Solution.py

April 14, 2020

Dataset is from the link https://www.kaggle.com/blastchar/telco-customer-churn

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import norm
from scipy.stats import chi2_contingency
from scipy.stats import ttest_ind

class EDA():
    '''
    Very basic EDA class to show the very basic nature of data.
    dataframe is not required for the __init__() constructor to make it very␣
↪generic.
    A pandas dataframe is required to be in every method.
    '''


    def check_distribution(self,df,column_name):
        '''
        Checks the distribution of column in data given the column is␣
↪categorical in nature.
        input:
            class_name: name of the column in dataframe
        out:
            display a pie chart with respective percentages of the classes
        '''

        df[column_name].value_counts().plot(kind='pie',autopct='%1.2f%%',␣
↪rotatelabels=True)



    def skew_plot(self,df,num_cols,cols=4,width=15,height=12,tight=True):
        '''
        Find skewness of the numerical data
```

```
        args:
            df: pandas dataframe
            num_cols: list of names of numerical cols of the dataframe
            cols: number of columns for the subplots
            PAD: padding for spacing of subplots
            width: width of figure size
            height: figure height
            tight: plt.tight_layout()
        '''
        rows = len(num_cols)//2+1
        f,ax = plt.subplots(nrows=rows,ncols=cols, figsize=(width,height))
        ax = ax.ravel()

        i=0
        col_num=0
        while col_num<=len(num_cols)-1:
            column = num_cols[col_num]
            sns.distplot(df[column], fit=norm,ax=ax[i])
            stats.probplot(df[column], plot=ax[i+1])
            i+=2
            col_num+=1

        for axis_num in range(i,(cols*rows)): # delete the remaining empty plots
                f.delaxes(ax[axis_num])

        if tight:
            plt.tight_layout()
        plt.show()



    def box_vio(self,df,num_cols,cols=4,width=15,height=12,tight=True):
        '''
        plot box and violin plots of numerical plots
        args:
            df: pandas dataframe
            num_cols: list of names of numerical cols of the dataframe
            cols: number of columns for the subplots
            PAD: padding for spacing of subplots
            width: width of figure size
            height: figure height
            tight: plt.tight_layout()
        '''
        rows = len(num_cols)//2+1
        f,ax = plt.subplots(nrows=rows,ncols=cols, figsize=(width,height))
        ax = ax.ravel()
```

```
        i=0
        col_num=0
        while col_num<=len(num_cols)-1:
            column = num_cols[col_num]
            sns.boxplot(y=df[column],ax=ax[i],color='m')
            ax[i].set_xlabel(column+ ' Box')
            sns.violinplot(y=df[column],ax=ax[i+1],color='teal')
            ax[i+1].set_xlabel(column+ ' Violin')
            i+=2
            col_num+=1

        for axis_num in range(i,(cols*rows)): # delete the remaining empty plots
                f.delaxes(ax[axis_num])

        if tight:
            plt.tight_layout()
        plt.show()


    def fix_heatmap(self):
        '''
        fix the half cut upper and lower boxes of a heatmap in newer versions␣
↪of Seaborn
        '''
        b, t = plt.ylim() # discover the values for bottom and top
        b += 0.5 # Add 0.5 to the bottom
        t -= 0.5 # Subtract 0.5 from the top
        plt.ylim(b, t) # update the ylim(bottom, top) values
        plt.show()


    def show_corr(self,df,plot=True,return_corr_df=False,size=(10,10)):
        '''
        Show the correlation within a dataframe
        args:
            df: pandas datadframe
            plot: {bool} default False. Whether to show the heatmap or not
            return_corr_df: return the correlated df or not
            size: tuple (w,h) for the size of plot
        out:
            returns correlated squared DataFrame
        '''

        if plot:
            fig, ax = plt.subplots(figsize=size)
            sns.heatmap(df.corr(),annot=True,␣
↪cmap='coolwarm',facecolor='b',lw=2, ax=ax)
```

