
```

In [25]:
```
sns.set(style="ticks")
sns.countplot(df['Partner'])
plt.grid()
```



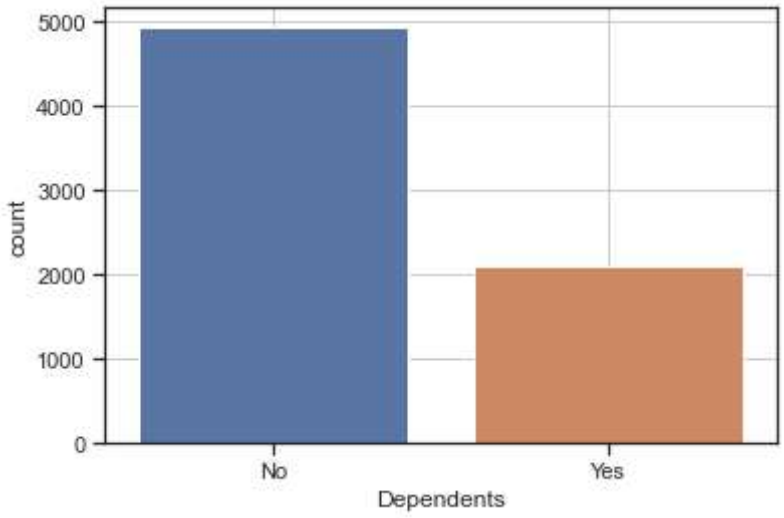
In [26]:
```
df['Partner'].value_counts(normalize=True)*100
```

Out[26]:
```
No     51.69672
Yes    48.30328
Name: Partner, dtype: float64
```

In [27]:
```
df['Dependents'].unique()
```

Out[27]:
```
array(['No', 'Yes'], dtype=object)
```

In [28]:
```
sns.countplot(df['Dependents'])
plt.grid()
```



In [29]:
```
df['Dependents'].value_counts(normalize=True)*100
```

Out[29]:
```
No     70.041176
Yes    29.958824
Name: Dependents, dtype: float64
```

tenure

In [30]:
```
df['tenure'].unique()
```

Out[30]:
```
array([ 1, 34,  2, 45,  8, 22, 10, 28, 62, 13, 16, 58, 49, 25, 69, 52, 71,
       21, 12, 30, 47, 72, 17, 27,  5, 46, 11, 70, 63, 43, 15, 60, 18, 66,
        9,  3, 31, 50, 64, 56,  7, 42, 35, 48, 29, 65, 38, 68, 32, 55, 37,
       36, 41,  6,  4, 33, 67, 23, 57, 61, 14, 20, 53, 40, 59, 24, 44, 19,
       54, 51, 26,  0, 39], dtype=int64)
```

In [31]:
```
df['tenure'].nunique()
```

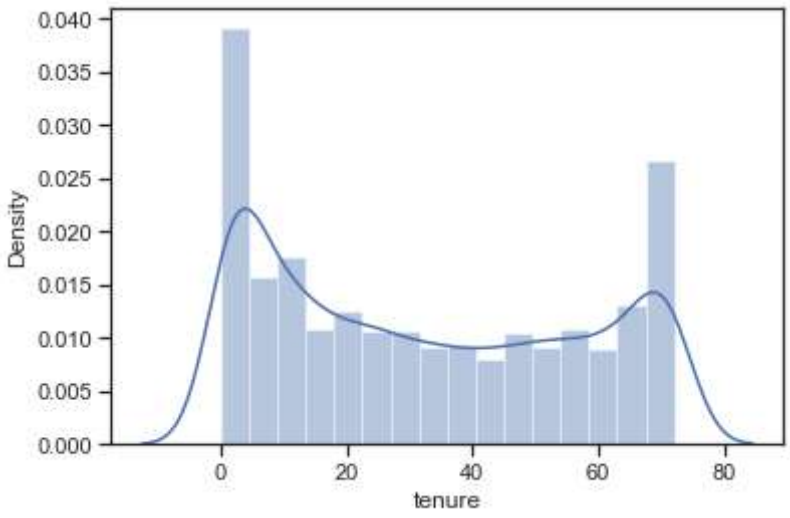
Out[31]:
```
73
```

In [32]:
```
df['tenure'].value_counts()
```

Out[32]:
```
1     613
72    362
2     238
3     200
4     176
     ...
28     57
39     56
44     51
36     50
0      11
Name: tenure, Length: 73, dtype: int64
```

In [33]:
```
sns.distplot(df['tenure'])
```

Out[33]:
```
<AxesSubplot:xlabel='tenure', ylabel='Density'>
```

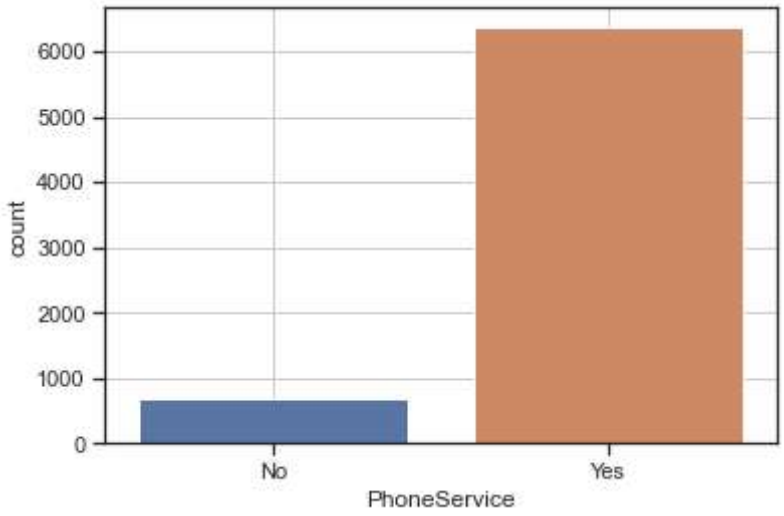


PhoneService

In [34]:
```
df['PhoneService'].unique()
```

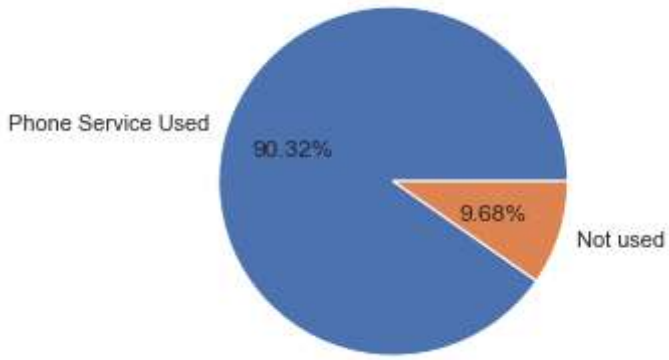
Out[34]:
```
array(['No', 'Yes'], dtype=object)
```

In [35]:
```
sns.countplot(df['PhoneService'])
plt.grid()
```



In [36]:
```
plt.pie(df['PhoneService'].value_counts().values,labels=['Phone Service Used','No
```

Out[36]:
```
([<matplotlib.patches.Wedge at 0x14ef80d9160>,
  <matplotlib.patches.Wedge at 0x14ef80d98e0>],
 [Text(-1.0494915966975706, 0.3294956577273573, 'Phone Service Used'),
  Text(1.049491627547176, -0.32949555946686127, 'Not used')],
 [Text(-0.5724499618350385, 0.17972490421492215, '90.32%'),
  Text(0.5724499786620959, -0.17972485061828794, '9.68%')])
```



MultipleLines

In [37]:
```
df['MultipleLines'].unique()
```

Out[37]:
```
array(['No phone service', 'No', 'Yes'], dtype=object)
```

In [38]:
```
df['MultipleLines'].value_counts(normalize=True)*100
```

Out[38]:
```
No                  48.132898
Yes                 42.183729
No phone service     9.683374
Name: MultipleLines, dtype: float64
```

MultipleLines have the information about Phone Service

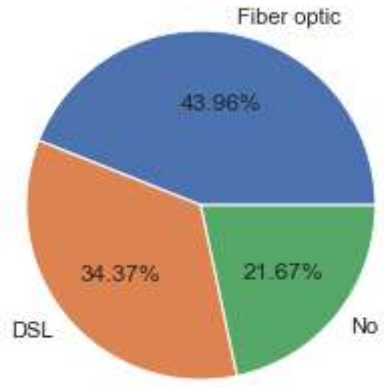
InternetService

In [39]:
```
df['InternetService'].unique()
```

Out[39]:
```
array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

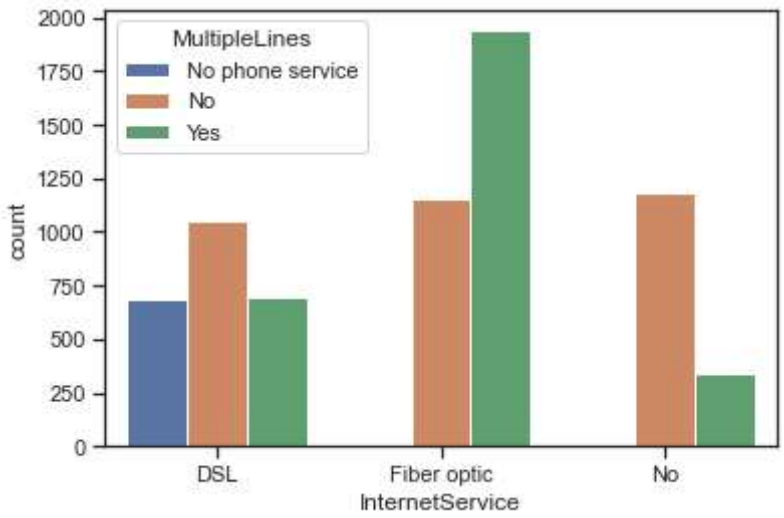
In [40]:
```
plt.pie(df['InternetService'].value_counts().values,labels=df['InternetService'].
```

Out[40]:
```
([<matplotlib.patches.Wedge at 0x14ef812c0a0>,
  <matplotlib.patches.Wedge at 0x14ef812c8e0>,
  <matplotlib.patches.Wedge at 0x14ef812cfd0>],
 [Text(0.20752658810331015, 1.0802465992680557, 'Fiber optic'),
  Text(-0.8411049099748845, -0.7089023419457307, 'DSL'),
  Text(0.8548554654106661, -0.6922587184409551, 'No')],
 [Text(0.11319632078362371, 0.5892254177825758, '43.96%'),
  Text(-0.4587844963499369, -0.38667400469767127, '34.37%'),
  Text(0.46628479931490874, -0.37759566460415733, '21.67%')])
```



In [41]:
```
sns.countplot(df['InternetService'],hue=df['MultipleLines'])
```

Out[41]:
```
<AxesSubplot:xlabel='InternetService', ylabel='count'>
```



In [42]:
```
df.columns
```

Out[42]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

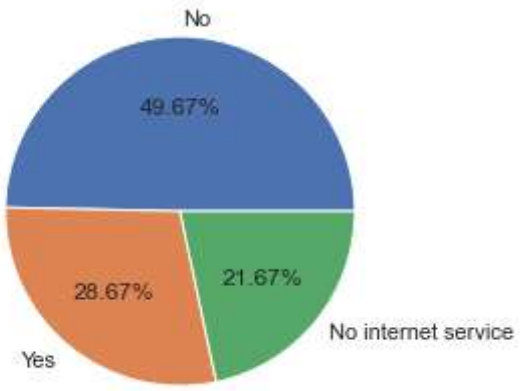
Some extra services like

In [43]:
```
df['OnlineSecurity'].unique()
```

Out[43]:
```
array(['No', 'Yes', 'No internet service'], dtype=object)
```

In [44]:
```
plt.pie(df['OnlineSecurity'].value_counts().values,labels=df['OnlineSecurity'].va
```

