Data Analytics Coursework_1 (ECS784P)

Surya Chandra Selvaraj - 210383171

Telecom Chrun rate prediction using supervised learning machine learning methods

The customer churn rate means the number of customers leaving the company or withdrawing from the service provided by the company. We have used supervised learning machine learning methods to predict this rate of users/customers leaving the services provided by the telecommunication company.

In [1]:
```
import pandas as pd                    # library for data manipulation and analysis
import numpy as np                     # library for high-level mathematical functions
import seaborn as sns                  # visualisation library for plotting with advanced features
from matplotlib import pyplot as plt   # visualisation library used for plotting
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('Customer-Churn.csv',index_col=False) # loading the CSV data set into a dataFrame name df
```

In [3]:
```
df.columns # returns all the column names in the dataframe df
```

Out[3]:
```
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 [4]:
```
df.head() # returns the first 5 lines of the dataframe
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtectio
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	N
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Ye
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	N
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Ye
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	N

5 rows × 21 columns

In [5]:
```
df.tail() # returns the last 5 lines of the dataframe
```

Out[5]:

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

5 rows × 21 columns

In [6]:
```
df.dtypes # returns the datatype of each column in the dataframe
```

Out[6]:
```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                object
dtype: object
```

In [7]:
```
df.shape # returns the shape of data rows and column respectively
```

Out[7]:
```
(7043, 21)
```

In [8]:
```
df['Churn'].value_counts() # return a series containing counts of unique rows in the dataframe df
```

Out[8]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

Pre-processing the data set

In [9]:
```
df = df.drop(['customerID'],axis=1) # drop the customerID column because it is useless for prediction
```

In [10]:
```
df.describe() #function computes a summary of statistics pertaining to the DataFrame columns
```

Out[10]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.00000
mean	0.162147	32.371149	64.761692	2281.77900
std	0.368612	24.559481	30.090047	2265.42947
min	0.000000	0.000000	18.250000	18.80000
25%	0.000000	9.000000	35.500000	402.22500
50%	0.000000	29.000000	70.350000	1397.30000
75%	0.000000	55.000000	89.850000	3786.60000
max	1.000000	72.000000	118.750000	8684.80000

In [11]:
```
df.info() #prints information about the dataframe
```

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

In [12]:
```
df.dtypes #returns all the columns with corresponding datatypes
```

Out[12]:
```
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                object
dtype: object
```

In [13]:
```
df.isnull().sum() #returns the sum of null values of every column
```