```
            self.fix_heatmap()

        if return_corr_df:
            print(f'Correlation of each column:\n\n')
            return df.corr()


    def␣
↪cat_vs_nums(self,df,cat_col_name,numeric_cols,cols=3,width=15,height=7,tight=False,**kwargs
↪
        '''
        method to plot the distribution categorical within all  of the␣
↪numerical columns
        args:
            df: pandas dataframe
            cat_col_name: name of the column whose distribution we want to␣
↪check among all the numreical columns
            numeric_cols: {list} of all  numerical the column names
            cols: number of graphs to plot per row. default 3
            width: width of plot
            height: height of plot
            tight: plt.tight_layout()
            **kwargs: sns.kdeplot() arguments
        '''

        class_values = df[cat_col_name].value_counts().index.tolist() # unique␣
↪target classes
        rows = len(numeric_cols)//cols+1
        f,ax = plt.subplots(rows,cols,figsize=(width,height))
        ax = ax.ravel()
        for i,column in enumerate(numeric_cols):
            for j in class_values:
                sns.kdeplot(df.loc[df[cat_col_name] == j, column], label =␣
↪f'{cat_col_name} {j}',ax=ax[i],**kwargs)
            ax[i].set_xlabel(column)

        for axis_num in range(i+1,(cols*rows)): # delete the remaining empty␣
↪plots
            f.delaxes(ax[axis_num])

        if tight:
                plt.tight_layout()
        plt.show()
```

```
    def␣
↪cat_vs_cats(self,df,col_name,cat_cols,cols=4,width=15,height=9,tight=False,**kwargs):
↪
        '''
        method to distribution of classes of a categorical among all other␣
↪categorical columns
        args:
            df: pandas dataframe
            col_name: name of categorical column whose distribution we want to␣
↪check
            cat_cols: {list/tuple} of all the other categorical columns
            cols: number of figures to plot at each row
            width: width of plot
            height: height of plot
            tight: plt.tight_layout()
            **kwargs: arguments for seaborn.countplot()
        '''
        rows = len(cat_cols)//cols+1
        f,ax = plt.subplots(rows,cols,figsize=(width,height))
        ax = ax.ravel()
        for i,column in enumerate(cat_cols):
            sns.countplot(x=column, hue=col_name, data=df,ax=ax[i],**kwargs)

        for axis_num in range(i+1,(cols*rows)): # delete the remaining empty␣
↪plots
            f.delaxes(ax[axis_num])

        if tight:
                plt.tight_layout()
        plt.show()



eda = EDA()
```

[2]:
```
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
df.head()
```

[2]:

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

	MultipleLines	InternetService	OnlineSecurity	…	DeviceProtection
0	No phone service	DSL	No	…	No

```
1                No             DSL            Yes  …              Yes
2                No             DSL            Yes  …               No
3  No phone service             DSL            Yes  …              Yes
4                No     Fiber optic             No  …               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

[3]:
```
df.drop('customerID',axis=1,inplace=True)
```

[4]:
```
df.shape
```

[4]:
```
(7043, 20)
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
Contract            7043 non-null object
```

```
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

[11]:
```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],errors='coerce')
df['tenure'] = pd.to_numeric(df['tenure'],errors='coerce')
```

[12]:
```
df.isna().sum()
```

[12]:
```
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
```

[13]:
```
df['TotalCharges'].fillna(df['TotalCharges'].median(),inplace=True)
```

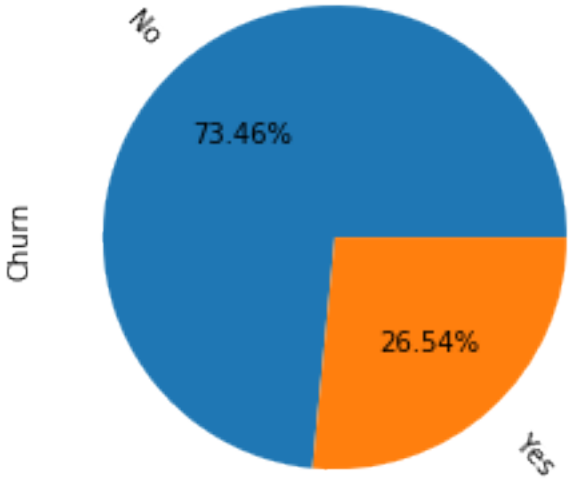
[14]:
```
df.columns.tolist()
cat = ['gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
```

```
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies','Contract',
 'PaperlessBilling',
 'PaymentMethod',]

num = ['MonthlyCharges','TotalCharges','tenure']
```

0.0.1 Distribution of Churn in the whole population

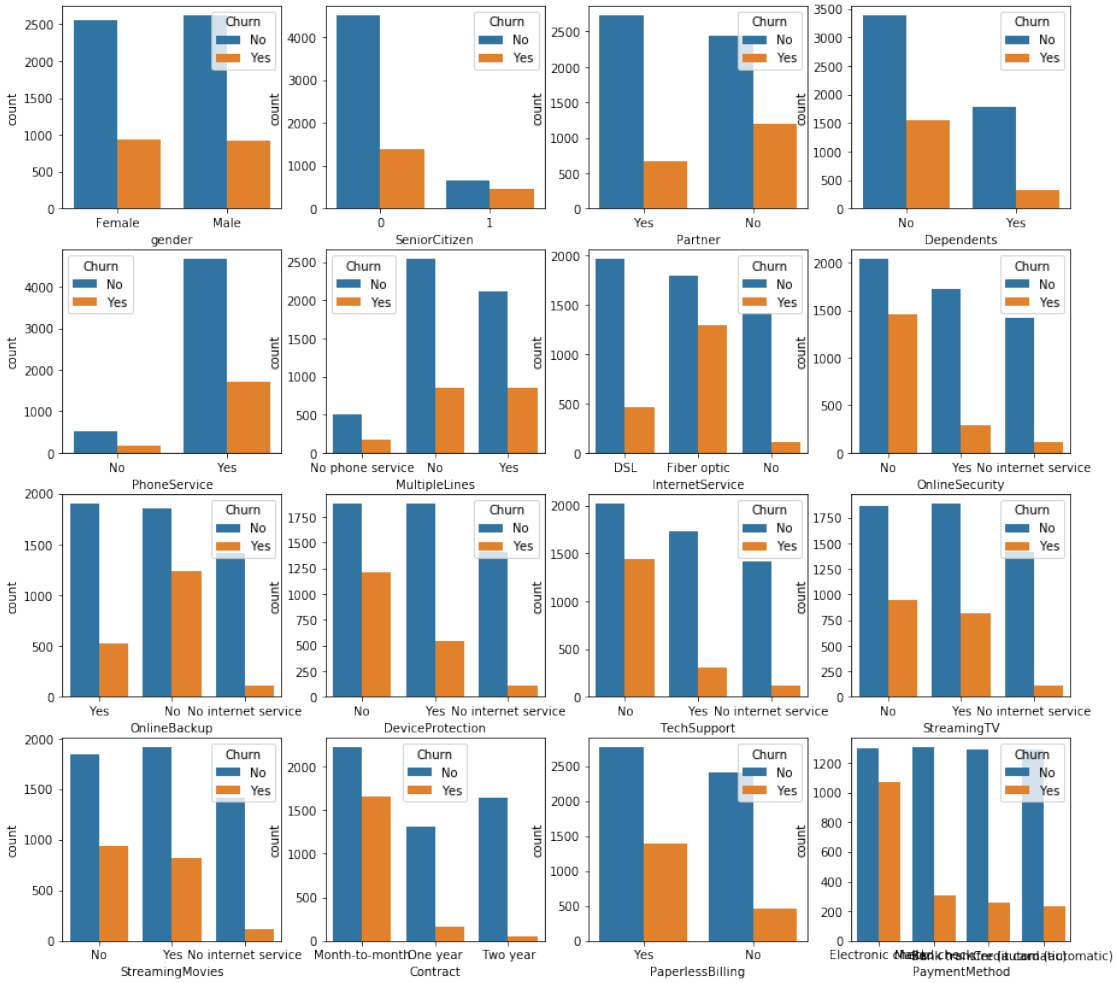
[15]:
```
eda.check_distribution(df,'Churn')
```



There are 26% Customers who Churn and 73% do not

0.0.2 DIstribution of Churn inside Categorical classes

[17]:
```
eda.cat_vs_cats(df,'Churn',cat,height=18)
```



`It Just Looks at the First Glance that`

Device Protection: People who do not have any DeviceProtection leave the services more than who do have

Online Security: People with no onlone Security Leave the company after using and those who have Online Security, tend to stay with the company

Tech Support: People with No tech Support seem to leave the Services and Churn while on the other hand people who have Internet Services, Do not Churn

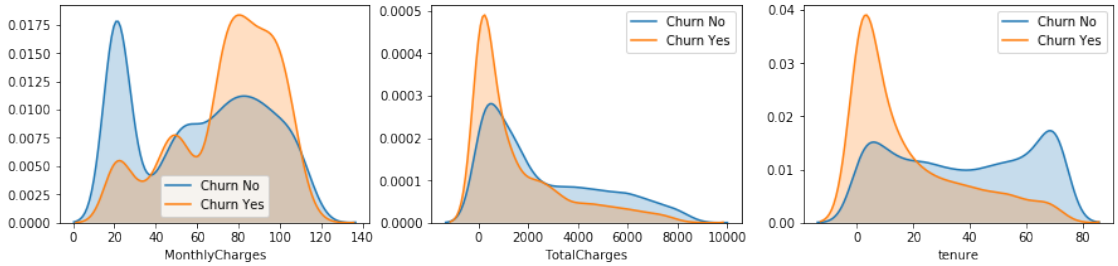
Contract: People with Month to Month seem to Churn the most and the ones who have Two Year Contract, are rarely to churn so it seems that Churn is affected by Contract

PaperlessBilling: Who prefer paperless billing, tend to leave the services more than who do not. This may be because of the old generation people tend to stick to the old things or they are too tired of switching the company every now and then.

`But These are just assumptions. We will implement some tests to check whether these make sense or not`

0.0.3 Distribution of Churn inside Numerical values

[18]:
```
eda.cat_vs_nums(df,'Churn',num,shade=True)
```

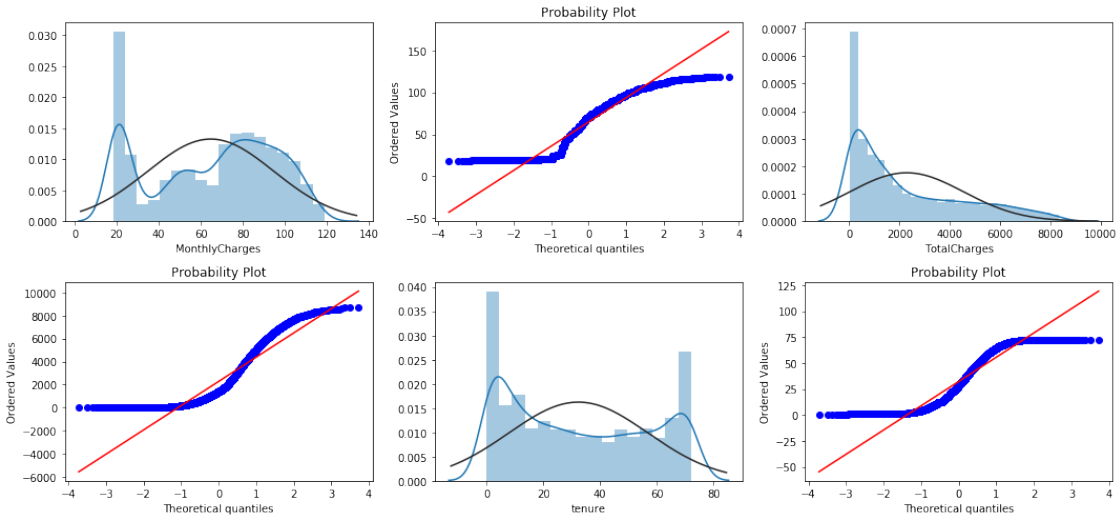


It looks like Customers who have less Tenure tend to Churn.

Most of the Customers with High Monthly Charges Churn

0.0.4 Distribution of Numerical Plots with Normality

[22]:
```
eda.skew_plot(df,num,cols=3,height=7)
```



Data is not Normally Distributed

[24]:
```
for i,col in enumerate(cat):
    ch_gen = pd.crosstab(df['Churn'],df[col])
    #print(ch_gen)

    chi2,p,dof,expected = chi2_contingency(ch_gen)
```

```
    print(f'Chi Square values is {chi2} with a p-value of {p}')

    if p <=0.01:
        print(f"-{col}- has a  significant relationship with Churn\n{'*'*20}\n")
    else:
        print(f"-{col}- does not have a significant relationship with␣
↪Churn\n{'*'*20}\n")
```

```
Chi Square values is 0.4840828822091383 with a p-value of 0.48657873605618596
-gender- does not have a significant relationship with Churn
**********

Chi Square values is 159.42630036838742 with a p-value of 1.510066805092378e-36
-SeniorCitizen- has a  significant relationship with Churn
**********

Chi Square values is 158.7333820309922 with a p-value of 2.1399113440759935e-36
-Partner- has a  significant relationship with Churn
**********

Chi Square values is 189.12924940423474 with a p-value of 4.9249216612154196e-43
-Dependents- has a  significant relationship with Churn
**********

Chi Square values is 0.9150329892546948 with a p-value of 0.3387825358066928
-PhoneService- does not have a significant relationship with Churn
**********

Chi Square values is 11.33044148319756 with a p-value of 0.0034643829548773
-MultipleLines- has a  significant relationship with Churn
**********

Chi Square values is 732.309589667794 with a p-value of 9.571788222840544e-160
-InternetService- has a  significant relationship with Churn
**********

Chi Square values is 849.9989679615962 with a p-value of 2.6611496351768565e-185
-OnlineSecurity- has a  significant relationship with Churn
**********

Chi Square values is 601.8127901134089 with a p-value of 2.0797592160865457e-131
-OnlineBackup- has a  significant relationship with Churn
**********

Chi Square values is 558.419369407389 with a p-value of 5.505219496457244e-122
-DeviceProtection- has a  significant relationship with Churn
```

```
**********

Chi Square values is 828.1970684587393 with a p-value of 1.4430840279999813e-180
-TechSupport- has a  significant relationship with Churn
**********

Chi Square values is 374.20394331098134 with a p-value of 5.528994485739024e-82
-StreamingTV- has a  significant relationship with Churn
**********

Chi Square values is 375.6614793452656 with a p-value of 2.667756755723681e-82
-StreamingMovies- has a  significant relationship with Churn
**********

Chi Square values is 1184.5965720837926 with a p-value of 5.863038300673391e-258
-Contract- has a  significant relationship with Churn
**********

Chi Square values is 258.27764906707307 with a p-value of 4.073354668665985e-58
-PaperlessBilling- has a  significant relationship with Churn
**********

Chi Square values is 648.1423274814 with a p-value of 3.6823546520097993e-140
-PaymentMethod- has a  significant relationship with Churn
**********
```

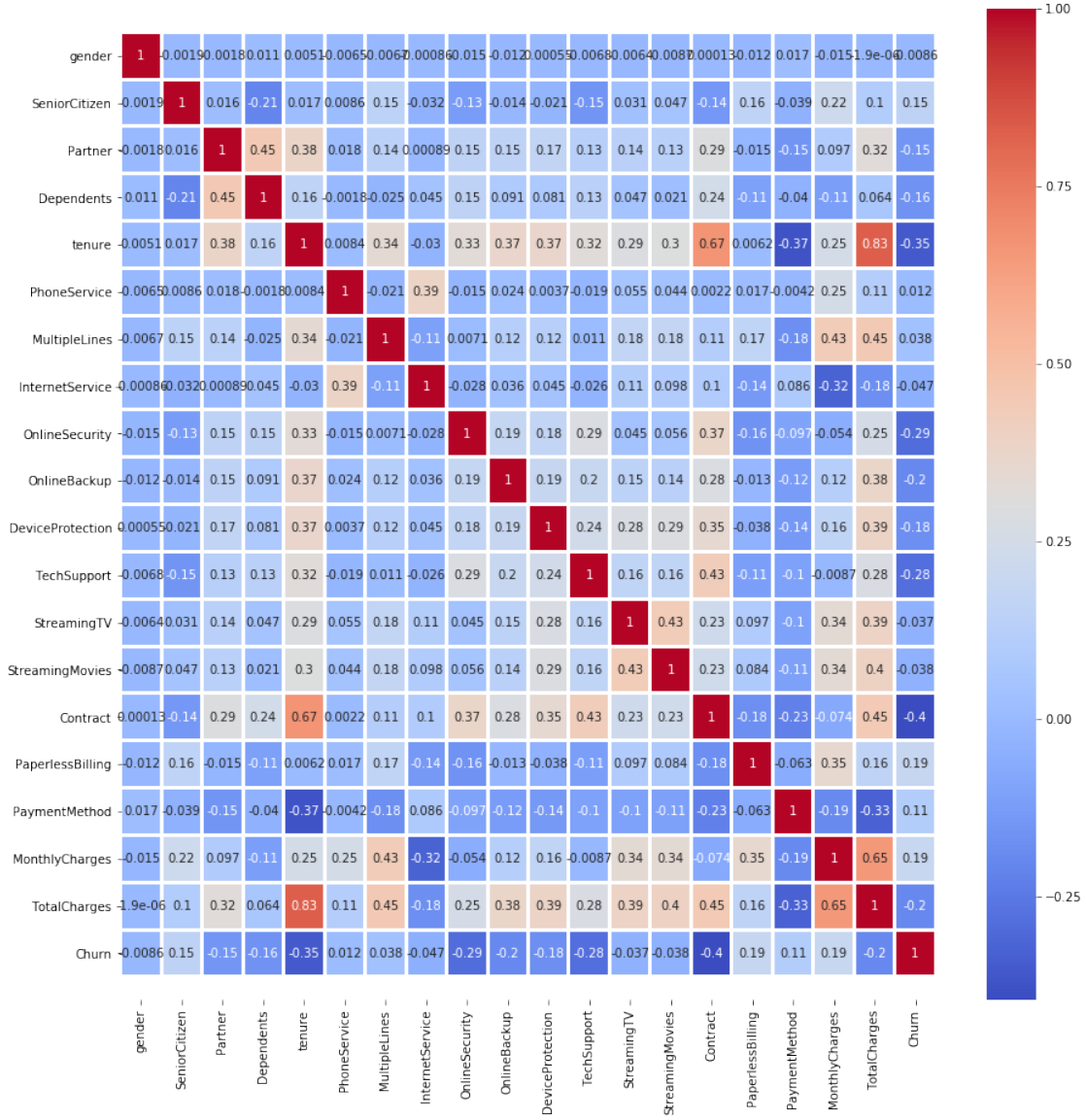
[44]:
```
for i,col in enumerate(num):
    churn_yes = df[df['Churn']=='Yes']
    churn_no = df[df['Churn']=='No']
    yes = churn_yes[col].values
    no = churn_no[col].values
    if abs(yes.var() - no.var())>10:
        print(f"Variance of two samples of -{col}- are un-equal. Performing␣
↪Welsh's test in place of t-test")
        score,p = ttest_ind(yes,no,equal_var=False)
    else:
        print(f"Variance of two samples of -{col}- are equal. Performing␣
↪t-test")
        score,p = ttest_ind(yes,no,equal_var=True)

    if p<=0.01:
        print(f"We have a score of {score} and p value of {p} shows that means␣
↪of two samples are significantly different\n{'*'*20}")
    else:
        print(f"We have a score of {score} and p value of {p} shows that means␣
↪of two samples are equal\n{'*'*20}")
```

```
Variance of two samples of -MonthlyCharges- are un-equal. Performing Welsh's
test in place of t-test
We have a score of 18.407526676414673 and p value of 8.592449331547539e-73 shows
that means of two samples are significantly different
**********
Variance of two samples of -TotalCharges- are un-equal. Performing Welsh's test
in place of t-test
We have a score of -18.767950751475944 and p value of 2.0590752866918164e-75
shows that means of two samples are significantly different
**********
Variance of two samples of -tenure- are un-equal. Performing Welsh's test in
place of t-test
We have a score of -34.82381869631297 and p value of 1.1954945472607148e-232
shows that means of two samples are significantly different
**********
```

[53]:
```
from sklearn.preprocessing import LabelEncoder
for i,col in enumerate(cat+['Churn']):
    df[col] = LabelEncoder().fit_transform(df[col])

eda.show_corr(df,size=(15,15))
```



Correlation Shows us that Contract,tenure,OnlineSecurity,TechSupport and Total Charges are the ones in order that affect the Churn Mostly with an absolute correlation value of more than 0.20

0.1 Results

If you want customers to stop leaving your services,

1. Offer them a tenure of 30+
2. Try to persuade them to have 2 year plan
3. Keep the Monthly Charges low under 60
4. Offer them Device PRotection, Online Security, Online Backup and Tech Support
5. Company has good Internet Service `(DSL)` but `Fiber Optic` is either bad or too costly so

people are leaving more than DSL. (Reflected by 2 facts that Streaming requies high speed and people are Churning more when they Stream Movies and people with Fiber Optics are Churning at a high proportion)

6. Phone Service is bad so try to improve the Telephone Services

[ ]: