Project Name: Knowledge Based Customer Churan Prediction For Telecom Services Using Python

Project Done By:

Pranjal Chowdhury (CSE / 4th / IEM)

Partho Protim Sarkar (CSE / 4th / IEM)

Problem Statement:

- Customer churn is a major problem and one of the most important concerns for large companies. Due to the direct effect on the revenues of the companies, especially in the telecom field, companies are seeking to develop means to predict potential customer to churn. Therefore, finding factors that increase customer churn is important to take necessary actions to reduce thisChurn. Churn prediction helps business to gain a better understanding of future expected revenue , allows to target individual in an attempt to prevent them from discontinuing their subscription with the company and helps to understand what preventative steps are necessary to ensure lost revenue is minimized.

In [ ]:

Importing the dataset and understanding the dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas_profiling
import plotly.offline as po
import plotly.graph_objs as go
import math
%matplotlib inline
df=pd.read_csv("Telco-Customer-Churn.csv")
```

In [2]:
```
#df.head(20)
#df.info()
#df.describe()
#df.count()
#df.shape
```

In [3]:
```
df.head(50)
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	Tech
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	
5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No	...	Yes	
6	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	...	No	

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
customerID          7043 non-null object
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(18)
memory usage: 1.1+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000

50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

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

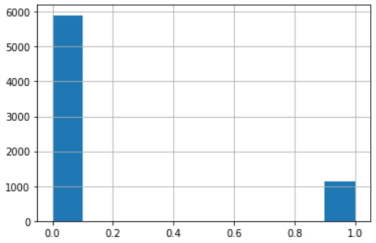
Out[6]:
```
customerID          7043
gender              7043
SeniorCitizen       7043
Partner             7043
Dependents          7043
tenure              7043
PhoneService        7043
MultipleLines       7043
InternetService     7043
OnlineSecurity      7043
OnlineBackup        7043
DeviceProtection    7043
TechSupport         7043
StreamingTV         7043
StreamingMovies     7043
Contract            7043
PaperlessBilling    7043
PaymentMethod       7043
MonthlyCharges      7043
TotalCharges        7043
Churn               7043
dtype: int64
```

In [7]:
```
df.shape
```

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

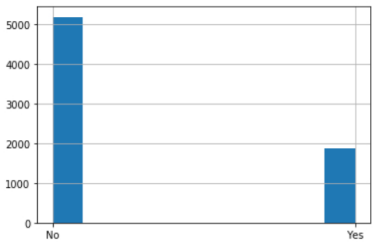
In [8]:
```
df.SeniorCitizen.hist()
plt.plot()
```

Out[8]:
```
[]
```



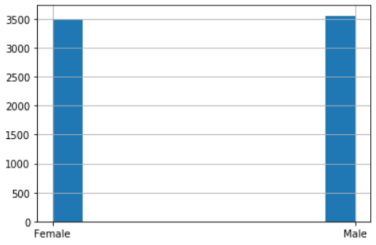
In [9]:
```
df.Churn.hist()
plt.plot()
```

Out[9]:
```
[]
```



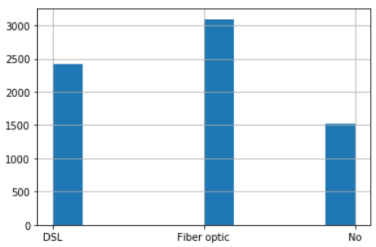
In [10]:
```
df.gender.hist()
plt.plot()
```

Out[10]:
```
[]
```



In [11]:
```
df.InternetService.hist()
plt.plot()
```

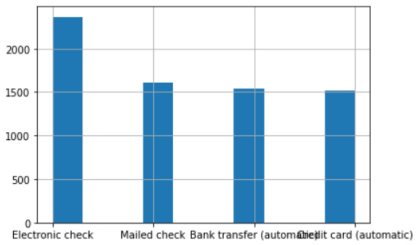
Out[11]:
```
[]
```



In [12]: df.PaymentMethod.hist()

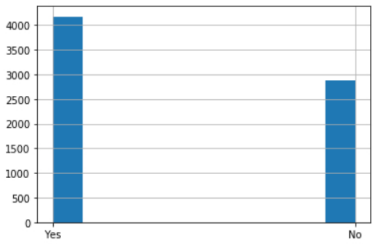
```
plt.plot()
```

Out[12]: []



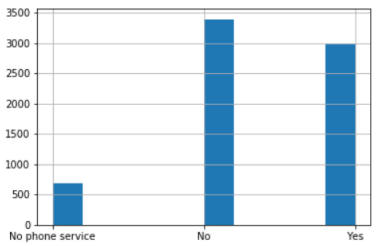
In [13]:
```
df.PaperlessBilling.hist()
plt.plot()
```

Out[13]: []



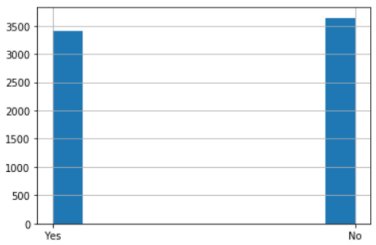
In [14]:
```
df.MultipleLines.hist()
plt.plot()
```

Out[14]: []



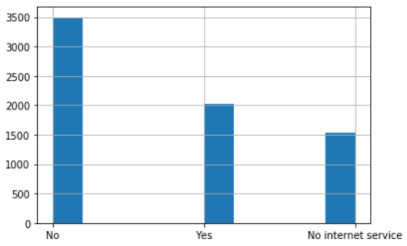
In [15]:
```
df.Partner.hist()
plt.plot()
```

Out[15]: []



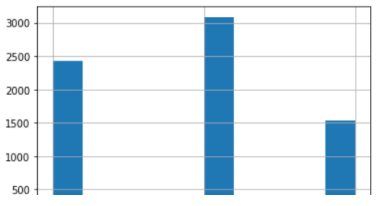
In [16]:
```
df.OnlineSecurity.hist()
plt.plot()
```

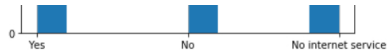
Out[16]: []



In [17]:
```
df.OnlineBackup.hist()
plt.plot()
```

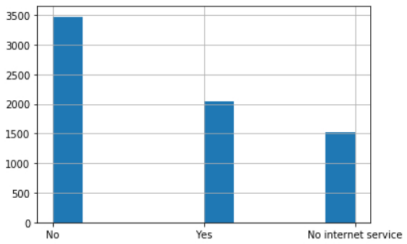
Out[17]: []





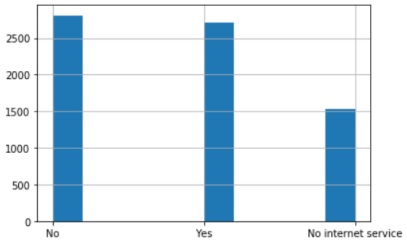
In [18]:
```
df.TechSupport.hist()
plt.plot()
```

Out[18]: []



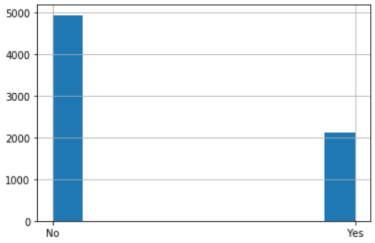
In [19]:
```
df.StreamingTV.hist()
plt.plot()
```

Out[19]: []



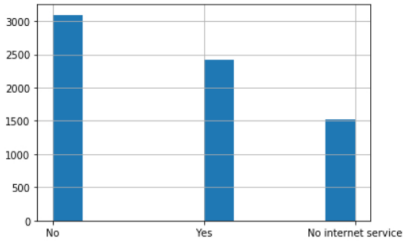
In [20]:
```
df.Dependents.hist()
plt.plot()
```

Out[20]: []



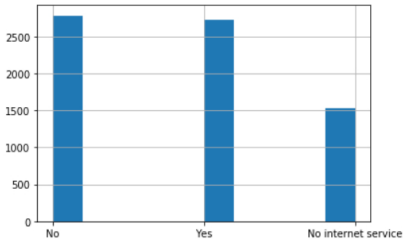
In [21]:
```
df.DeviceProtection.hist()
plt.plot()
```

Out[21]: []



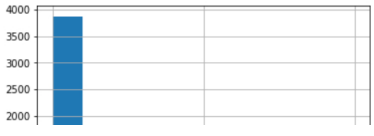
In [22]:
```
df.StreamingMovies.hist()
plt.plot()
```

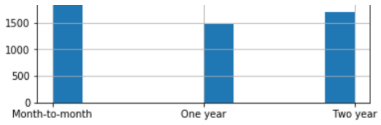
Out[22]: []



In [23]:
```
df.Contract.hist()
plt.plot()
```

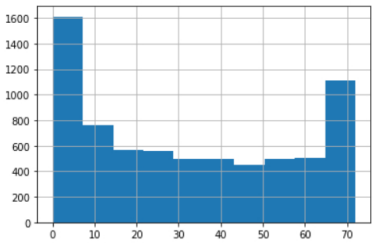
Out[23]: []





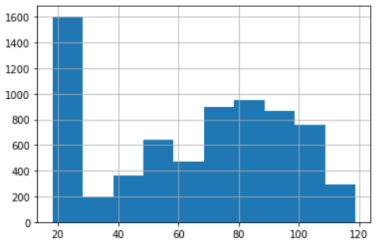
In [24]:
```
df.tenure.hist()
plt.plot()
```

Out[24]: []



In [25]:
```
df.MonthlyCharges.hist()
plt.plot()
```

Out[25]: []

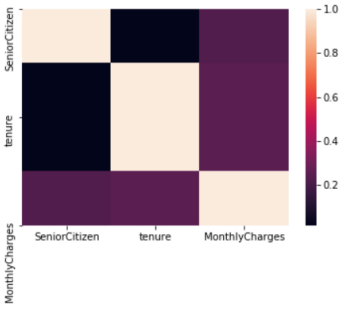


In [26]:
```
df.TotalCharges.hist()
plt.plot()
```

Out[26]: []

In [27]:
```
#correlation
sns.heatmap(df.corr())
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x21fadfc39c8>



In [28]:
```
#convert string values (yes and no) of churn colum to 1 and 0
df.loc[df.Churn=='No','Churn']=0
df.loc[df.Churn=='Yes','Churn']=1
```

In [29]:
```
# Convert 'No internet service' to 'No' for the below mentioned columns
cols = ['OnlineBackup', 'StreamingMovies','DeviceProtection',
                'TechSupport','OnlineSecurity','StreamingTV']
for i in cols :
    df[i]  = df[i].replace({'No internet service' : 'No'})
```

In [30]:
```

# Replace all the spaces with null values
df['TotalCharges'] = df["TotalCharges"].replace(" ",np.nan)

# Drop null values of 'Total Charges' feature
df = df[df["TotalCharges"].notnull()]
df= df.reset_index()[df.columns]

# Convert 'Total Charges' column values to float data type
df["TotalCharges"] = df["TotalCharges"].astype(float)
```

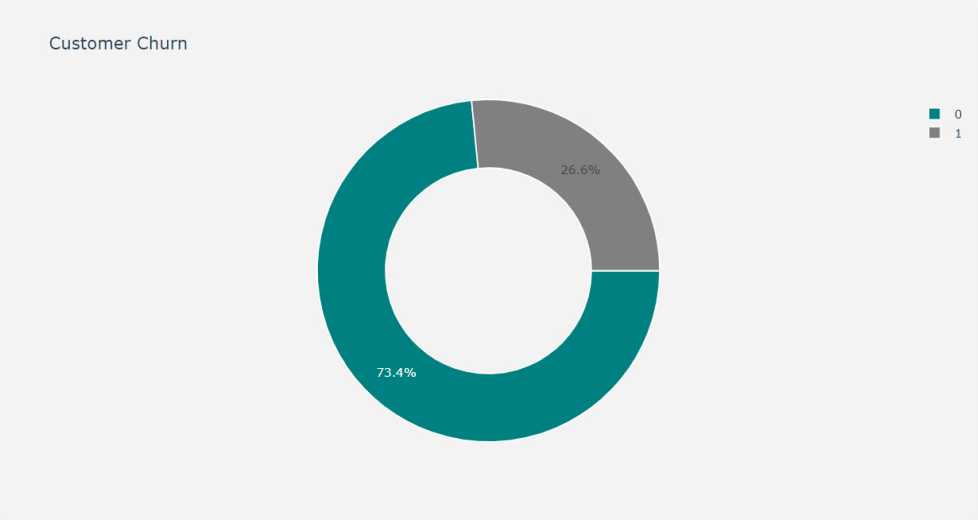
In [31]:
```
df["Churn"].value_counts().values
```

Out[31]:
```
array([5163, 1869], dtype=int64)
```

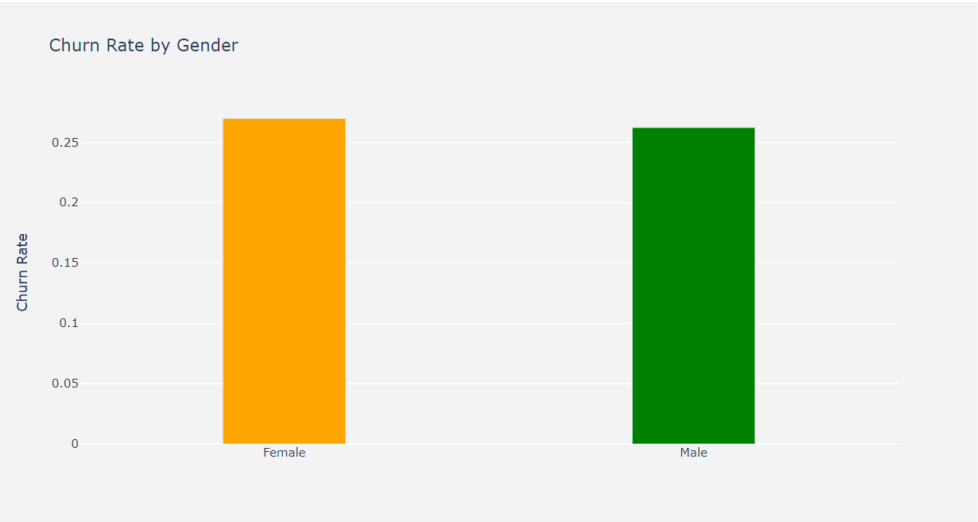
In [32]:
```
# Visualize Total Customer Churn
plot_by_churn_labels = df["Churn"].value_counts().keys().tolist()
plot_by_churn_values = df["Churn"].value_counts().values.tolist()

plot_data= [
    go.Pie(labels = plot_by_churn_labels,
           values = plot_by_churn_values,
           marker = dict(colors =  [ 'Teal' ,'Grey'],
                         line = dict(color = "white",
                                     width =  1.5)),
           rotation = 90,
           hoverinfo = "label+value+text",
           hole = .6)
]
plot_layout = go.Layout(dict(title = "Customer Churn",
                  plot_bgcolor  = "rgb(243,243,243)",
                  paper_bgcolor = "rgb(243,243,243)",))


fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```

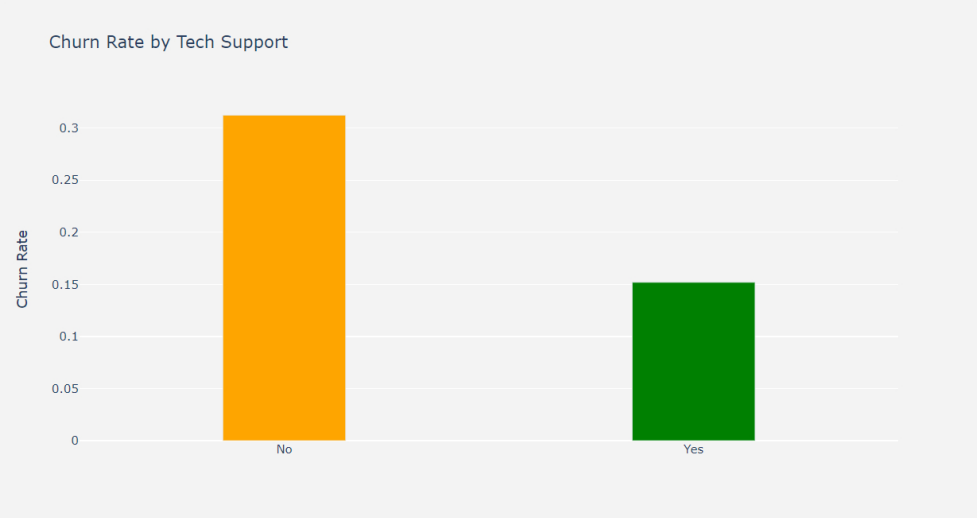


In [33]:
```
# Visualize Churn Rate by Gender
plot_by_gender = df.groupby('gender').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=plot_by_gender['gender'],
        y=plot_by_gender['Churn'],
        width = [0.3, 0.3],
        marker=dict(
        color=['orange', 'green'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        yaxis={"title": "Churn Rate"},
        title='Churn Rate by Gender',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```

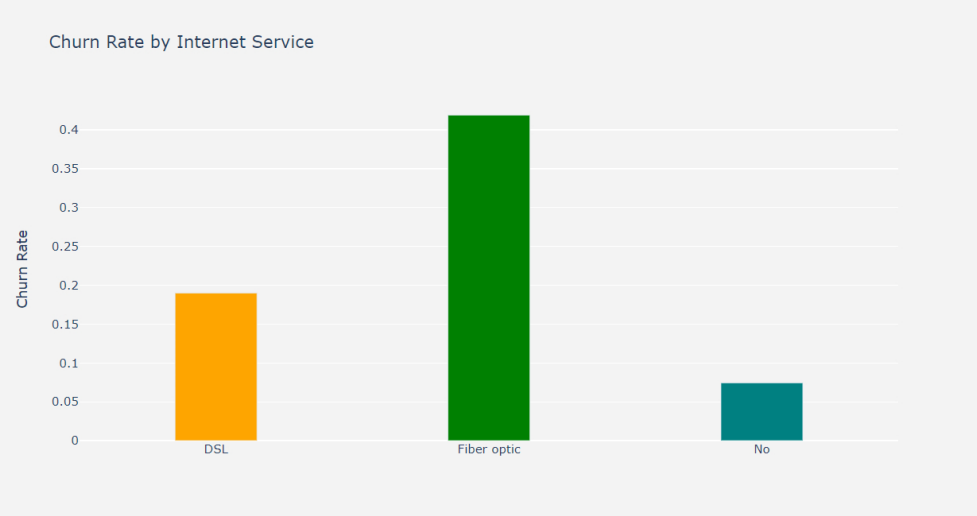


In [34]:
```
# Visualize Churn Rate by Tech Support
plot_by_techsupport = df.groupby('TechSupport').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=plot_by_techsupport['TechSupport'],
        y=plot_by_techsupport['Churn'],
        width = [0.3, 0.3, 0.3],
        marker=dict(
```

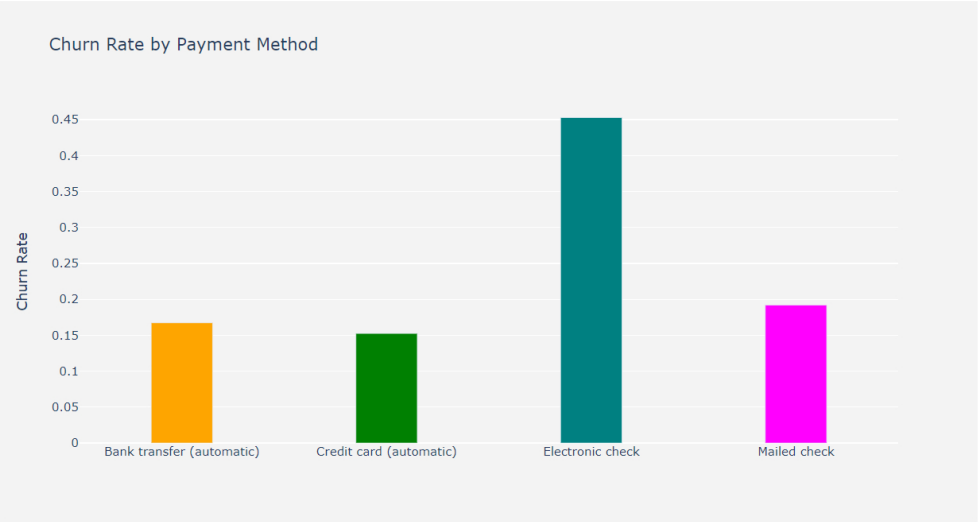
```
        color=['orange', 'green', 'teal'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        yaxis={"title": "Churn Rate"},
        title='Churn Rate by Tech Support',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```



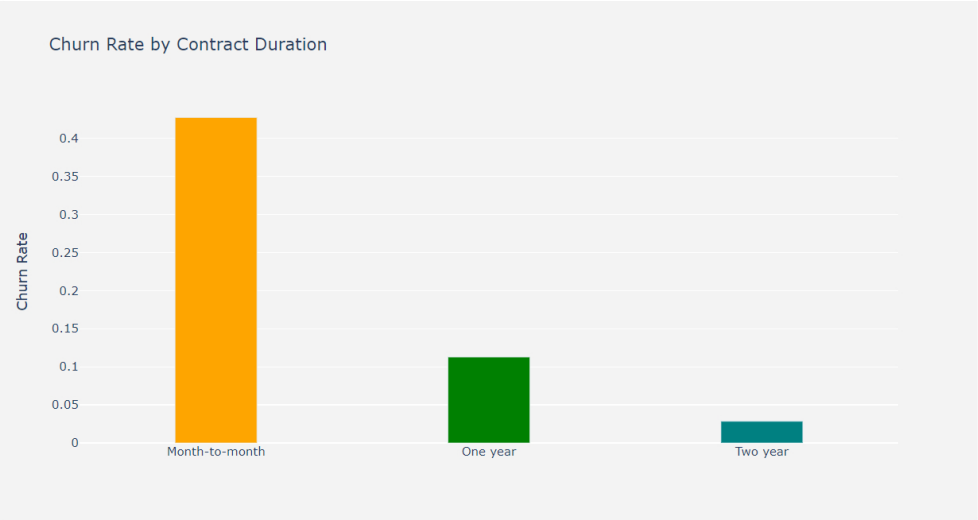
In [35]:
```
# Visualize Churn Rate by Internet Services
plot_by_internet_service = df.groupby('InternetService').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=plot_by_internet_service['InternetService'],
        y=plot_by_internet_service['Churn'],
        width = [0.3, 0.3, 0.3],
        marker=dict(
        color=['orange', 'green', 'teal'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        yaxis={"title": "Churn Rate"},
        title='Churn Rate by Internet Service',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```



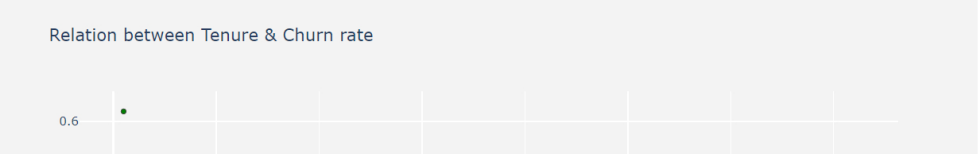
In [36]:
```
# Visualize Churn Rate by Payment Method
plot_by_payment = df.groupby('PaymentMethod').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=plot_by_payment['PaymentMethod'],
        y=plot_by_payment['Churn'],
        width = [0.3, 0.3,0.3,0.3],
        marker=dict(
        color=['orange', 'green','teal','magenta'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        yaxis={"title": "Churn Rate"},
        title='Churn Rate by Payment Method',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```

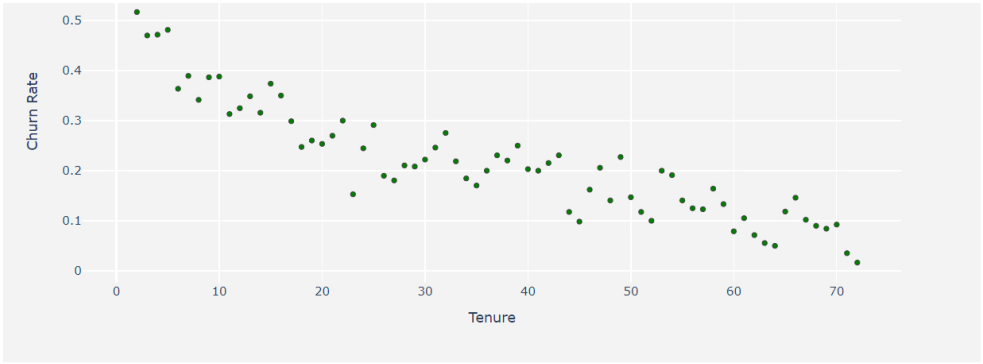


In [37]:
```
# Visualize Churn Rate by Contract Duration
plot_by_contract = df.groupby('Contract').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=plot_by_contract['Contract'],
        y=plot_by_contract['Churn'],
        width = [0.3, 0.3,0.3],
        marker=dict(
        color=['orange', 'green','teal'])
    )
]
plot_layout = go.Layout(
        xaxis={"type": "category"},
        yaxis={"title": "Churn Rate"},
        title='Churn Rate by Contract Duration',
        plot_bgcolor  = 'rgb(243,243,243)',
        paper_bgcolor  = 'rgb(243,243,243)',
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```



In [38]:
```
# Visualize Relation between Tenure & Churn rate
plot_by_tenure = df.groupby('tenure').Churn.mean().reset_index()
plot_data = [
    go.Scatter(
        x=plot_by_tenure['tenure'],
        y=plot_by_tenure['Churn'],
        mode='markers',
        name='Low',
        marker= dict(size= 5,
            line= dict(width=0.8),
            color= 'green'
           ),
    )
]
plot_layout = go.Layout(
        yaxis= {'title': "Churn Rate"},
        xaxis= {'title': "Tenure"},
        title='Relation between Tenure & Churn rate',
        plot_bgcolor  = "rgb(243,243,243)",
        paper_bgcolor  = "rgb(243,243,243)",
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
po.iplot(fig)
```





In [39]:
```
#Perform One Hot Encoding using get_dummies method
df = pd.get_dummies(df, columns = ['Contract','Dependents','DeviceProtection','gender',
                                   'InternetService','MultipleLines','OnlineBackup',
                                   'OnlineSecurity','PaperlessBilling','Partner',
                                   'PaymentMethod','PhoneService','SeniorCitizen',
                                   'StreamingMovies','StreamingTV','TechSupport'],
                    drop_first=True)
```

In [40]:
```
#Perform Feature Scaling and One Hot Encoding
from sklearn.preprocessing import StandardScaler

#Perform Feature Scaling on 'tenure', 'MonthlyCharges', 'TotalCharges' in order to bring them on same scale
standardScaler = StandardScaler()
columns_for_ft_scaling = ['tenure', 'MonthlyCharges', 'TotalCharges']

#Apply the feature scaling operation on dataset using fit_transform() method
df[columns_for_ft_scaling] = standardScaler.fit_transform(df[columns_for_ft_scaling])
```

In [41]:
```
# See subset of values
df.head()
```

Out[41]:

	customerID	tenure	MonthlyCharges	TotalCharges	Churn	Contract_One year	Contract_Two year	Dependents_Yes	DeviceProtection_Yes	gender_Male	...	PaperlI
0	7590-VHVEG	-1.280248	-1.161694	-0.994194	0	0	0	0	0	0	...	
1	5575-GNVDE	0.064303	-0.260878	-0.173740	0	1	0	0	1	1	...	
2	3668-QPYBK	-1.239504	-0.363923	-0.959649	1	0	0	0	0	1	...	
3	7795-CFOCW	0.512486	-0.747850	-0.195248	0	1	0	0	1	1	...	
4	9237-HQITU	-1.239504	0.196178	-0.940457	1	0	0	0	0	0	...	

5 rows × 26 columns

In [42]:
```
#Number of columns increased and have suffixes attached, as a result of get_dummies method.
df.columns
```

Out[42]:
```
Index(['customerID', 'tenure', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'Contract_One year', 'Contract_Two year', 'Dependents_Yes',
       'DeviceProtection_Yes', 'gender_Male', 'InternetService_Fiber optic',
       'InternetService_No', 'MultipleLines_No phone service',
       'MultipleLines_Yes', 'OnlineBackup_Yes', 'OnlineSecurity_Yes',
       'PaperlessBilling_Yes', 'Partner_Yes',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check',
       'PhoneService_Yes', 'SeniorCitizen_1', 'StreamingMovies_Yes',
       'StreamingTV_Yes', 'TechSupport_Yes'],
      dtype='object')
```

In [43]:
```
#Create Feature variable X and Target variable y
y = df['Churn']
X = df.drop(['Churn','customerID'], axis = 1)
```

In [44]:
```
#Split the data into training set (70%) and test set (30%)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 50)
```

In [45]:
```
# Machine Learning classification model libraries
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

In [46]:
```
#Fit the Perceptron Model
per_clf = Perceptron(random_state=42)
per_clf.fit(X_train,y_train)

#Predict the value for new, unseen data
per_pred = per_clf.predict(X_test)

# Find Accuracy using accuracy_score method
per_accuracy= round(metrics.accuracy_score(y_test, per_pred ) * 100, 2)
```

In [47]:
```
#Fit the Logistic Regression Model
logmodel = LogisticRegression(random_state=50)
logmodel.fit(X_train,y_train)

#Predict the value for new, unseen data
pred = logmodel.predict(X_test)

# Find Accuracy using accuracy_score method
logmodel_accuracy = round(metrics.accuracy_score(y_test, pred) * 100, 2)
```

```
C:\Users\user\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
```

```
C:\Users\user\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:

Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
```

In [48]:
```
#Fit the Support Vector Machine Model
svcmodel = SVC(kernel='linear', random_state=50, probability=True)
svcmodel.fit(X_train,y_train)

#Predict the value for new, unseen data
svc_pred = svcmodel.predict(X_test)

# Find Accuracy using accuracy_score method
svc_accuracy = round(metrics.accuracy_score(y_test, svc_pred) * 100, 2)
```

In [49]:
```
#Fit the K-Nearest Neighbor Model
from sklearn.neighbors import KNeighborsClassifier
knnmodel = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2) #p=2 represents Euclidean distance, p=1 represents Manha
knnmodel.fit(X_train, y_train)

#Predict the value for new, unseen data
knn_pred = knnmodel.predict(X_test)

# Find Accuracy using accuracy_score method
knn_accuracy = round(metrics.accuracy_score(y_test, knn_pred) * 100, 2)
```

In [50]:
```
#Fit the Decision Tree Classification Model
from sklearn.tree import DecisionTreeClassifier
dtmodel = DecisionTreeClassifier(criterion = "gini", random_state = 50)
dtmodel.fit(X_train, y_train)

#Predict the value for new, unseen data
dt_pred = dtmodel.predict(X_test)

# Find Accuracy using accuracy_score method
dt_accuracy = round(metrics.accuracy_score(y_test, dt_pred) * 100, 2)
```

In [51]:
```
#Fit the Random Forest Classification Model
from sklearn.ensemble import RandomForestClassifier
rfmodel = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', random_state = 0)
rfmodel.fit(X_train, y_train)

#Predict the value for new, unseen data
rf_pred = rfmodel.predict(X_test)

# Find Accuracy using accuracy_score method
rf_accuracy = round(metrics.accuracy_score(y_test, rf_pred) * 100, 2)
```

In [52]:
```
#using neural network
```

In [53]:
```
#Create Feature variable X and Target variable y
y = df['Churn']
X = df.drop(['Churn','customerID'], axis = 1)
```

In [54]:
```
#Split the data into training set (70%) and test set (30%)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 50)
```

In [55]:
```
import keras
```

```
Using TensorFlow backend.
```

In [56]:
```
#Let's build the model
model = keras.models.Sequential()

model.add(keras.layers.Dense(30, activation='relu', input_shape=(24,)))
model.add(keras.layers.Dense(28, activation='relu'))
model.add(keras.layers.Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
```

In [57]:
```
#summary()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 30)                750
_________________________________________________________________
dense_2 (Dense)              (None, 28)                868
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 29
=================================================================
Total params: 1,647
Trainable params: 1,647
Non-trainable params: 0
_________________________________________________________________
```

In [58]:
```
#Now fit the model
model.fit(X_train, y_train, epochs=100, callbacks=[keras.callbacks.EarlyStopping(patience=3)])
```

```
Epoch 1/100
4922/4922 [==============================] - 21s 4ms/step - loss: 0.1633
Epoch 2/100
1056/4922 [=====>........................] - ETA: 0s - loss: 0.1407
C:\Users\user\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py:846: RuntimeWarning:

Early stopping conditioned on metric `val_loss` which is not available. Available metrics are: loss

4922/4922 [==============================] - 1s 150us/step - loss: 0.1387
Epoch 3/100
4922/4922 [==============================] - 1s 172us/step - loss: 0.1359
Epoch 4/100
4922/4922 [==============================] - 1s 173us/step - loss: 0.1343
Epoch 5/100
4922/4922 [==============================] - 1s 160us/step - loss: 0.1329
Epoch 6/100
4922/4922 [==============================] - 1s 153us/step - loss: 0.1315
Epoch 7/100
```

In [59]:
```
DNN_pra= model.predict(X_test)
DNN_accuracy= round(metrics.accuracy_score(y_test, per_pred ) * 100, 2)
```

In [60]:
```
# Compare Several models according to their Accuracies
Model_Comparison = pd.DataFrame({
    'Model': ['Perceptron','Danse neural network','Logistic Regression', 'Support Vector Machine', 'K-Nearest Neighbor',
              'Decision Tree', 'Random Forest'],
    'Score': [per_accuracy,DNN_accuracy,logmodel_accuracy, svc_accuracy, knn_accuracy,
              dt_accuracy, rf_accuracy]})
Model_Comparison_df = Model_Comparison.sort_values(by='Score', ascending=False)
Model_Comparison_df = Model_Comparison_df.set_index('Score')
Model_Comparison_df.reset_index()
```

Out[60]:

	Score	Model
0	81.14	Logistic Regression
1	80.66	Support Vector Machine
2	79.38	Random Forest
3	78.10	Perceptron
4	78.10	Danse neural network
5	76.87	K-Nearest Neighbor
6	73.27	Decision Tree

In [61]:
```
#Generate confusion matrix for logistics regression model as it has maximum Accuracy
from sklearn.metrics import confusion_matrix
conf_mat_logmodel = confusion_matrix(y_test,pred)
conf_mat_logmodel
```

Out[61]:
```
array([[1395,  166],
       [ 232,  317]], dtype=int64)
```

In [62]:
```
# Predict the probability of Churn of each customer
df['Probability_of_Churn'] = logmodel.predict_proba(df[X_test.columns])[:,1]
```

In [63]:
```
# Create a Dataframe showcasing probability of Churn of each customer
df[['customerID','Probability_of_Churn']].head(10)
```

Out[63]:

	customerID	Probability_of_Churn
0	7590-VHVEG	0.649626
1	5575-GNVDE	0.044389
2	3668-QPYBK	0.338442
3	7795-CFOCW	0.027125
4	9237-HQITU	0.696231
5	9305-CDSKC	0.781197
6	1452-KIOVK	0.488613
7	6713-OKOMC	0.291929
8	7892-POOKP	0.593652
9	6388-TABGU	0.012203

In [ ]: