Predicting Telco Customer Churn

Creating a machine learning model to predict customer churn. In this notebook we will build the prediction model using the SparkML library.

In [2]:
```
import pandas as pd
import numpy as np
import json
import os

# Import the Project Library to read/write project assets
from project_lib import Project
project = Project.access()

import warnings
warnings.filterwarnings("ignore")
```

1.0 Load and Clean data

We'll load our data as a pandas data frame.

In [3]:

```
# Place cursor below and insert the Pandas DataFrame for the Telco churn data
df_data_1 = pd.read_csv('Telco-Customer-Churn.csv')
df_data_1.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:

```
# for virtualized data
# df = data_df_1

# for local upload
df = df_data_1
```

1.1 Drop CustomerID feature (column)

In [5]:
```
df = df.drop('customerID', axis=1)
df.head(5)
```

Out[5]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtectio
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	N
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Ye
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	N
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Ye
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	N

1.2 Examine the data types of the features

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

In [7]:
```
# Statistics for the columns (features). Set it to all, since default is to describe just the numeric features.
df.describe(include = 'all')
```

Out[7]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	Devi
count	7043	7043.000000	7043	7043	7043.000000	7043	7043	7043	7043	7043	
unique	2	NaN	2	2	NaN	2	3	3	3	3	
top	Male	NaN	No	No	NaN	Yes	No	Fiber optic	No	No	
freq	3555	NaN	3641	4933	NaN	6361	3390	3096	3498	3088	
mean	NaN	0.162147	NaN	NaN	32.371149	NaN	NaN	NaN	NaN	NaN	
std	NaN	0.368612	NaN	NaN	24.559481	NaN	NaN	NaN	NaN	NaN	
min	NaN	0.000000	NaN	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	
25%	NaN	0.000000	NaN	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	
50%	NaN	0.000000	NaN	NaN	29.000000	NaN	NaN	NaN	NaN	NaN	
75%	NaN	0.000000	NaN	NaN	55.000000	NaN	NaN	NaN	NaN	NaN	
max	NaN	1.000000	NaN	NaN	72.000000	NaN	NaN	NaN	NaN	NaN	

1.3 Check for need to Convert TotalCharges column to numeric if it is detected as object

If the above `df.info` shows the "TotalCharges" columnn as an object, we'll need to convert it to numeric. If you have already done this during a previous exercise for "Data Visualization with Data Refinery", you can skip to step `2.4`.

In [8]:
```
totalCharges = df.columns.get_loc("TotalCharges")
new_col = pd.to_numeric(df.iloc[:, totalCharges], errors='coerce')
df.iloc[:, totalCharges] = pd.Series(new_col)
```

In [9]:
```
# Statistics for the columns (features). Set it to all, since default is to describe just the numeric features.
df.describe(include = 'all')
```

Out[9]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	Devi
count	7043	7043.000000	7043	7043	7043.000000	7043	7043	7043	7043	7043	
unique	2	NaN	2	2	NaN	2	3	3	3	3	
top	Male	NaN	No	No	NaN	Yes	No	Fiber optic	No	No	
freq	3555	NaN	3641	4933	NaN	6361	3390	3096	3498	3088	
mean	NaN	0.162147	NaN	NaN	32.371149	NaN	NaN	NaN	NaN	NaN	
std	NaN	0.368612	NaN	NaN	24.559481	NaN	NaN	NaN	NaN	NaN	
min	NaN	0.000000	NaN	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	
25%	NaN	0.000000	NaN	NaN	9.000000	NaN	NaN	NaN	NaN	NaN	
50%	NaN	0.000000	NaN	NaN	29.000000	NaN	NaN	NaN	NaN	NaN	
75%	NaN	0.000000	NaN	NaN	55.000000	NaN	NaN	NaN	NaN	NaN	
max	NaN	1.000000	NaN	NaN	72.000000	NaN	NaN	NaN	NaN	NaN	

1.4 Any NaN values should be removed to create a more accurate model.

In [10]:
```
# Check if we have any NaN values and see which features have missing values that should be addressed
print(df.isnull().values.any())
df.isnull().sum()
```

```
True
```

Out[10]:
```
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
```

We should see that the `TotalCharges` column has missing values. There are various ways we can address this issue:

- Drop records with missing values
- Fill in the missing value with one of the following strategies: Zero, Mean of the values for the column, Random value, etc).

In [11]:
```
# 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])
```

In [12]:
```
# Validate that we have addressed any NaN values
print(df.isnull().values.any())
df.isnull().sum()
```

```
False
```

Out[12]:
```
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

1.5 Categorize Features

We will categorize some of the columns / features based on wether they are categorical values or continuous (i.e numerical) values. We will use this in later sections to build visualizations.

In [13]:
```
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)
```

1.6 Visualize data

Data visualization can be used to find patterns, detect outliers, understand distribution and more. We can use graphs such as:

- Histograms, boxplots, etc: To find distribution / spread of our continuous variables.
- Bar charts: To show frequency in categorical values.

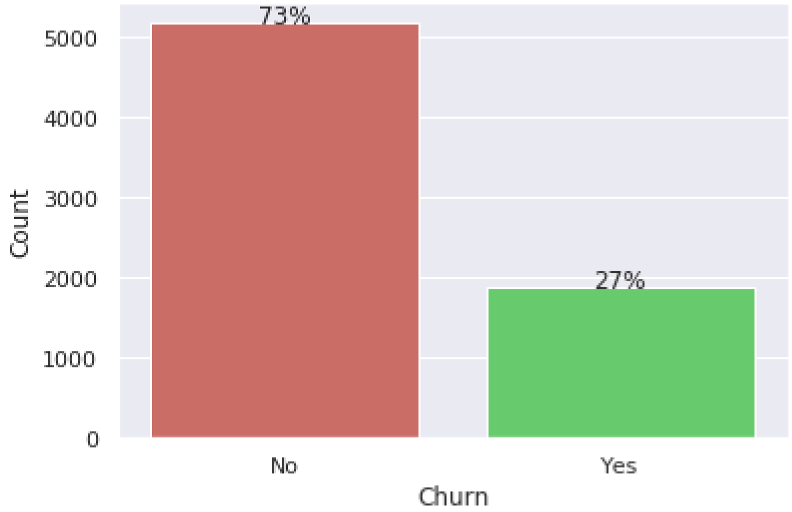
In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

%matplotlib inline
sns.set(style="darkgrid")
sns.set_palette("hls", 3)
```

In [15]:
```
print(df.groupby(['Churn']).size())
churn_plot = sns.countplot(data=df, x='Churn', order=df.Churn.value_counts().index)
plt.ylabel('Count')
for p in churn_plot.patches:
    height = p.get_height()
    churn_plot.text(p.get_x()+p.get_width()/2., height + 1,'{0:.0%}'.format(height/float(len(df))),ha="center")
plt.show()
```

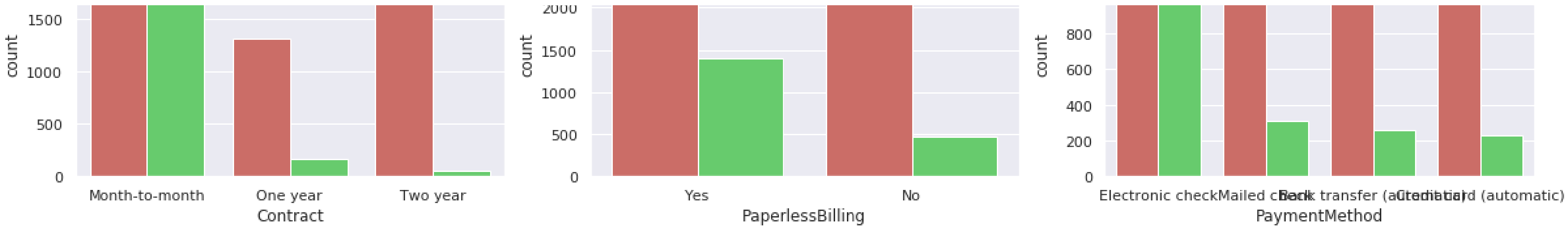
```
Churn
No     5174
Yes    1869
dtype: int64
```



In [16]:
```
# Categorical feature count plots
f, ((ax1, ax2, ax3), (ax4, ax5, ax6), (ax7, ax8, ax9), (ax10, ax11, ax12), (ax13, ax14, ax15)) = plt.subplots(5, 3, fi
ax = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10, ax11, ax12, ax13, ax14, ax15 ]

for i in range(len(categorical_features)):
    sns.countplot(x = categorical_features[i], hue="Churn", data=df, ax=ax[i])
```

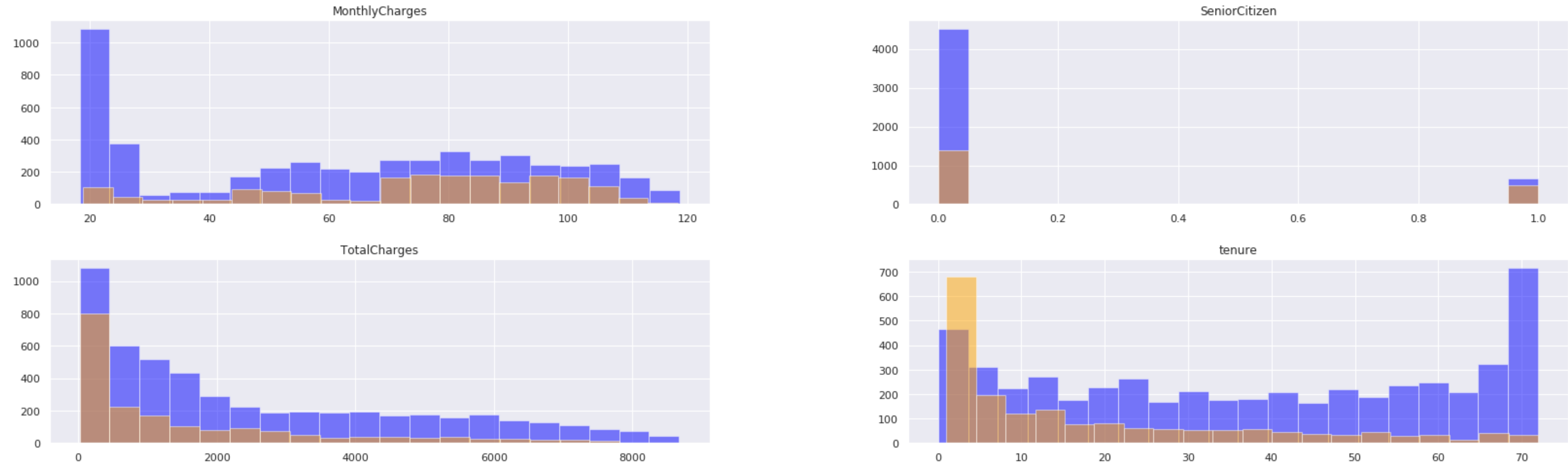

count
Churn
No
Yes
Female
Male
gender
Yes
No
Partner
No
Yes
Dependents
No
Yes
PhoneService
No phone service
No
Yes
MultipleLines
DSL
Fiber optic
No
InternetService
No
Yes
No internet service
OnlineSecurity
Yes
No
No internet service
OnlineBackup
No
Yes
No internet service
DeviceProtection
No
Yes
No internet service
TechSupport
No
Yes
No internet service
StreamingTV
No
Yes
No internet service
StreamingMovies


count
1500
1000
500
0
Month-to-month
One year
Two year
Contract
count
2000
1500
1000
500
0
Yes
No
PaperlessBilling
count
800
600
400
200
0
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod

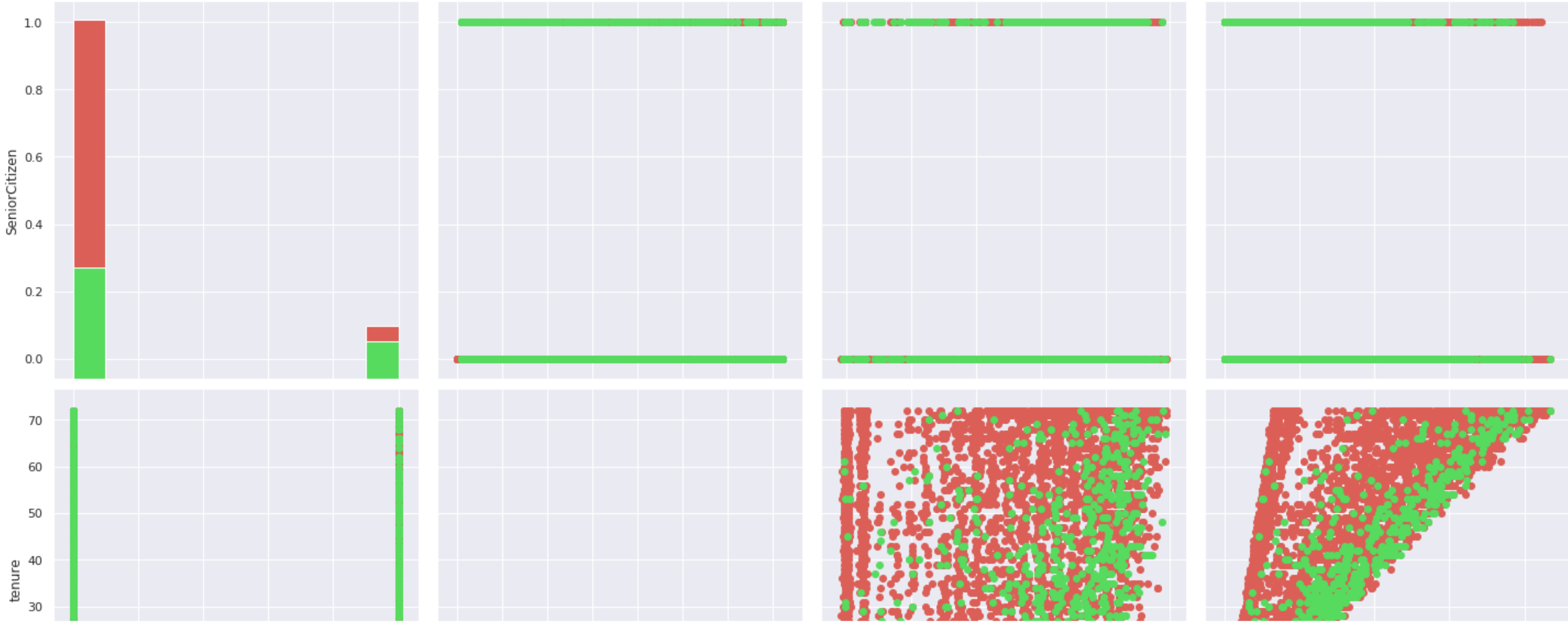
In [17]:
```
# Continuous feature histograms.
fig, ax = plt.subplots(2, 2, figsize=(28, 8))
df[df.Churn == 'No'][continuous_features].hist(bins=20, color="blue", alpha=0.5, ax=ax)
df[df.Churn == 'Yes'][continuous_features].hist(bins=20, color="orange", alpha=0.5, ax=ax)

# Or use displots
#sns.set_palette("hls", 3)
#f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(25, 25))
#ax = [ax1, ax2, ax3, ax4]
#for i in range(len(continuous_features)):
#    sns.distplot(df[continuous_features[i]], bins=20, hist=True, ax=ax[i])
```

Out[17]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f76c0cfda20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f76c0cedeb8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f76c0c96898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f76c0c51278>],
      dtype=object)
```

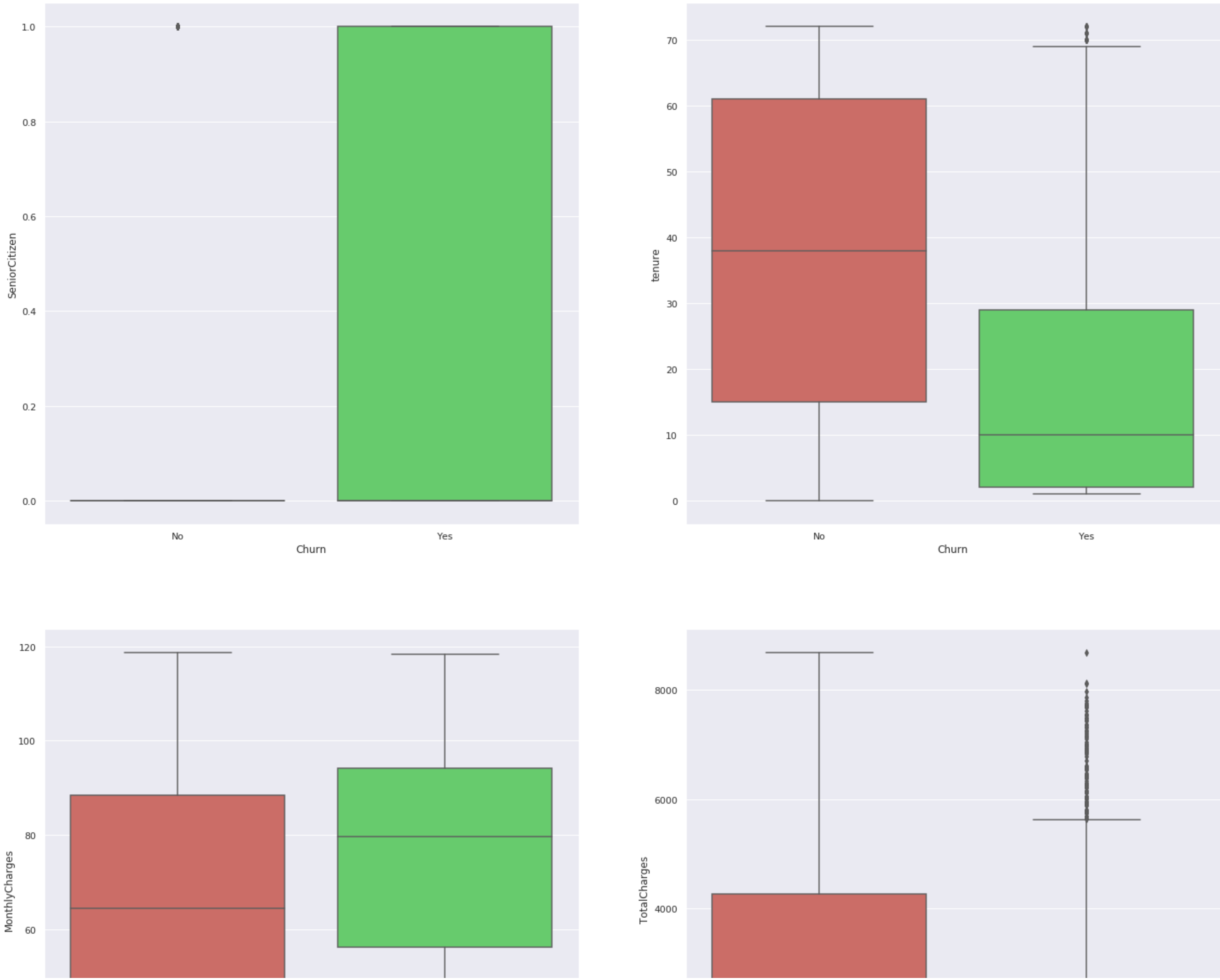


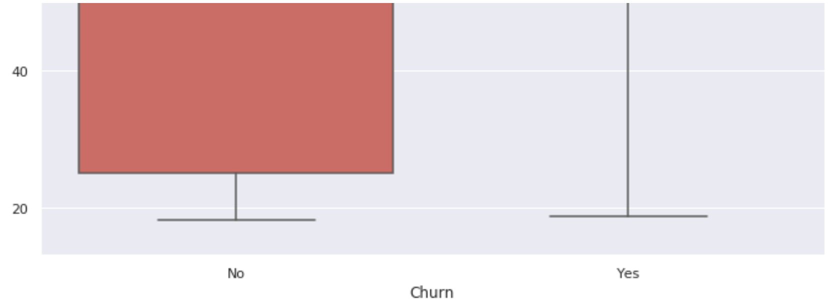
In [18]:
```
# Create Grid for pairwise relationships
gr = sns.PairGrid(df, height=5, hue="Churn")
gr = gr.map_diag(plt.hist)
gr = gr.map_offdiag(plt.scatter)
gr = gr.add_legend()
```

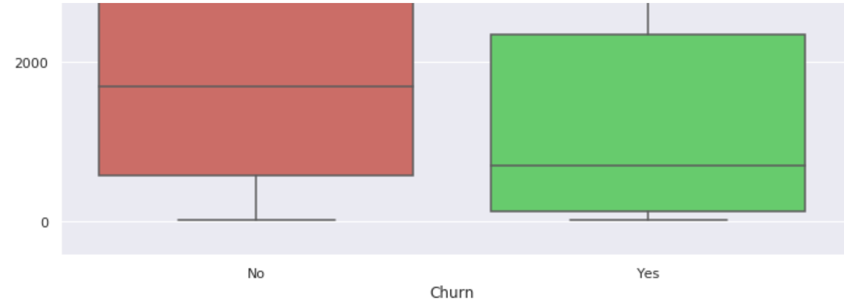


In [19]:
```
# Plot boxplots of numerical columns. More variation in the boxplot implies higher significance.
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(25, 25))
ax = [ax1, ax2, ax3, ax4]

for i in range(len(continuous_features)):
    sns.boxplot(x = 'Churn', y = continuous_features[i], data=df, ax=ax[i])
```


1.0
0.8
0.6
0.4
0.2
0.0
SeniorCitizen
No
Yes
Churn
70
60
50
40
30
20
10
0
tenure
No
Yes
Churn
120
100
80
60
MonthlyCharges
8000
6000
4000
TotalCharges





2.0 Create a model

In [20]:
```
from pyspark.sql import SparkSession
import pandas as pd
import json

spark = SparkSession.builder.getOrCreate()
df_data = spark.createDataFrame(df)
df_data.head()
```

Out[20]:
```
Row(gender='Female', SeniorCitizen=0, Partner='Yes', Dependents='No', tenure=1, PhoneService='No', MultipleLines='No
phone service', InternetService='DSL', OnlineSecurity='No', OnlineBackup='Yes', DeviceProtection='No', TechSupport='N
o', StreamingTV='No', StreamingMovies='No', Contract='Month-to-month', PaperlessBilling='Yes', PaymentMethod='Electro
nic check', MonthlyCharges=29.85, TotalCharges=29.85, Churn='No')
```

2.1 Split the data into training and test sets

In [21]:
```
spark_df = df_data
(train_data, test_data) = spark_df.randomSplit([0.8, 0.2], 24)

print("Number of records for training: " + str(train_data.count()))
print("Number of records for evaluation: " + str(test_data.count()))
```

```
Number of records for training: 5662
Number of records for evaluation: 1381
```

2.2 Examine the Spark DataFrame Schema

data types to determine requirements for feature engineering

In [22]:
```
spark_df.printSchema()
```

```
root
 |-- gender: string (nullable = true)
 |-- SeniorCitizen: long (nullable = true)
 |-- Partner: string (nullable = true)
 |-- Dependents: string (nullable = true)
 |-- tenure: long (nullable = true)
 |-- PhoneService: string (nullable = true)
 |-- MultipleLines: string (nullable = true)
 |-- InternetService: string (nullable = true)
 |-- OnlineSecurity: string (nullable = true)
 |-- OnlineBackup: string (nullable = true)
 |-- DeviceProtection: string (nullable = true)
 |-- TechSupport: string (nullable = true)
 |-- StreamingTV: string (nullable = true)
 |-- StreamingMovies: string (nullable = true)
 |-- Contract: string (nullable = true)
 |-- PaperlessBilling: string (nullable = true)
 |-- PaymentMethod: string (nullable = true)
 |-- MonthlyCharges: double (nullable = true)
 |-- TotalCharges: double (nullable = true)
 |-- Churn: string (nullable = true)
```

2.3 Use StringIndexer to encode a string column of labels to a column of label indices

We are using the Pipeline package to build the development steps as pipeline.
We are using StringIndexer to handle categorical / string features from the dataset. StringIndexer encodes a string column of labels to a column of label indices

We then use VectorAssembler to asemble these features into a vector. Pipelines API requires that input variables are passed in a vector

In [23]:
```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import StringIndexer, IndexToString, VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline, Model


si_gender = StringIndexer(inputCol = 'gender', outputCol = 'gender_IX')
si_Partner = StringIndexer(inputCol = 'Partner', outputCol = 'Partner_IX')
si_Dependents = StringIndexer(inputCol = 'Dependents', outputCol = 'Dependents_IX')
si_PhoneService = StringIndexer(inputCol = 'PhoneService', outputCol = 'PhoneService_IX')
si_MultipleLines = StringIndexer(inputCol = 'MultipleLines', outputCol = 'MultipleLines_IX')
si_InternetService = StringIndexer(inputCol = 'InternetService', outputCol = 'InternetService_IX')
si_OnlineSecurity = StringIndexer(inputCol = 'OnlineSecurity', outputCol = 'OnlineSecurity_IX')
si_OnlineBackup = StringIndexer(inputCol = 'OnlineBackup', outputCol = 'OnlineBackup_IX')
si_DeviceProtection = StringIndexer(inputCol = 'DeviceProtection', outputCol = 'DeviceProtection_IX')
si_TechSupport = StringIndexer(inputCol = 'TechSupport', outputCol = 'TechSupport_IX')
si_StreamingTV = StringIndexer(inputCol = 'StreamingTV', outputCol = 'StreamingTV_IX')
si_StreamingMovies = StringIndexer(inputCol = 'StreamingMovies', outputCol = 'StreamingMovies_IX')
si_Contract = StringIndexer(inputCol = 'Contract', outputCol = 'Contract_IX')
si_PaperlessBilling = StringIndexer(inputCol = 'PaperlessBilling', outputCol = 'PaperlessBilling_IX')
si_PaymentMethod = StringIndexer(inputCol = 'PaymentMethod', outputCol = 'PaymentMethod_IX')
```

In [24]:
```
si_Label = StringIndexer(inputCol="Churn", outputCol="label").fit(spark_df)
label_converter = IndexToString(inputCol="prediction", outputCol="predictedLabel", labels=si_Label.labels)
```

2.4 Create a single vector

In [25]:
```
va_features = VectorAssembler(inputCols=['gender_IX',  'SeniorCitizen', 'Partner_IX', 'Dependents_IX', 'PhoneService_I
                                         'OnlineSecurity_IX', 'OnlineBackup_IX', 'DeviceProtection_IX', 'TechSupport_I
                                         'Contract_IX', 'PaperlessBilling_IX', 'PaymentMethod_IX', 'TotalCharges', 'Mc
```

2.5 Create a pipeline, and fit a model using RandomForestClassifier

Assemble all the stages into a pipeline. We don't expect a clean linear regression, so we'll use RandomForestClassifier to find the best decision tree for the data.

In [26]:
```
classifier = RandomForestClassifier(featuresCol="features")

pipeline = Pipeline(stages=[si_gender, si_Partner, si_Dependents, si_PhoneService, si_MultipleLines, si_InternetServic
                            si_TechSupport, si_StreamingTV, si_StreamingMovies, si_Contract, si_PaperlessBilling, si_P
                            classifier, label_converter])

model = pipeline.fit(train_data)
```

In [27]:
```
predictions = model.transform(test_data)
evaluatorDT = BinaryClassificationEvaluator(rawPredictionCol="prediction")
area_under_curve = evaluatorDT.evaluate(predictions)

evaluatorDT = BinaryClassificationEvaluator(rawPredictionCol="prediction",  metricName='areaUnderROC')
area_under_curve = evaluatorDT.evaluate(predictions)
evaluatorDT = BinaryClassificationEvaluator(rawPredictionCol="prediction",  metricName='areaUnderPR')
area_under_PR = evaluatorDT.evaluate(predictions)
print("areaUnderROC = %g" % area_under_curve)
```

```
areaUnderROC = 0.709654
```