Out[13]:
```
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 [14]:
```
#Removing any  missing values in the dataframe even though if the spaces are as string
df = df.dropna(how='any')
```

Exploratory Data Analysis

In [15]:
```
df.head() #returns the top 5 rows in the dataframe
```

Out[15]:

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

In [16]:
```
df.tail() #returns the last 5 rows in the dataframe
```

Out[16]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtect
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	No	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	Yes	
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	No	
7041	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	No	
7042	Male	0	No	No	66	Yes	No	Fiber optic	Yes	No	

In [17]:
```
df['gender'].value_counts() # returns the count of male and female in the column gender
```

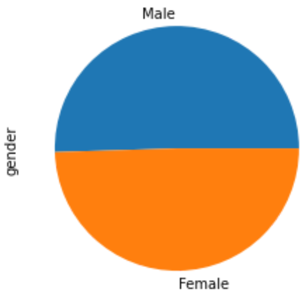
Out[17]:
```
Male      3555
Female    3488
Name: gender, dtype: int64
```

In [18]:
```
df['gender'].value_counts(normalize=True) # returns the normalized value count
```

Out[18]:
```
Male      0.504756
Female    0.495244
Name: gender, dtype: float64
```

In [19]:
```
df['gender'].value_counts().plot.pie() #Plot as pie-chart
```

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



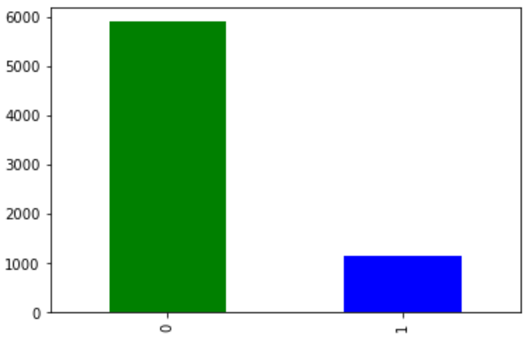
We can infer from the pie chart that the number of male and female in the dataset is approximately 50 percent each.

In [20]:
```
df['SeniorCitizen'].value_counts() #returns the count of senior citizens 0 indicating false and 1 is true
```

Out[20]:
```
0    5901
1    1142
Name: SeniorCitizen, dtype: int64
```

In [21]:
```
df['SeniorCitizen'].value_counts().plot(kind='bar', color=['g','b']) # this plots the distribution of Senior Citizen
```

Out[21]:
```
<AxesSubplot:>
```



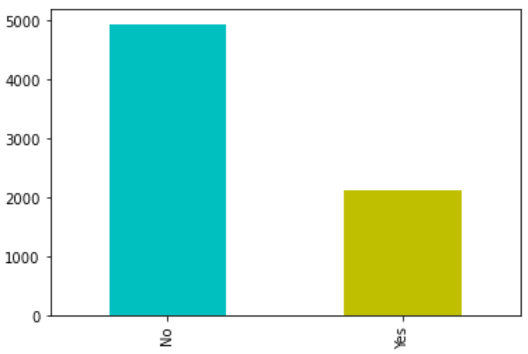
The number of senior citizen is 1142 compared to adults whose count is 5901.

In [22]:
```
df['Dependents'].value_counts() #returns the count of dependents
```

Out[22]:
```
No     4933
Yes    2110
Name: Dependents, dtype: int64
```

In [23]:
```
df['Dependents'].value_counts().plot(kind='bar',color=['c','y']) # this plots the distribution of dependents
```

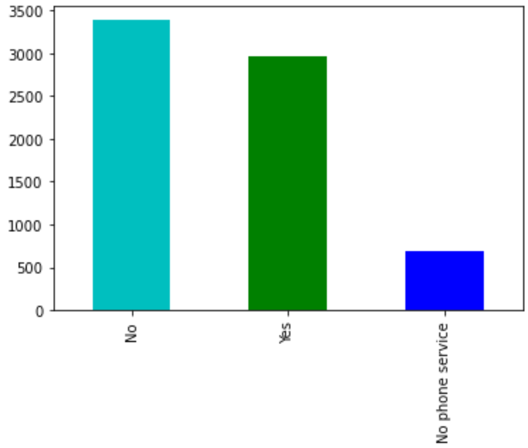
Out[23]:
```
<AxesSubplot:>
```



The number of customers who have dependents is 2110 and others who do not have dependents is 4933.

In [24]:
```
df['MultipleLines'].value_counts().plot(kind='bar', color=['c','g','b']) # this plots the distribution of network connectio
df['MultipleLines'].value_counts()
```

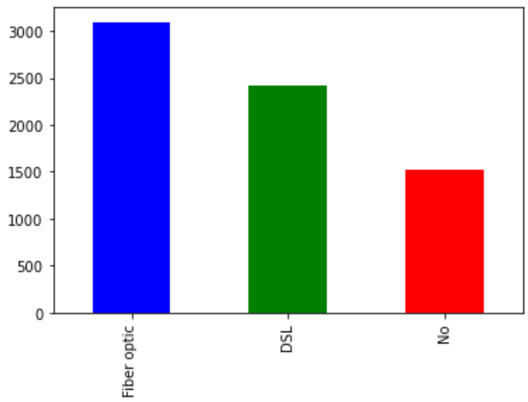
Out[24]:
```
No                  3390
Yes                 2971
No phone service     682
Name: MultipleLines, dtype: int64
```



The number of customers with multiple line connections is 2971 and those who do not is 3390, and no phone service is 682.

In [25]:
```
df['InternetService'].value_counts().plot(kind='bar', color=['b','g','r']) # this plots the distribution of fiber optic, DS
df['InternetService'].value_counts()
```

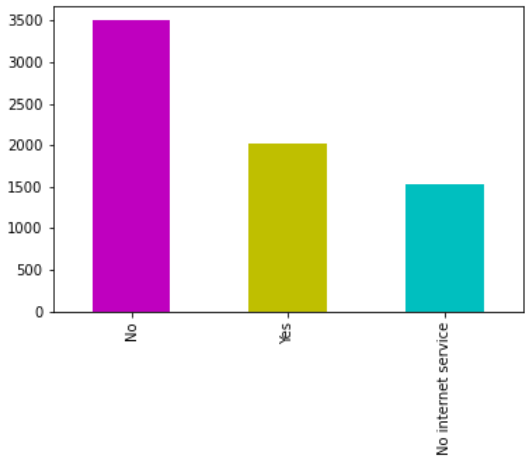
Out[25]:
```
Fiber optic    3096
DSL            2421
No             1526
Name: InternetService, dtype: int64
```



The number of customers with fiber optic connections is 3096, DSL is 2421 and no internet connection is 1526.

In [26]:
```
df['OnlineSecurity'].value_counts().plot(kind='bar', color=['m','y','c']) # this plots the distribution of online security
df['OnlineSecurity'].value_counts()
```

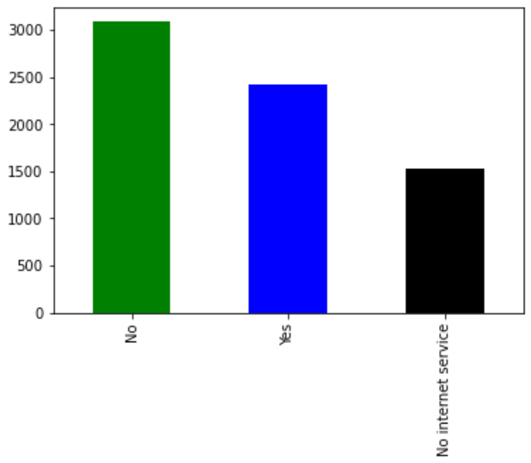
Out[26]:
```
No                     3498
Yes                    2019
No internet service    1526
Name: OnlineSecurity, dtype: int64
```



The number of customers opted for online security is 2019, not-opted is 3498 and no internet service is 1526.

In [27]:
```
df['OnlineBackup'].value_counts().plot(kind='bar', color=['g','b','k']) # this plots the distribution of online backup
df['OnlineBackup'].value_counts()
```

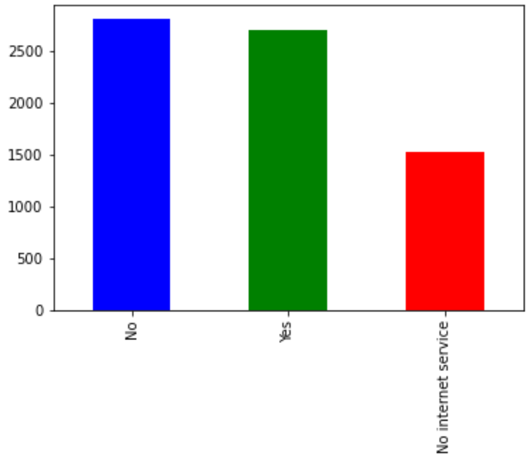
Out[27]:
```
No                     3088
Yes                    2429
No internet service    1526
Name: OnlineBackup, dtype: int64
```



The number of customers opted for online backup is 2429, not-opted is 3088 and no internet service is 1526.

In [28]:
```
df['StreamingTV'].value_counts().plot(kind='bar', color=['b','g','r']) # this plots the distribution of streamingTV
df['StreamingTV'].value_counts()
```

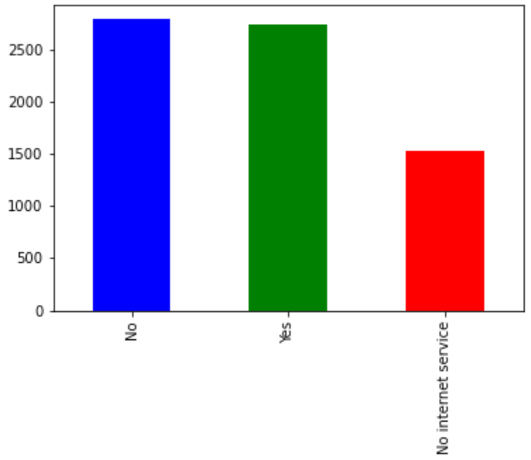
Out[28]:
```
No                     2810
Yes                    2707
No internet service    1526
Name: StreamingTV, dtype: int64
```



The number of customers streaming TV is 2707, not-streaming TV is 2810 and no internet service is 1526.

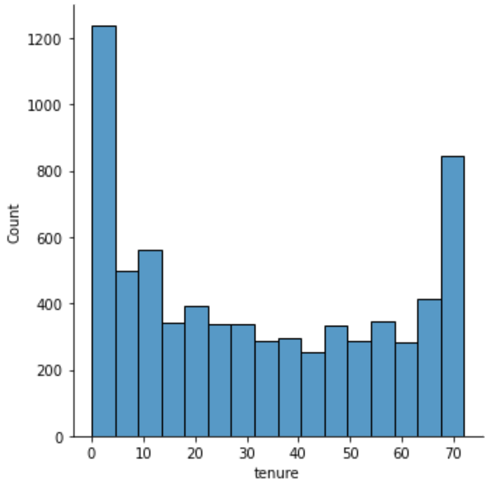
In [29]:
```
df['StreamingMovies'].value_counts().plot(kind='bar', color=['b','g','r']) # this plots the distribution of StreamingMovies
df['StreamingMovies'].value_counts()
```

Out[29]:
```
No                     2785
Yes                    2732
No internet service    1526
Name: StreamingMovies, dtype: int64
```



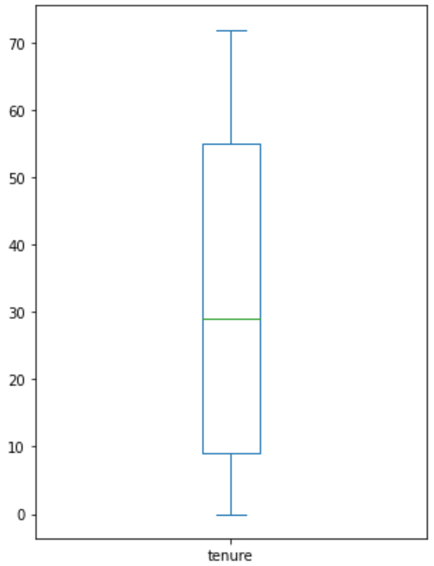
The number of customers streaming movies is 2732, not-streaming TV is 2785 and no internet service is 1526.

In [30]:
```
