CHURN PREDICTION

K184060744 - Nguyễn Thị Hồng Nhung

Input libraries

In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from pylab import rcParams
%matplotlib inline

from sklearn.preprocessing import MinMaxScaler
#from IPython.display import Image
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

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

Exploratory Data Analysis

In [3]:
```python
df.head(3)
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServi
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	D
1	5575-GNVDE	Male	0	No	No	34	Yes	No	D
2	3668-QPYBK	Male	0	No	No	2	Yes	No	D

3 rows × 21 columns

In [4]:
```python
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      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

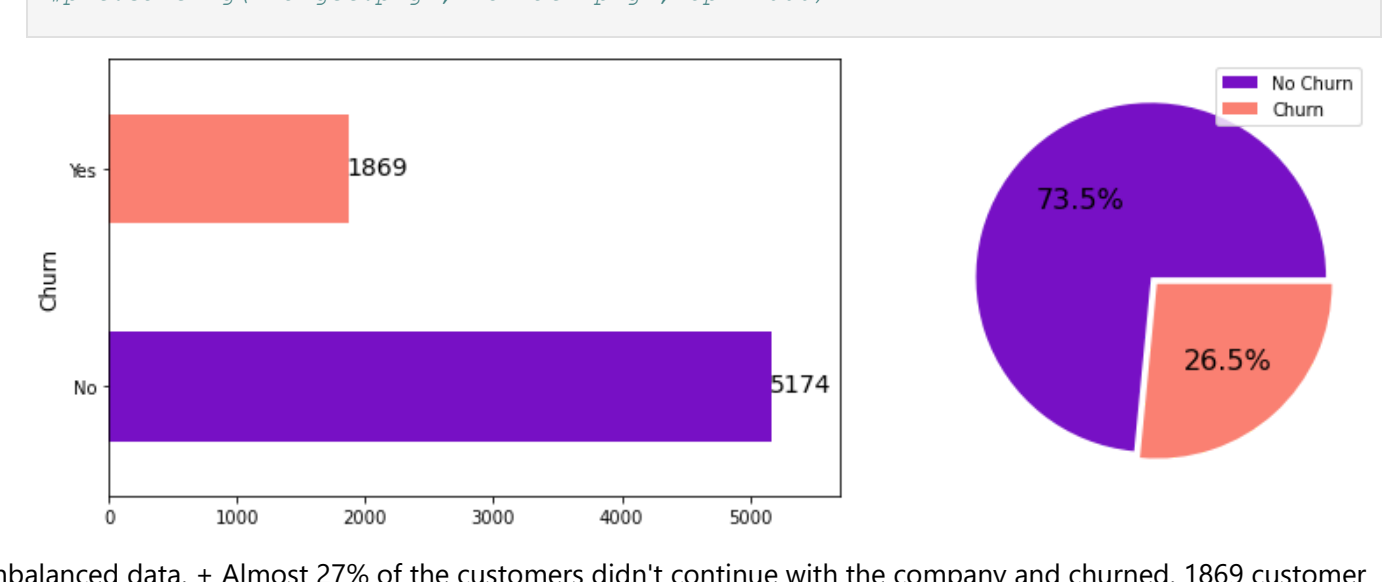
Basic data cleaning

In [5]:
```python
df['SeniorCitizen'] = df['SeniorCitizen'].apply(str)

#Totalcharges is given as object datatype but it is float datatype
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
```

Visualization

Target variable

In [6]:
```python
colors = ['#7710c5','salmon','#FFFDD0','#b1ede8','#6d435a']

fig,ax = plt.subplots(1,2,figsize = (12,4))

df['Churn'].value_counts().plot(kind='barh', color =colors, ax=ax[0])
for p in ax[0].patches:
    ax[0].text(p.get_width() + 230,
            p.get_y() + 0.2,
            p.get_width(),
            color='black',
            fontsize=14,
            ha='center',
            va='bottom')
ax[0].set_ylabel('Churn',fontsize=12)
ax[0].set_xlim(0,5700)


#pie chart
df['Churn'].value_counts().plot(kind='pie',labels = None, colors = colors,
                                 autopct='%.1f%%',  explode = [0,0.05], textprops = {":
plt.ylabel("")
plt.legend(labels=['No Churn', 'Churn'])
plt.tight_layout()
#plt.savefig('Target.png', format='png', dpi=1000)
```



We have imbalanced data. + Almost 27% of the customers didn't continue with the company and churned. 1869 customer churned. + Almost 73% of the customers continue with the company and didn't churn. 5174 customer didn't churn.

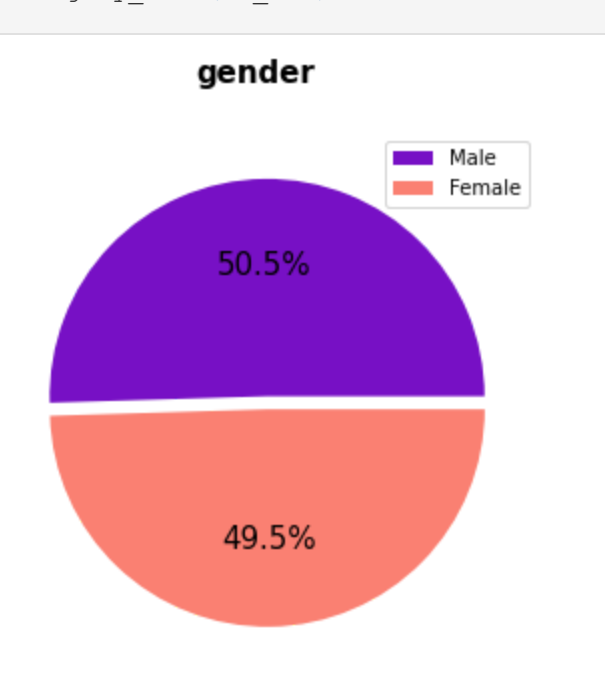
Categorial Features

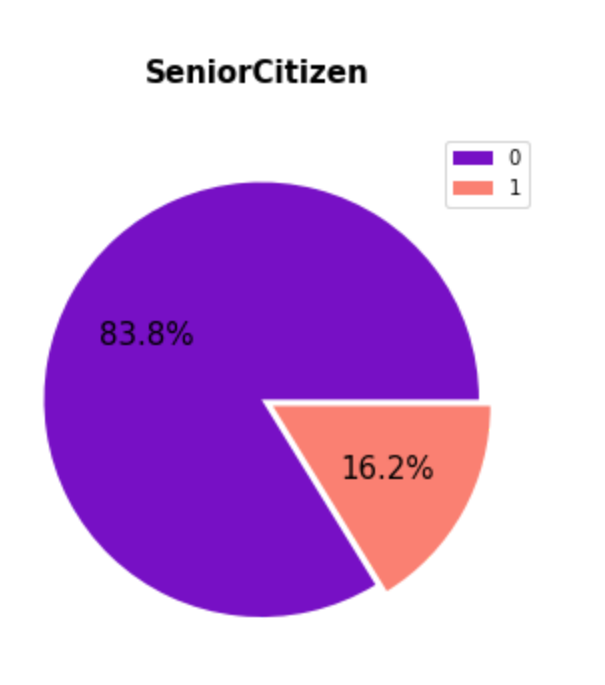
In [7]:
```python
df_cat = df.select_dtypes(include='object').copy()
df_cat.columns
```

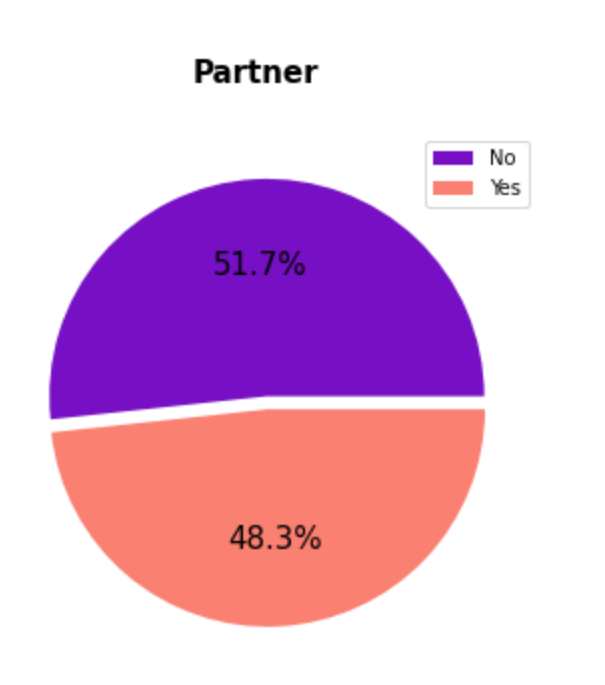
Out[7]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
       'Churn'],
      dtype='object')
```

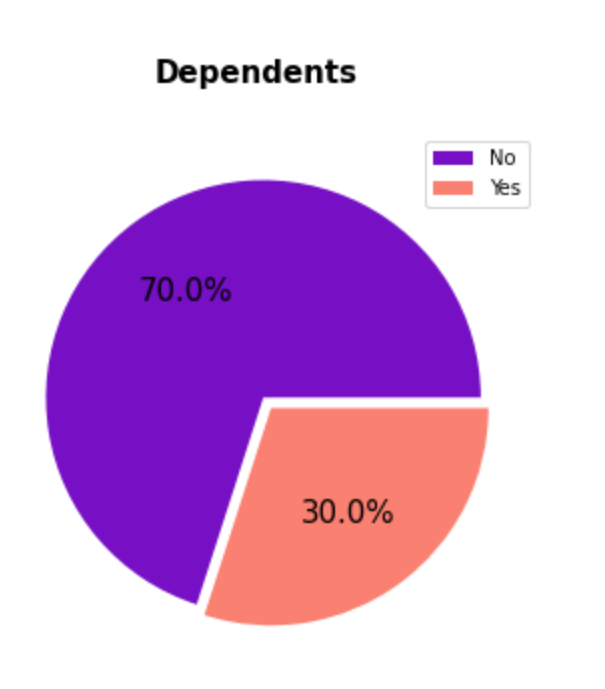
In [8]:
```python
#Create func to visualize categorical feature
def category_dist(df):
    X = df.columns

    for j in range(1,len(X)-1):
        plt.figure(figsize=(6,5))
        counts = df[X[j]].value_counts()
        percent = df[X[j]].value_counts(normalize=True).mul(100).round(1).astype(str)
        table=pd.DataFrame({'Counts': counts, 'Percent': percent})
        #print(table)

    #Draw plot
        i = counts.index
        counts.plot.pie(autopct='%.1f%%', labels = None, colors = colors, explode = [(
    #pie chart
        plt.ylabel("")
        plt.legend(labels=i)
        plt.suptitle (df[X[j]].name,weight = 'bold', fontsize = 15)
        #plt.savefig(df[X[j]].name +'.png', format='png', dpi=1000)
```

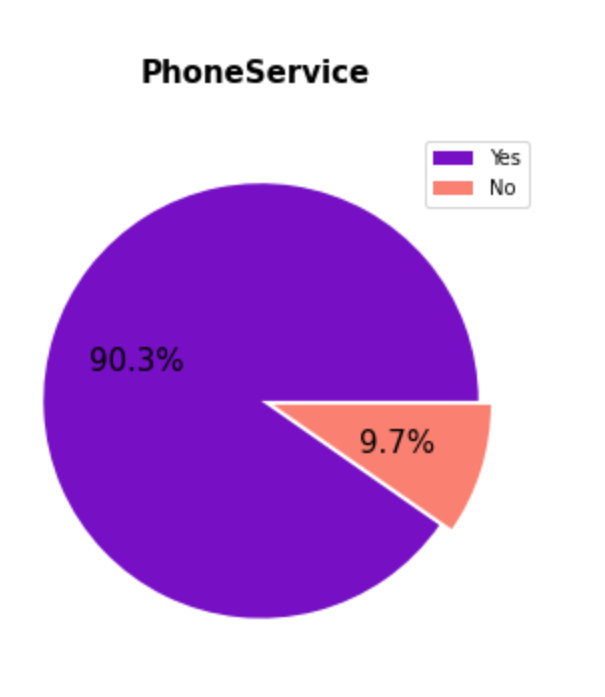
In [9]:
```python
category_dist(df_cat)
```

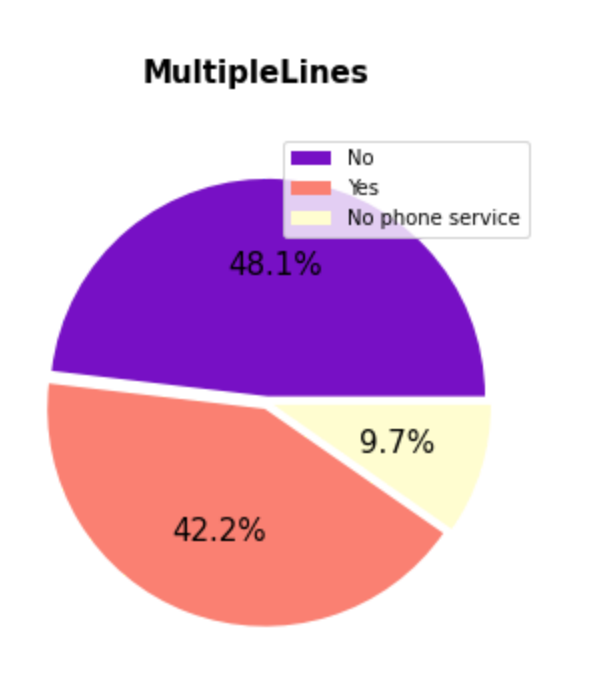


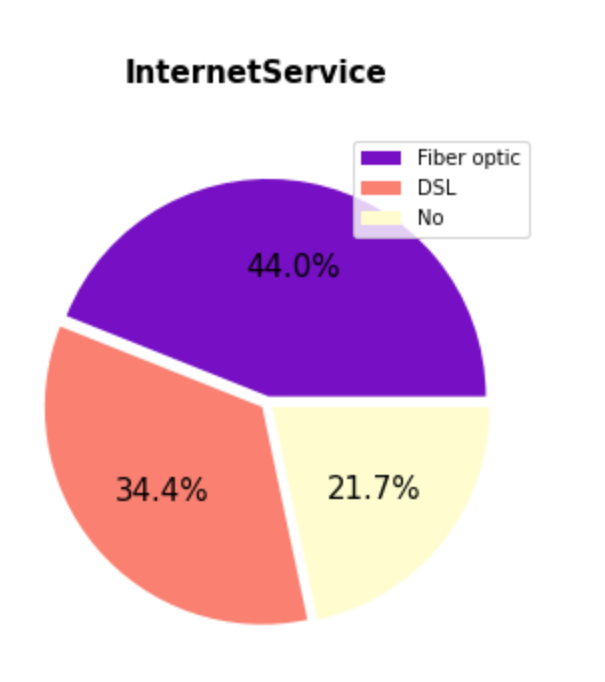


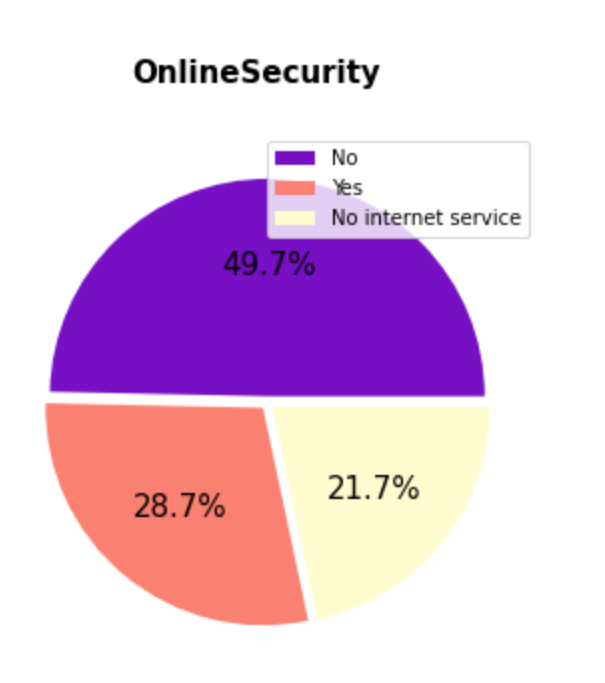


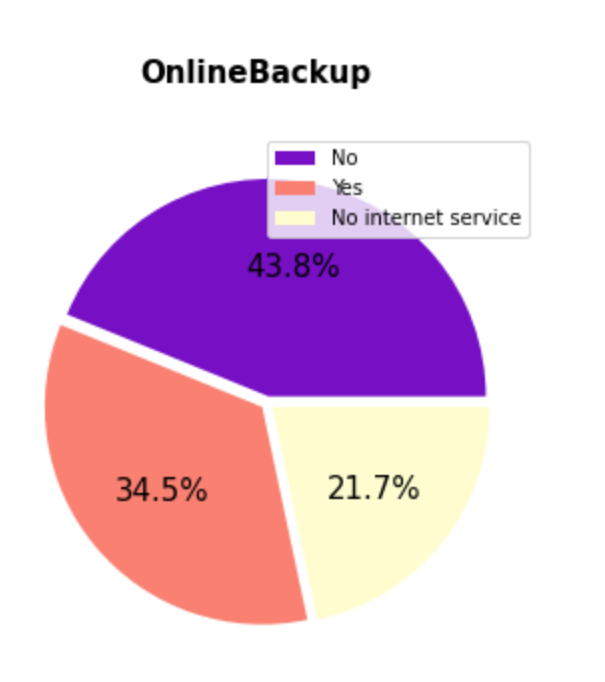


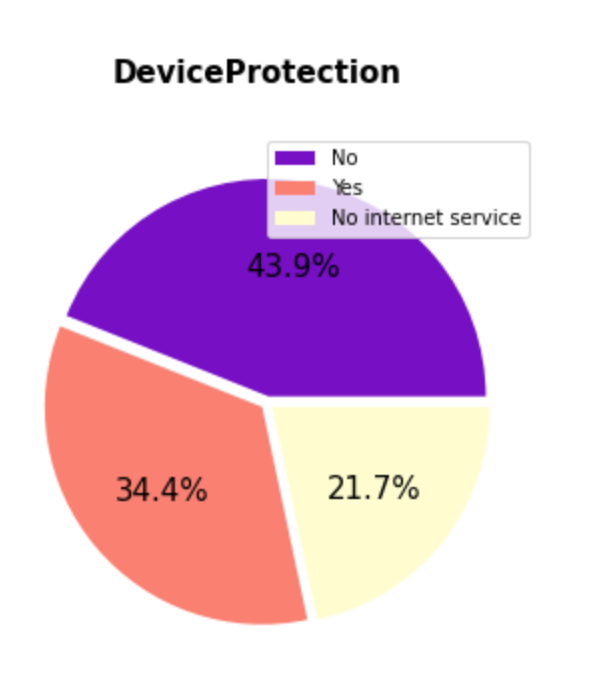


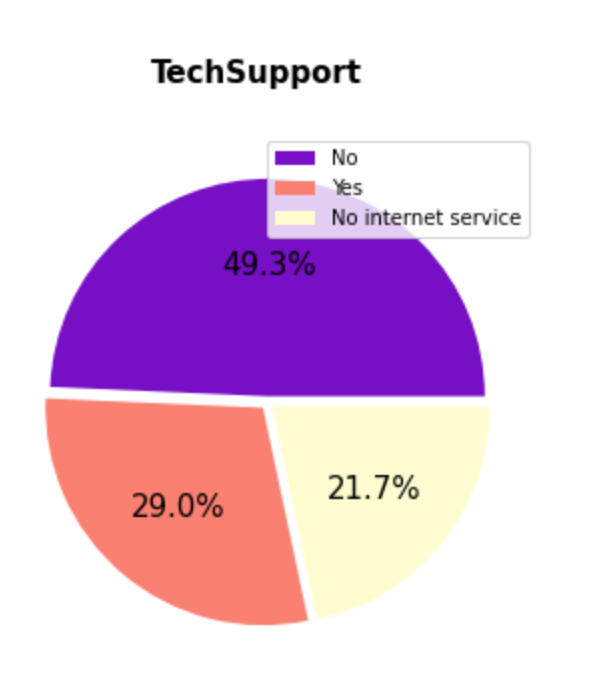


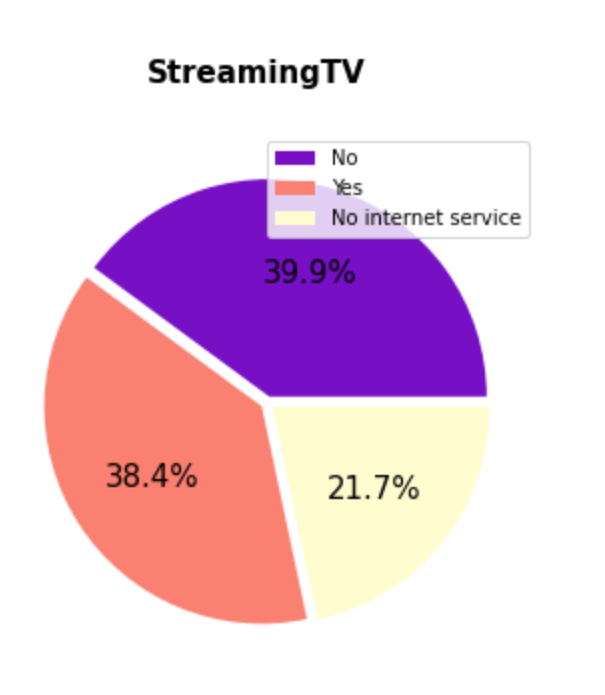


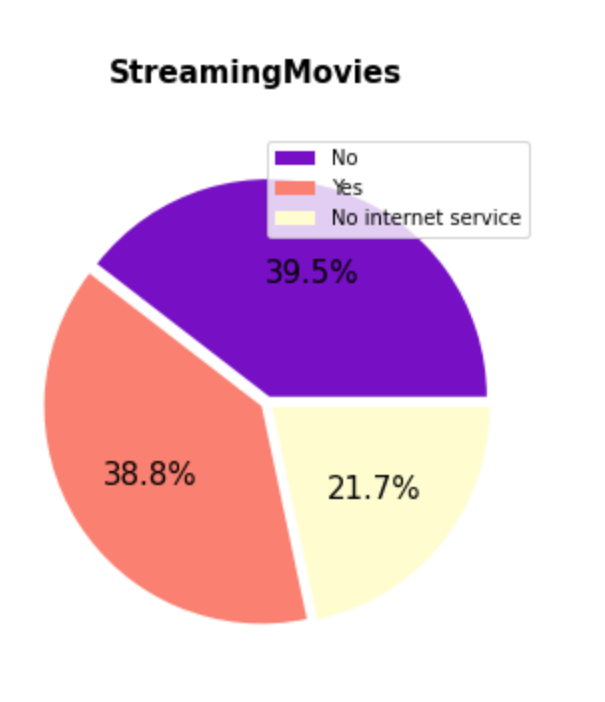


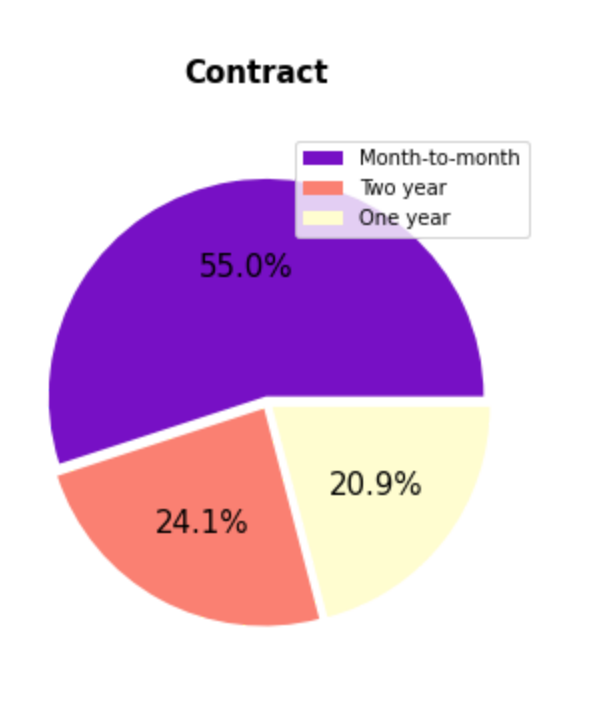


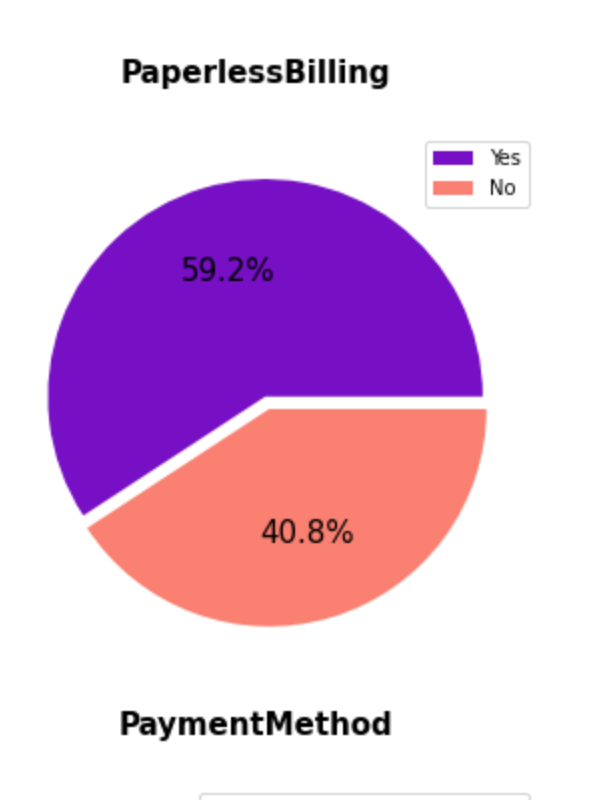


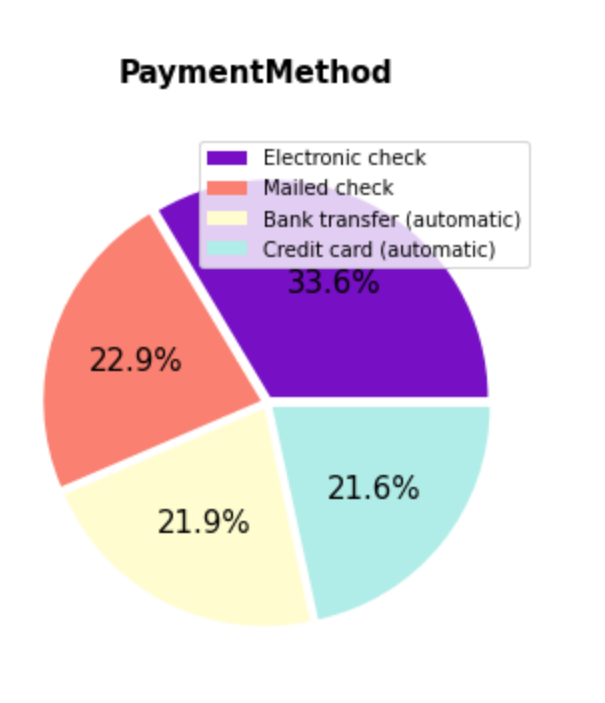








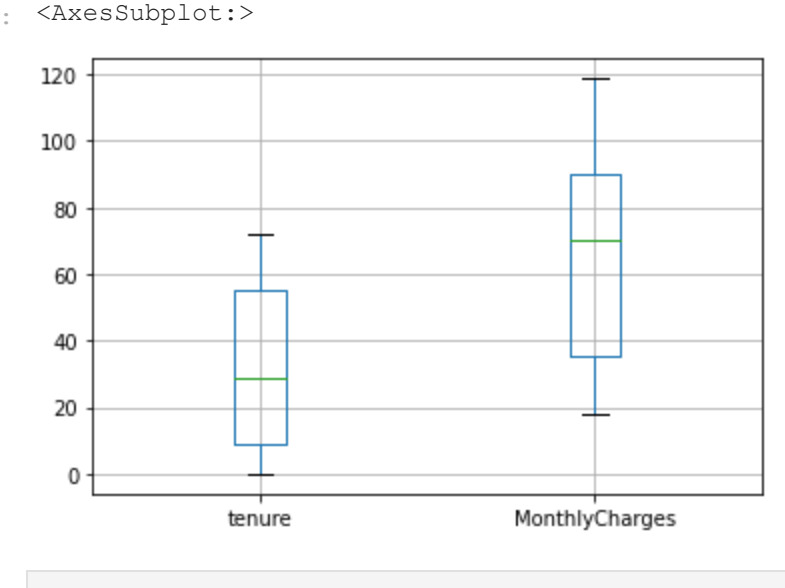




Numerical variables

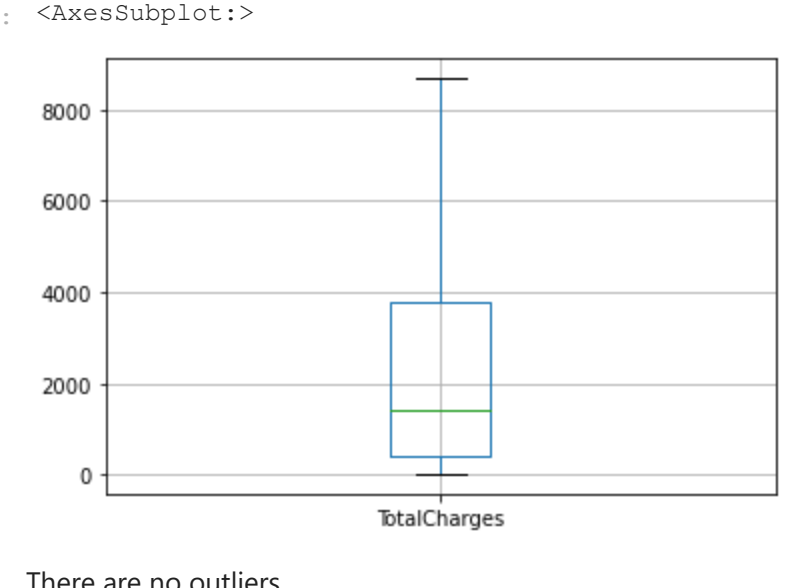
In [10]:
```python
num_cols = ['tenure','MonthlyCharges']
df[num_cols].boxplot()
#plt.savefig('Boxplot1.png',format='png', dpi=1000)
```

Out[10]: <AxesSubplot:>



In [11]:
```python
df.boxplot(column='TotalCharges')
#plt.savefig('Boxplot2.png',format='png', dpi=1000)
```

Out[11]: <AxesSubplot:>



There are no outliers

In [12]:
```python
def distplot(feature):
    plt.figure(figsize=(6,3))
    plt.title("Distribution for {}".format(feature),weight = 'bold', fontsize = 15)
    ax = sns.distplot(df[feature])
    plt.xlabel('')
    plt.ylabel('')
    plt.tick_params(axis='x',labelsize=13)
    plt.yticks(color='w')
    #plt.savefig(feature + 'Dist.png', format='png',dpi=1000)

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

In [13]:
```python
df[numerical_cols].describe().T
```

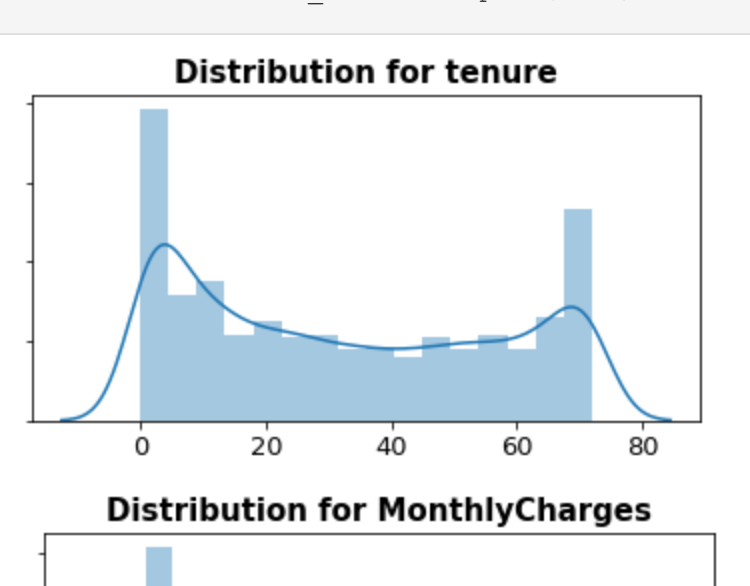
Out[13]:

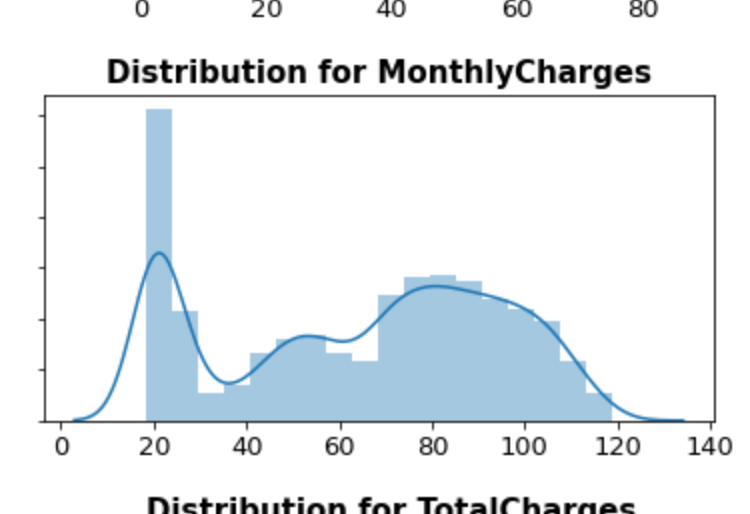
	count	mean	std	min	25%	50%	75%	max
tenure	7043.0	32.371149	24.559481	0.00	9.00	29.000	55.0000	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.50	70.350	89.8500	118.75
TotalCharges	7032.0	2283.300441	2266.771362	18.80	401.45	1397.475	3794.7375	8684.80

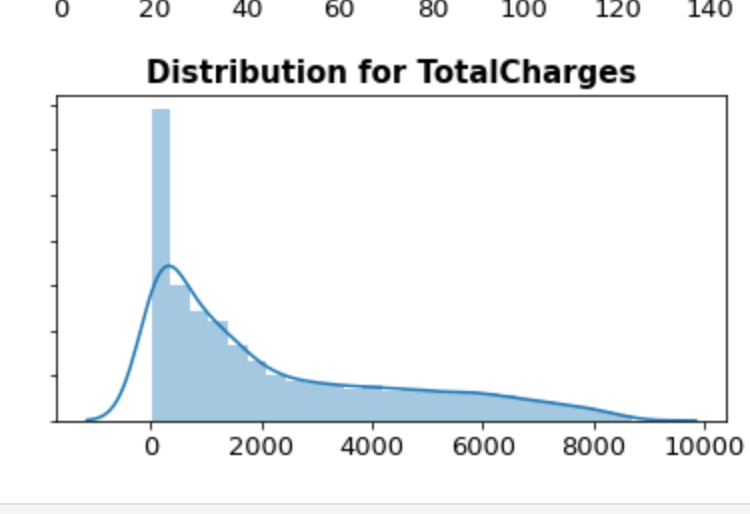
In [14]:
```python
df[numerical_cols].skew()
```

Out[14]:
```
tenure            0.239540
MonthlyCharges   -0.220524
TotalCharges      0.961642
dtype: float64
```

In [15]:
```python
for feat in numerical_cols: distplot(feat)
```







In [16]:
```python
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)
```

In [17]:
```python
df['tenure']=df['tenure'].replace(0,df['tenure'].median())
```

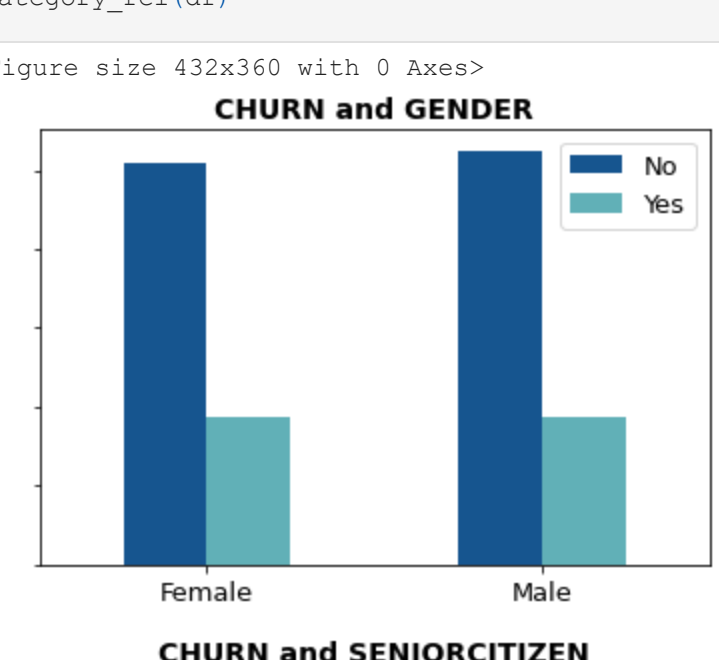
In [18]:
```python
df.info()
```

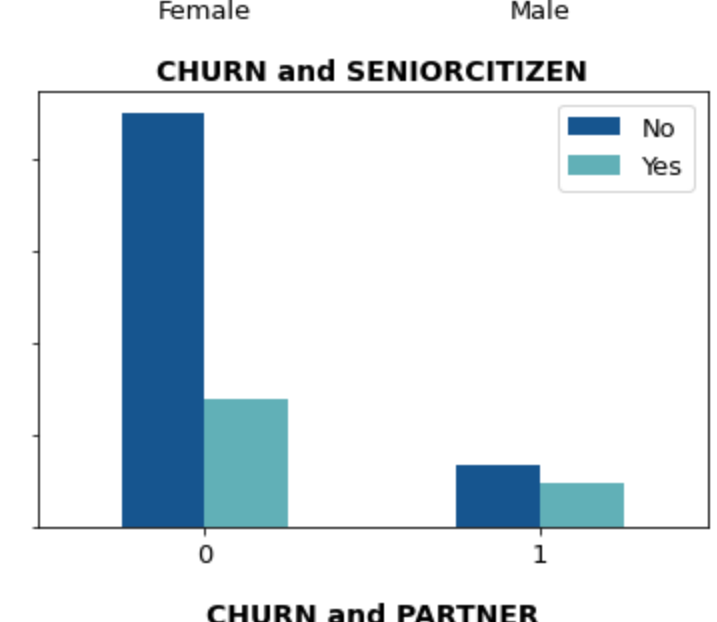
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   object 
 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      7043 non-null   float64
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(1), object(18)
memory usage: 1.1+ MB
```

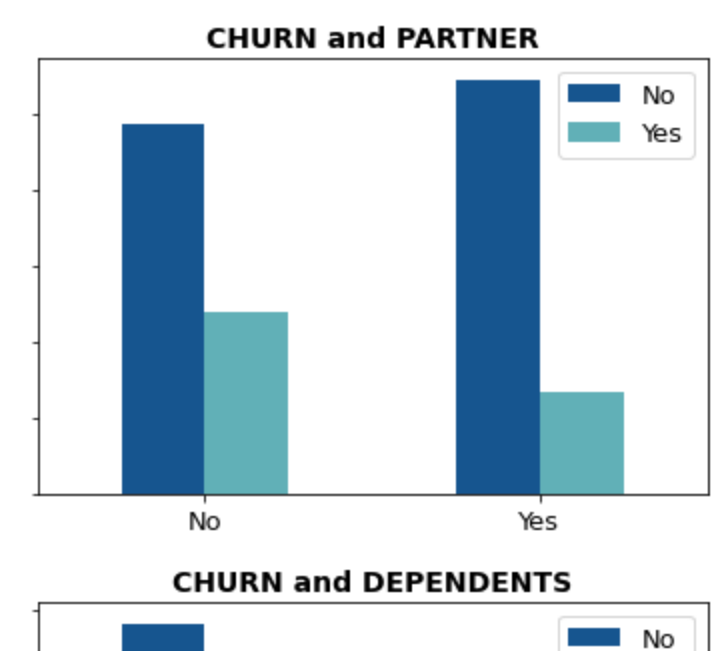
Dependent and Independent variables

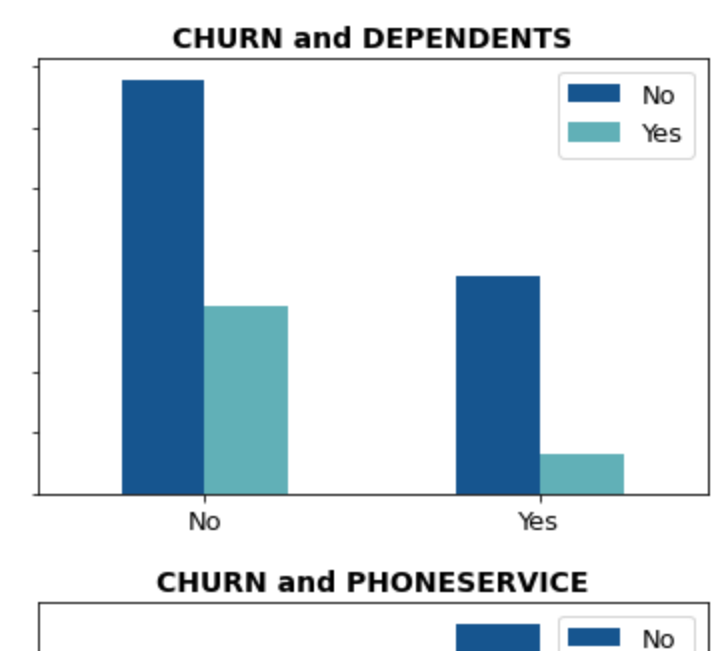
In [19]:
```python
def category_rel(df):
    plt.figure(figsize=(6,5))
    X = df.select_dtypes(include = ['object']).columns
    #print(X)
    for i in range(1,len(X)-1):
        tab_values=pd.crosstab(df[X[i]],df.iloc[:,20])
        tab_values.plot(kind='bar', color=['#16558F','#61B0B7','#0583D2','#B8E3FF'],st

        plt.xlabel('')
        plt.tick_params(axis='x', rotation=0,labelsize=13)
        plt.yticks(color='w')

        #sns.barplot(tab)
                #axs[0].set_title('Actual counts for CHURN output versus '+X[i].upper
                #axs[0] = axs[0].facet_axis(0,0)
                #for p in axs[0].patches:
                  #  axs[0].text(p.get_x() + p.get_width()/2,
                  #  p.get_height() + 0.5 ,
                  #  p.get_height(),
                  #  color='black',
                   # ha='center',
                  # va='bottom')
        plt.legend(fontsize=13)
        plt.title("CHURN and "+(X[i]).upper(),weight = 'bold', fontsize = 14)
        #plt.tight_layout()
        #plt.savefig(X[i] + 'Corr.png', format='png', dpi=1000)
        plt.show()

category_rel(df)
```

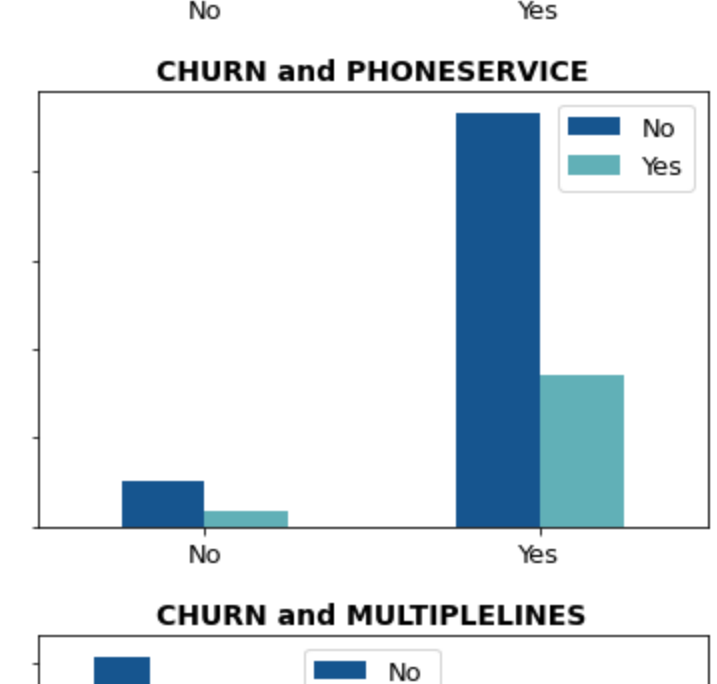
<Figure size 432x360 with 0 Axes>

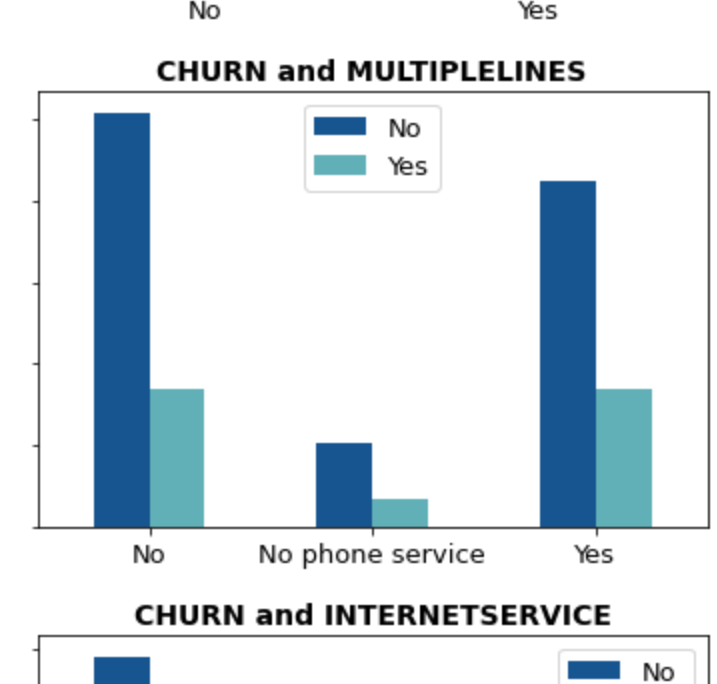


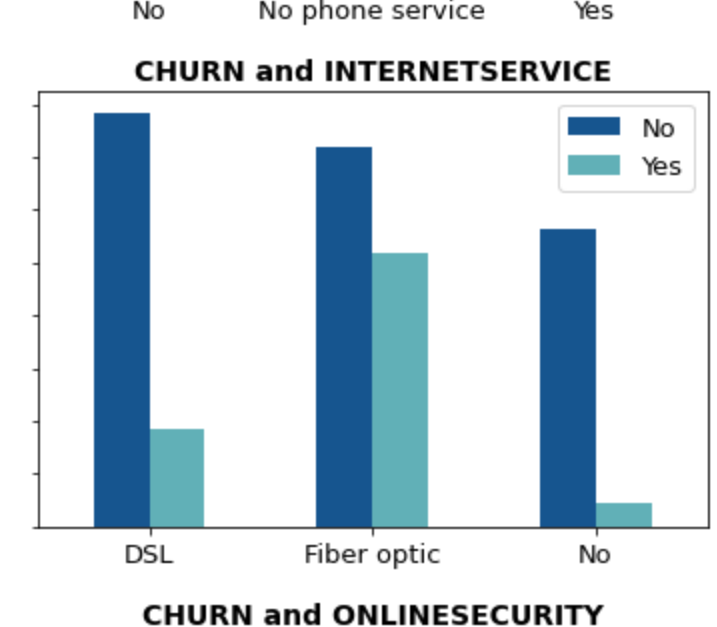


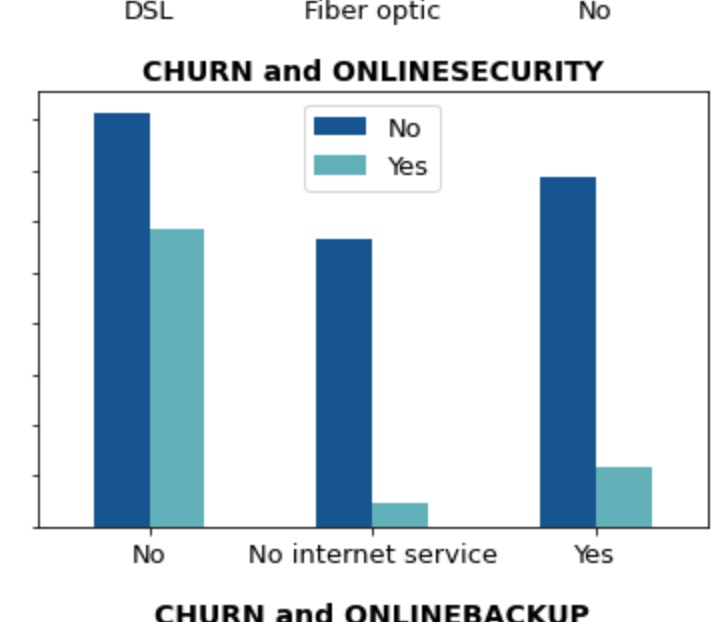


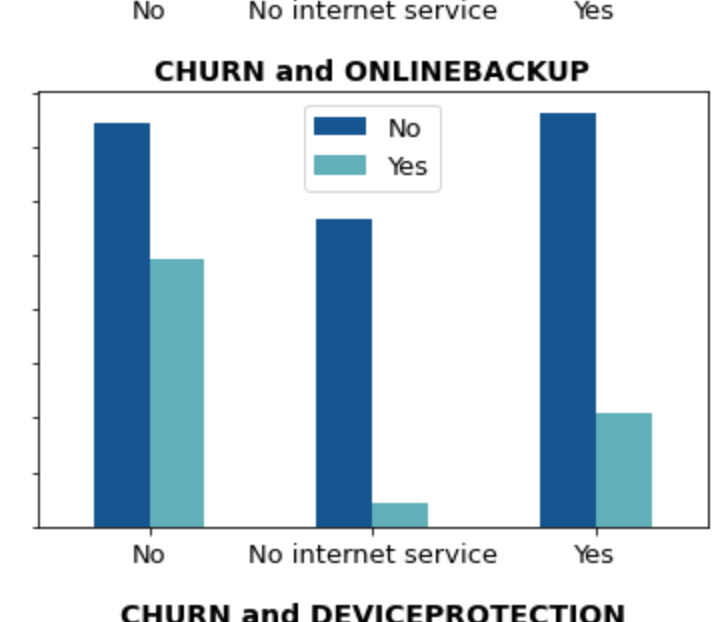


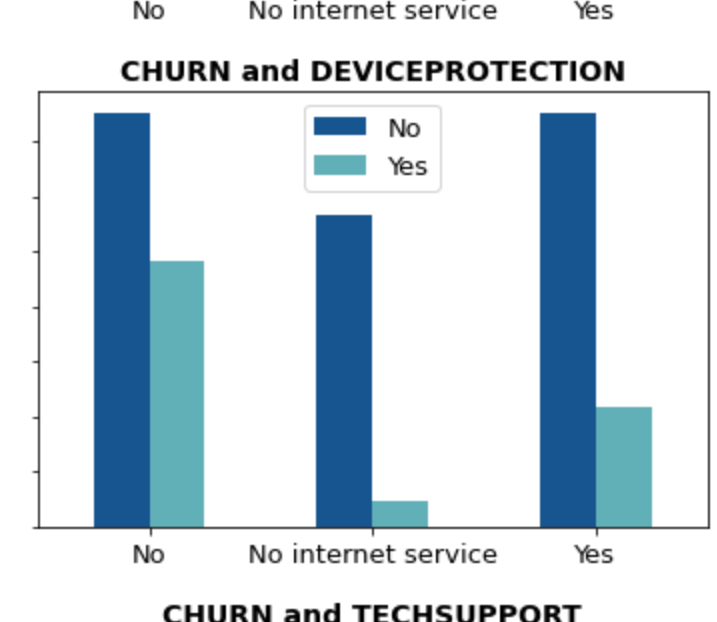


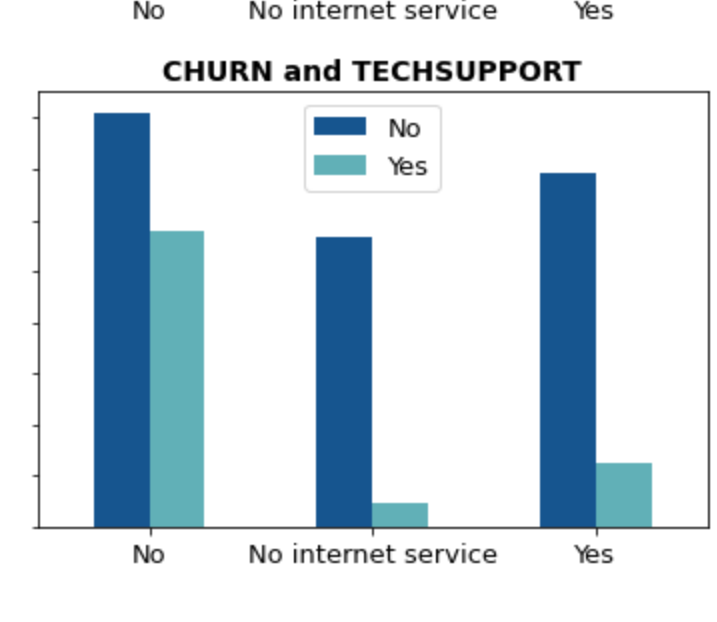


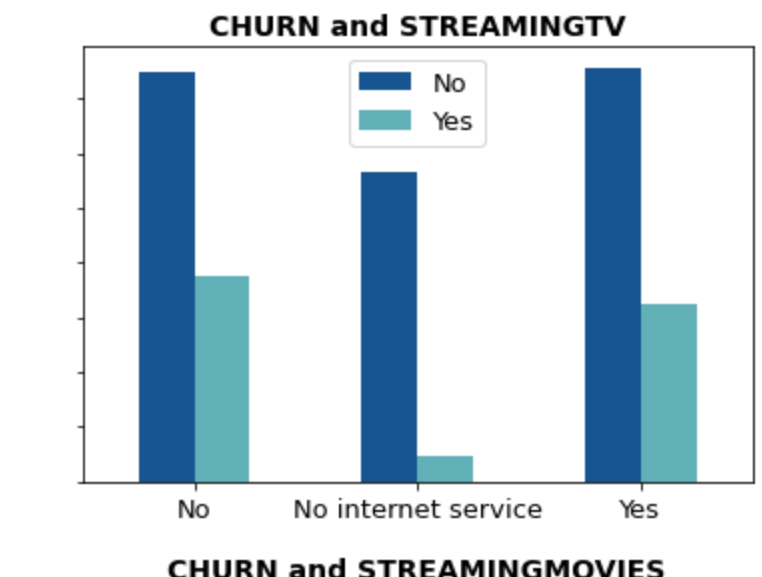


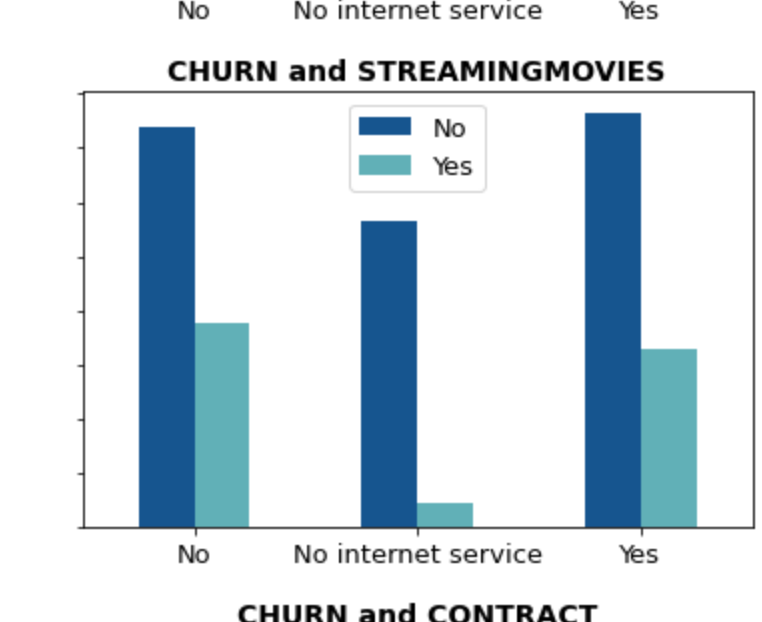


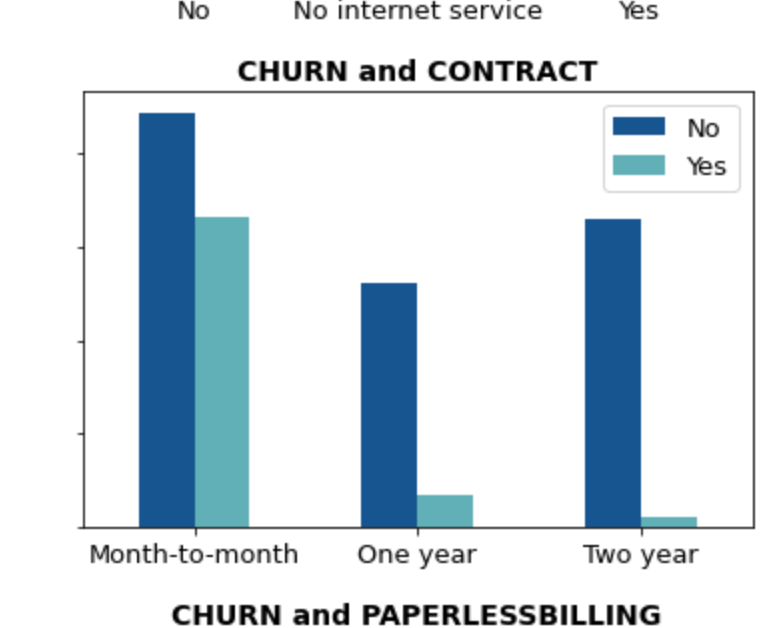


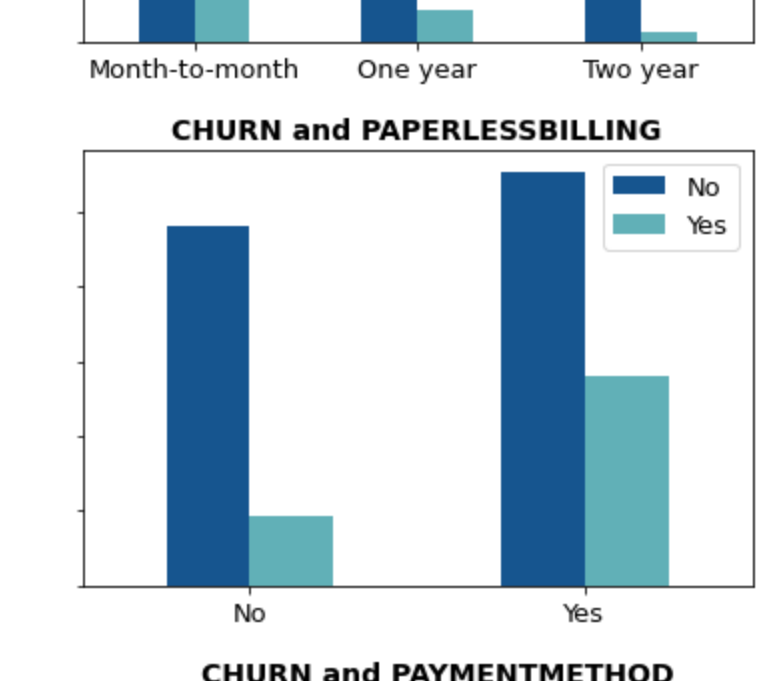


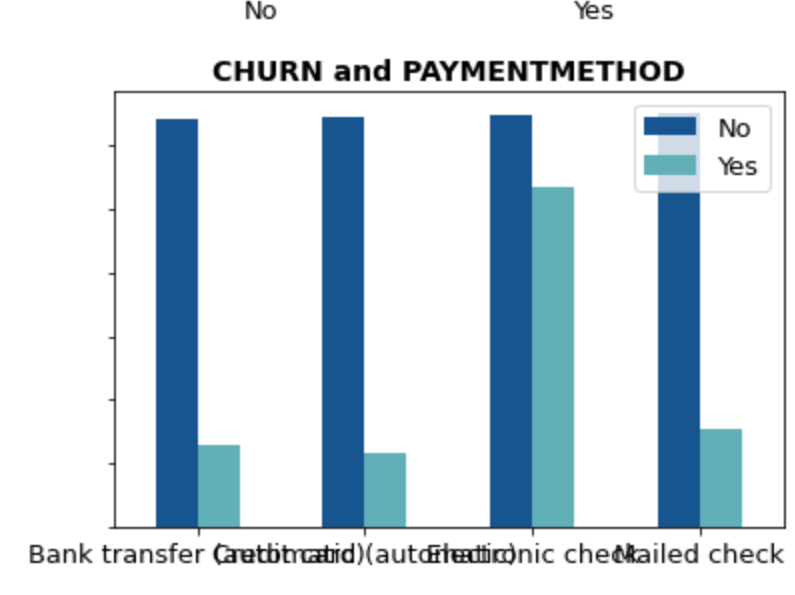












plt.figure(figsize=(6,5)) tab_values=pd.crosstab(df['PaymentMethod'],df['Churn']) #Creating % of total values cross tab by using 'normalize=True' in normal tab #print(tab_values) tab_values.plot(kind='bar', color= ['#16558F','#61B0B7','#0583D2','#B8E3FF'],stacked=False) label =['Bank Transfer', 'Credit Card', 'Electronic Check', 'Mail Check'] plt.xlabel('') plt.tick_params(axis='x', rotation=0, labelsize=12) plt.xticks([0,1,2,3],label) plt.yticks(color='w') plt.legend(fontsize=13) plt.title("CHURN and PAYMENT METHOD",weight = 'bold', fontsize = 14) #plt.tight_layout() #plt.savefig('PaymentMethodCorr.png', format='png', dpi=1000) plt.show()

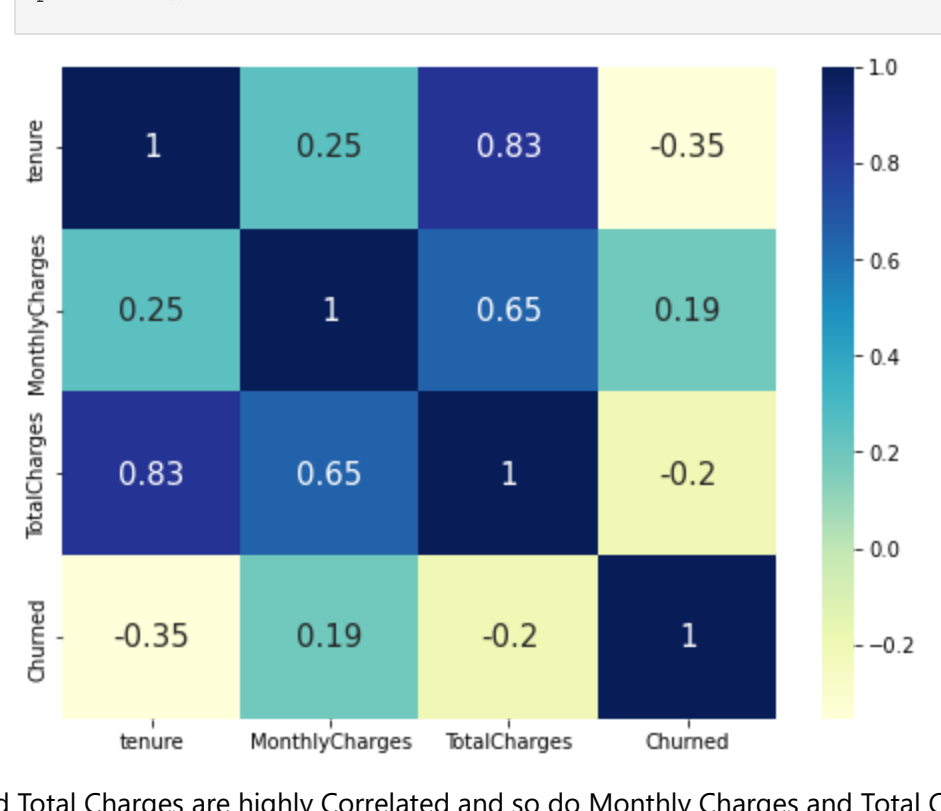
In [20]:
```
df_num = df[['tenure','MonthlyCharges','TotalCharges','Churn']].copy()
df_num
```

Out[20]:

	tenure	MonthlyCharges	TotalCharges	Churn
0	1	29.85	29.85	No
1	34	56.95	1889.50	No
2	2	53.85	108.15	Yes
3	45	42.30	1840.75	No
4	2	70.70	151.65	Yes
...	...	...	...	...
7038	24	84.80	1990.50	No
7039	72	103.20	7362.90	No
7040	11	29.60	346.45	No
7041	4	74.40	306.60	Yes
7042	66	105.65	6844.50	No

7043 rows × 4 columns

In [21]:
```
#Convert Churn into numerical values of 0 and 1 to get correlation with other numerica
churn = pd.get_dummies(df_num["Churn"],drop_first=True)
churn.rename(columns = {'Yes':'Churned'}, inplace = True)
df_num=pd.concat([df_num,churn],axis=1)

corrMatrix=df_num.corr()

#Draw heatmap
fig=plt.subplots(figsize=(8,6))
sns.heatmap(corrMatrix, annot=True, cmap ='YlGnBu', annot_kws={"size": 15})
plt.savefig('NumericalCorr.png', format='png', dpi=1000)
#plt.title('Numerical Attributes Correlation Matrix')
plt.show()
```



1.Tenure and Total Charges are highly Correlated and so do Monthly Charges and Total Charges 2. Week correlation exists between tenure and Monthly Charges Both of the remaining variables are not highly correlated. Therefore I will drop Total Charges feature and keep Tenure and Monnthlycharge, thus now only tenure and Monthly Charges are left as Numeric variables.

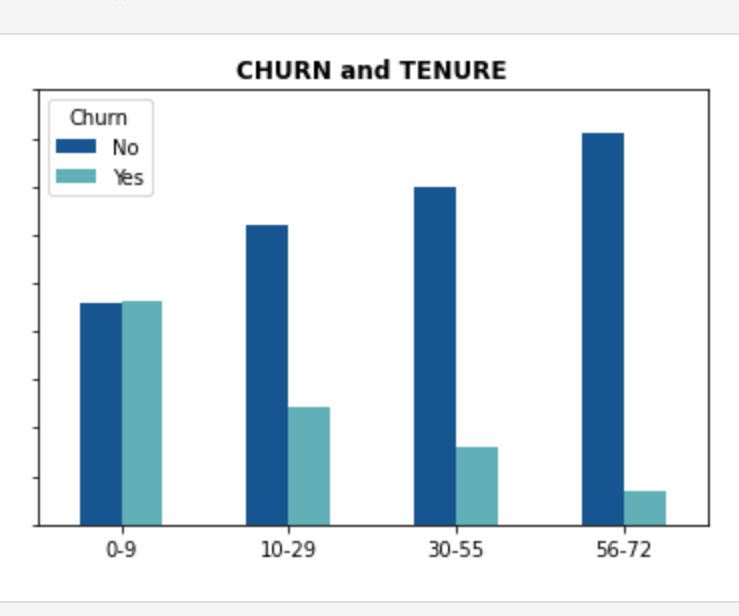
In [22]:
```
df_num['tenure'].describe()
```

Out[22]:
```
count    7043.000000
mean       32.416442
std        24.526454
min         1.000000
25%         9.000000
50%        29.000000
75%        55.000000
max        72.000000
Name: tenure, dtype: float64
```

In [23]:
```
# Divide the tenure basket
df_num['quantile_tenure'] = pd.qcut(df_num['tenure'], q = 4)
df_num['quantile_tenure'].value_counts().sort_values(ascending=True)
```

Out[23]:
```
(29.0, 55.0]     1719
(9.0, 29.0]      1726
(55.0, 72.0]     1755
(0.999, 9.0]     1843
Name: quantile_tenure, dtype: int64
```

In [24]:
```
def bins(x):
    if x>0 and x<=9:
        return '0-9'
    elif x>=10 and x<=29:
        return '10-29'
    elif x>=30 and x<=55:
        return '30-55'
    #elif x>=41 and x<=60:
    #    return '31-40'
    elif x>=56 and x<=72:
        return '56-72'

df_num['tenure_bins']=df_num['tenure'].map(bins)
#print(df[['tenure_bins','tenure']])

x = (pd.crosstab(df_num['tenure_bins'],df_num['Churn']))

ax = x.plot(kind='bar',color=['#16558F','#61B0B7','#0583D2','#B8E3FF'], stacked=False)
#for p in ax.patches:
#    ax.text(p.get_x() + p.get_width()/2,
#    p.get_height() + 0.5 ,
#    p.get_height(),
#    color='black',
#    ha='center',
#    va='bottom')
plt.xlabel('')
plt.ylabel('')
plt.yticks(color='w')
plt.tick_params(axis='x', rotation=0)

plt.ylim(top=1800)
plt.title("CHURN and TENURE",weight = 'bold')
#plt.savefig('tenure-churn.png',format='png',dpi=1000)
plt.show()
```



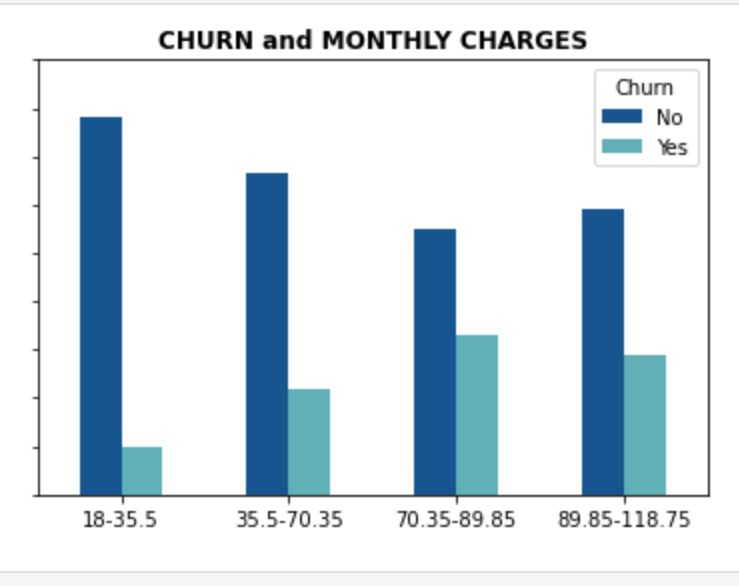
In [25]:
```
df_num['MonthlyCharges'].describe()
```

Out[25]:
```
count    7043.000000
mean       64.761692
std        30.090047
min        18.250000
25%        35.500000
50%        70.350000
75%        89.850000
max       118.750000
Name: MonthlyCharges, dtype: float64
```

In [26]:
```
# Divide the MonthlyCharges basket

df_num['quantile_MonthlyCharges'] = pd.qcut(df_num['MonthlyCharges'], q = 4)
df_num['quantile_MonthlyCharges'].value_counts().sort_values(ascending=True)
```

Out[26]:
```
(70.35, 89.85]     1757
(89.85, 118.75]    1758
(18.249, 35.5]     1762
(35.5, 70.35]      1766
Name: quantile_MonthlyCharges, dtype: int64
```

In [27]:
```
def MCbins(x):
    if x>18 and x<=35.5:
        return '18-35.5'
    elif x>35.5 and x<=70.35:
        return '35.5-70.35'
    #elif x>=21 and x<=40:
    #   return '21-40'
    elif x>70.35 and x<=89.85:
        return '70.35-89.85'
    elif x>89.85 and x<=118.75:
        return '89.85-118.75'
df_num['MonthlyCharges_bins']=df_num['MonthlyCharges'].map(MCbins)
#print(df[['tenure_bins','tenure']])

x=(pd.crosstab(df_num['MonthlyCharges_bins'],df_num['Churn']))

ax = x.plot(kind='bar', color=['#16558F','#61B0B7','#0583D2','#B8E3FF'], stacked=False
#for p in ax.patches:
#        ax.text(p.get_x() + p.get_width()/2,
#        p.get_height() + 0.5 ,
#        p.get_height(),
#        color='black',
#        ha='center',
#        va='bottom')
plt.xlabel('')
plt.ylabel('')
plt.yticks(color='w')
plt.tick_params(axis='x', rotation=0)

plt.ylim(top=1800)
plt.title("CHURN and MONTHLY CHARGES",weight = 'bold')
#plt.savefig('monthlycharge-churn.png',format='png',dpi=1000)
plt.show()
```



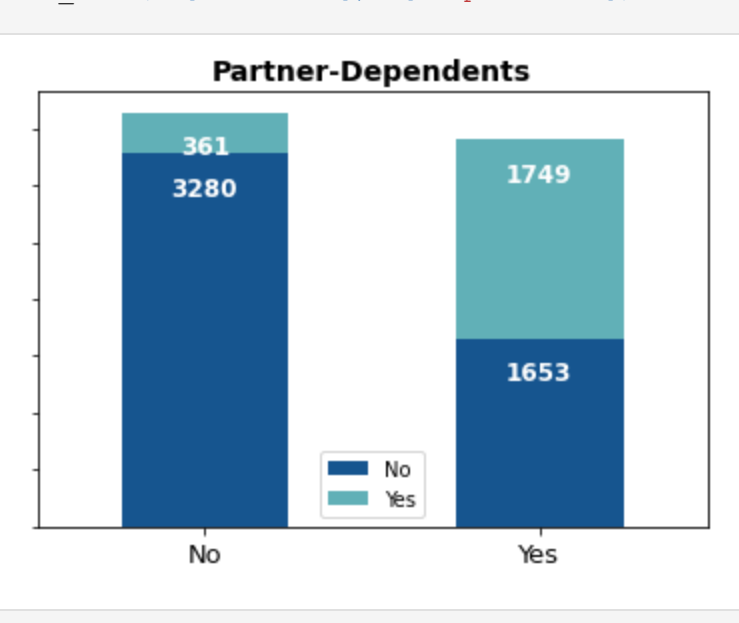
In [28]:
```
df = df.drop(columns=['customerID', 'TotalCharges','gender'])
```

Data Tranformation

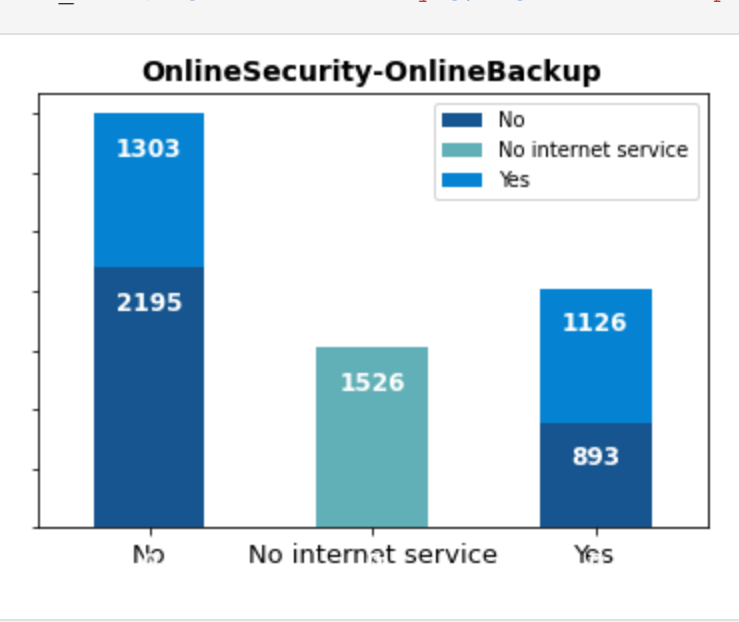
Data aggregation

In [29]:
```
def check_dist(X,Y):
    tab_values=pd.crosstab(X,Y)
    #print(tab_values)
    ax = tab_values.plot(kind='bar', color=['#16558F','#61B0B7','#0583D2','#B8E3FF'],s
    plt.tick_params(axis='x', rotation=0,labelsize=13)
    plt.xlabel('')
    for bar in ax.patches:

        ax.text(
      # Put the text in the middle of each bar. get_x returns the start
      # so we add half the width to get to the middle.
          bar.get_x() + bar.get_width() / 2,
      # Vertically, add the height of the bar to the start of the bar,
      # along with the offset.
          bar.get_height() + bar.get_y()-361,
      # This is actual value we'll show.
          round(bar.get_height()),
      # Center the labels and style them a bit.
          ha='center',
          color='w',
          weight='bold',
          size=12
  )
    plt.yticks(color='w')

    figname = X.name + "-" + Y.name
    plt.title(figname ,weight = 'bold', fontsize = 14)
    #plt.savefig( figname + 'Checkdist.png',format='png',dpi=1000)
    ax.legend()
```

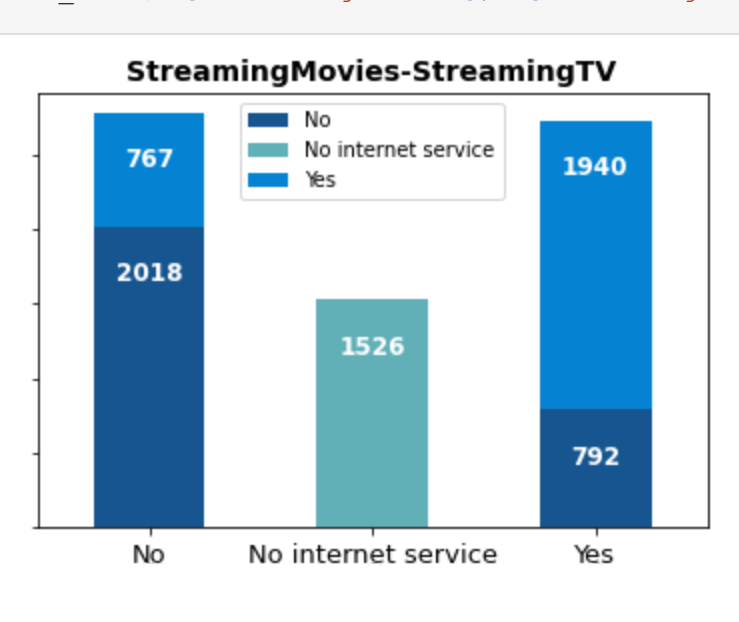
In [30]:
```
check_dist(df['Partner'],df['Dependents'])
```



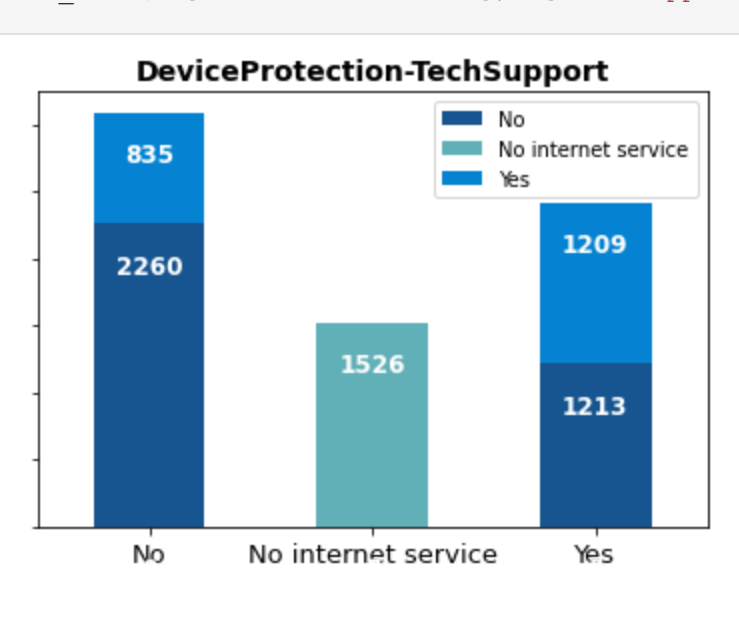
In [31]:
```
check_dist(df['OnlineSecurity'],df['OnlineBackup'])
```



In [32]:
```
check_dist(df['StreamingMovies'],df['StreamingTV'])
```



In [33]:
```
check_dist(df['DeviceProtection'],df['TechSupport'])
```



In [34]:
```
#Creating a new column for Online Services (Online Security & Online Backup) . If a cu
#then , I am considering it as "Yes" else "No"
list_Family = []
for rows_os in range(len(df['Partner'])):
    if ((df['Partner'][rows_os] == 'No') and (df['Dependents'][rows_os] == 'No')):
        list_Family.append('No')
    else:
        list_Family.append('Yes')
df['Family'] = list_Family
```

In [35]:
```
#Creating a new column for Online Services (Online Security & Online Backup) . If a cu
#then , I am considering it as "Yes" else "No"
list_online_services = []
for rows_os in range(len(df['OnlineSecurity'])):
    if ((df['OnlineSecurity'][rows_os] == 'No') and (df['OnlineBackup'][rows_os] == 'N
        list_online_services.append('No')
    else:
        list_online_services.append('Yes')
df['OnlineServices'] = list_online_services
```

In [36]:
```
#Creating a new column for Streaming Services (StreamingTV & StreamingMovies) . If a c
#then , I am considering it as "Yes" else "No"
list_streaming_services = []
for rows_stv in range(len(df['StreamingTV'])):
    if ((df['StreamingTV'][rows_stv] == 'No') and (df['StreamingMovies'][rows_stv] ==
        list_streaming_services.append('No')
    else:
        list_streaming_services.append('Yes')
df['StreamingServices'] = list_streaming_services
```

In [37]:
```
df = df.drop(columns=['StreamingTV','StreamingMovies','OnlineSecurity','OnlineBackup',
```

Feature Scaling

def distplot(feature, frame, color='g'): plt.figure(figsize=(8,3)) plt.title("Distribution for {}".format(feature)) ax = sns.distplot(frame[feature], color= color)

In [38]:
```
large_cols = ["tenure", "MonthlyCharges"]
df[large_cols].describe()
```

Out[38]:

	tenure	MonthlyCharges
count	7043.000000	7043.000000
mean	32.416442	64.761692
std	24.526454	30.090047
min	1.000000	18.250000
25%	9.000000	35.500000
50%	29.000000	70.350000
75%	55.000000	89.850000
max	72.000000	118.750000

In [39]:
```
scaler = MinMaxScaler()
df[large_cols] = scaler.fit_transform(df[large_cols])
df[large_cols].describe()
```

Out[39]:

	tenure	MonthlyCharges
count	7043.000000	7043.000000
mean	0.442485	0.462803
std	0.345443	0.299403
min	0.000000	0.000000
25%	0.112676	0.171642
50%	0.394366	0.518408
75%	0.760563	0.712438
max	1.000000	1.000000

Normalization

The dataset has 16 categorical columns and six of them have binary values (yes or no). The approach used here is: - In binary columns, just replace values with 1 and 0 - Create a new column for each value in the remaining columns (and assign 1 or 0) This is equivalent to one-hot-encode, but we avoid some extra columns for yes and no features.

In [40]:
```
categorical_cols = [c for c in df.columns if df[c].dtype == 'object']
for col in categorical_cols:
    if df[col].nunique() == 2:
        df[col], _ = pd.factorize(df[col])
    else:
        df = pd.get_dummies(df, columns=[col])

df.head(3)
```

Out[40]:

	SeniorCitizen	tenure	PhoneService	PaperlessBilling	MonthlyCharges	Churn	Family	OnlineServices	Stre
0	0	0.000000	0	0	0.115423	0	0	0	
1	0	0.464789	1	1	0.385075	0	1	0	
2	0	0.014085	1	0	0.354229	1	1	0	

3 rows × 28 columns

Modelling

Splitting the data

In [41]:
```
#Separate attributes and output into two tables: feature and label
feature = df.drop(['Churn'],axis=1)
label = df['Churn']
```

In [42]:
```
#Divide the data into train and test according to the ratio train:test = 8:2
X_train,X_test,y_train,y_test = train_test_split(feature ,label,test_size=0.2,random_s
```

Handle imbalanced classes with SMOTE

In [43]:
```
#pip install imblearn
```

In [44]:
```
from imblearn import over_sampling
from imblearn.over_sampling import SMOTE

oversample = SMOTE(random_state = 42)
X_train_smote, y_train_smote = oversample.fit_resample(X_train,y_train)
```

In [45]:
```
from collections import Counter
print("Before SMOTE:", Counter(y_train))
print("After SMOTE:", Counter(y_train_smote))
```

```
Before SMOTE: Counter({0: 4138, 1: 1496})
After SMOTE: Counter({0: 4138, 1: 4138})
```

In [46]:
```
def cm(test,predict):
    cm = confusion_matrix(test,predict)
# %% confusion matrix visualization
    f, ax = plt.subplots(figsize = (5,5))
    sns.heatmap(cm, annot = True, linewidths = 0.5, color = "red", fmt = ".0f", annot_

    plt.xlabel("Predict")
    plt.ylabel("Actual")
    TP = cm[1,1]
    TN = cm[0,0]
    FP = cm[0,1]
    FN = cm[1,0]
    print('TP =',TP,'TN =',TN,'FP =',FP,'FN =',FN)
```

In [47]:
```
def eval_metrics(test,predict):
    #presision, recall, f1, accuracy
    print(classification_report(test, predict))

    #AUC score
    auc = metrics.roc_auc_score(test, predict)
    print('AUC score:',round(auc,2))
```

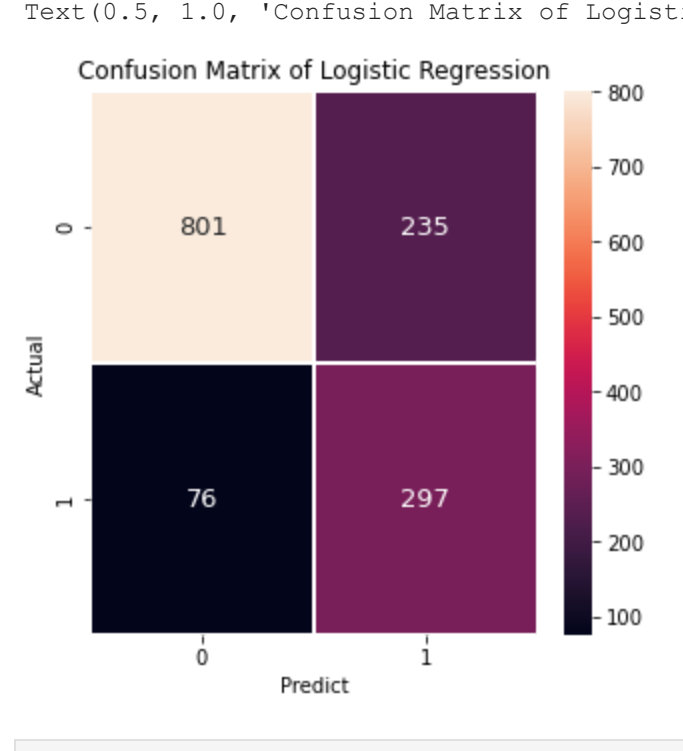
Logistic Regresssion

In [48]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X_train_smote, y_train_smote)

y_pred_log = logreg.predict(X_test)
```

In [49]:
```
#Viz Confuse Matrix
cm(y_test,y_pred_log)
plt.savefig('CMofLR.png', format='png', dpi = 1000)
plt.title("Confusion Matrix of Logistic Regression")
```

```
TP = 297 TN = 801 FP = 235 FN = 76
```

Out[49]:
```
Text(0.5, 1.0, 'Confusion Matrix of Logistic Regression')
```



In [50]:
```
eval_metrics(y_test,y_pred_log)
```

```
              precision    recall  f1-score   support

           0       0.91      0.77      0.84      1036
           1       0.56      0.80      0.66       373

    accuracy                           0.78      1409
   macro avg       0.74      0.78      0.75      1409
weighted avg       0.82      0.78      0.79      1409

AUC score: 0.78
```

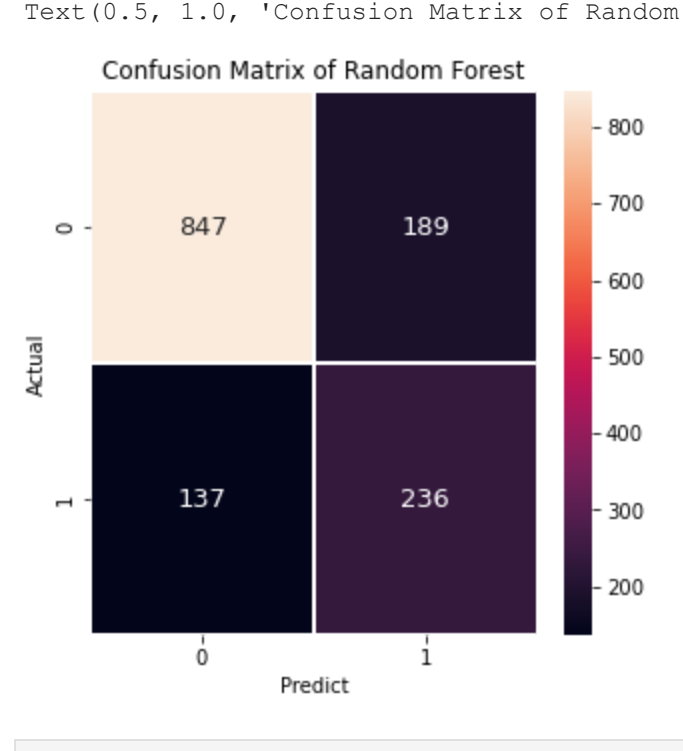
Random Forest

In [51]:
```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(100)
random_forest.fit(X_train_smote, y_train_smote)

y_pred_rf = random_forest.predict(X_test)
```

In [52]:
```
#Viz Confuse Matrix
cm(y_test,y_pred_rf)
plt.savefig('CMofRF.png', format='png', dpi = 1000)
plt.title("Confusion Matrix of Random Forest")
```

```
TP = 236 TN = 847 FP = 189 FN = 137
```

Out[52]:
```
Text(0.5, 1.0, 'Confusion Matrix of Random Forest')
```



In [53]:
```
eval_metrics(y_test, y_pred_rf)
```

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84      1036
           1       0.56      0.63      0.59       373

    accuracy                           0.77      1409
   macro avg       0.71      0.73      0.72      1409
weighted avg       0.78      0.77      0.77      1409

AUC score: 0.73
```

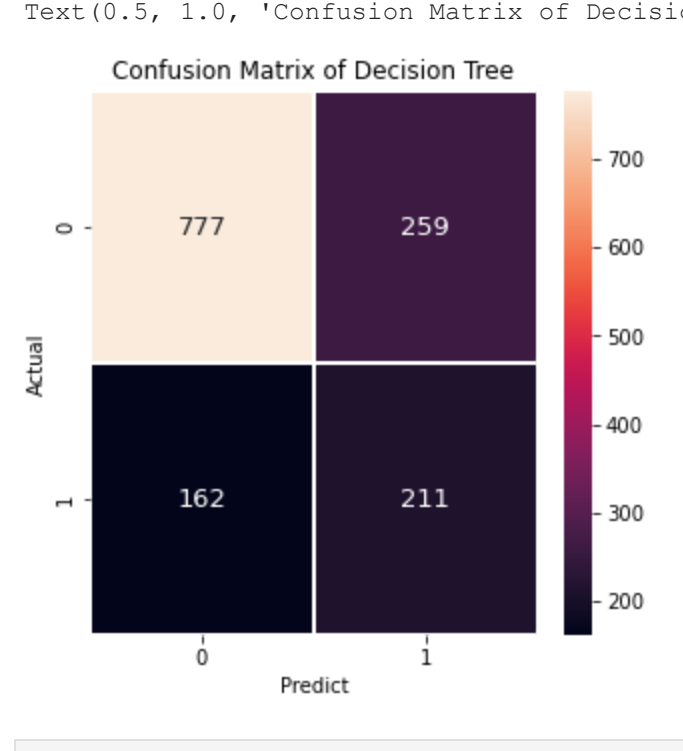
Decision Tree

In [54]:
```
#Creating decision trees with CART
from sklearn import tree
dtcart= tree.DecisionTreeClassifier(criterion ='gini', random_state=0)

#Train DT model
dtcart.fit(X_train_smote,y_train_smote)

tree_pred = dtcart.predict(X_test)
```

In [55]:
```
#Viz Confuse Matrix
cm(y_test, tree_pred)
plt.savefig('CMofDT.png', format='png', dpi = 1000)
plt.title("Confusion Matrix of Decision Tree")
```

```
TP = 211 TN = 777 FP = 259 FN = 162
```

Out[55]:
```
Text(0.5, 1.0, 'Confusion Matrix of Decision Tree')
```



In [56]:
```
eval_metrics(y_test, tree_pred)
```

```
              precision    recall  f1-score   support

           0       0.83      0.75      0.79      1036
           1       0.45      0.57      0.50       373

    accuracy                           0.70      1409
   macro avg       0.64      0.66      0.64      1409
weighted avg       0.73      0.70      0.71      1409

AUC score: 0.66
```

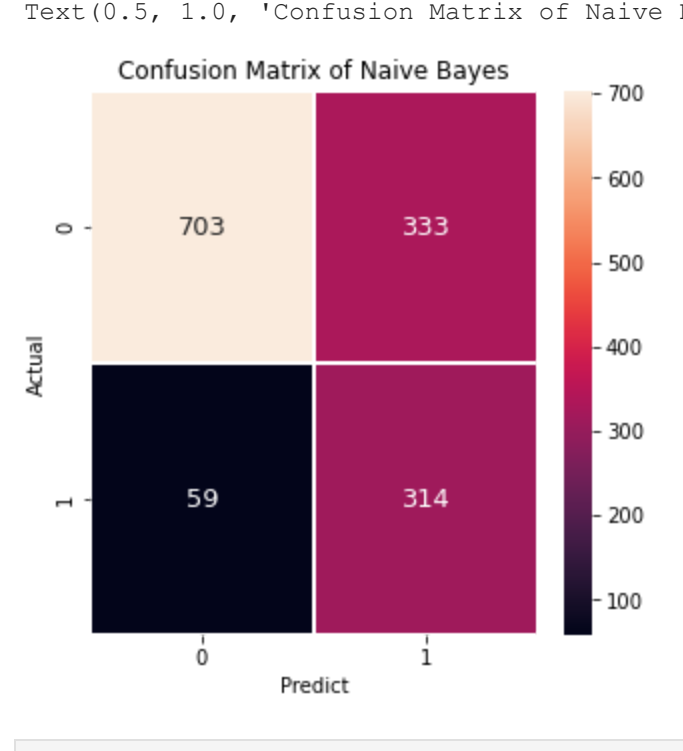
Naive Bayes

In [57]:
```
from sklearn.naive_bayes import GaussianNB
#Initialize NB Algorithm
gnb = GaussianNB()

#Training NB model
y_pred_bayes = gnb.fit(X_train,y_train).predict(X_test)
```

In [58]:
```
#Viz Confuse Matrix
cm(y_test,y_pred_bayes)
plt.savefig('CMofNB.png', format='png', dpi = 1000)
plt.title("Confusion Matrix of Naive Bayes")
```

```
TP = 314 TN = 703 FP = 333 FN = 59
```

Out[58]:
```
Text(0.5, 1.0, 'Confusion Matrix of Naive Bayes')
```



In [59]:
```
eval_metrics(y_test,y_pred_bayes)
```

```
              precision    recall  f1-score   support

           0       0.92      0.68      0.78      1036
           1       0.49      0.84      0.62       373

    accuracy                           0.72      1409
   macro avg       0.70      0.76      0.70      1409
weighted avg       0.81      0.72      0.74      1409

AUC score: 0.76
```