Out[44]:
```
([<matplotlib.patches.Wedge at 0x14ef91d12e0>,
  <matplotlib.patches.Wedge at 0x14ef91d1af0>,
  <matplotlib.patches.Wedge at 0x14ef91e01f0>],
 [Text(0.011530382742536482, 1.0999395666461913, 'No'),
  Text(-0.7011814628356199, -0.8475520964375585, 'Yes'),
  Text(0.8548554654106661, -0.6922587184409551, 'No internet service')],
 [Text(0.006289299677747172, 0.5999670363524678, '49.67%'),
  Text(-0.38246261609215626, -0.46230114351139545, '28.67%'),
  Text(0.46628479931490874, -0.37759566460415733, '21.67%')])
```



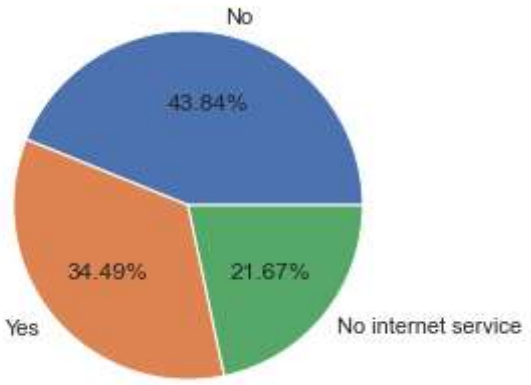
OnlineBackup

In [45]:
```
df['OnlineBackup'].unique()
```

Out[45]:
```
array(['Yes', 'No', 'No internet service'], dtype=object)
```

In [46]:
```
plt.pie(df['OnlineBackup'].value_counts().values,labels=df['OnlineBackup'].value_
```

Out[46]:
```
([<matplotlib.patches.Wedge at 0x14ef9221640>,
  <matplotlib.patches.Wedge at 0x14ef9221df0>,
  <matplotlib.patches.Wedge at 0x14ef9230550>],
 [Text(0.21138010801255863, 1.0794991662509976, 'No'),
  Text(-0.8436292603186925, -0.7058963600516268, 'Yes'),
  Text(0.8548554654106661, -0.6922587184409551, 'No internet service')],
 [Text(0.11529824073412287, 0.5888177270459987, '43.84%'),
  Text(-0.46016141471928673, -0.38503437820997827, '34.49%'),
  Text(0.46628479931490874, -0.37759566460415733, '21.67%')])
```



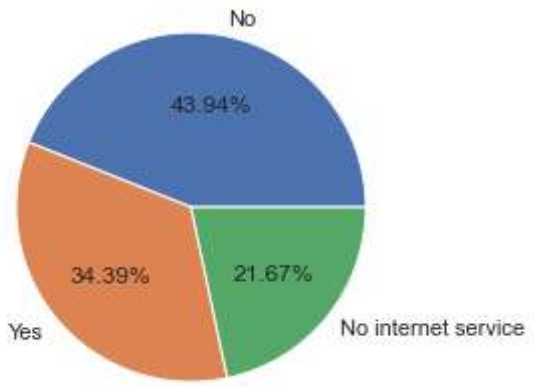
DeviceProtection

In [47]:
```
df['DeviceProtection'].unique()
```

Out[47]:
```
array(['No', 'Yes', 'No internet service'], dtype=object)
```

In [48]:
```
plt.pie(df['DeviceProtection'].value_counts().values,labels=df['DeviceProtection'
```

Out[48]:
```
([<matplotlib.patches.Wedge at 0x14ef9273760>,
  <matplotlib.patches.Wedge at 0x14ef9273f40>,
  <matplotlib.patches.Wedge at 0x14ef92836a0>],
 [Text(0.20800849946792185, 1.080153907621087, 'No'),
  Text(-0.8414211562608911, -0.7085269492380549, 'Yes'),
  Text(0.8548553357827734, -0.692258878515633, 'No internet service')],
 [Text(0.11345918152795736, 0.589174858702411, '43.94%'),
  Text(-0.45895699432412235, -0.386469245038939, '34.39%'),
  Text(0.46628472860878545, -0.3775957519176179, '21.67%')])
```

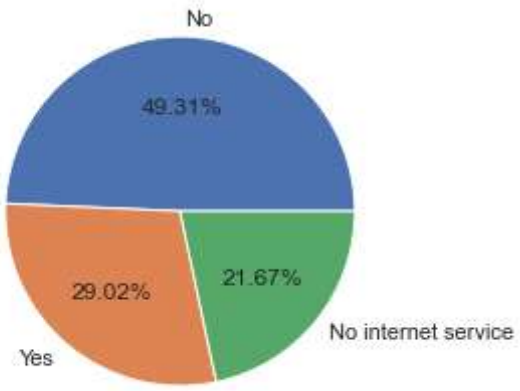


In [49]:
```
df['TechSupport'].unique()
```

Out[49]:
```
array(['No', 'Yes', 'No internet service'], dtype=object)
```

In [50]:
```
plt.pie(df['TechSupport'].value_counts().values,labels=df['TechSupport'].value_co
```

Out[50]:
```
([<matplotlib.patches.Wedge at 0x14ef92c7940>,
  <matplotlib.patches.Wedge at 0x14ef92d61c0>,
  <matplotlib.patches.Wedge at 0x14ef92d68e0>],
 [Text(0.023795397047164612, 1.0997425967376948, 'No'),
  Text(-0.7105891537173198, -0.8396803288271695, 'Yes'),
  Text(0.8548554654106661, -0.6922587184409551, 'No internet service')],
 [Text(0.012979307480271604, 0.5998595982205608, '49.31%'),
  Text(-0.38759408384581073, -0.45800745208754695, '29.02%'),
  Text(0.46628479931490874, -0.37759566460415733, '21.67%')])
```



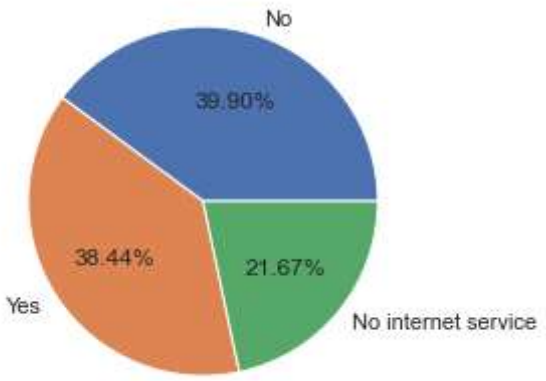
StreamingTV

In [51]:
```
df['StreamingTV'].unique()
```

Out[51]:
```
array(['No', 'Yes', 'No internet service'], dtype=object)
```

In [52]:
```
plt.pie(df['StreamingTV'].value_counts().values,labels=df['StreamingTV'].value_co
```

Out[52]:
```
([<matplotlib.patches.Wedge at 0x14ef8077100>,
  <matplotlib.patches.Wedge at 0x14ef805d910>,
  <matplotlib.patches.Wedge at 0x14ef805d640>],
 [Text(0.3432768520552361, 1.0450650711046885, 'No'),
  Text(-0.9244614473012523, -0.5961300465952661, 'Yes'),
  Text(0.8548553357827734, -0.692258878515633, 'No internet service')],
 [Text(0.18724191930285602, 0.5700354933298301, '39.90%'),
  Text(-0.5042516985279557, -0.3251618435974179, '38.44%'),
  Text(0.46628472860878545, -0.3775957519176179, '21.67%')])
```



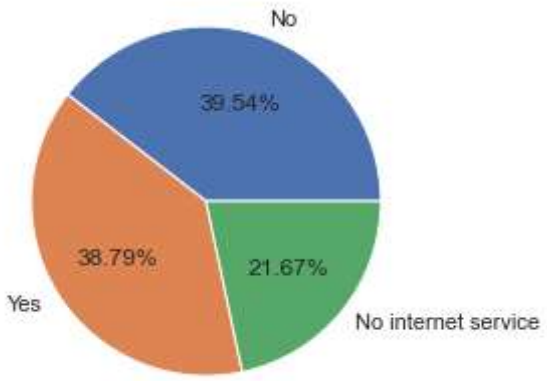
StreamingMovies

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

Out[53]:
```
array(['No', 'Yes', 'No internet service'], dtype=object)
```

In [54]:
```
plt.pie(df['StreamingMovies'].value_counts().values,labels=df['StreamingMovies'].
```

Out[54]:
```
([<matplotlib.patches.Wedge at 0x14ef7f2bbe0>,
  <matplotlib.patches.Wedge at 0x14ef7f07040>,
  <matplotlib.patches.Wedge at 0x14ef7f07f10>],
 [Text(0.3549092210143265, 1.0411721494733732, 'No'),
  Text(-0.9310514684948953, -0.5857842290583618, 'Yes'),
  Text(0.8548554654106661, -0.6922587184409551, 'No internet service')],
 [Text(0.19358684782599625, 0.5679120815309308, '39.54%'),
  Text(-0.50784625554267, -0.31951867039547005, '38.79%'),
  Text(0.46628479931490874, -0.37759566460415733, '21.67%')])
```



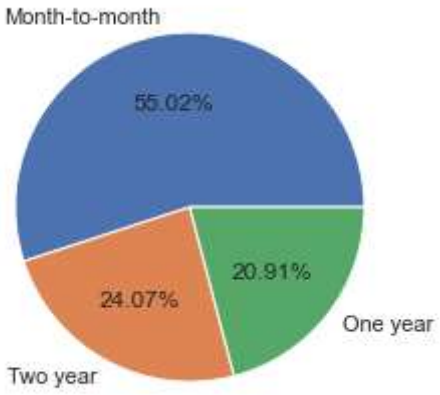
Contract

In [55]:
```
df['Contract'].unique()
```

Out[55]:
```
array(['Month-to-month', 'One year', 'Two year'], dtype=object)
```

In [56]:
```
plt.pie(df['Contract'].value_counts().values,labels=df['Contract'].value_counts()
```

Out[56]:
```
([<matplotlib.patches.Wedge at 0x14ef8051280>,
  <matplotlib.patches.Wedge at 0x14ef92f5b20>,
  <matplotlib.patches.Wedge at 0x14ef7ee2280>],
 [Text(-0.17273219343040325, 1.0863533446134006, 'Month-to-month'),
  Text(-0.526752873107977, -0.9656766594841628, 'Two year'),
  Text(0.8709808402501051, -0.6718574074290027, 'One year')],
 [Text(-0.09421756005294721, 0.5925563697891275, '55.02%'),
  Text(-0.2873197489679874, -0.5267327233549979, '24.07%'),
  Text(0.4750804583182391, -0.36646767677945596, '20.91%')])
```



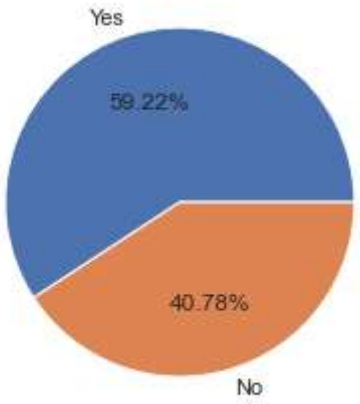
PaperlessBilling

In [57]:
```
df['PaperlessBilling'].unique()
```

Out[57]:
```
array(['Yes', 'No'], dtype=object)
```

In [58]:
```
plt.pie(df['PaperlessBilling'].value_counts().values,labels=df['PaperlessBilling'
```

Out[58]:
```
([<matplotlib.patches.Wedge at 0x14ef7f46790>,
  <matplotlib.patches.Wedge at 0x14ef7f463a0>],
 [Text(-0.31424730634794096, 1.0541577825226467, 'Yes'),
  Text(0.31424730634794035, -1.0541577825226467, 'No')],
 [Text(-0.17140762164433143, 0.5749951541032617, '59.22%'),
  Text(0.1714076216443311, -0.5749951541032617, '40.78%')])
```



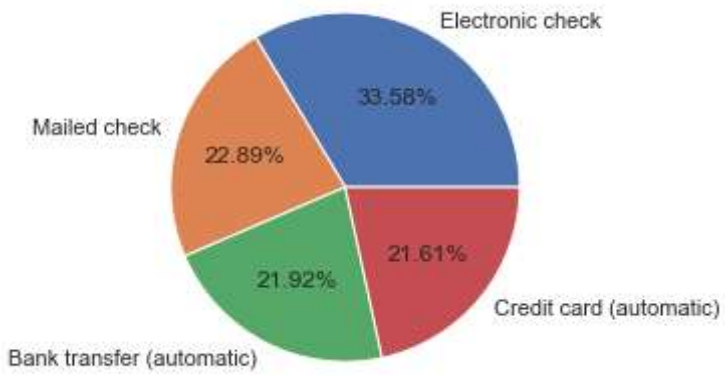
PaymentMethod

In [59]:
```
df['PaymentMethod'].unique()
```

Out[59]:
```
array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',
       'Credit card (automatic)'], dtype=object)
```

In [60]:
```
plt.pie(df['PaymentMethod'].value_counts().values,labels=df['PaymentMethod'].valu
```

Out[60]:
```
([<matplotlib.patches.Wedge at 0x14ef9341fa0>,
  <matplotlib.patches.Wedge at 0x14ef934f760>,
  <matplotlib.patches.Wedge at 0x14ef934fe80>,
  <matplotlib.patches.Wedge at 0x14ef935a5e0>],
 [Text(0.5426181731482101, 0.956851878907754, 'Electronic check'),
  Text(-1.0466614167922357, 0.33837830692653775, 'Mailed check'),
  Text(-0.5037781622923265, -0.9778586621784185, 'Bank transfer (automatic)'),
  Text(0.8560892632593112, -0.690732345653604, 'Credit card (automatic)')],
 [Text(0.2959735489899328, 0.5219192066769567, '33.58%'),
  Text(-0.5709062273412193, 0.18456998559629328, '22.89%'),
  Text(-0.27478808852308717, -0.5333774520973191, '21.92%'),
  Text(0.46695777995962423, -0.3767630976292385, '21.61%')])
```



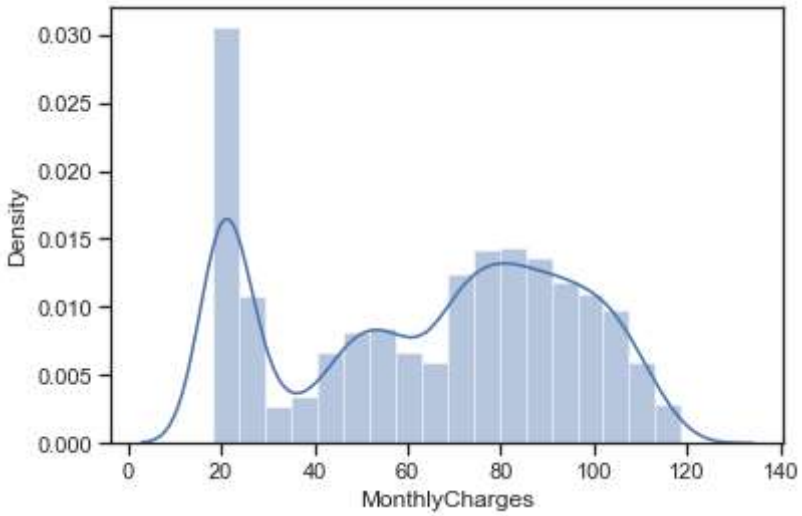
MonthlyCharges

In [61]:
```
df['MonthlyCharges'].unique()
```

Out[61]:
```
array([29.85, 56.95, 53.85, ..., 63.1 , 44.2 , 78.7 ])
```

In [62]:
```
sns.distplot(df['MonthlyCharges'])
```

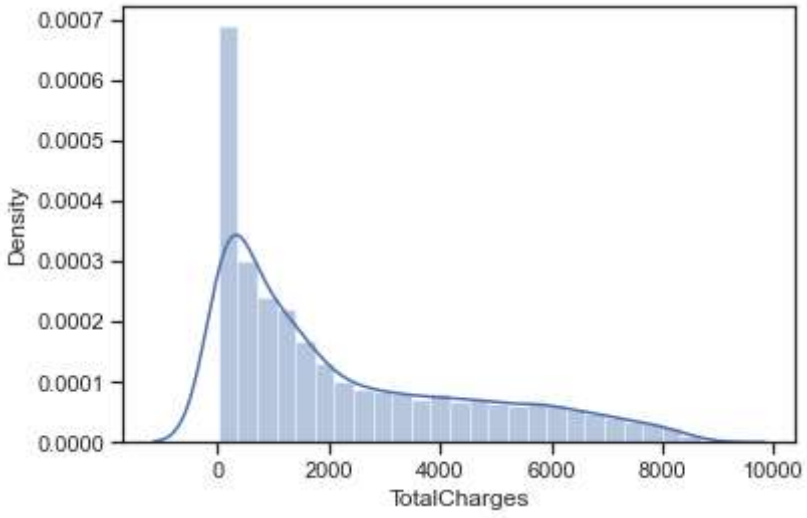
Out[62]:
```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='Density'>
```



TotalCharges

In [63]:
```
sns.distplot(df['TotalCharges'])
```

Out[63]:
```
<AxesSubplot:xlabel='TotalCharges', ylabel='Density'>
```



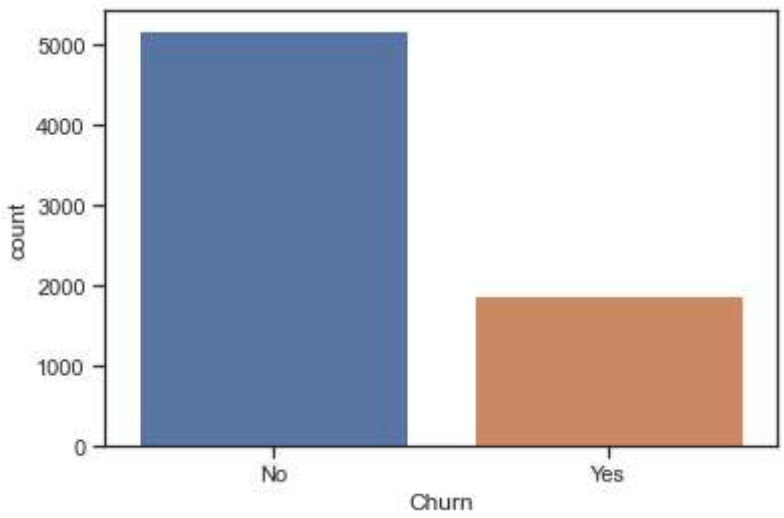
In [64]:
```
df['TotalCharges']=df['TotalCharges'].astype(float)
```

In [65]:
```
df['Churn'].unique()
```

Out[65]:
```
array(['No', 'Yes'], dtype=object)
```

In [66]:
```
sns.countplot(df['Churn'])
```

Out[66]:
```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



In [67]:
```
df['Churn'].value_counts()
```

Out[67]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

In [68]:
```
df['Churn'].value_counts(normalize=True)*100
```

Out[68]:
```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

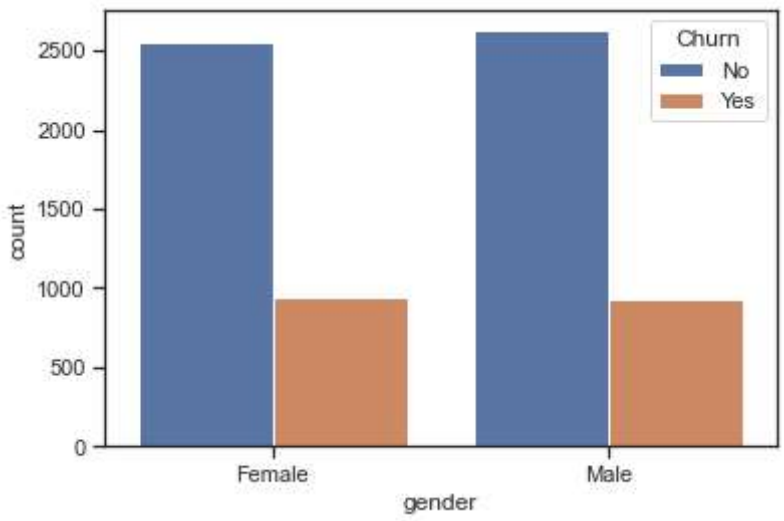
BiVarient Analysis

In [69]:
```
df.columns
```

Out[69]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [70]:
```
sns.countplot('gender',hue='Churn',data=df)
```

Out[70]:
```
<AxesSubplot:xlabel='gender', ylabel='count'>
```



In [71]:
```
df.groupby('gender')['Churn'].count()
```

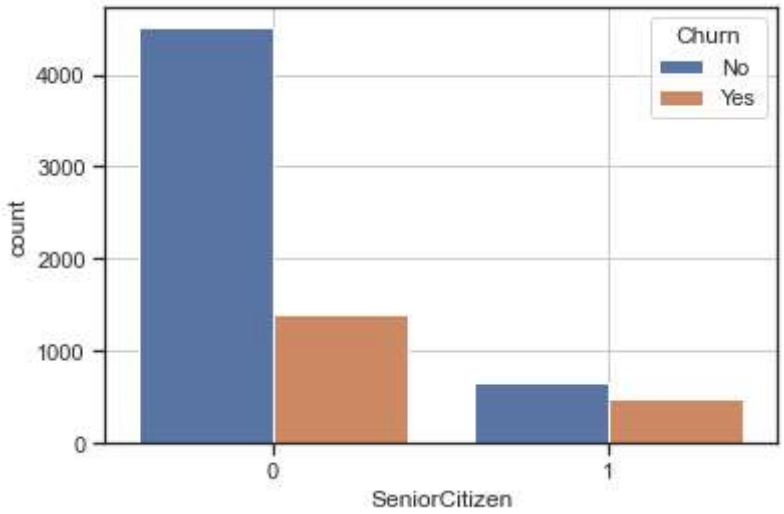
Out[71]:
```
gender
Female    3488
Male      3555
Name: Churn, dtype: int64
```

In [72]:
```
# Approx equal no of Male and Female have churn out.
```

In [73]:
```
df['SeniorCitizen'].value_counts()
```

Out[73]:
```
0    5901
1    1142
Name: SeniorCitizen, dtype: int64
```

In [74]:
```
sns.countplot('SeniorCitizen',hue='Churn',data=df)
plt.grid()
```



In [75]:
```
df.groupby('SeniorCitizen')['Churn'].value_counts()
```

Out[75]:
```
SeniorCitizen  Churn
0              No       4508
               Yes      1393
1              No        666
               Yes       476
Name: Churn, dtype: int64
```

In [76]:
```
1393/5901  # 23%  not seniors have churned out
```

Out[76]:
```
0.23606168446026096
```

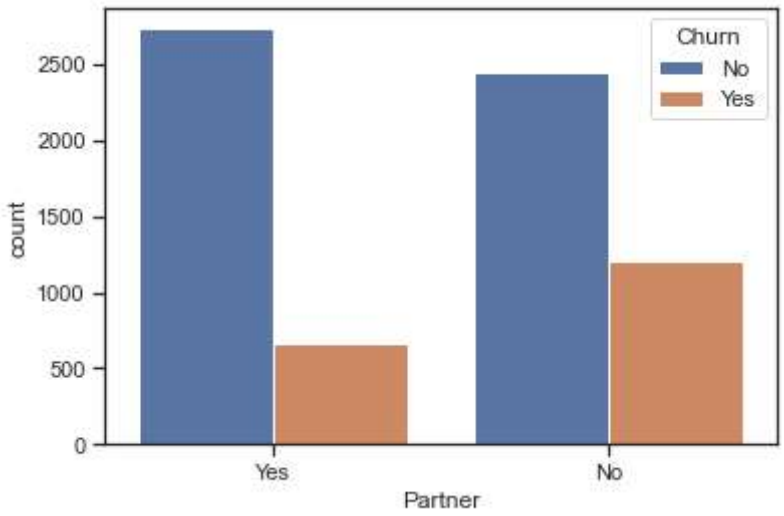
In [77]:
```
476/1142   #   41.6% Senior customers have churned out
```

Out[77]:
```
0.4168126094570928
```

In [78]:
```
# However, Senior customers have more churned out
```

In [79]:
```
sns.countplot('Partner',hue='Churn',data=df)
```

Out[79]:
```
<AxesSubplot:xlabel='Partner', ylabel='count'>
```



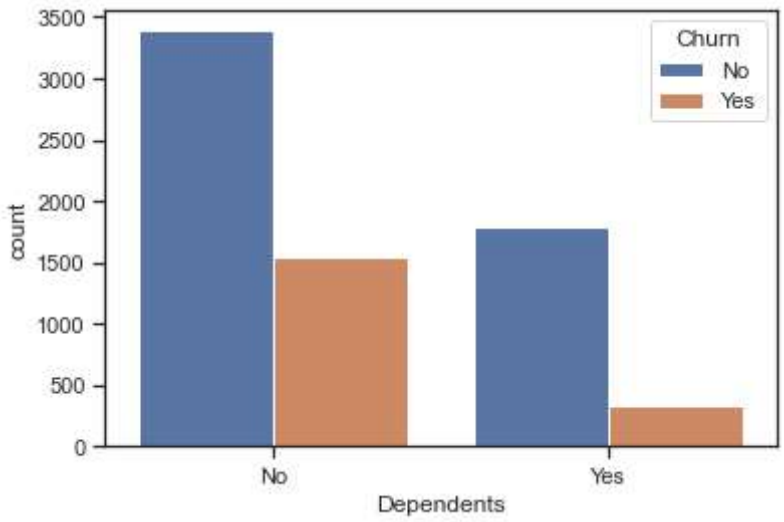
In [80]:
```
df.groupby('Partner')['Churn'].value_counts()
```

Out[80]:
```
Partner  Churn
No       No       2441
         Yes      1200
Yes      No       2733
         Yes       669
Name: Churn, dtype: int64
```

In [81]:
```
# Single customers, who were not have partners have churned out more
```

In [82]:
```
sns.countplot('Dependents',hue='Churn',data=df)
```

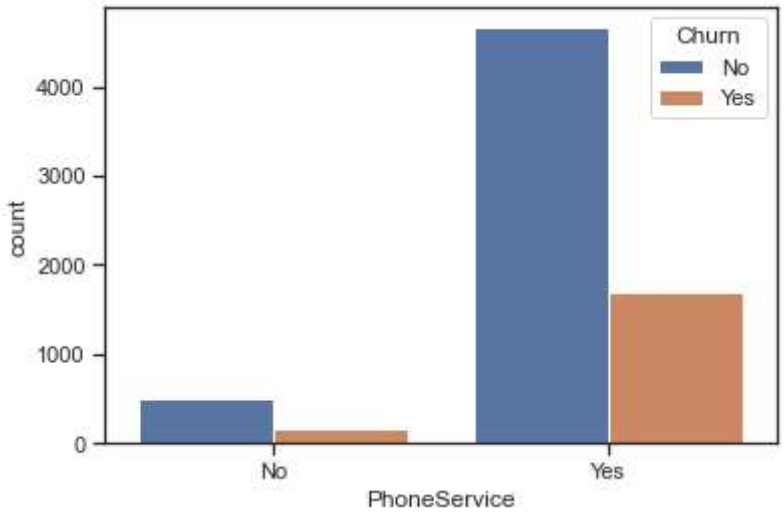
Out[82]: <AxesSubplot:xlabel='Dependents', ylabel='count'>



In [83]:
```
# Customers who have dependents are churned less, whil independent customers have
```

In [84]:
```
sns.countplot('PhoneService',hue='Churn',data=df)
```

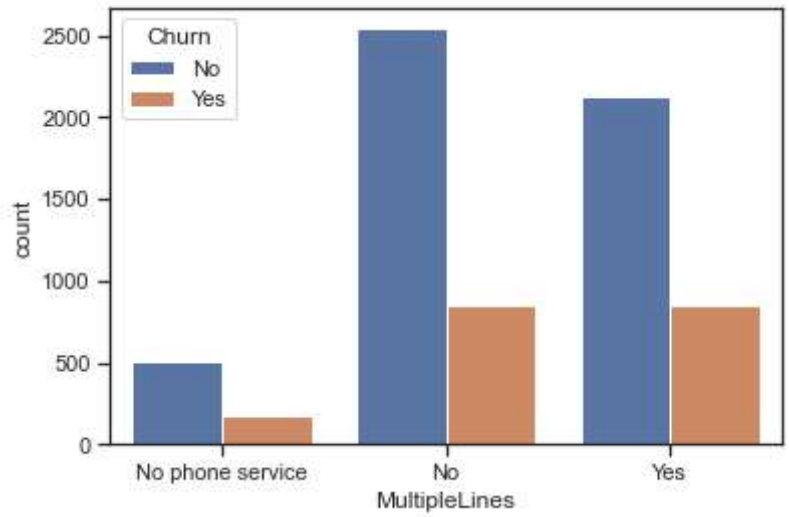
Out[84]: <AxesSubplot:xlabel='PhoneService', ylabel='count'>



In [85]:
```
# nothing insightfull info from here, 90% customers were using Phone service whil
```

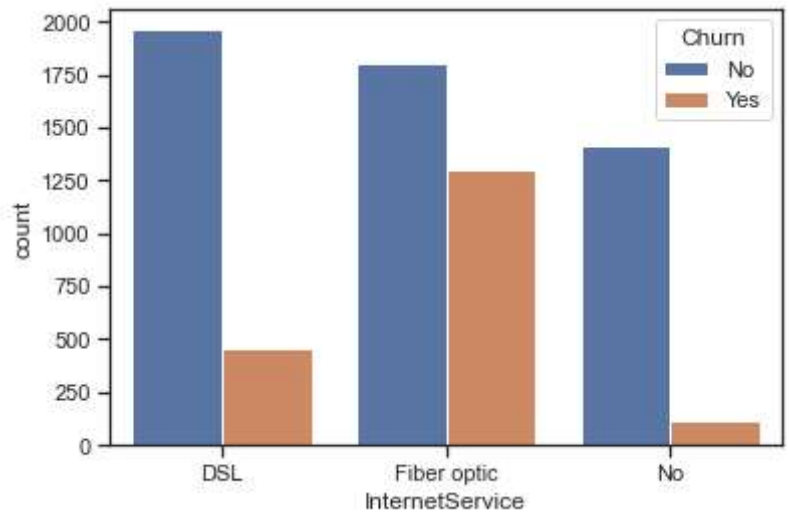
In [86]:
```
sns.countplot('MultipleLines',hue='Churn',data=df)
```

Out[86]:
```
<AxesSubplot:xlabel='MultipleLines', ylabel='count'>
```



In [87]:
```
sns.countplot('InternetService',hue='Churn',data=df)
```

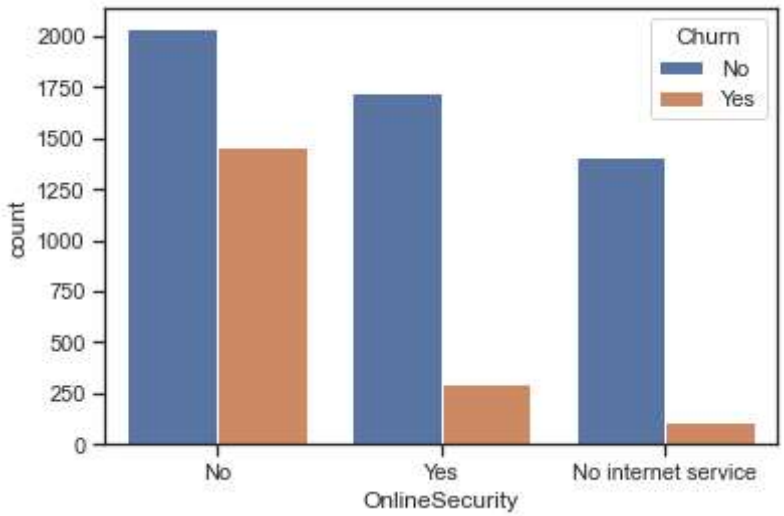
Out[87]:
```
<AxesSubplot:xlabel='InternetService', ylabel='count'>
```



In [88]:
```
# Majority of customers who churned out were using Fibre service, they might be r
```

In [89]:
```
sns.countplot('OnlineSecurity',hue='Churn',data=df)
```

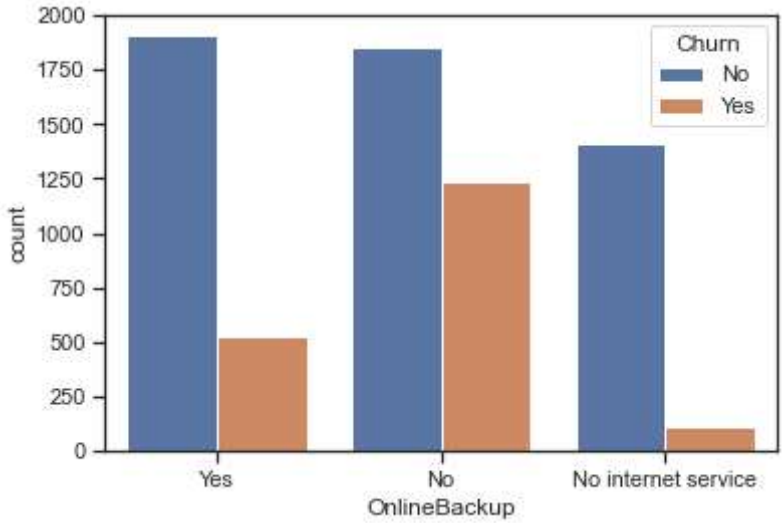
Out[89]: <AxesSubplot:xlabel='OnlineSecurity', ylabel='count'>



In [90]:
```
# Majority churned out customers were using fibre and were not using Online Secur
```

In [91]:
```
sns.countplot('OnlineBackup',hue='Churn',data=df)
```

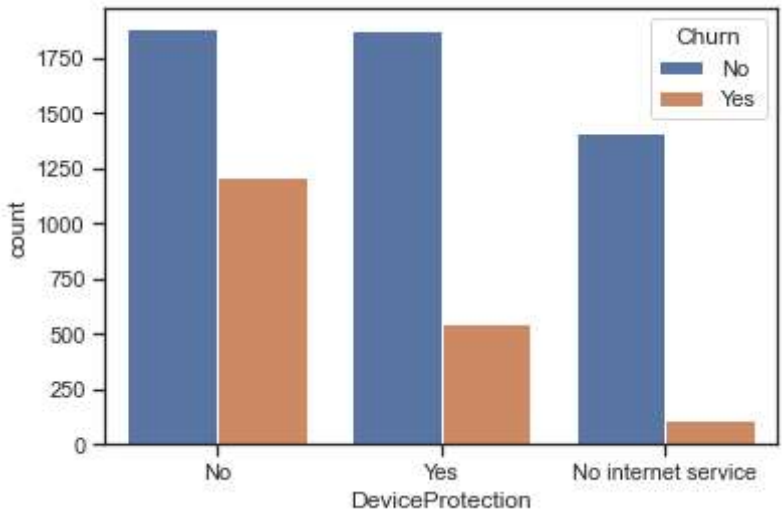
Out[91]: <AxesSubplot:xlabel='OnlineBackup', ylabel='count'>



In [92]:
```
# Majority churned ut customers were not using OnlineBackup service
```

In [93]:
```
sns.countplot('DeviceProtection',hue='Churn',data=df)
```

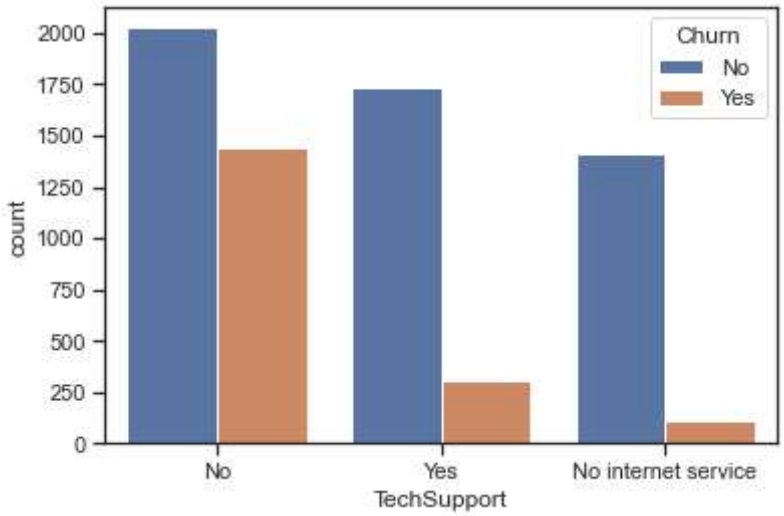
Out[93]: <AxesSubplot:xlabel='DeviceProtection', ylabel='count'>



In [94]:
```
# Majority id churned out customers were not using Device Protection
```

In [95]:
```
sns.countplot('TechSupport',hue='Churn',data=df)
```

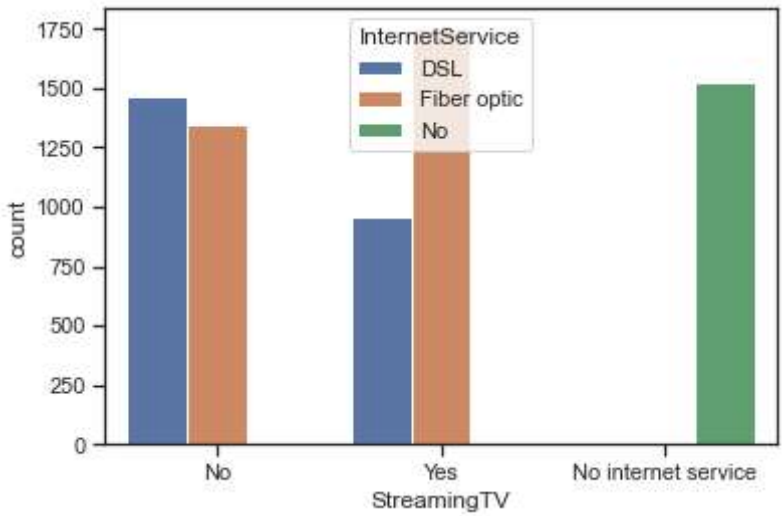
Out[95]: <AxesSubplot:xlabel='TechSupport', ylabel='count'>



In [96]:
```
# Majority of churned out customers were not using TechSupport
```

In [97]:
```
sns.countplot('StreamingTV',hue='InternetService',data=df)
```

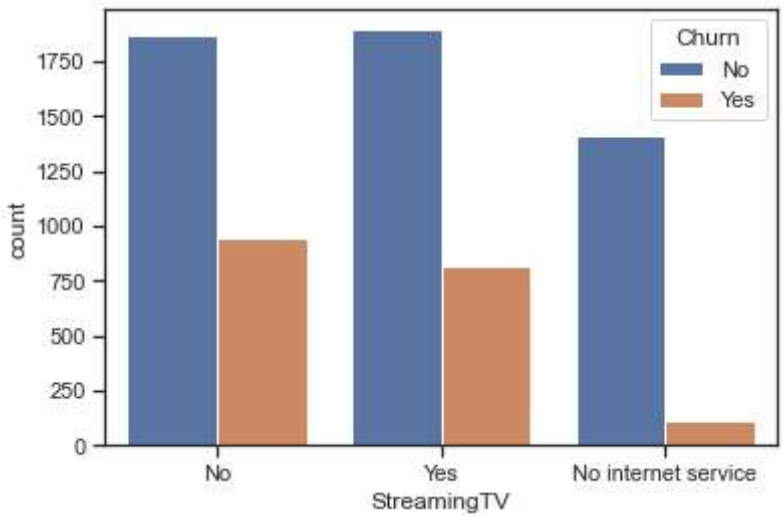
Out[97]:
```
<AxesSubplot:xlabel='StreamingTV', ylabel='count'>
```



In [98]:
```
# More customers were streaming Tv on Fibre
```

In [99]:
```
sns.countplot('StreamingTV',hue='Churn',data=df)
```

Out[99]:
```
<AxesSubplot:xlabel='StreamingTV', ylabel='count'>
```

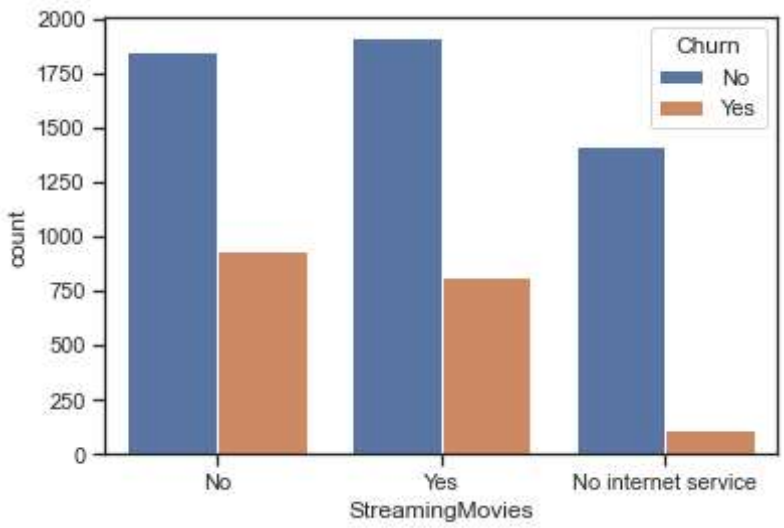


In [100]:
```
df.columns
```

Out[100]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

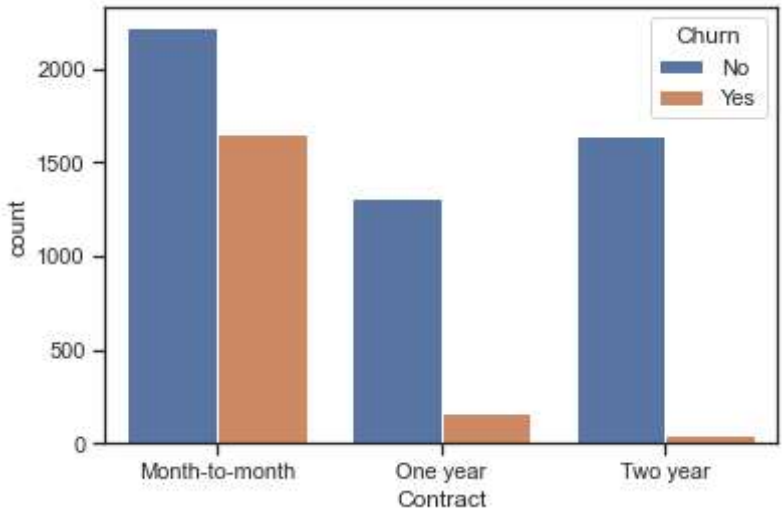
In [101]:
```
sns.countplot('StreamingMovies',hue='Churn',data=df)
```

Out[101]:
```
<AxesSubplot:xlabel='StreamingMovies', ylabel='count'>
```



In [102]:
```
sns.countplot('Contract',hue='Churn',data=df)
```

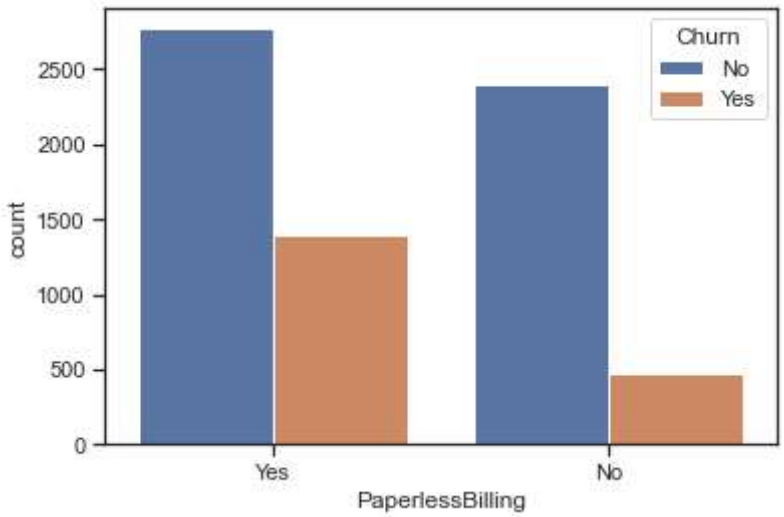
Out[102]:
```
<AxesSubplot:xlabel='Contract', ylabel='count'>
```



In [103]:
```
# Month to Month renewal customers have churned out more
```

In [104]:
```
sns.countplot('PaperlessBilling',hue='Churn',data=df)
```

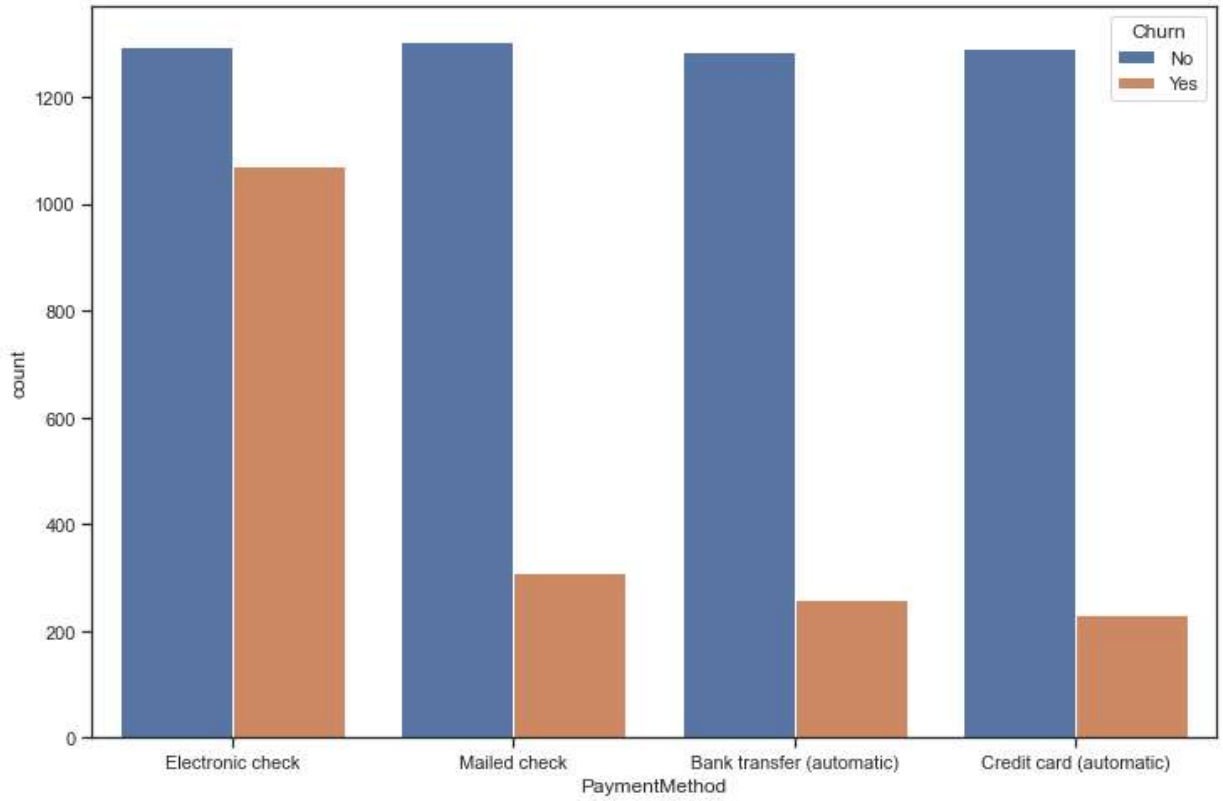
Out[104]: <AxesSubplot:xlabel='PaperlessBilling', ylabel='count'>



In [105]:
```
# more customes with paperless billing churned out more
```

In [106]:
```
plt.figure(figsize=(12,8))
sns.countplot('PaymentMethod',hue='Churn',data=df)
```

Out[106]: <AxesSubplot:xlabel='PaymentMethod', ylabel='count'>



In [107]:
```
# Electronic transfer customers have churned out more
```

In [108]:
```
df[['tenure','MonthlyCharges','TotalCharges']]
```

Out[108]:

	tenure	MonthlyCharges	TotalCharges
0	1	29.85	29.85
1	34	56.95	1889.50
2	2	53.85	108.15
3	45	42.30	1840.75
4	2	70.70	151.65
...	...	...	...
7038	24	84.80	1990.50
7039	72	103.20	7362.90
7040	11	29.60	346.45
7041	4	74.40	306.60
7042	66	105.65	6844.50

7043 rows × 3 columns

In [109]:
```
56.95*34
```

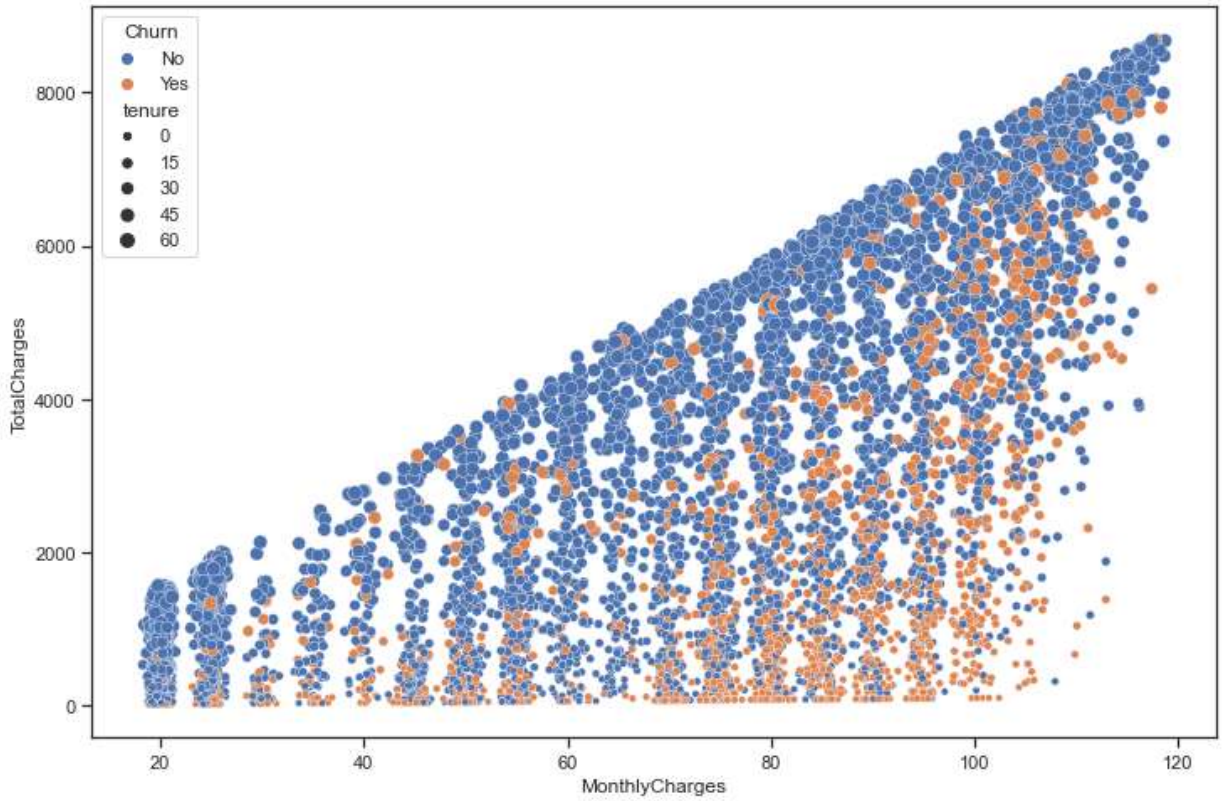
Out[109]:
```
1936.3000000000002
```

In [110]:
```
53.85*2
```

Out[110]:
```
107.7
```

In [111]:
```
plt.figure(figsize=(12,8))
sns.scatterplot('MonthlyCharges','TotalCharges',hue='Churn',size='tenure',data=df
```

Out[111]: <AxesSubplot:xlabel='MonthlyCharges', ylabel='TotalCharges'>



In [112]:
```
df['MonthlyCharges'].max()
```

Out[112]: 118.75

In [113]:
```
df['MonthlyCharges'].sort_values()
```

Out[113]:
```
3719     18.25
1529     18.40
6652     18.55
6906     18.70
1156     18.70
         ...
5127    118.35
3894    118.60
4804    118.60
2115    118.65
4586    118.75
Name: MonthlyCharges, Length: 7043, dtype: float64
```

In [114]:
```
df.sort_values('MonthlyCharges',ascending=False).head(150)
```

Out[114]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
4586	7569-NMZYQ	Female	0	Yes	Yes	72	Yes	Yes
2115	8984-HPEMB	Female	0	No	No	71	Yes	Yes
3894	5989-AXPUC	Female	0	Yes	No	68	Yes	Yes
4804	5734-EJKXG	Female	0	No	No	61	Yes	Yes
5127	8199-ZLLSA	Male	0	No	No	67	Yes	Yes
...	...	...	...	...	...	...	...	...
1357	7130-YXBRO	Male	0	Yes	No	48	Yes	Yes
1568	3292-PBZEJ	Male	1	No	No	11	Yes	Yes
197	6168-YBYNP	Male	0	No	No	59	Yes	Yes
2204	2659-VXMWZ	Male	0	Yes	Yes	67	Yes	Yes
3901	2773-OVBPK	Male	0	Yes	No	67	Yes	Yes

150 rows × 21 columns

In [115]:
```
df.sort_values('TotalCharges').head(20)
```

Out[115]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
1654	2967-MXRAV	Male	0	Yes	Yes	1	Yes	No
6489	9318-NKNFC	Male	0	No	No	1	Yes	No
1151	8992-CEUEN	Female	0	No	No	1	Yes	No
4939	9975-SKRNR	Male	0	No	No	1	Yes	No
583	1423-BMPBQ	Female	0	Yes	Yes	1	Yes	No
1733	1015-OWJKI	Male	0	No	No	1	Yes	No
3110	6569-KTMDU	Female	0	No	No	1	Yes	No
4348	6121-VZNQB	Female	0	No	No	1	Yes	No
5336	9441-QHEVC	Male	0	No	No	1	Yes	No
367	7302-ZHMHP	Female	0	No	No	1	Yes	No
1816	1663-MHLHE	Male	0	No	No	1	Yes	No
2989	3308-MHOOC	Male	0	No	Yes	1	Yes	No
4392	3373-YZZYM	Male	0	Yes	Yes	1	Yes	No
3313	4232-JGKIY	Male	0	No	No	1	Yes	No
310	1098-TDVUQ	Female	0	No	No	1	Yes	No
6251	9374-YOLBJ	Female	0	Yes	Yes	1	Yes	No
5646	5510-BOIUJ	Male	0	No	Yes	1	Yes	No
5585	4667-OHGKG	Male	0	No	No	1	Yes	No
4221	7926-IJOOU	Male	0	No	No	1	Yes	No
2096	0761-AETCS	Female	0	No	No	1	Yes	No

In [116]:
```
# More customers have churned out with Monthly charges were more
```

In [117]:
```
df.sort_values('Churn')
```

Out[117]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
4334	3969-GYXEL	Female	0	No	No	11	Yes	No
4333	5325-UWTWJ	Male	0	Yes	No	31	Yes	Yes
4332	4194-FJARJ	Female	0	Yes	Yes	54	Yes	Yes
4331	5743-KHMNA	Male	0	No	No	71	Yes	Yes
...	...	...	...	...	...	...	...	...
5441	3512-IZIKN	Female	0	Yes	No	70	Yes	Yes
1161	5868-CZJDR	Male	0	No	Yes	1	No	No phone service
1159	4086-YQSNZ	Female	1	Yes	No	3	Yes	No
1163	0135-NMXAP	Female	0	No	No	12	Yes	Yes
2974	1481-ZUWZA	Male	0	No	No	28	Yes	Yes

7043 rows × 21 columns

In [118]:
```
# We have to fill Null values in TotalCharges
```

Missing value in TotalCharge

In [119]:
```
df.isnull().sum()
```

Out[119]:
```
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
```

In [120]:
```
df[['tenure','MonthlyCharges','TotalCharges']]
```

Out[120]:

	tenure	MonthlyCharges	TotalCharges
0	1	29.85	29.85
1	34	56.95	1889.50
2	2	53.85	108.15
3	45	42.30	1840.75
4	2	70.70	151.65
...	...	...	...
7038	24	84.80	1990.50
7039	72	103.20	7362.90
7040	11	29.60	346.45
7041	4	74.40	306.60
7042	66	105.65	6844.50

7043 rows × 3 columns

In [121]:
```
105.65*66
```

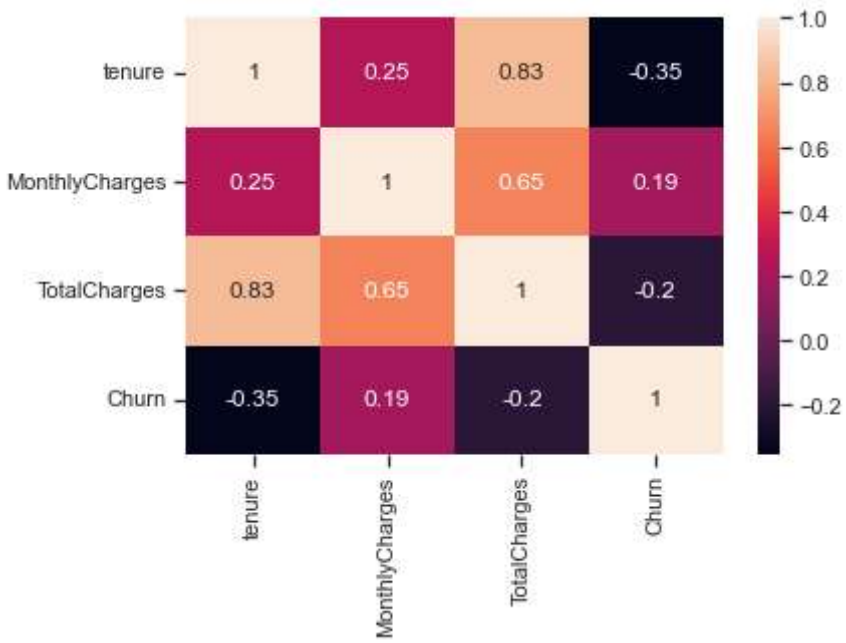
Out[121]:
```
6972.900000000001
```

In [122]:
```
# Correlation check:
data=df[['tenure','MonthlyCharges','TotalCharges','Churn']]

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['Churn']=le.fit_transform(data['Churn'])

sns.heatmap(data.corr(),annot=True)
```

Out[122]: <AxesSubplot:>



In [123]:
```
data['diff']= data['TotalCharges']  - (data['tenure']* data['MonthlyCharges'])
```

In [124]:
```
data
```

Out[124]:

	tenure	MonthlyCharges	TotalCharges	Churn	diff
0	1	29.85	29.85	0	0.00
1	34	56.95	1889.50	0	-46.80
2	2	53.85	108.15	1	0.45
3	45	42.30	1840.75	0	-62.75
4	2	70.70	151.65	1	10.25
...	...	...	...	...	...
7038	24	84.80	1990.50	0	-44.70
7039	72	103.20	7362.90	0	-67.50
7040	11	29.60	346.45	0	20.85
7041	4	74.40	306.60	1	9.00
7042	66	105.65	6844.50	0	-128.40

7043 rows × 5 columns

In [125]:
```
data[data['TotalCharges'].isnull()]
```

Out[125]:

	tenure	MonthlyCharges	TotalCharges	Churn	diff
488	0	52.55	NaN	0	NaN
753	0	20.25	NaN	0	NaN
936	0	80.85	NaN	0	NaN
1082	0	25.75	NaN	0	NaN
1340	0	56.05	NaN	0	NaN
3331	0	19.85	NaN	0	NaN
3826	0	25.35	NaN	0	NaN
4380	0	20.00	NaN	0	NaN
5218	0	19.70	NaN	0	NaN
6670	0	73.35	NaN	0	NaN
6754	0	61.90	NaN	0	NaN

In [126]:
```
data[data['TotalCharges'].isnull()]
```

Out[126]:

	tenure	MonthlyCharges	TotalCharges	Churn	diff
488	0	52.55	NaN	0	NaN
753	0	20.25	NaN	0	NaN
936	0	80.85	NaN	0	NaN
1082	0	25.75	NaN	0	NaN
1340	0	56.05	NaN	0	NaN
3331	0	19.85	NaN	0	NaN
3826	0	25.35	NaN	0	NaN
4380	0	20.00	NaN	0	NaN
5218	0	19.70	NaN	0	NaN
6670	0	73.35	NaN	0	NaN
6754	0	61.90	NaN	0	NaN

In [127]:
```
df['TotalCharges'].min()
```

Out[127]: 18.8

In [128]:
```
missing=data[data['tenure']==0]
missing
```

Out[128]:

	tenure	MonthlyCharges	TotalCharges	Churn	diff
488	0	52.55	NaN	0	NaN
753	0	20.25	NaN	0	NaN
936	0	80.85	NaN	0	NaN
1082	0	25.75	NaN	0	NaN
1340	0	56.05	NaN	0	NaN
3331	0	19.85	NaN	0	NaN
3826	0	25.35	NaN	0	NaN
4380	0	20.00	NaN	0	NaN
5218	0	19.70	NaN	0	NaN
6670	0	73.35	NaN	0	NaN
6754	0	61.90	NaN	0	NaN

In [129]:
```
df['TotalCharges']=np.where(df['tenure']==0,df['TotalCharges'].fillna(df['Monthly
df['tenure'].value_counts()
```

Out[129]:
```
1     613
72    362
2     238
3     200
4     176
     ...
28     57
39     56
44     51
36     50
0      11
Name: tenure, Length: 73, dtype: int64
```

In [130]:
```
df['tenure'].replace(0,1,inplace=True)
df['tenure'].value_counts()
```

Out[130]:
```
1     624
72    362
2     238
3     200
4     176
     ...
38     59
28     57
39     56
44     51
36     50
Name: tenure, Length: 72, dtype: int64
```

In [131]:
```
df.head()
```

Out[131]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Ir
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	

In [132]:
```
# CustomerID is releavant for our model prediction
df.drop('customerID',axis=1,inplace=True)
```

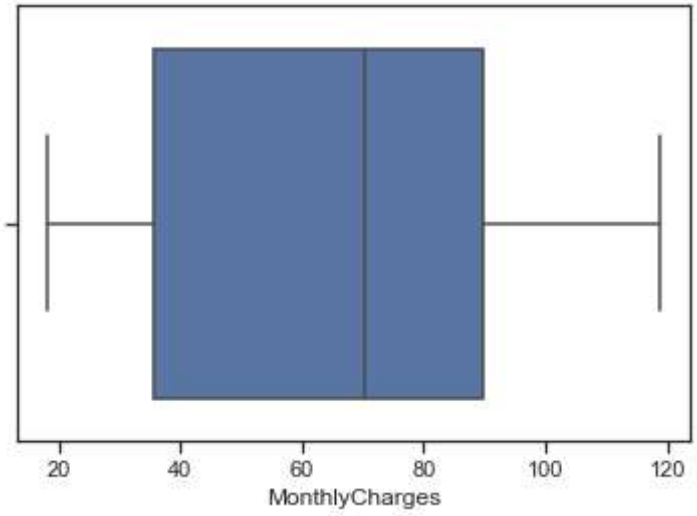
In [133]:
```
df.head()
```

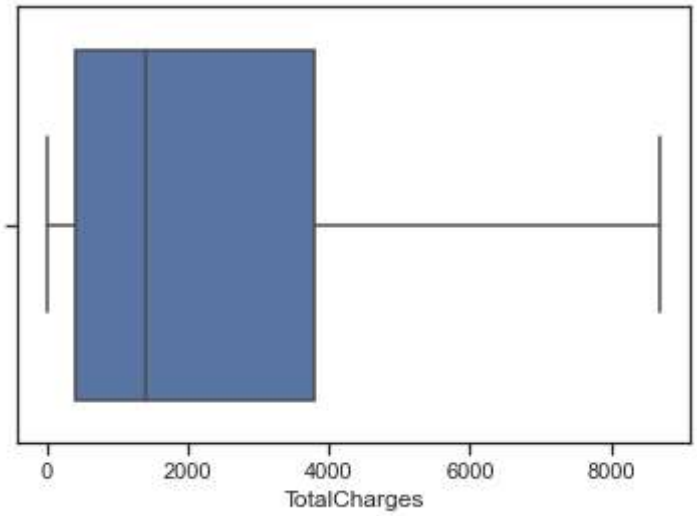
Out[133]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	1	No	No phone service	DSL
1	Male	0	No	No	34	Yes	No	DSL
2	Male	0	No	No	2	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
4	Female	0	No	No	2	Yes	No	Fiber optic

Outliers

In [134]:
```
for i in df.columns:
    if df[i].dtypes==np.number:
        sns.boxplot(df[i])
        plt.show()
```





Encoding

In [135]:
```
df.head()
```

Out[135]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	1	No	No phone service	DSL
1	Male	0	No	No	34	Yes	No	DSL
2	Male	0	No	No	2	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
4	Female	0	No	No	2	Yes	No	Fiber optic

In [136]:
```
X=df.drop('Churn',axis=1)
Y=df['Churn']
```

In [137]:
```
X.shape , Y.shape
```

Out[137]:
```
((7043, 19), (7043,))
```

In [138]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
Y=le.fit_transform(Y)
Y
```

Out[138]:
```
array([0, 0, 1, ..., 0, 1, 0])
```

In [139]:
```
X.head()
```

Out[139]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	1	No	No phone service	DSL
1	Male	0	No	No	34	Yes	No	DSL
2	Male	0	No	No	2	Yes	No	DSL
3	Male	0	No	No	45	No	No phone service	DSL
4	Female	0	No	No	2	Yes	No	Fiber optic

Transformation

In [140]:
```
X.skew()
```

Out[140]:
```
SeniorCitizen     1.833633
tenure            0.239730
MonthlyCharges   -0.220524
TotalCharges      0.963316
dtype: float64
```

In [141]:
```
X['tenure']=X['tenure'].astype(float)
```

In [142]:
```
X['tenure'].dtypes
```

Out[142]:
```
dtype('float64')
```

In [143]:
```
num=X[['tenure','MonthlyCharges','TotalCharges']]
num
```

Out[143]:

	tenure	MonthlyCharges	TotalCharges
0	1.0	29.85	29.85
1	34.0	56.95	1889.50
2	2.0	53.85	108.15
3	45.0	42.30	1840.75
4	2.0	70.70	151.65
...	...	...	...
7038	24.0	84.80	1990.50
7039	72.0	103.20	7362.90
7040	11.0	29.60	346.45
7041	4.0	74.40	306.60
7042	66.0	105.65	6844.50

7043 rows × 3 columns

In [144]:
```
from sklearn.preprocessing import power_transform
trans=power_transform(num)
trans
```

Out[144]:
```
array([[-1.64619766, -1.1585412 , -1.80520648],
       [ 0.29737751, -0.23949171,  0.25686123],
       [-1.49696035, -0.34266505, -1.38133786],
       ...,
       [-0.72548575, -1.16724064, -0.85408054],
       [-1.26620843,  0.33431207, -0.91709707],
       [ 1.20193472,  1.33886338,  1.48322527]])
```

In [145]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
num=sc.fit_transform(trans)
number=pd.DataFrame(num)
number
```

Out[145]:

	0	1	2
0	-1.646198	-1.158541	-1.805206
1	0.297378	-0.239492	0.256861
2	-1.496960	-0.342665	-1.381338
3	0.646504	-0.731079	0.235865
4	-1.496960	0.213545	-1.244141
...	...	...	...
7038	-0.078009	0.671510	0.299107
7039	1.342177	1.260981	1.565492
7040	-0.725486	-1.167241	-0.854081
7041	-1.266208	0.334312	-0.917097
7042	1.201935	1.338863	1.483225

7043 rows × 3 columns

In [146]:
```
cat=df[['gender','SeniorCitizen','Partner','Dependents','PhoneService','MultipleL
cat
```

Out[146]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	On
0	Female	0	Yes	No	No	No phone service	DSL	
1	Male	0	No	No	Yes	No	DSL	
2	Male	0	No	No	Yes	No	DSL	
3	Male	0	No	No	No	No phone service	DSL	
4	Female	0	No	No	Yes	No	Fiber optic	
...	...	...	...	...	...	...	...	...
7038	Male	0	Yes	Yes	Yes	Yes	DSL	
7039	Female	0	Yes	Yes	Yes	Yes	Fiber optic	
7040	Female	0	Yes	Yes	No	No phone service	DSL	
7041	Male	1	Yes	No	Yes	Yes	Fiber optic	
7042	Male	0	No	No	Yes	No	Fiber optic	

7043 rows × 16 columns

In [147]:
```
category=pd.get_dummies(cat,drop_first=True)
category
```

Out[147]:

	SeniorCitizen	gender_Male	Partner_Yes	Dependents_Yes	PhoneService_Yes	MultipleLines_N phone servic
0	0	0	1	0	0	
1	0	1	0	0	1	
2	0	1	0	0	1	
3	0	1	0	0	0	
4	0	0	0	0	1	
...	...	...	...	...	...	
7038	0	1	1	1	1	
7039	0	0	1	1	1	
7040	0	0	1	1	0	
7041	1	1	1	0	1	
7042	0	1	0	0	1	

7043 rows × 27 columns

In [148]:
```
number.shape , category.shape
```

Out[148]: ((7043, 3), (7043, 27))

In [149]:
```
type(number)  , type(category)
```

Out[149]: (pandas.core.frame.DataFrame, pandas.core.frame.DataFrame)

In [150]:
```
X=pd.concat([number,category],axis=1)
```

Balance the dataset

SMOTE

In [151]:
```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
bal_x,bal_y=sm.fit_resample(X,Y)
```

In [152]:
```
bal_x.shape , bal_y.shape
```

Out[152]:
```
((10348, 30), (10348,))
```

Machine Learning

In [153]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [154]:
```
# FInd best Random_state

maxaccu=0
maxRS=0

for i in range(0,200):
    x_train,x_test,y_train,y_test= train_test_split(X,Y,random_state=i,test_size=
    LR= LogisticRegression()
    LR.fit(x_train,y_train)
    pred= LR.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>maxaccu:
        maxaccu=acc
        maxRS=i
print("Best accuracy is ",maxaccu,"on Random State =",maxRS)
```

```
Best accuracy is  0.8282469836763662 on Random State = 158
```

In [155]:
```
x_train,x_test,y_train,y_test= train_test_split(bal_x,bal_y,random_state=158,test
```

In [156]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [157]:
```
LR_model= LogisticRegression()
RD_model= RidgeClassifier()
DT_model= DecisionTreeClassifier()
SV_model= SVC()
KNR_model= KNeighborsClassifier()
RFR_model= RandomForestClassifier()
XGB_model= XGBClassifier()
SGH_model= SGDClassifier()
Bag_model=BaggingClassifier()
ADA_model=AdaBoostClassifier()
GB_model= GradientBoostingClassifier()

model=[LR_model,RD_model,DT_model,SV_model,KNR_model,RFR_model,XGB_model,SGH_mode
```

In [158]:
```
for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    pred= m.predict(x_test)
    print('Accuracy_Score of ',m, 'is', accuracy_score(y_test,pred)*100)
    print('Confusion Matrix of ',m,' is \n', confusion_matrix(y_test,pred) )
    print(classification_report(y_test,pred))
    print('*'*50)
```

```
Accuracy_Score of  LogisticRegression() is 80.57971014492755
Confusion Matrix of  LogisticRegression()  is
 [[1239  341]
 [ 262 1263]]
              precision    recall  f1-score   support

           0       0.83      0.78      0.80      1580
           1       0.79      0.83      0.81      1525

    accuracy                           0.81      3105
   macro avg       0.81      0.81      0.81      3105
weighted avg       0.81      0.81      0.81      3105

**************************************************
Accuracy_Score of  RidgeClassifier() is 80.03220611916264
Confusion Matrix of  RidgeClassifier()  is
 [[1222  358]
 [ 262 1263]]
              precision    recall  f1-score   support
```

CROSS Validation

In [159]:
```
from sklearn.model_selection import cross_val_score
```

In [160]:
```
for i in model:
    print('Accuracy_Score of ',i, 'is', accuracy_score(y_test,i.predict(x_test))*
    print("cross Validation accuracy score of ",i ," is ",cross_val_score(i,X,Y,c
    print('*'*50)
```

```
Accuracy_Score of  LogisticRegression() is 80.57971014492755
cross Validation accuracy score of  LogisticRegression()  is  80.76118822182077
**************************************************
Accuracy_Score of  RidgeClassifier() is 80.03220611916264
cross Validation accuracy score of  RidgeClassifier()  is  80.34935721659463
**************************************************
Accuracy_Score of  DecisionTreeClassifier() is 76.16747181964574
cross Validation accuracy score of  DecisionTreeClassifier()  is  72.9663909768
3722
**************************************************
Accuracy_Score of  SVC() is 80.5475040257649
cross Validation accuracy score of  SVC()  is  80.34945802955029
**************************************************
Accuracy_Score of  KNeighborsClassifier() is 79.54911433172302
cross Validation accuracy score of  KNeighborsClassifier()  is  77.637660090973
61
**************************************************
Accuracy_Score of  RandomForestClassifier() is 84.66988727858293
cross Validation accuracy score of  RandomForestClassifier()  is  78.9011690270
3402
**************************************************
Accuracy_Score of  XGBClassifier(base_score=0.5, booster='gbtree', callbacks=No
ne,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...) is 83.41384863123994
cross Validation accuracy score of  XGBClassifier(base_score=0.5, booster='gbtr
ee', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)  is  78.46124145106135
**************************************************
Accuracy_Score of  SGDClassifier() is 78.71175523349436
cross Validation accuracy score of  SGDClassifier()  is  80.13653098587005
**************************************************
Accuracy_Score of  BaggingClassifier() is 81.06280193236715
cross Validation accuracy score of  BaggingClassifier()  is  77.9782566617201
**************************************************
Accuracy_Score of  AdaBoostClassifier() is 79.2914653784219
cross Validation accuracy score of  AdaBoostClassifier()  is  80.5196403800245
```

```
************************************************
Accuracy_Score of  GradientBoostingClassifier() is 79.77455716586151
cross Validation accuracy score of  GradientBoostingClassifier()  is  80.420359
78127622
************************************************
```

Hypertuning GradientBoostingClasifier()

In [161]:
```
from sklearn.model_selection import GridSearchCV
```

In [162]:
```
params= {"learning_rate"    : [0.01,.05,.1,.2,.3,.5 ] ,
         'n_estimators':[5,50,100,200,300,400],
         "max_depth"        : [ 3, 4, 5, 6, 8]
          }
```

In [163]:
```
GCV= GridSearchCV(GB_model,params,cv=5,scoring='accuracy', n_jobs=-1)
GCV.fit(x_train,y_train)
```

Out[163]:

GridSearchCV

estimator: GradientBoostingClassifier

GradientBoostingClassifier

In [164]:
```
GCV.best_estimator_
```

Out[164]:

GradientBoostingClassifier

```
GradientBoostingClassifier(learning_rate=0.3, max_depth=8, n_estimators=400)
```

In [165]:
```
GCV.best_params_
```

Out[165]:
```
{'learning_rate': 0.3, 'max_depth': 8, 'n_estimators': 400}
```

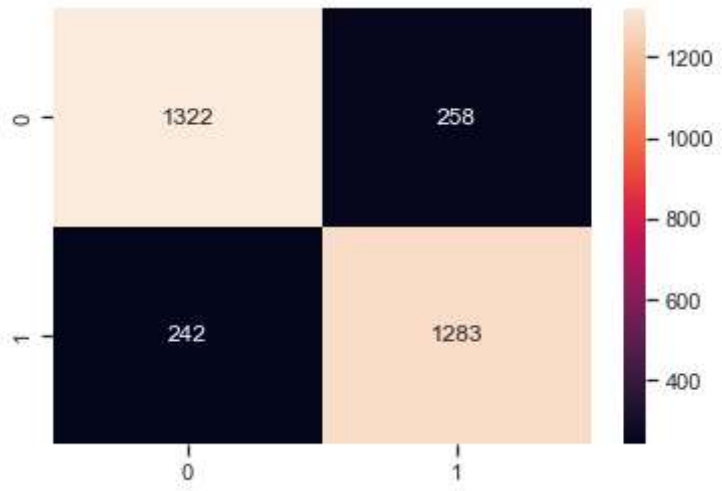
In [166]:
```
pred=GCV.best_estimator_.predict(x_test)
```

In [167]:
```
print('Accuracy score:', round(accuracy_score(y_test,pred) * 100, 2))
print('F1 score:', round(f1_score(y_test,pred) * 100, 2))
```

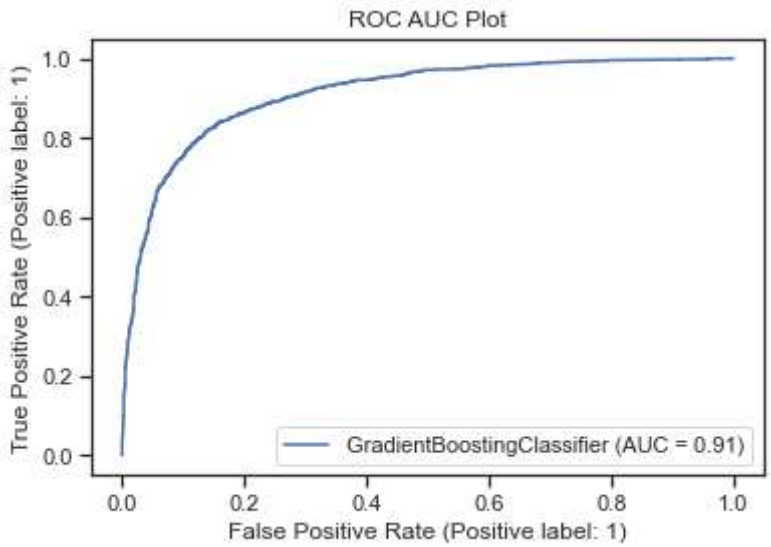
```
Accuracy score: 83.9
F1 score: 83.69
```

In [168]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,pred)
sns.heatmap(confusion_matrix(y_test,pred),annot=True, fmt='d')
```

Out[168]: <AxesSubplot:>



In [170]:
```
from sklearn.metrics import roc_auc_score,roc_curve,plot_roc_curve
plot_roc_curve(GCV.best_estimator_,x_test,y_test)
plt.title('ROC AUC Plot')
plt.show()
```



Saving the Model ¶

In [171]:
```
import joblib
joblib.dump(GCV.best_estimator_,"Customer_Churn_Analysis.pkl")
```

Out[171]:
```
['Customer_Churn_Analysis.pkl']
```

In [ ]:
```

```