sns.displot(df['tenure']) # Similarly, we can visualise the distribution of the numerical variables of tenure
plt.show()
```



The bar plot depicts the tenure years with the number of customers. We can infer that 0 to 5 month contract is higher than all the tenure segments.

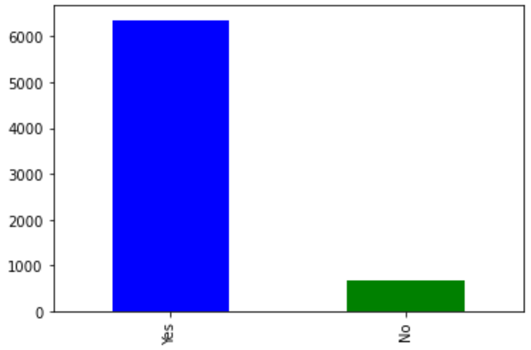
In [31]:
```
df['tenure'].plot.box(figsize=(5,7)) # Creates a box plot with five-number summary
plt.show()
```



This box plot for tenure shows the high,low,1st quartile, median quartile and the 3rd quartile. Median lies around tenure 30 months.

In [32]:
```
df['PhoneService'].value_counts().plot(kind='bar', color=['b','g'],) # this plots the distribution of users with phone serv
df['PhoneService'].value_counts()
```

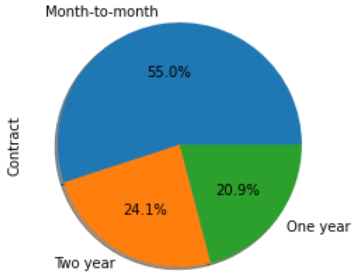
Out[32]:
```
Yes    6361
No      682
Name: PhoneService, dtype: int64
```



The number of customers who have opted for phone service is 6361 and those who are not is 682.

In [33]:
```
df['Contract'].value_counts().plot(kind='pie',shadow = True,autopct='%1.1f%%') # this pie plots the distribution of contrac
```

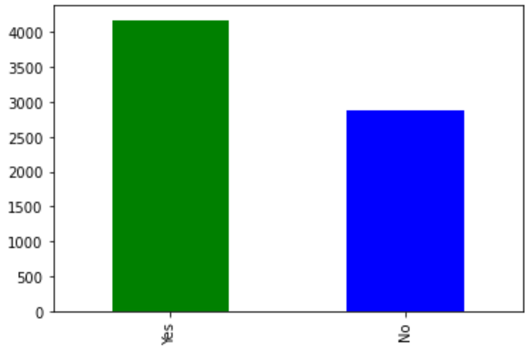
Out[33]:
```
<AxesSubplot:ylabel='Contract'>
```



The number of customers who have opted for month-to-month tenure percentage is 55, one-year is 20.9 and two-year is 24.1 percent.

In [34]:
```
df['PaperlessBilling'].value_counts().plot(kind='bar',color=['g','b']) # this plots the distribution of paperlessbilling
df['PaperlessBilling'].value_counts()
```

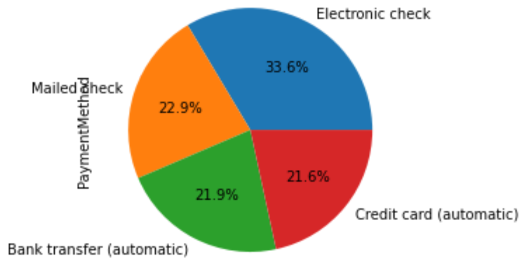
Out[34]:
```
Yes     4171
No      2872
Name: PaperlessBilling, dtype: int64
```



The number of customers who have opted for paperless billing is 4171 and those who are not is 2872.

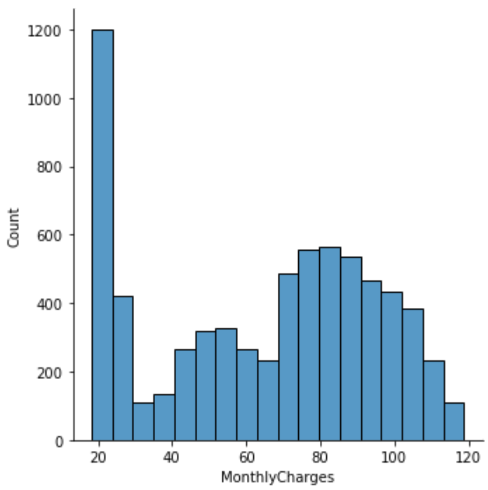
In [35]:
```
df['PaymentMethod'].value_counts().plot(kind='pie',autopct='%1.1f%%') # this plots the distribution of payment method
```

Out[35]:
```
<AxesSubplot:ylabel='PaymentMethod'>
```



The payment method opted by most customers is electronic check and the least is credit card method of payment.

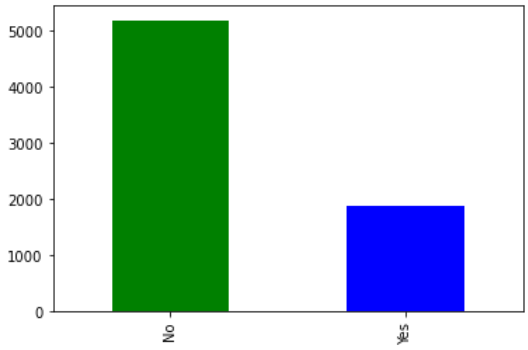
In [36]:
```
sns.displot(df['MonthlyCharges']) # distribution of monthly charges from 20 to 120 unit cash
plt.show()
```



The count of units of monthly charges vs customers depicts that most customers most customers pay 20 to 25 units of cash .

In [37]:
```
df['Churn'].value_counts().plot(kind='bar',color=['g','b']) # this plots the distribution of the target value churn
df['Churn'].value_counts()
```

Out[37]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

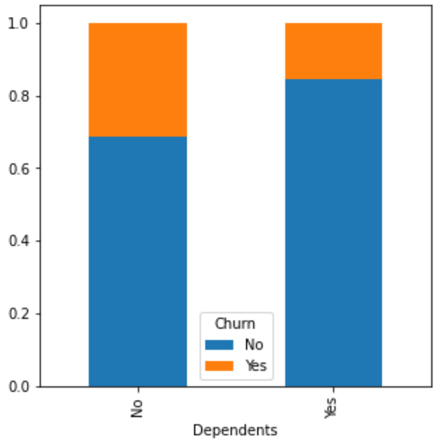


This bar chart depicts the number of users that leave the service which is 1869 and those who do not is 5174

Bivariate Analysis

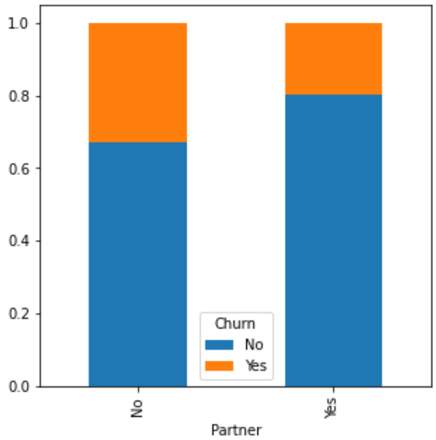
Categorical Independent Variable vs Target Variable

In [38]:
```
Dependents=pd.crosstab(df['Dependents'],df['Churn']) #matrix format that displays the frequency distribution of the variabl
Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) #normalize the table
plt.show()
```



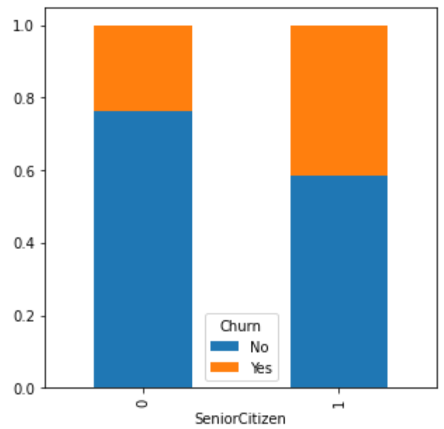
The bar chart shows normalized bars of number of dependents vs churn rate. we can infer that those who do not have dependents has the major losing customer compared to those with dependents.

In [39]:
```
Partner=pd.crosstab(df['Partner'],df['Churn']) # matrix format that displays the frequency distribution of the variables he
Partner.div(Partner.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize the table
plt.show()
```



The bar chart shows normalized bars of number of partner vs churn rate. we can note that those without partners has the highest churn rate compared to those who are without the partners.

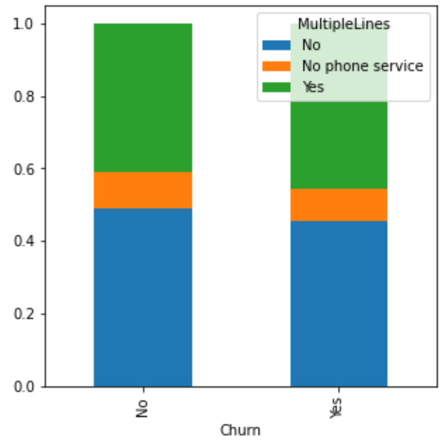
In [40]:
```
SeniorCitizen=pd.crosstab(df['SeniorCitizen'],df['Churn']) # matrix format that displays the frequency distribution of the
SeniorCitizen.div(SeniorCitizen.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize the t
plt.show()
```



The bar chart shows normalized bars of number of senior citizen vs churn rate. we can note that those who fall in to the senior citizen category has the highest churn rate.

In [41]:

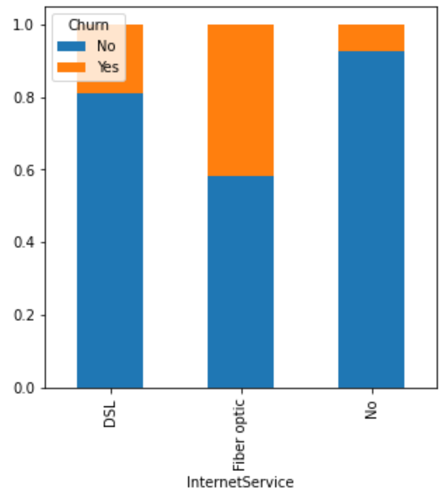
```
MultipleLines=pd.crosstab(df['Churn'],df['MultipleLines']) # matrix format that displays the frequency distribution of the
MultipleLines.div(MultipleLines.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize the t
plt.show()
```



The bar chart shows normalized bars of churn rate vs multiple lines of connection. Clearly, those who has multiple connections move out of the customer base.

In [42]:

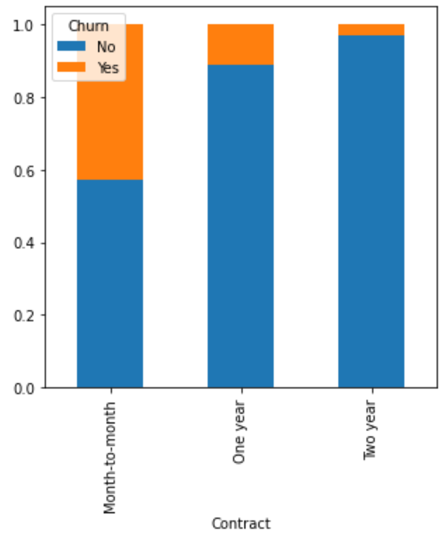
```
InternetService=pd.crosstab(df['InternetService'],df['Churn']) #matrix format that displays the frequency distribution of t
InternetService.div(InternetService.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize t
plt.show()
```



The bar chart shows normalized bars of type of internet service opted vs churn rate. Those customers who has opted for fiber optic internet service might be experiencing few problems and thus that particular segment has the highest churn rate.

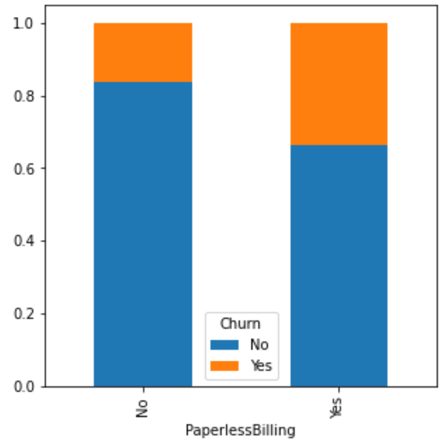
In [43]:

```
Contract=pd.crosstab(df['Contract'],df['Churn']) # matrix format that displays the frequency distribution of the variables
Contract.div(Contract.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize the table
plt.show()
```



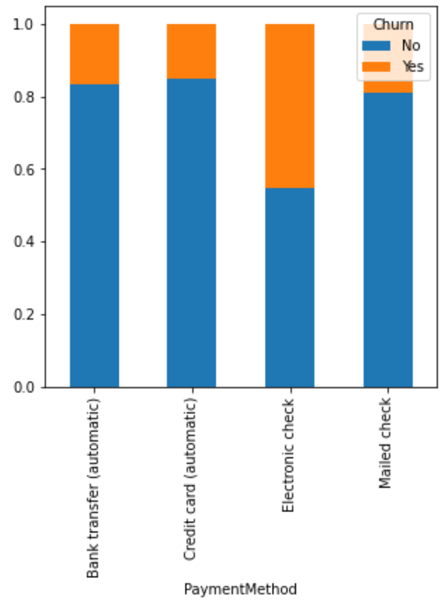
Here, those who took month-to-month contract has the highest churn rate since they are not bound by any contracts after a month and can switch to other services easily.

In [44]:
```
PaperlessBilling=pd.crosstab(df['PaperlessBilling'],df['Churn']) #matrix format that displays the frequency distribution of
PaperlessBilling.div(PaperlessBilling.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize
plt.show()
```



Here paperless billing has the highest churn rate than other billings.

In [45]:
```
PaymentMethod=pd.crosstab(df['PaymentMethod'],df['Churn']) # matrix format that displays the frequency distribution of the
PaymentMethod.div(PaymentMethod.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(5,5)) # normalize the t
plt.show()
```

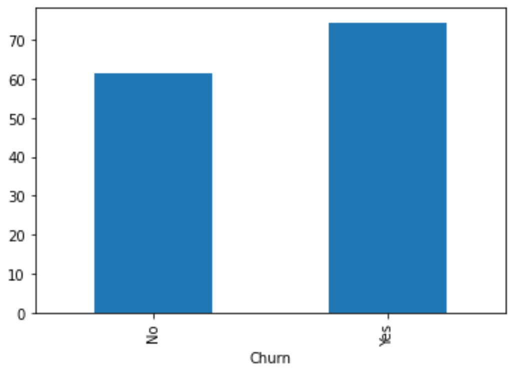


Here electronic check method of payment has the highest churn rate and credit card automatic has the least churn rate.

Numerical Independent Variable vs Target Variable

In [46]:
```
df.groupby('Churn')['MonthlyCharges'].mean().plot.bar() # displays bar chart of monthly charges by churn rate
```

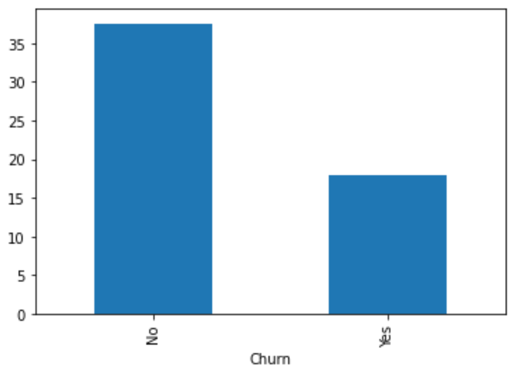
Out[46]: <AxesSubplot:xlabel='Churn'>



Churn rate vs the monthly charges shown in bar chart.

In [47]:
```python
df.groupby('Churn')['tenure'].mean().plot.bar() # displays bar chart of tenure by churn rate
```

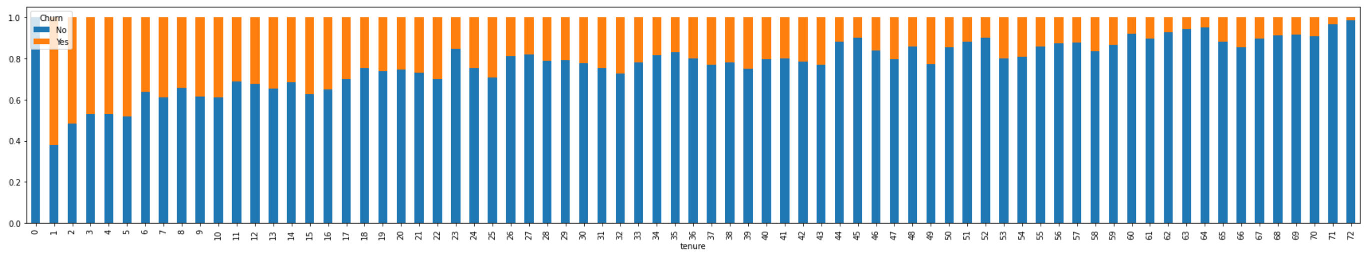
Out[47]: <AxesSubplot:xlabel='Churn'>



Churn rate vs the tenure shown in bar chart.

In [48]:
```python
df['x']=pd.cut(df['tenure'],1,0) # displays bar chart of tenure by churn data
x=pd.crosstab(df['tenure'],df['Churn'])
x.div(x.sum(1).astype(float), axis=0).plot(kind="bar",stacked=True,figsize=(30,5))
```

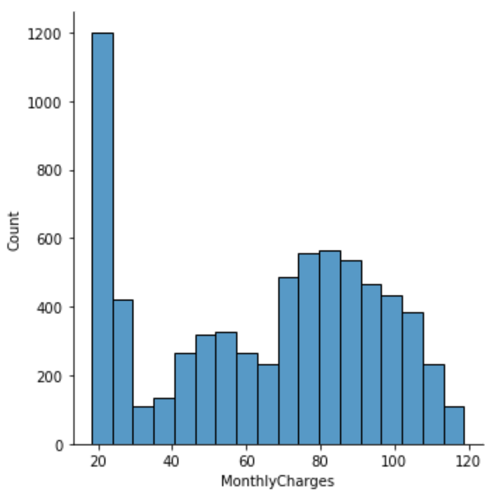
Out[48]: <AxesSubplot:xlabel='tenure'>

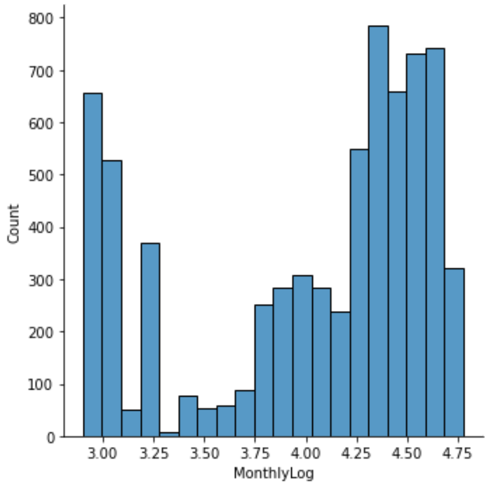


Here, the churn rate is shown for every tenure months. We can clearly notice that churn rate decreases with increase in tenure months.

In [49]:
```python
sns.displot(df['MonthlyCharges']) # displays bar chart of monthly charges by count
df['MonthlyLog'] = np.log(df['MonthlyCharges'])
sns.displot(df['MonthlyLog']) # Let's view the log-scaled distributions of monthly charges by count
```

Out[49]: <seaborn.axisgrid.FacetGrid at 0x7fcc5361d2e0>

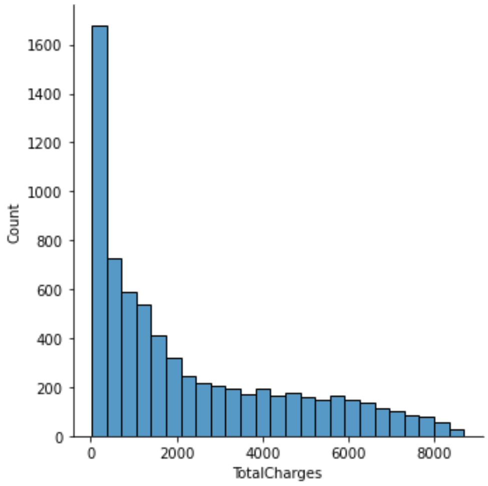


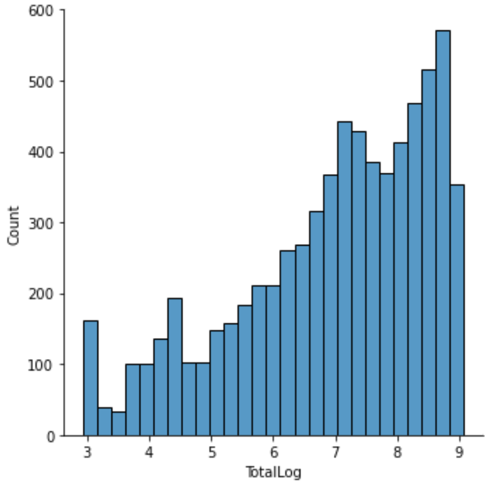


Clearly, most customers has opted for the affordable plans as we can infer from the bar chart. we can also plot the monthly log vs count of customers.

In [50]:
```
sns.displot(df['TotalCharges'])# displays bar chart of total charges by count
df['TotalLog'] = np.log(df['TotalCharges'])
sns.displot(df['TotalLog']) # Let's view the log-scaled distributions of total charges by count
```

Out[50]: <seaborn.axisgrid.FacetGrid at 0x7fcc53650670>



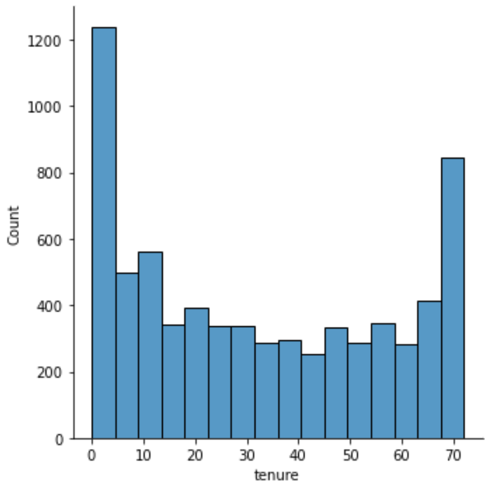


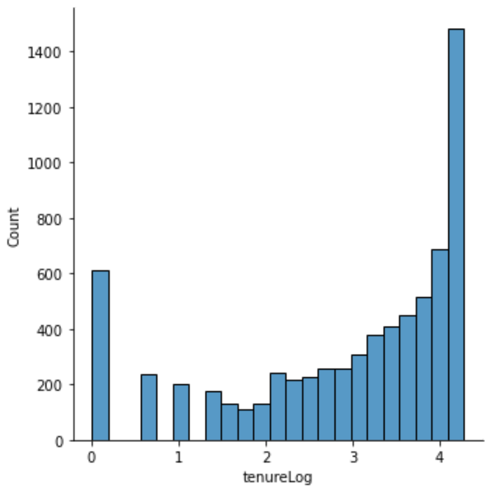
similary, total charges are the highest for entry level connection plans.

In [51]:
```
sns.displot(df['tenure'])# displays bar chart of tenure by count
df['tenureLog'] = np.log(df['tenure'])
sns.displot(df['tenureLog']) # Let's view the log-scaled distributions of tenure by count
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/arraylike.py:364: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[51]: <seaborn.axisgrid.FacetGrid at 0x7fcc53625b80>





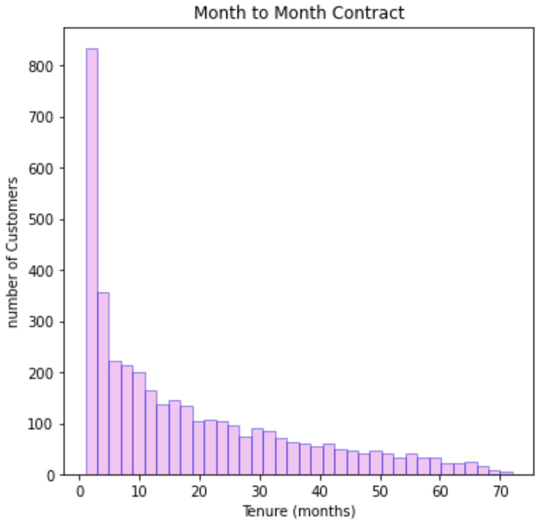
In [52]:

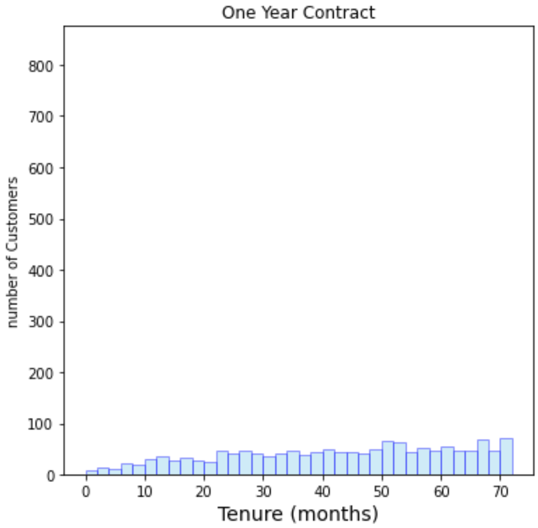
```
fig, (ax1,ax2,ax3) = plt.subplots(nrows=3, ncols=1, sharey = True, figsize = (6,20))
# #displays the tenure months for month to month vs the count of customers
ax = sns.distplot(df[df['Contract']=='Month-to-month']['tenure'],hist=True, kde=False,bins=int(180/5), color = 'orchid',his
ax.set_ylabel('number of Customers')
ax.set_xlabel('Tenure (months)')
ax.set_title('Month to Month Contract')

#displays the tenure months for one year contract vs the count of customers
ax = sns.distplot(df[df['Contract']=='One year']['tenure'],hist=True, kde=False,bins=int(180/5), color = 'skyblue',hist_kws
ax.set_xlabel('Tenure (months)',size = 14)
ax.set_ylabel('number of Customers')
ax.set_title('One Year Contract')

#displays the tenure months for two year contract vs the count of customers
ax = sns.distplot(df[df['Contract']=='Two year']['tenure'],hist=True, kde=False,bins=int(180/5), color = 'yellowgreen',hist_
ax.set_xlabel('Tenure (months)')
ax.set_ylabel('number of Customers')
ax.set_title('Two Year Contract')
```

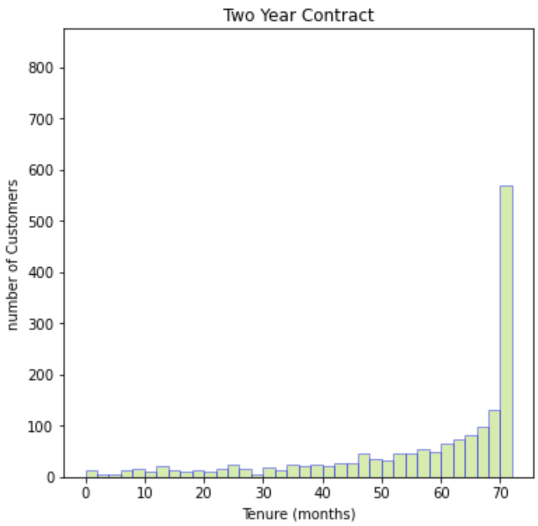
```
/opt/conda/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function an
d will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar
flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[52]:

```
Text(0.5, 1.0, 'Two Year Contract')
```







Here, we can see three plots for month-to-month contract, one year contract and two year contract vs number of customers

In [53]:
```
from sklearn.preprocessing import LabelEncoder #library to convert categorical data to integer values
```

In [54]:
```
# setting integer values to all the below columns with categorical values
cols = ['gender','Partner','Dependents','MultipleLines','InternetService','OnlineSecurity','OnlineBackup','DeviceProtection
le = LabelEncoder()                          # initialising the necessary function taken from the LabelEncoder library
for col in cols:                             # iterate over all variables in cols
    df[col] = le.fit_transform(df[col])     # convert categorical values into integer values
```

In [55]:
```
df.head() # returns the first 5 lines of the dataframe
```

Out[55]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	...	Contract	Pa
0	0	0	1	0	1	0	1	0	0	2	...	0	
1	1	0	0	0	34	1	0	0	2	0	...	1	
2	1	0	0	0	2	1	0	0	2	2	...	0	
3	1	0	0	0	45	0	1	0	2	0	...	1	
4	0	0	0	0	2	1	0	1	0	0	...	0	

5 rows × 24 columns

In [56]:

```
df = df.drop(['x'],axis=1) # dropping the x variable because it is useless for prediction
```

In [57]:

```
df = df.drop(['MonthlyLog'],axis=1) # We drop the monthlylog variable because it is useless for prediction
```

In [58]:

```
df = df.drop(['TotalLog'],axis=1) # We drop the totallog because it is useless for prediction
df = df.drop(['tenureLog'],axis=1) # We drop the tenurelog because it is useless for prediction
df.head()
```

Out[58]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0	0	1	0	1	0	1	0	0	2	0
1	1	0	0	0	34	1	0	0	2	0	2
2	1	0	0	0	2	1	0	0	2	2	0
3	1	0	0	0	45	0	1	0	2	0	2
4	0	0	0	0	2	1	0	1	0	0	0

Correlation Matrix

Let's view the correlation between features

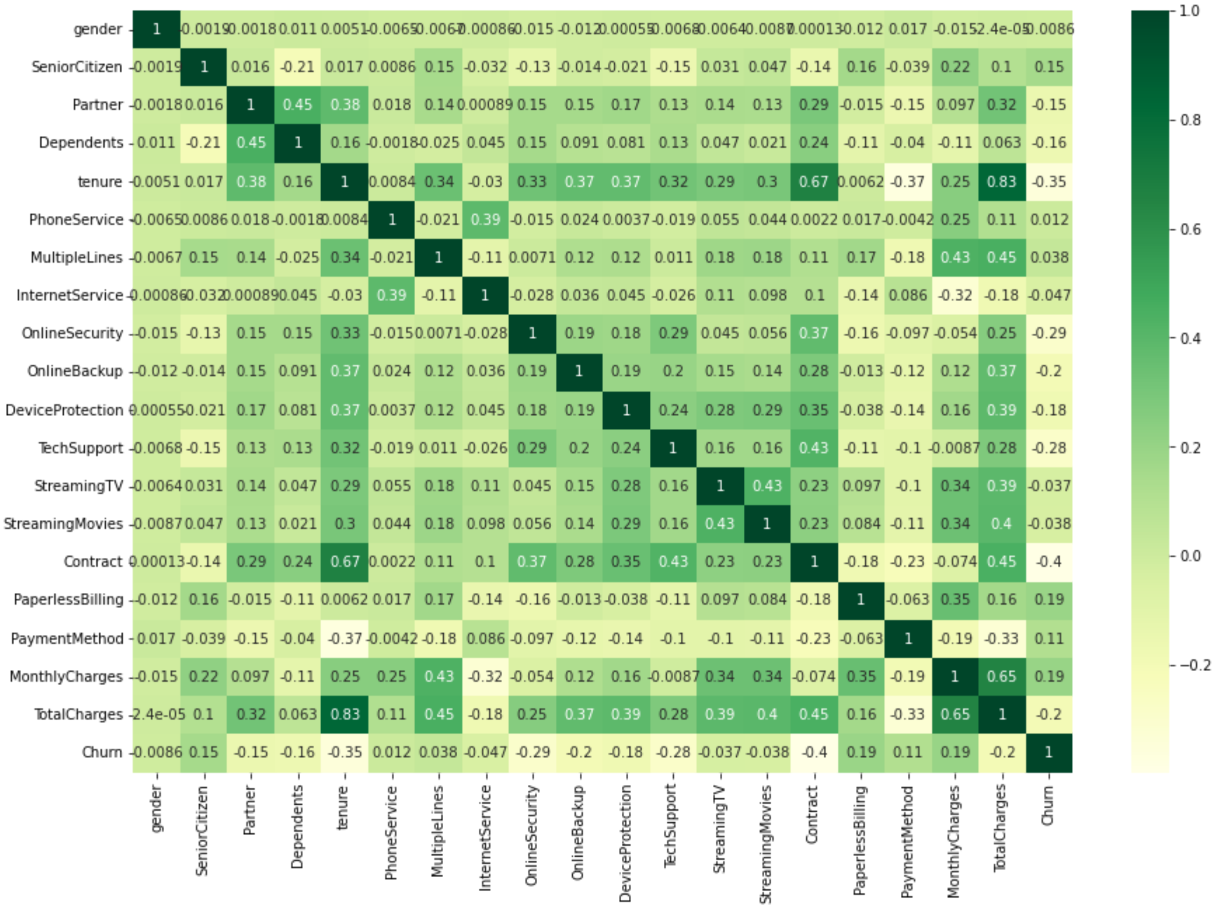
In [59]:

```
corr=df.corr() # gives us the correlation values
```

In [60]:

```
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot = True, cmap="YlGn")  # visualise the correlation matrix
```

Out[60]: <AxesSubplot:>



Correlation matrix was plotted for all the column attributes and the correlation co-efficient was found with respect to other attributes. We can remove highly correlated features.

In [61]:
```
# specify inputs 'x' and output 'y' attributes - target value
X = df.drop(['Churn'],axis=1)
y = df['Churn']
```

k-best method

In [62]:
```
# using the k-best method to get to know the most highest predictive features of 'y' attribute
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [63]:
```
# passing x and y as inputs to the k-best method
k = SelectKBest(score_func = chi2, k = 'all')
features = k.fit(X,y)
```

In [64]:
```
# new dataframe for the feature scores
orders = pd.DataFrame(features.scores_, columns=['scores'])
```

In [65]:
```
#new dataframe for the feature feature_name
df_columns = pd.DataFrame(X.columns, columns = ['Feature_name'])
```

In [66]:
```
#combining the above two feature dataframe into a dataframe
feature_rank = pd.concat([orders,df_columns],axis=1) # combine the two dataFrames
```

In [67]:
```
#ranks displayed based on the scores and the chi2 scoring func for the 19 attribute columns
feature_rank.nlargest(19,'scores')
```

Out[67]:

	scores	Feature_name
18	627147.530404	TotalCharges
4	16278.923685	tenure
17	3680.787699	MonthlyCharges
14	1115.780167	Contract
8	551.611529	OnlineSecurity
11	523.303866	TechSupport
9	230.086520	OnlineBackup
10	191.303140	DeviceProtection
1	134.351545	SeniorCitizen
3	133.036443	Dependents
15	105.680863	PaperlessBilling
2	82.412083	Partner
16	58.492250	PaymentMethod
7	9.821028	InternetService
6	9.746921	MultipleLines
13	8.235399	StreamingMovies
12	7.490203	StreamingTV
0	0.258699	gender
5	0.097261	PhoneService

Here, total charges is the most important feature that is necessary for prediction and the phone service feature is the least important feature.

In [68]:

```
df.head()
```

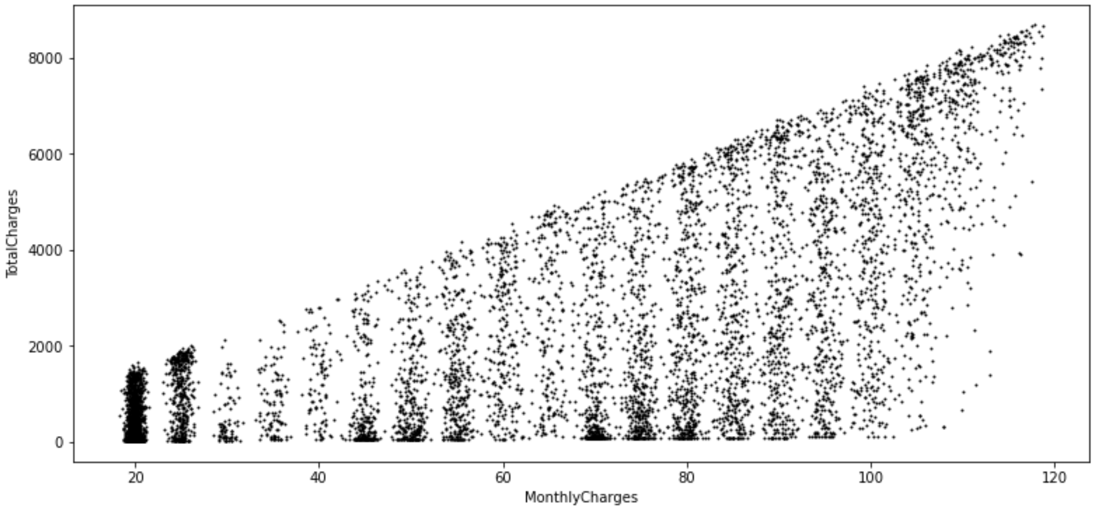
Out[68]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0	0	1	0	1	0	1	0	0	2	0
1	1	0	0	0	34	1	0	0	2	0	2
2	1	0	0	0	2	1	0	0	2	2	0
3	1	0	0	0	45	0	1	0	2	0	2
4	0	0	0	0	2	1	0	1	0	0	0

In [69]:

```
# scatter plot of monthlycharges vs totalcharges
df[['MonthlyCharges', 'TotalCharges']].plot.scatter(x = 'MonthlyCharges',y='TotalCharges',color="black",s=1,figsize = (13,6
```

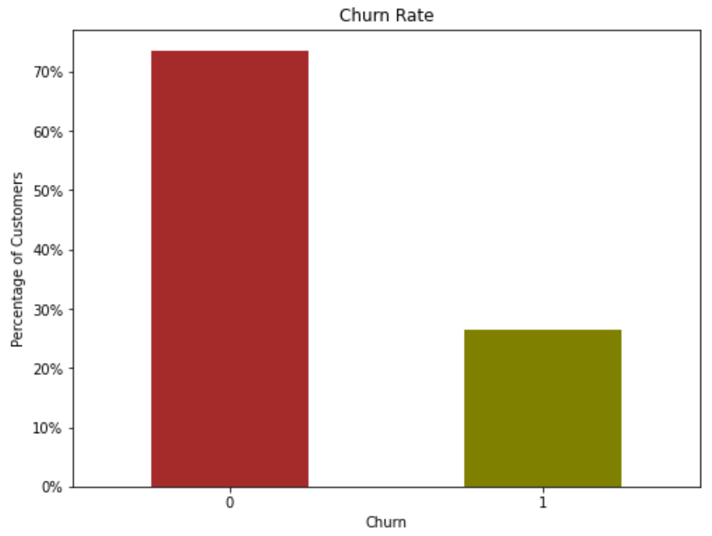
Out[69]: <AxesSubplot:xlabel='MonthlyCharges', ylabel='TotalCharges'>



In [70]:

```
# plotting the churn rate in percentage with respect to the number of customers
import matplotlib.ticker as mtick
colors = ['brown','olive']
ax = (df['Churn'].value_counts()*100.0 /len(df)).plot(kind='bar',stacked = True,rot = 0,color = colors,figsize = (8,6))
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.set_ylabel('Percentage of Customers')
ax.set_xlabel('Churn')
ax.set_title('Churn Rate')
```

Out[70]: Text(0.5, 1.0, 'Churn Rate')



The churn rate == true is 26.5 percent and churn rate == false is 73.5 percent.

1. Logistic Regression

In [71]:

```
df_dummies = pd.get_dummies(df)
df_dummies.head()
```

Out[71]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0	0	1	0	1	0	1	0	0	2	0
1	1	0	0	0	34	1	0	0	2	0	2
2	1	0	0	0	2	1	0	0	2	2	0
3	1	0	0	0	45	0	1	0	2	0	2
4	0	0	0	0	2	1	0	1	0	0	0

In [72]:

```
df.head()
```

Out[72]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	0	0	1	0	1	0	1	0	0	2	0
1	1	0	0	0	34	1	0	0	2	0	2
2	1	0	0	0	2	1	0	0	2	2	0
3	1	0	0	0	45	0	1	0	2	0	2
4	0	0	0	0	2	1	0	1	0	0	0

In [73]:

```
# We will use the data frame where we had created dummy variables
y = df_dummies['Churn'].values
X = df_dummies.drop(columns = ['Churn'])

# Scaling all the variables to a range of 0 to 1
from sklearn.preprocessing import MinMaxScaler
features = X.columns.values
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X))
X.columns = features
```

In [74]:

```
# Create Train & Test Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [75]:

```
# Running logistic regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result = model.fit(X_train, y_train)
```

In [76]:

```
#importing the metrics package
from sklearn import metrics
prediction_test = model.predict(X_test)
# Print the prediction accuracy
print(metrics.accuracy_score(y_test, prediction_test)*100)
```

```
80.50165641268339
```

For logistic regression we have got an accuracy of 80.50 percent.

In [77]:
```
# printing all the metrics for analysis
print("Precision: "+str(round(metrics.precision_score(y_test,prediction_test.round())*100,5)))
print("Accuracy: "+str(round(metrics.accuracy_score(y_test,prediction_test.round())*100,5)))
print("Recall: "+str(round(metrics.recall_score(y_test,prediction_test.round(),average="binary")*100,5)))
print("F1 score: "+str(round(metrics.f1_score(y_test,prediction_test.round(),average="binary")*100,5)))
print("ROC_AUC: "+str(round(metrics.roc_auc_score(y_test,prediction_test.round())*100,5)))
```

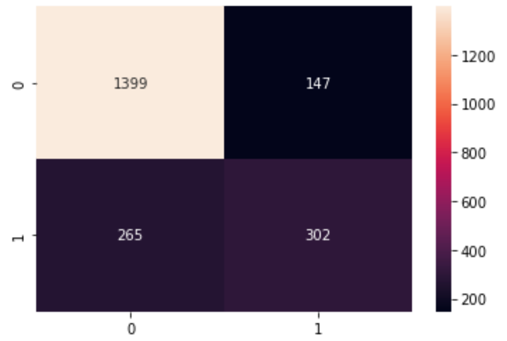
```
Precision: 67.26058
Accuracy: 80.50166
Recall: 53.26279
F1 score: 59.44882
ROC_AUC: 71.87719
```

All metrics such as precision, accuracy, recall, F1_score and ROC_AUC was observed as above.

In [78]:
```
# Create the Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,prediction_test))
cm = confusion_matrix(y_test,prediction_test)
ax= plt.subplot()
sns.heatmap(cm,annot=True,fmt='g',ax=ax)
# plots the confusion matrix
```

```
[[1399  147]
 [ 265  302]]
```

Out[78]:
```
<AxesSubplot:>
```



[TP - TRUE POSITIVE - 302 - Correctly predicted churn == Yes]
[FP - FALSE POSITIVE - 147 - Falsely predicted churn == Yes]
[TN - TRUE NEGATIVE - 1399 - Correctly predicted churn == No]
[FN - FALSE NEGATIVE - 265 - Falsely predicted churn == No].

K fold cross validation

In [79]:
```
# k fold cross validation where k = 10 here
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
```

```
Accuracy: 79.84 %
```

The accuracy of logistic regression is 79.84 percent for k = 10 , 10 fold cross validation.

In [80]:
```
# k fold cross validation where k=9
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 9)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
```

```
Accuracy: 80.06 %
```

The accuracy of logistic regression is 79.84 percent for k = 9 , 9 fold cross validation. Here 9 fold cv has accuracy higher compared to 10 fold.

The recall score signifies the number of correct positive predictions made out of all positive predictions.
Previously, we have got an recall score of 53.26 %. So we can increase this by making including class_weight = {0:1,1:2}. This will help to overcome any class imbalance in our dataset. model = LogisticRegression(class_weight={0:1,1:2})

In [81]:
```
# We will use the data frame where we had created dummy variables
y = df_dummies['Churn'].values
X = df_dummies.drop(columns = ['Churn'])

# Scaling all the variables to a range of 0 to 1
from sklearn.preprocessing import MinMaxScaler
features = X.columns.values
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X)
X = pd.DataFrame(scaler.transform(X))
X.columns = features
```

In [82]:
```
# Create Train & Test Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [83]:
```
# Running logistic regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(class_weight={0:1,1:2})
result = model.fit(X_train, y_train)
```

In [84]:
```
#importing the metrics package
from sklearn import metrics
prediction_test = model.predict(X_test)
# Print the prediction accuracy
print(metrics.accuracy_score(y_test, prediction_test)*100)
```

```
77.1888310459063
```

In [85]:
```
print("Precision: "+str(round(metrics.precision_score(y_test,prediction_test.round())*100,5)))
print("Accuracy: "+str(round(metrics.accuracy_score(y_test,prediction_test.round())*100,5)))
print("Recall: "+str(round(metrics.recall_score(y_test,prediction_test.round(),average="binary")*100,5)))
print("F1 score: "+str(round(metrics.f1_score(y_test,prediction_test.round(),average="binary")*100,5)))
print("ROC_AUC: "+str(round(metrics.roc_auc_score(y_test,prediction_test.round())*100,5)))
```

```
Precision: 55.94406
Accuracy: 77.18883
Recall: 70.54674
F1 score: 62.4025
ROC_AUC: 75.08579
```

Although, the accuracy has drop a 3 percent the recall score went up noticeably from 53.26 to 70.54 percent.

2. Support Vecor Machine (SVM)

In [86]:
```
# Create Train & Test Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=99)
```

In [87]:
```
#Support Vector Machine for classification using SVC
from sklearn.svm import SVC

model.svm = SVC(kernel='linear')
model.svm.fit(X_train,y_train)
preds = model.svm.predict(X_test)
metrics.accuracy_score(y_test, preds)*100
```

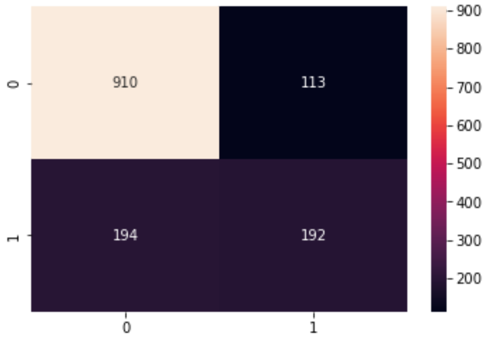
Out[87]:
```
78.21149751596877
```

For Support Vector Machine we have trained a model with accuracy of 78.21 percent.

In [88]:
```
# Create the Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,preds))
cm = confusion_matrix(y_test,preds)
ax= plt.subplot()
sns.heatmap(cm,annot=True,fmt='g',ax=ax)
# plots the confusion matrix
```

```
[[910 113]
 [194 192]]
```

Out[88]:
```
<AxesSubplot:>
```



[TP - TRUE POSITIVE - 192 - Correctly predicted churn == Yes]

[FP - FALSE POSITIVE - 113 - Falsely predicted churn == Yes]

[TN - TRUE NEGATIVE - 910 - Correctly predicted churn == No]

[FN - FALSE NEGATIVE - 194 - Falsely predicted churn == No].

In [89]:
```
df['Churn'].value_counts()
```

Out[89]:
```
0    5174
1    1869
Name: Churn, dtype: int64
```

In [90]:
```
# performing k fold cross validation where K = 10 as most cases
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = model.svm, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
```

```
Accuracy: 79.84 %
```

When we set the K-fold cross validation to 10 fold, we get an accuracy of 79.84 percent from 78.21 percent