Springboard Data Science Bootcamp – Capstone 2 – Predict Customer Churn

Objective:

Customer Churn measures the loss of customers and service provider companies use this metric to understand the customer retention. The objective is to predict behavior to retain customers by analyzing all relevant customer data and develop focused customer retention programs.

Problem:
Is the customer going churn?

Outcome:
When a customer stops service or company losing customer is referred to as Customer Churn. This is an important measure for any service-based company. The model predictions an provide the propensity of churning and gives the companies with the feature's importance that leads the customer to churn. With the list of potential customers who are likely to churn, the marketing/retention teams can then take measure to reduce their churn probability. This project helps companies in identifying customer who are at risk of churning and we have used this IBM sample data set provided for a telecom company. We will be using statistical analysis to understand variables that are associated with customer churn.

Dataset:

- Customers who left within the last month – the column is called Churn
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

Data set contains 7043 rows and 21 columns, see below for more information:

- *customerID*: Customer ID
- *genderCustomer*: gender (female, male)
- *SeniorCitizen*: Whether the customer is a senior citizen or not (1, 0)
- *PartnerWhether*: the customer has a partner or not (Yes, No)
- *Dependents*: Whether the customer has dependents or not (Yes, No)
- *tenure*: Number of months the customer has stayed with the company
- *PhoneService*: Whether the customer has a phone service or not (Yes, No)
- *MultipleLines*: Whether the customer has multiple lines or not (Yes, No, No phone service)
- *InternetService*: Customer's internet service provider (DSL, Fiber optic, No)
- *OnlineSecurity*: Whether the customer has online security or not (Yes, No, No internet service)
- *OnlineBackup*: Whether the customer has online backup or not (Yes, No, No internet service)
- *DeviceProtection*: Whether the customer has device protection or not (Yes, No, No internet service)
- *TechSupport*: Whether the customer has tech support or not (Yes, No, No internet service)
- *StreamingTV*: Whether the customer has streaming TV or not (Yes, No, No internet service)
- *StreamingMovies*: Whether the customer has streaming movies or not (Yes, No, No internet service)
- *Contract*: The contract term of the customer (Month-to-month, One year, Two year)
- *PaperlessBilling*: Whether the customer has paperless billing or not (Yes, No)
- *PaymentMethod*: The customer's payment method (Electronic check, mailed check, Bank transfer (automatic), Credit card (automatic))
- *MonthlyCharges*: The amount charged to the customer monthly
- *TotalCharges*: The total amount charged to the customer
- *Churn*: Whether the customer churned or not (Yes or No)

There are many categorical variables in this data set. The numerical features are Tenure, MonthlyCharges and TotalCharges.

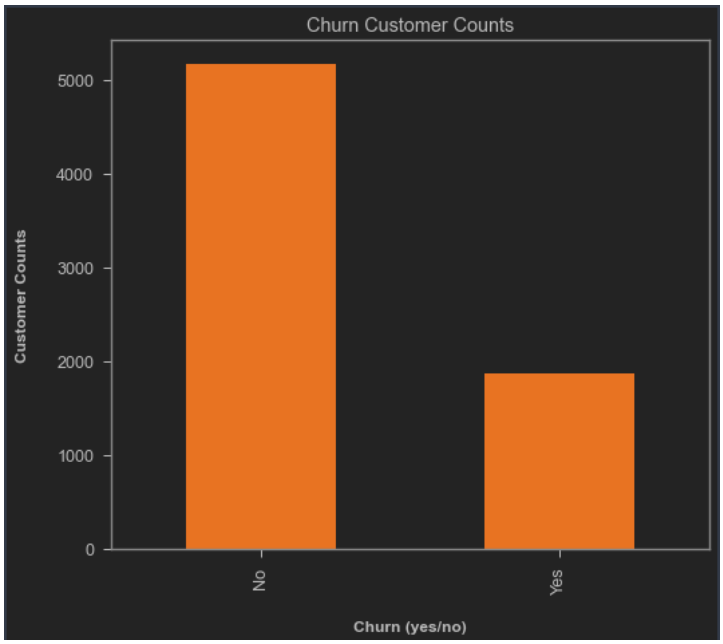
Descriptive Stats:

We see that Tenure ranges from 0 (new customer) to 6 years, Monthly charges range from $18 to $118, etc

tenure
7043.000000
NaN
NaN
NaN
32.371149
24.559481
0.000000
9.000000
29.000000
55.000000
72.000000

MonthlyCharges
7043.000000
NaN
NaN
NaN
64.761692
30.090047
18.250000
35.500000
70.350000
89.850000
118.750000

Roughly a quarter of the customers have churned in this data set.



Looking at the coloration matrix, there seems to be some positive correlation between Monthly Charges and Churn and some negative correlation between tenure and Churn.

Data Wrangling:

Replacing the Churn string value (yes/no) to numbers (0 – no and 1 – yes)

```
# converting churn to numerical variable for analyzing
df.loc[df.Churn=='No','Churn'] = 0
df.loc[df.Churn=='Yes','Churn'] = 1
df['Churn'] = df['Churn'].astype(int)
```

Total Charges column was read as object, to fix this, we are converting its data type.

```
#Total charges is obect in the data types converting to number
totalCharges = df.columns.get_loc("TotalCharges")
new_col = pd.to_numeric(df.iloc[:, totalCharges], errors='coerce')
df.iloc[:, totalCharges] = pd.Series(new_col)
```

Then investigate for missing values, it looks like TotalCharges has missing values.

```
print(df.isnull().values.any())
df.isnull().sum()

# Looks like Total charges has missing values
```

```
True
```

```
customerID          0
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       11
Churn               0
dtype: int64
```

Applying imputation to fix the issue with missing values with means of TotalCharges using SimpleImputer from sklearn.impute.

```python
# Handle missing values for nan_column (TotalCharges)
from sklearn.impute import SimpleImputer

# Find the column number for TotalCharges (starting at 0).
total_charges_idx = df.columns.get_loc("TotalCharges")
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

df.iloc[:, total_charges_idx] = imputer.fit_transform(df.iloc[:, total_charges_idx].values.reshape(-1, 1))
df.iloc[:, total_charges_idx] = pd.Series(df.iloc[:, total_charges_idx])
```

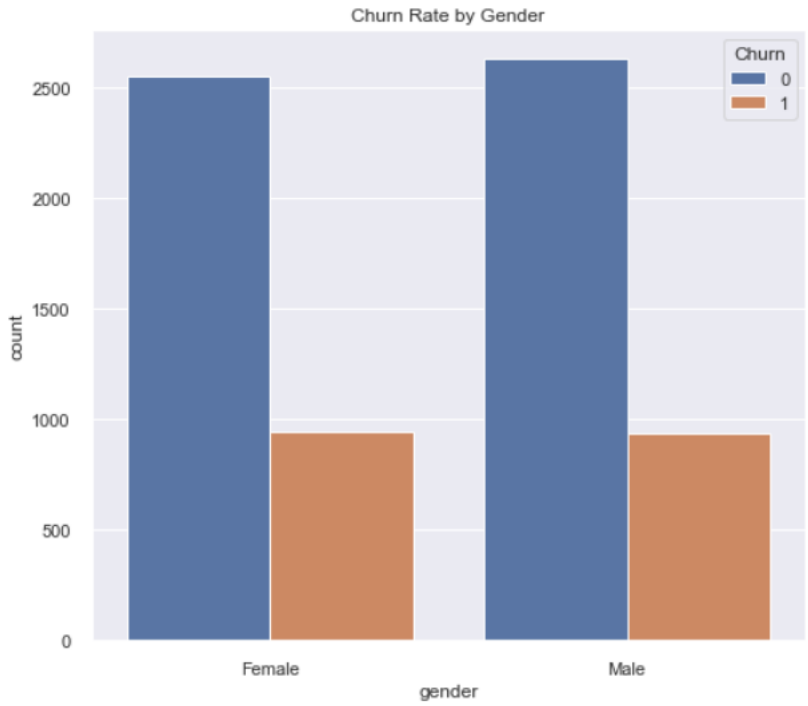
Identified the categorical, numerical and continuous features in the data set.

```python
columns_idx = np.s_[0:] # Slice of first row(header) with all columns.
first_record_idx = np.s_[0] # Index of first record

string_fields = [type(fld) is str for fld in df.iloc[first_record_idx, columns_idx]] # All string fields
all_features = [x for x in df.columns if x != 'Churn']
categorical_columns = list(np.array(df.columns)[columns_idx][string_fields])
categorical_features = [x for x in categorical_columns if x != 'Churn']
continuous_features = [x for x in all_features if x not in categorical_features]

print('All Features: ', all_features)
print('\nCategorical Features: ', categorical_features)
print('\nContinuous Features: ', continuous_features)
print('\nAll Categorical Columns: ', categorical_columns)
```

```
All Features:  ['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtecti
on', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges']

Categorical Features:  ['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']

Continuous Features:  ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges']

All Categorical Columns:  ['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSuppor
t', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']
```

EDA - Data visualization:

Looking at the churn rate by gender there seems to a very little higher % of female customer churning than compared to male customers, however, I don't think this is a big enough difference to be considered.

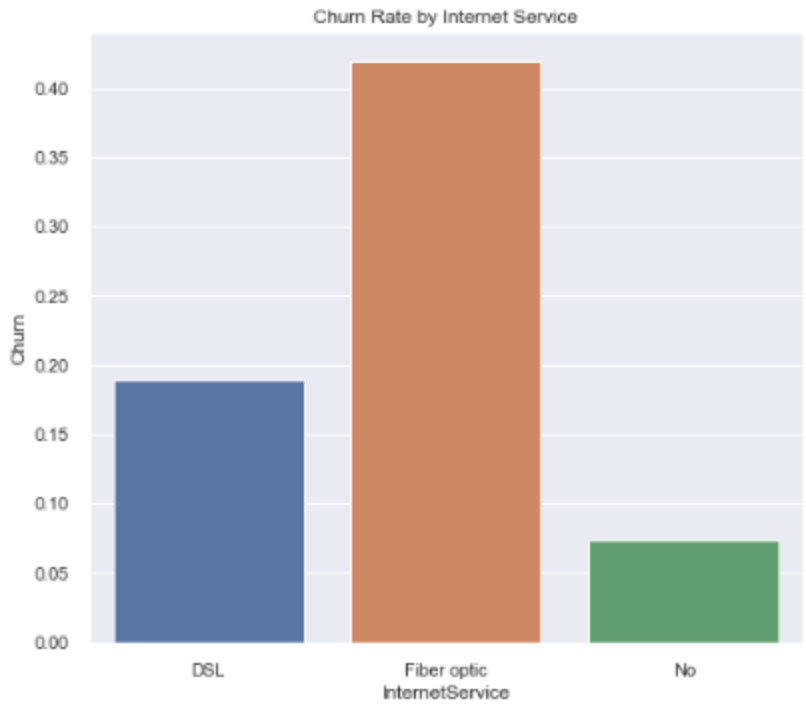



```
# plotting gender
mean_churn = df[['gender','Churn']].groupby('gender').mean()
print(mean_churn)
print("Female churn is little higher, but not that big difference")
```

```
           Churn
gender
Female  0.269209
Male    0.261603
Female churn is little higher, but not that big difference
```

Looking at the churn rate by Internet service type there seems to a high % of customers churning that are Fiber Optic service. This also could be due to Fiber optic being the most used service type.

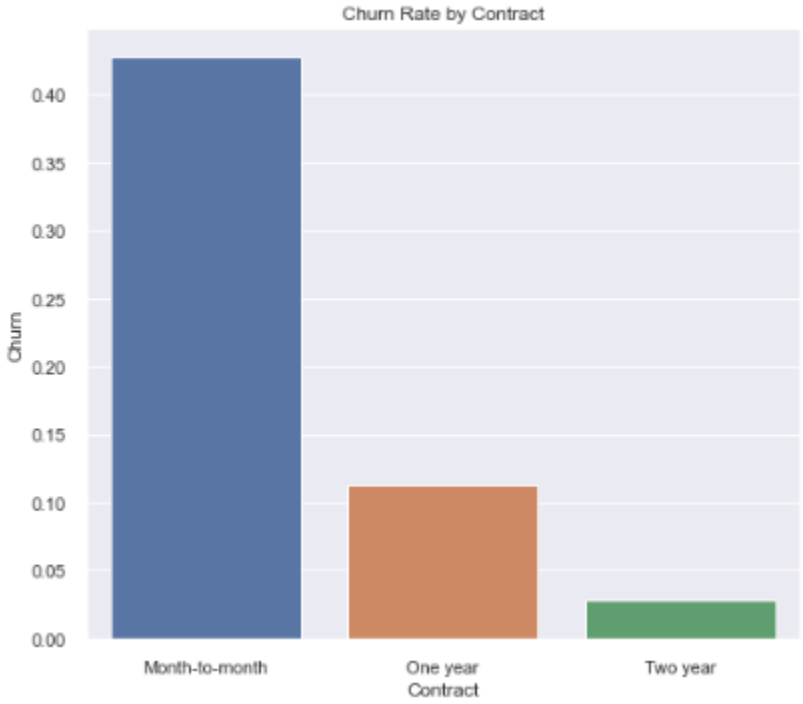
Fiber Optic customers are churning at a higher rate:




```
# plotting Internet Servive type
isp_churn = df[['InternetService','Churn']].groupby('InternetService').mean()
print(isp_churn)
print("Fiber Optic customers are churning at a higher rate")
```

```
                   Churn
InternetService
DSL             0.189591
Fiber optic     0.418928
No              0.074050
Fiber Optic customers are churning at a higher rate
```

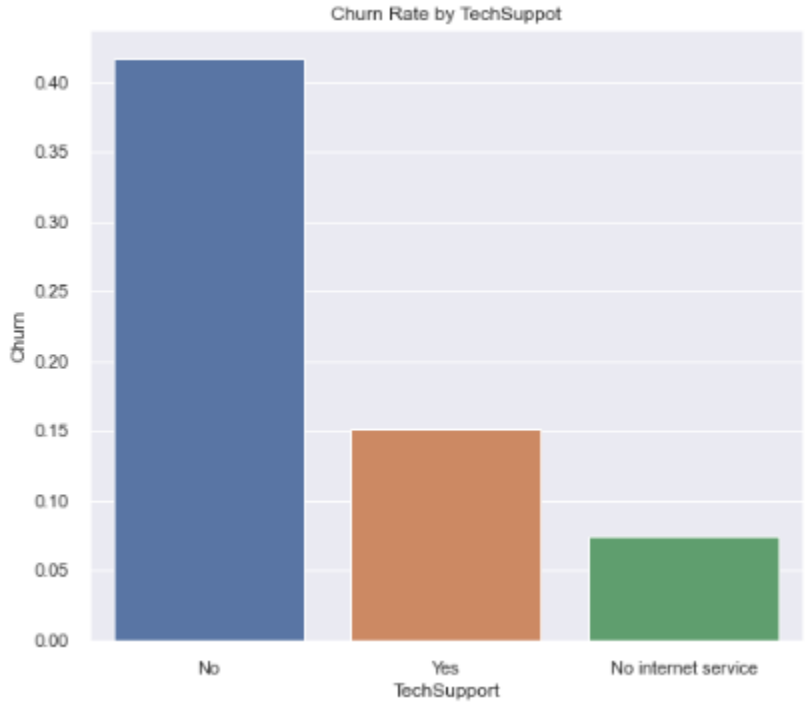
Month to Month subscribers are churning at a higher rate




```
# plotting Churn rate by Contract type
cont_churn = df[['Contract','Churn']].groupby('Contract').mean()
print(cont_churn)
print("Month to Month subscribers are churning at a higher rate")
```

```
                 Churn
Contract
Month-to-month  0.427097
One year        0.112695
Two year        0.028319
Month to Month subscribers are churning at a higher rate
```

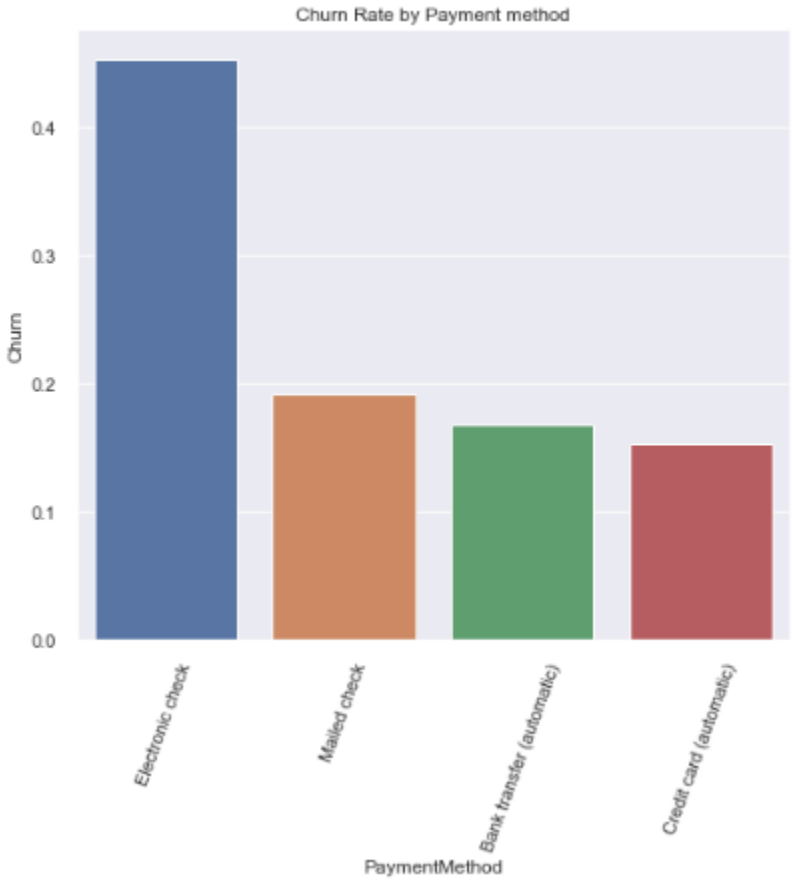
Customers who did not use Tech Support are churning at a higher rate




```
# plotting Churn rate by TechSupport
tech_churn = df[['TechSupport','Churn']].groupby('TechSupport').mean()
print(tech_churn)
print("Customers who did not use  Tech Support are churning at a higher rate")
```

```
                       Churn
TechSupport
No                  0.416355
No internet service  0.074050
Yes                 0.151663
Customers who did not use  Tech Support are churning at a higher rate
```

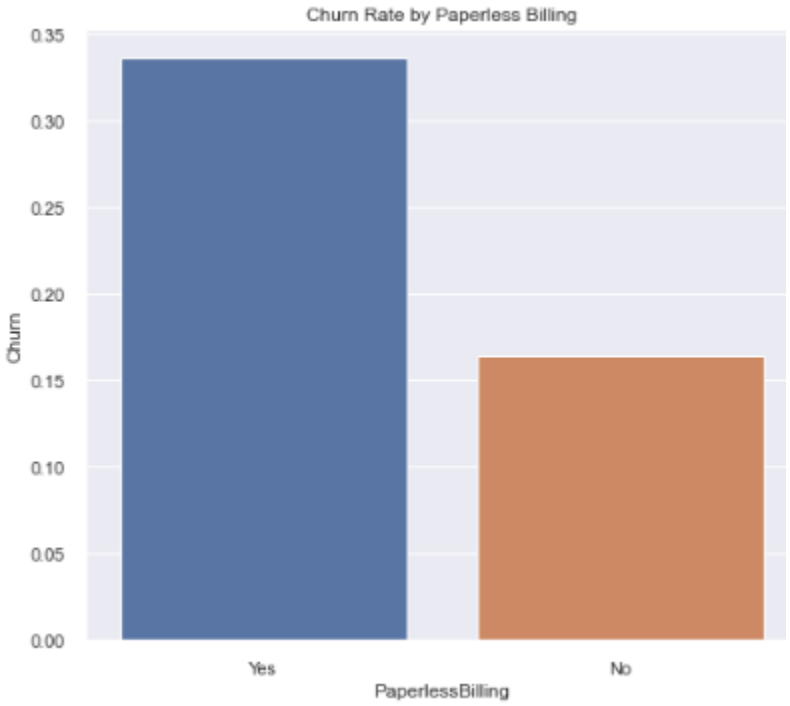
Customers who use checks as payment method are churning at higher rate




```
# plotting PaymentMethod
paym_churn = df[['PaymentMethod','Churn']].groupby('PaymentMethod').mean()
print(paym_churn)
print("Customers who use checks as payment method are churning at higher rate")
```

```
                           Churn
PaymentMethod
Bank transfer (automatic)  0.167098
Credit card (automatic)    0.152431
Electronic check           0.452854
Mailed check               0.191067
Customers who use checks as payment method are churning at higher rate
```

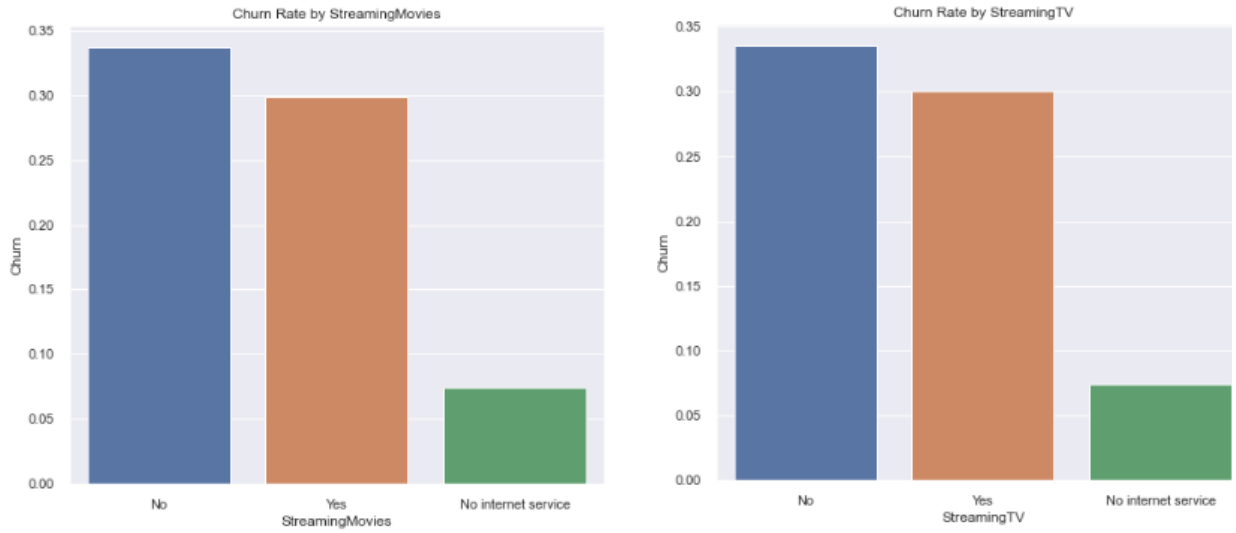
Customers who are on paperless billing have a higher churn rate. But this is could be very well because a lot of customers are enrolled into paperless billing.




```
# plotting PaperlessBilling
ebill_churn = df[['PaperlessBilling','Churn']].groupby('PaperlessBilling').mean()
print(ebill_churn)
print("Customers who are on paperless billing are churning at higher rate")
```

```
                   Churn
PaperlessBilling
No              0.163301
Yes             0.335651
Customers who are on paperless billing are churning at higher rate
```

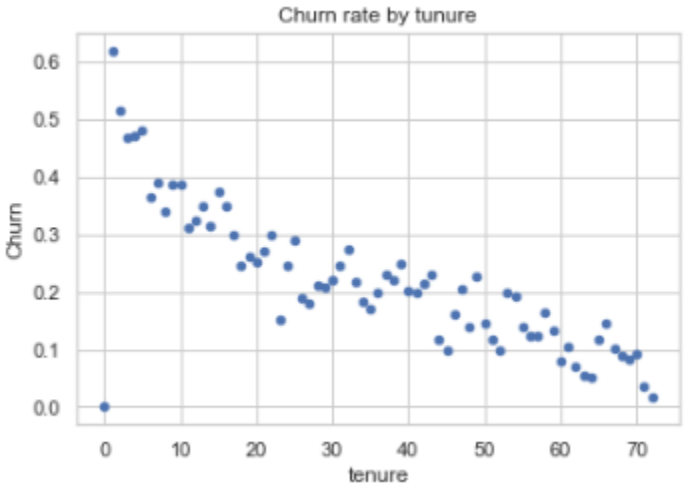
Customer who are Steaming movies vs Steaming tv seems to have very similar Churn rates:



Let's look at some summary stats for tenure variable:

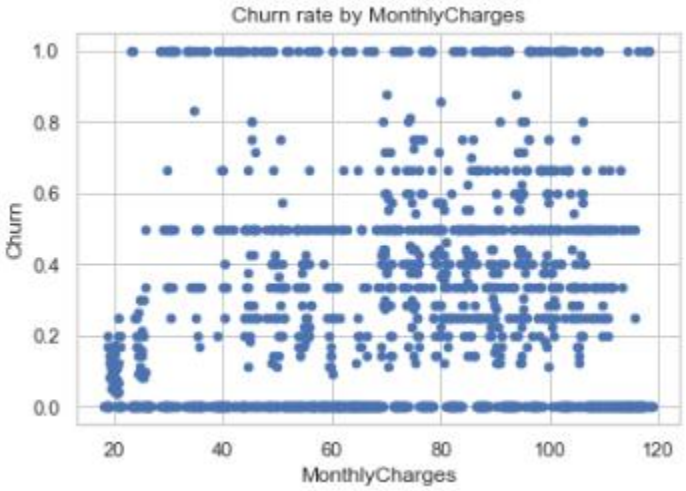
```
count    7043.000000
mean       32.371149
std        24.559481
min         0.000000
25%         9.000000
50%        29.000000
75%        55.000000
max        72.000000
Name: tenure, dtype: float64
```

Now, we will take a look at how mean churn rates are doing when compared with tenure.



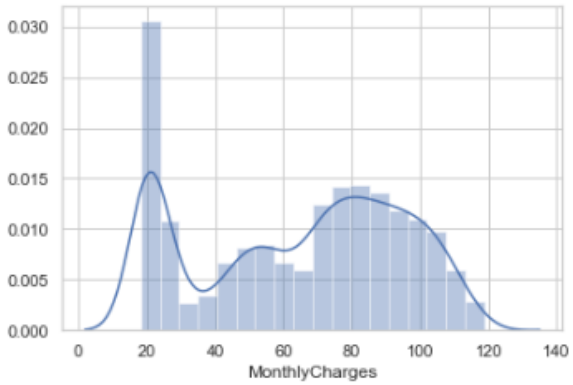
It shows that if the customer's tenure is long then churn rate is low.

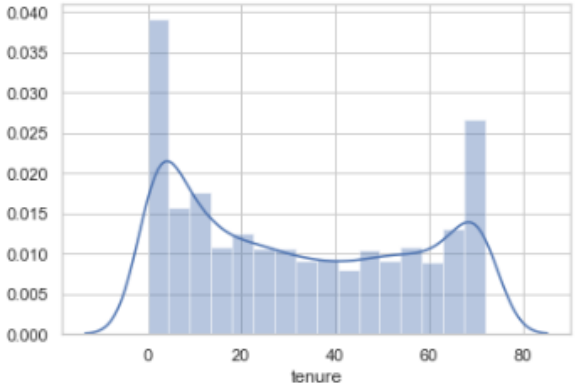
Let's also look at other numerical variables such as MonthlyCharges:



There seems to be no relation for Churn and MonthlyCharges

Distribution plots for MonthlyCharges and tenure:





Monthly Charges seems to be roughly normal distribution and Tenure Distribution seems to be high at the ends, so a portion of the customers have either had lowest and highest tenure periods.

EDA Summary:

- More customers are using Fiber Optic for Internet Service have left the company than compared to DSL.
- Customers who do not use online security have left the company.
- Customers not using technical support have left the company.
- Customers who pay month to month are the most who leave the company.
- Customer's gender has almost equal rates of churn between them.
- The Monthly Charges for customers who churned tends to pay higher monthly fees than those that stay.
- Customers that churn tend to be relatively new customers when looking at tenure distribution.

Data Pre-processing:

- Categorical data label encoding
- Machine learning works with only numerical values. Therefore, we need to convert our categorical values to numerical values. By using the Pandas function “get_dummies()”, we can replace the gender column with “gender_Female” and “gender_Male”. We will use df.info() to show us which ones are categorical and numerical.

```
#import Label Encoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
dummy_columns = [] #array for multiple value columns

for column in df.columns:
    if df[column].dtype == object and column != 'customerID':
        if df[column].nunique() == 2:
            #apply Label Encoder for binary ones
            df[column] = le.fit_transform(df[column])
        else:
            dummy_columns.append(column)

#apply get dummies for selected columns
df_clean_data = pd.get_dummies(data = df,columns = dummy_columns)
```

- Dropping unwanted variables and target variable
- In our dataset, we can see that customer ID is not needed for our model, so we drop the variable. We do not need to treat missing values as there are none in this dataset.

```
# Dropping varibles that are not neeeded and also target variables
X = df_clean_data.drop(['Churn','customerID'], axis=1)

# target variable
y = df_clean_data['Churn']
```

- Train on one set and test and measure on the test set.
- Our model needs to be trained; second our model needs to be tested. Therefore, it is best to have two different datasets.

- Splitting the data 70% for training and 30% for testing.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=123,
                                                    stratify=y)
```

- Realized that the data set is facing a class imbalance so apply Under Sampling technique to standardize the dataset.

```
# import the StandardScaler
from sklearn.preprocessing import StandardScaler

# initiate a StandardScaler and fit to training feature set.
scaler = StandardScaler().fit(X_train)

# Scale the training data using transform
X_train = scaler.transform(X_train)

from imblearn.under_sampling import RandomUnderSampler

# initiante a random under sampler
rus = RandomUnderSampler()

# Store in new random undersampled variables
X_train_rus, y_train_rus = rus.fit_sample(X_train, y_train)
```

Machine Learning:

The customer churn is a classification problem and have choose the following classification models to predict if a customer churns by fitting the training data set and testing on the test set. Used scikit-learn and other libraries in the python machine learning world.

1. Logistic Regression
2. Decision Tree model
3. Random Forest Classifier
4. Extreme gradient boosting (xGBoost)

Model performance metrics

Accuracy = Pct of correctly classified predictions (i.e, churn and non-churn)

Precision = Pct of total positive class predictions that were correctly classified

Recall = Pct of total positive class samples that were correctly classified

<u>Logistic Regression:</u>

This type of statistical classification modelling is commonly used for predicting binary variables. It models the log odds of the probability of the target variable.

The accuracy train accuracy and the testing accuracy was around 80%

Confusion matrix shows the True Positives, True Negatives, False Positives and False Negatives.

The models seem to have maintaining recall rate between training and test models.

```python
# import Logistic regression model
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import accuracy_score

#Initialize the log reg model instance
lr = LogisticRegression()

# fit the training data
lr.fit(X_train, y_train)

# Measureing the model accuracy
y_pred_train = lr.predict(X_train)
y_pred_test = lr.predict(X_test)

train_acc = accuracy_score(y_train, y_pred_train)
test_acc = accuracy_score(y_test, y_pred_test)

print('Train accuracy:', round(train_acc, 4))
print('Test accuracy:', round(test_acc, 4))
print('\n')

train_prec = round(precision_score(y_train,y_pred_train),4)
test_prec = round(precision_score(y_test,y_pred_test),4)

train_rec = round(recall_score(y_train,y_pred_train),4)
test_rec = round(recall_score(y_test,y_pred_test),4)

print('Training precision: {}'.format(train_prec))
print('Training recall: {}'.format(train_rec))
print('\n')
print('Test precision: {}'.format(test_prec))
print('Test recall: {}'.format(test_rec))
print('\n')

print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred_test))
print('\n')
print("Accuracy Score:")
print(metrics.accuracy_score(y_test, y_pred_test))
```

Evaluation metrics:

```
Train accuracy: 0.8049
Test accuracy: 0.8064


Training precision: 0.6629
Training recall: 0.5382


Test precision: 0.6767
Test recall: 0.5187


Confusion Matrix:
[[1413  139]
 [ 270  291]]


Accuracy Score:
0.8064363464268812
```

Decision Tree model:

First tree-based model that splits the data multiple times based on the cutoff values defined in the features.

Model parameters: criterion = gini; max_depth=5;random_state=1

The accuracy train accuracy and the testing accuracy was around 79%

TP=1374

TN=303

FP=178

FN=258

The models seem to decent accuracy and precision and recall rates between train and test data set.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Instantiate a DecisionTreeClassifier 'dt' with a maximum depth of 6
dt = DecisionTreeClassifier(criterion ='gini', max_depth=5, random_state=1)

# Fit dt to the training set
dt.fit(X_train,y_train)

# Predict test set labels
y_pred_dt_train = dt.predict(X_train)
y_pred_dt_test = dt.predict(X_test)

# Compute test set accuracy
#acc = accuracy_score(y_test, y_pred)
#print("Test set accuracy: {:.2f}".format(acc))

train_acc_dt = accuracy_score(y_train, y_pred_dt_train)
test_acc_dt = accuracy_score(y_test, y_pred_dt_test)

print('Train accuracy:', round(train_acc_dt, 4))
print('Test accuracy:', round(test_acc_dt, 4))
print('\n')

train_prec_dt = round(precision_score(y_train,y_pred_dt_train),4)
test_prec_dt = round(precision_score(y_test,y_pred_dt_test),4)

train_rec_dt = round(recall_score(y_train,y_pred_dt_train),4)
test_rec_dt = round(recall_score(y_test,y_pred_dt_test),4)

print('Training precision: {}'.format(train_prec_dt))
print('Training recall: {}'.format(train_rec_dt))
print('\n')
print('Test precision: {}'.format(test_prec_dt))
print('Test recall: {}'.format(test_rec_dt))
print('\n')

print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred_dt_test))
print('\n')
print("Accuracy Score:")
print(metrics.accuracy_score(y_test, y_pred_dt_test))
```

Evaluation metrics:

```
Train accuracy: 0.8065
Test accuracy: 0.7937


Training precision: 0.6547
Training recall: 0.5726


Test precision: 0.6299
Test recall: 0.5401


Confusion Matrix:
[[1374  178]
 [ 258  303]]


Accuracy Score:
0.7936583057264552
```

Random Forest

This is the second tree-based classification models we have experiments in the model phase. The difference between the previous model is it using bagging to define the boundaries in Random Forest models.

Model parameters include n_estimators=50; criterion=entropy' min_samples_split=285

The accuracy train accuracy and the testing accuracy was around 79%

The models seem to have low recall rate and higher precession rate.

The model predicted more False Negatives compared to the previous Decision tree model.

```
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier

# Start an instance of RandomForestClassifier
RF = RandomForestClassifier(n_estimators = 50, criterion = 'entropy', min_samples_split = 285)

# fit the model to trainig data
RF.fit(X_train, y_train)

# predict the labels
y_pred_rf_train = RF.predict(X_train)
y_pred_rf_test = RF.predict(X_test)

f1_rf = metrics.f1_score(y_pred_rf_test, y_test)

print("F1 Score (test):"+str(f1_rf))

metrics.classification_report(y_test, y_pred_rf_test)

print(metrics.classification_report(y_test, y_pred_rf_test))

train_prec_rf = round(precision_score(y_train,y_pred_rf_train),4)
test_prec_rf = round(precision_score(y_test,y_pred_rf_test),4)

train_rec_rf = round(recall_score(y_train,y_pred_rf_train),4)
test_rec_rf = round(recall_score(y_test,y_pred_rf_test),4)

print('Training precision: {}'.format(train_prec_rf))
print('Training recall: {}'.format(train_rec_rf))
print('\n')
print('Test precision: {}'.format(test_prec_rf))
print('Test recall: {}'.format(test_rec_rf))

print('\n')

print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred_rf_test))
print('\n')
print("Accuracy Score:")
print(metrics.accuracy_score(y_test, y_pred_rf_test))
print("\n")

importance = zip(RF.feature_importances_, X)

for rank in sorted(importance, key = lambda x: x[0], reverse = True):
    print(rank)
```

Evaluation metrics:

```
F1 Score (test):0.4942528735632184
              precision    recall  f1-score   support

           0       0.81      0.94      0.87      1552
           1       0.70      0.38      0.49       561

    accuracy                           0.79      2113
   macro avg       0.75      0.66      0.68      2113
weighted avg       0.78      0.79      0.77      2113

Training precision: 0.7055
Training recall: 0.4213


Test precision: 0.6958
Test recall: 0.3832


Confusion Matrix:
[[1458   94]
 [ 346  215]]


Accuracy Score:
0.7917652626597255
```

Feature importance:

```
(0.1879207270248737, 'Contract_Month-to-month')
(0.11284650561596918, 'Contract_Two year')
(0.10806257606188054, 'OnlineSecurity_No')
(0.10471715403124655, 'tenure')
(0.07692005102945117, 'TotalCharges')
(0.06968666921458776, 'TechSupport_No')
(0.05234377185200279, 'MonthlyCharges')
(0.039030635347902326, 'InternetService_Fiber optic')
(0.03319961592145481, 'PaymentMethod_Electronic check')
(0.030193193411427354, 'OnlineBackup_No')
(0.026137524973344362, 'Contract_One year')
(0.02172726731833357, 'InternetService_DSL')
(0.020071852660975175, 'InternetService_No')
(0.014122446762522518, 'StreamingMovies_No internet service')
(0.009771629958891456, 'OnlineBackup_No internet service')
(0.009043407444789216, 'OnlineSecurity_No internet service')
(0.008323410990405023, 'OnlineSecurity_Yes')
(0.008133091349550465, 'TechSupport_Yes')
(0.007402325649217218, 'OnlineBackup_Yes')
(0.006298065650423189, 'DeviceProtection_No internet service')
(0.006004197845796403, 'TechSupport_No internet service')
(0.005833233955297278, 'StreamingTV_No internet service')
(0.004656246491838335, 'SeniorCitizen')
(0.004613171514422696, 'StreamingMovies_Yes')
(0.004499559319886553, 'Partner')
(0.003827242713864532, 'DeviceProtection_No')
(0.0033673289736262427, 'StreamingTV_Yes')
(0.0032246318604002375, 'PaperlessBilling')
(0.0030617464634349087, 'Dependents')
(0.0026909529825503847, 'DeviceProtection_Yes')
(0.0025497907785194517, 'StreamingMovies_No')
(0.0022121197062000136, 'PaymentMethod_Mailed check')
(0.0016962921997061052, 'PaymentMethod_Credit card (automatic)')
(0.001641964225790931, 'MultipleLines_Yes')
(0.0012197791463175382, 'StreamingTV_No')
(0.0008576520544336781, 'PaymentMethod_Bank transfer (automatic)')
(0.0008308913869217064, 'MultipleLines_No')
(0.0004610386922999683 7, 'gender')
(0.0004506833882795072, 'MultipleLines_No phone service')
(0.0003495540311650109, 'PhoneService')
```

<u>xGBoost (Extreme Gradient Boosting)</u>

This is an open-source gradient boosted model which attempts to accurately predict target by combining the estimates of set of simpler, weaker models. This is a type of ensemble models.

The accuracy train accuracy and the testing accuracy was around 79%

The model prediction a lot of True negatives.

The models have low precision and recall scores compared to previous models.

```
import xgboost as xgb
from matplotlib import pyplot

# instansitate xbg classifier
xg_cl = xgb.XGBRFClassifier(objective='binary:logistic', n_estimators=10,seed=123)

# Fit on train
xg_cl.fit(X_train, y_train)

#Predict on Test
y_pred_xgb_train = xg_cl.predict(X_train)
y_pred_xgb_test = xg_cl.predict(X_test)

#Accuracy Scores
accuracy_xgb = float(np.sum(y_pred_xgb_test==y_test))/y_test.shape[0]

print('accuracy: %f' % (accuracy_xgb))

f1_xgb = metrics.f1_score(y_pred_xgb_test, y_test)

print("F1 Score:"+str(f1_xgb))

metrics.classification_report(y_test, y_pred_xgb_test)

print(metrics.classification_report(y_test, y_pred_xgb_test))

print("\n")

train_prec_xgb = round(precision_score(y_train,y_pred_xgb_train),4)
test_prec_xgb = round(precision_score(y_test,y_pred_xgb_test),4)

train_rec_xgb = round(recall_score(y_train,y_pred_xgb_train),4)
test_rec_xgb = round(recall_score(y_test,y_pred_xgb_test),4)

print('Training precision: {}'.format(train_prec_xgb))
print('Training recall: {}'.format(train_rec_xgb))
print('\n')
print('Test precision: {}'.format(test_prec_xgb))
print('Test recall: {}'.format(test_rec_xgb))

print('\n')

print("Confusion Matrix:")
print(metrics.confusion_matrix(y_test, y_pred_xgb_test))

print('\n')

print("Accuracy Score:")
print(metrics.accuracy_score(y_test, y_pred_xgb_test))

print("\n")
```

Evaluation metrics:

```
F1 Score (test):0.4942528735632184
              precision    recall  f1-score   support

           0       0.81      0.94      0.87      1552
           1       0.70      0.38      0.49       561

    accuracy                           0.79      2113
   macro avg       0.75      0.66      0.68      2113
weighted avg       0.78      0.79      0.77      2113

Training precision: 0.7055
Training recall: 0.4213


Test precision: 0.6958
Test recall: 0.3832


Confusion Matrix:
[[1458   94]
 [ 346  215]]


Accuracy Score:
0.7917652626597255
```

Feature importance's: