In [1]:
```python
# Import Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas_profiling #pip install pandas_profiling
import plotly.offline as po
import plotly.graph_objs as go
```

In [2]:
```python
#Perform Exploratory Data Analysis in just one line of code
pandas_profiling.ProfileReport(pd.read_csv('Tel_Customer_Churn_Dataset.csv'))
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	21
Number of observations	7043
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.1 MiB
Average record size in memory	168.0 B

Variable types

Categorical	14
Boolean	5
Numeric	2

Alerts

`customerID` has a high cardinality: 7043 distinct values	High cardinality
`TotalCharges` has a high cardinality: 6531 distinct values	High cardinality
`tenure` is highly overall correlated with `Contract`	High correlation

Out[2]:

In [3]:
```
#Import Customer Churn Dataset
df = pd.read_csv('Tel_Customer_Churn_Dataset.csv')
df.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:

```
df.shape
```

Out[4]:

```
(7043, 21)
```

In [5]:

```
# Convert String values (Yes and No) of Churn column to 1 and 0
df.loc[df.Churn=='No','Churn'] = 0
df.loc[df.Churn=='Yes','Churn'] = 1
```

In [6]:

```
# 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 [7]:

```
# 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)
df.head()
```

Out[7]:

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

5 rows × 21 columns

In [8]:

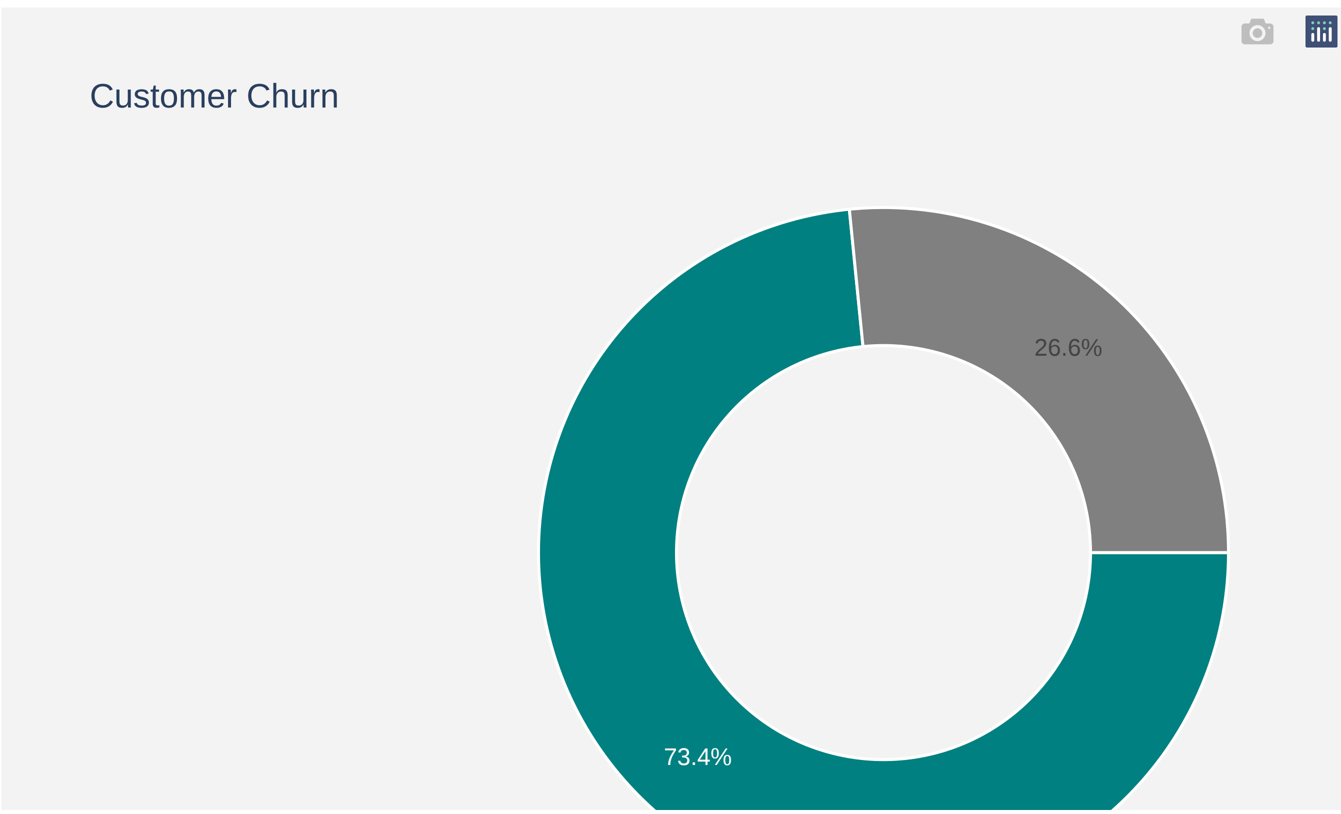
```
df["Churn"].value_counts().values
```

Out[8]: array([5163, 1869], dtype=int64)

In [9]:
```
# Visualize Total Customer Churn
df_labels = df["Churn"].value_counts().keys().tolist()
df_values = df["Churn"].value_counts().values.tolist()

plot_data= [
    go.Pie(labels = df_labels,
           values = df_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)
```

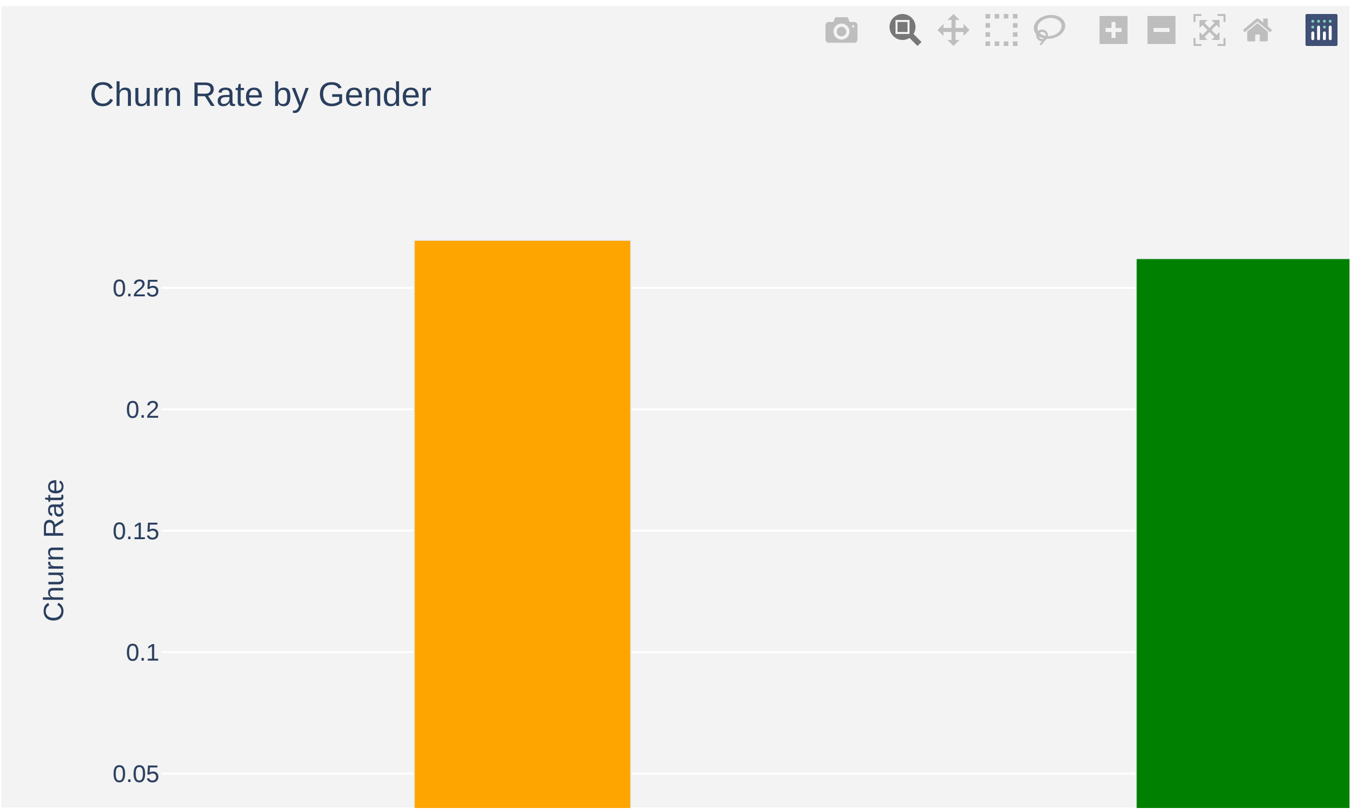


26.6% customer churn out while 73.4% customers stayed

In [10]:
```
# Visualize Churn Rate by Gender
df_gender = df.groupby('gender').Churn.mean().reset_index()
```

Loading [MathJax]/extensions/Safe.js

```
    go.Bar(
        x=df_gender['gender'],
        y=df_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)
print(df_gender)
```

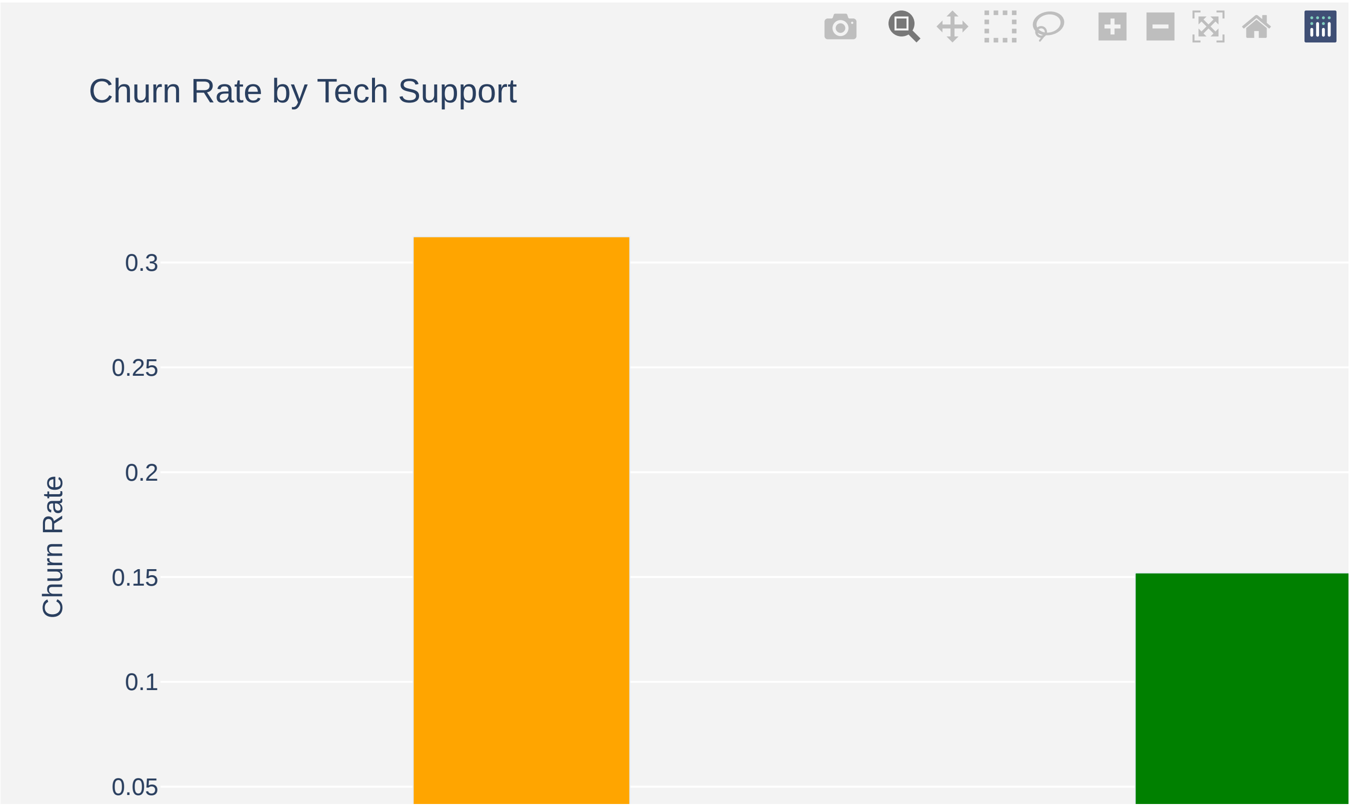



```
   gender     Churn
0  Female  0.269595
1    Male  0.262046
```

churn rate is slightly higher for females than males

In [11]:
```
# Visualize Churn Rate by Tech Support
df_techsupport = df.groupby('TechSupport').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_techsupport['TechSupport'],
        y=df_techsupport['Churn'],
        h = [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)
print(df_techsupport)
```




```
  TechSupport     Churn
0          No  0.312300
1         Yes  0.151961
```

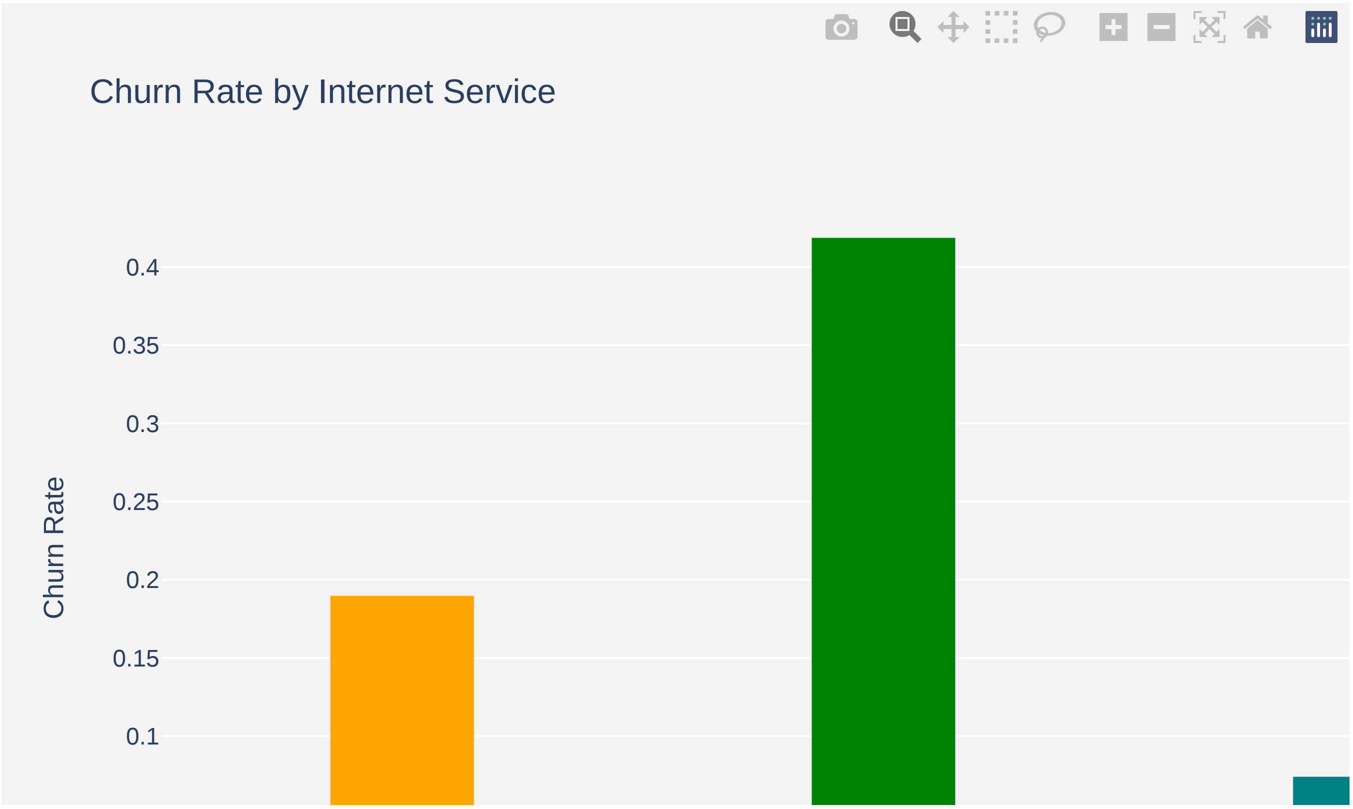
churn rate for those with no tech support is higher compared to those without tech support

In [12]:

```
# Visualize Churn Rate by Internet Services
df_internet_service = df.groupby('InternetService').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_internet_service['InternetService'],
        y=df_internet_service['Churn'],
        width = [0.3, 0.3, 0.3],
        marker=dict(
        color=['orange', 'green', 'teal'])
    )
```

Loading [MathJax]/extensions/Safe.js

```
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)
print(df_internet_service)
```



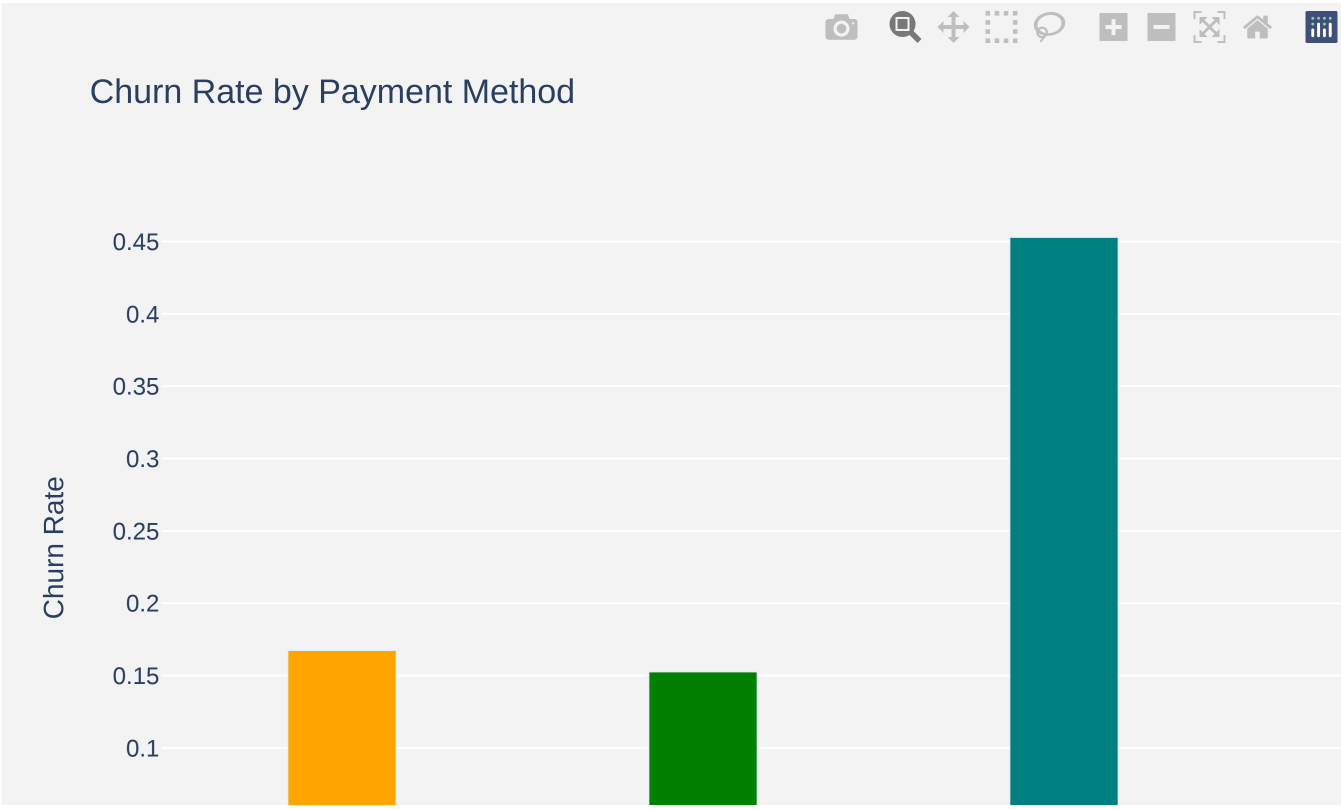

```
  InternetService     Churn
0             DSL  0.189983
1     Fiber optic  0.418928
2              No  0.074342
```

we have alot more people churning on fiber optics compared to DSL

In [13]:
```
# Visualize Churn Rate by Payment Method
df_payment = df.groupby('PaymentMethod').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_payment['PaymentMethod'],
        y=df_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"},
```

Loading [MathJax]/extensions/Safe.js

```python
        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)
print(df_payment)
```

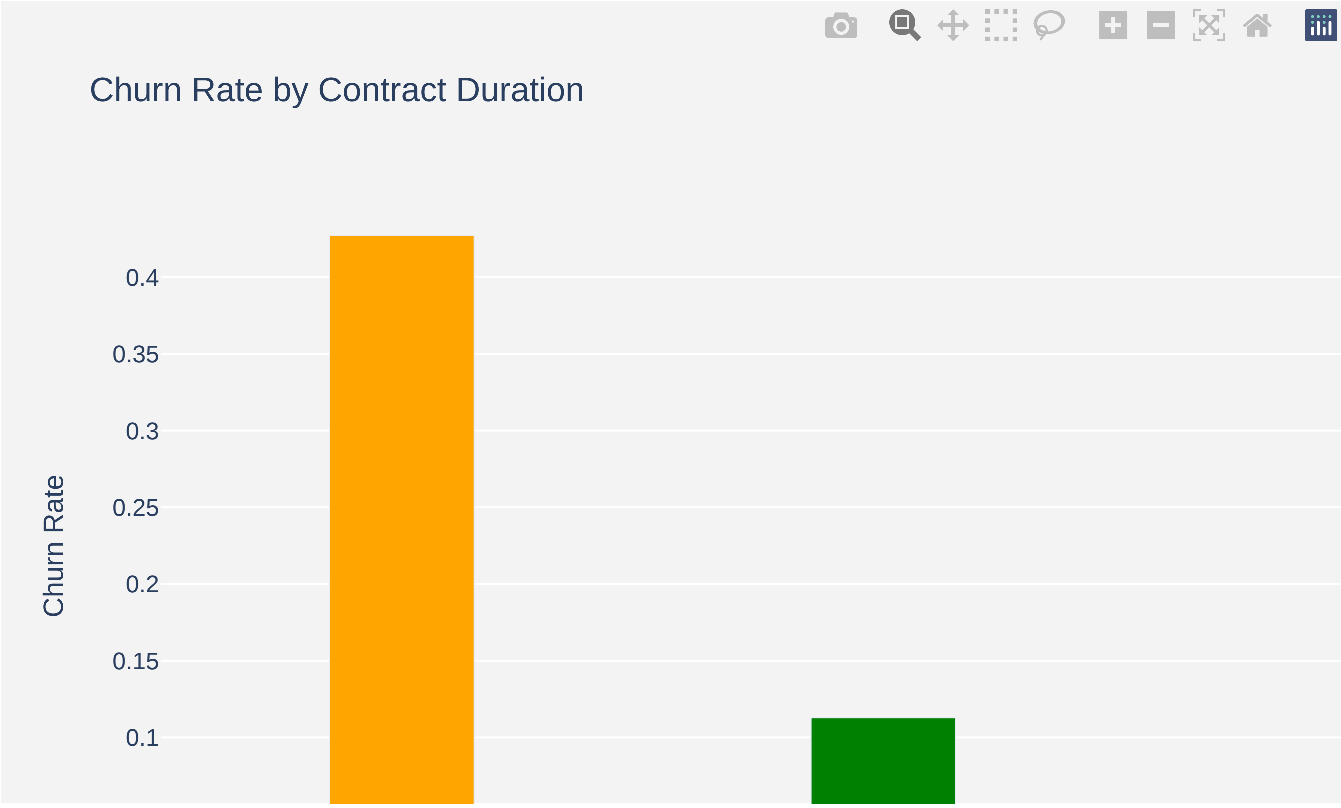



```
              PaymentMethod     Churn
0  Bank transfer (automatic)  0.167315
1    Credit card (automatic)  0.152531
2           Electronic check  0.452854
3               Mailed check  0.192020
```

we find more people who make payment through electronic check churning compared to the other payment methods

In [14]:
```python
# Visualize Churn Rate by Contract Duration
df_contract = df.groupby('Contract').Churn.mean().reset_index()
plot_data = [
    go.Bar(
        x=df_contract['Contract'],
        y=df_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)
print(df_contract)
```




```
        Contract     Churn
0  Month-to-month  0.427097
1        One year  0.112772
2        Two year  0.028487
```

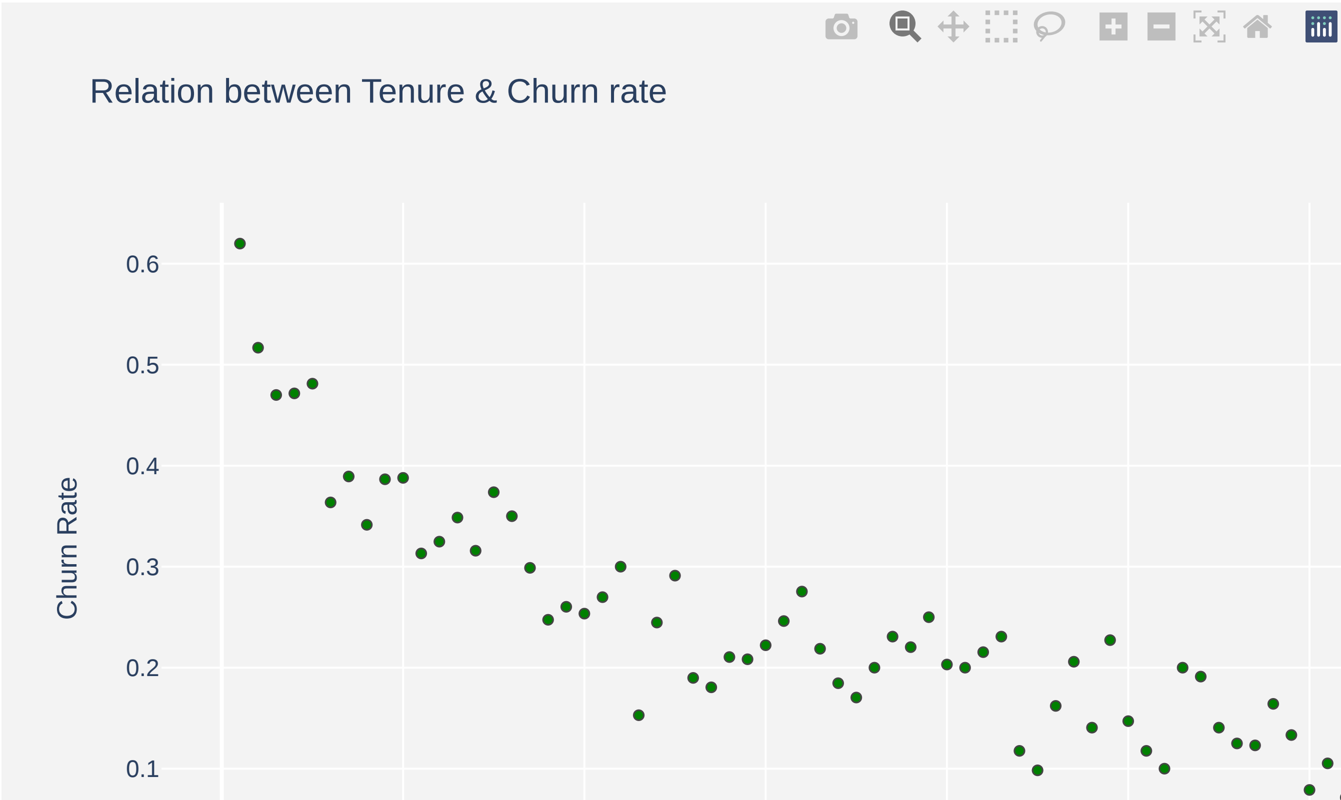
we have more people churning on the month to month contract compared to the one and two years contracts

In [15]:

```
# Visualize Relation between Tenure & Churn rate
df_tenure = df.groupby('tenure').Churn.mean().reset_index()
plot_data = [
    go.Scatter(
        x=df_tenure['tenure'],
        y=df_tenure['Churn'],
        mode='markers',
        name='Low',
        marker= dict(size= 5,
            line= dict(width=0.8),
            color= 'green'
           ),
    )
]
plot_layout = go.Layout(
```

Loading [MathJax]/extensions/Safe.js

```
        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)
print(df_tenure)
```




```
    tenure     Churn
0        1  0.619902
1        2  0.516807
2        3  0.470000
3        4  0.471591
4        5  0.481203
..     ...       ...
67      68  0.090000
68      69  0.084211
69      70  0.092437
70      71  0.035294
71      72  0.016575

[72 rows x 2 columns]
```

there is a negative relationship between tenure and churn rate, as tenure increase churn rate falls

Data Preprocessing

In [16]:

```
#Perform One Hot Encoding using get_dummies method
dummies(df, columns = ['Contract','Dependents','DeviceProtection','gender',
```

Loading [MathJax]/extensions/Safe.js

```
                                                    'InternetService','MultipleLines
                                                    'OnlineSecurity','PaperlessBilli
                                                    'PaymentMethod','PhoneService','
                                                    'StreamingMovies','StreamingTV',
                              drop_first=True)
df.columns
```

Out[16]:

```
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 [17]:

```
df.head()
```

Out[17]:

	customerID	tenure	MonthlyCharges	TotalCharges	Churn	Contract_One year	Contract_Two year	Dependents_Yes	De
0	7590-VHVEG	1	29.85	29.85	0	0	0	0	
1	5575-GNVDE	34	56.95	1889.50	0	1	0	0	
2	3668-QPYBK	2	53.85	108.15	1	0	0	0	
3	7795-CFOCW	45	42.30	1840.75	0	1	0	0	
4	9237-HQITU	2	70.70	151.65	1	0	0	0	

5 rows × 26 columns

In [18]:

```
#Perform Feature Scaling and One Hot Encoding
from sklearn.preprocessing import StandardScaler

#Perform Feature Scaling on 'tenure', 'MonthlyCharges', 'TotalCharges' in order to bring
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])
df.head()
```

Out[18]:

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

5 rows × 26 columns

In [19]:

```
df.columns
```

Out[19]:

```
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 [20]:

```
#Create Feature variable X and Target variable y

X = df.drop(['Churn','customerID'], axis = 1)

y = df['Churn'].astype("int")
```

In [21]:

```
#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
```

In [22]:

```
# Machine Learning classification model libraries
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

starting with logistic Regression model (used for binary classification)

In [23]:

```
#Fit the logistic Regression Model
log = LogisticRegression(random_state=50).fit(X_train,y_train)

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

# Find Accuracy using accuracy_score method
logmodel_accuracy = round(metrics.accuracy_score(y_test, pred) * 100, 2)
print("R-square of Logistic regression: ",logmodel_accuracy)
```

```
R-square of Logistic regression:  81.14
```

support vector machine model (svm)

In [24]:
```
#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)
print("R-square of SVC: ",svc_accuracy)
```

```
R-square of SVC:  80.66
```

K-nearest Neighbour model

In [25]:
```
#Fit the K-Nearest Neighbor Model
from sklearn.neighbors import KNeighborsClassifier
knnmodel = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2) #p=2 represents
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)
print("R-Squared for KNN: ",knn_accuracy)
```

```
2023-02-21 16:12:25,780 [14492] WARNING  py.warnings:109: [JupyterRequire] C:\Users\ADOW
UONA-OWOO\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWa
rning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mod
e` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will chang
e: the default value of `keepdims` will become False, the `axis` over which the statisti
c is taken will be eliminated, and the value None will no longer be accepted. Set `keepd
ims` to True or False to avoid this warning.
```

```
R-Squared for KNN:  76.82
```

Decision tree classification model

In [26]:
```
#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)
print("R-Squared for Decision tree: ", dt_accuracy)
```

```
R-Squared for Decision tree:  73.27
```

Random Forest Classification model

In [27]:
```
#Fit the Random Forest Classification Model
from sklearn.ensemble import RandomForestClassifier
rfmodel = RandomForestClassifier(n_estimators = 100, criterion = 'entropy', random_state
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)
print("R-square for Random forest: ", rf_accuracy)
```

```
R-square for Random forest:  79.38
```

In [28]:
```
# Compare Several models according to their Accuracies
Model_Comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Support Vector Machine', 'K-Nearest Neighbor',
              'Decision Tree', 'Random Forest'],
    'Score': [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[28]:

	Score	Model
0	81.14	Logistic Regression
1	80.66	Support Vector Machine
2	79.38	Random Forest
3	76.82	K-Nearest Neighbor
4	73.27	Decision Tree

Logistic regression has the highest accuracy score hence we compute the confusion matrix

In [29]:
```
#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[29]:
```
array([[1396,  165],
       [ 233,  316]], dtype=int64)
```

1396 and 316 are the correct predictions and 233 and 165 are incorrect predictions

In [31]:
```
# Predict the probability of Churn of each customer
df['Probability_of_Churn'] = log.predict_proba(df[X_test.columns])[:,1]
df.head(10)
```

Out[31]:

	customerID	tenure	MonthlyCharges	TotalCharges	Churn	Contract_One year	Contract_Two year	Dependents_Yes
0	7590-VHVEG	-1.280248	-1.161694	-0.994194	0	0	0	0
1	5575-GNVDE	0.064303	-0.260878	-0.173740	0	1	0	0
2	3668-QPYBK	-1.239504	-0.363923	-0.959649	1	0	0	0
3	7795-CFOCW	0.512486	-0.747850	-0.195248	0	1	0	0
4	9237-HQITU	-1.239504	0.196178	-0.940457	1	0	0	0
5	9305-CDSKC	-0.995040	1.158489	-0.645369	1	0	0	0
6	1452-KIOVK	-0.424625	0.807802	-0.147313	0	0	0	1
7	6713-OKOMC	-0.913552	-1.165018	-0.874169	0	0	0	0
8	7892-POOKP	-0.180161	1.329677	0.336516	1	0	0	0
9	6388-TABGU	1.205134	-0.287470	0.531476	0	1	0	1

10 rows × 27 columns

In [32]:

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

Out[32]:

	customerID	Probability_of_Churn
0	7590-VHVEG	0.649225
1	5575-GNVDE	0.043673
2	3668-QPYBK	0.340977
3	7795-CFOCW	0.026396
4	9237-HQITU	0.694569
5	9305-CDSKC	0.782005
6	1452-KIOVK	0.490814
7	6713-OKOMC	0.290564
8	7892-POOKP	0.594619
9	6388-TABGU	0.011939

for probability of churn that is close to 1 means the customer may leave soon so we need to take corrective actions for such customers.

In [ ]: