Customer Churn Case Study

We will take the case study through 3 steps in order to compute a good fit model for predicting a customer churn

<p style="text-align:center;color:red;">Prepared by Michael Le</p>

1. Data Summary and Feature Selection Using three methods
2. Seperating out Train and Test Data
3. Applying model to Train Data and test resulting model on Test Data 80/20 split respectively

In [60]:
```python
#Importing required packages to run a Univeariate Feature Selection method
import pandas as pd
import numpy as np

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier
```

In [61]:
```python
#Loading data
churn_data = pd.read_csv(r"C:\Users\lemic\Downloads\telco-customer-churn\WA_Fn
-UseC_-Telco-Customer-Churn.csv")
churn_data.head()
```

Out[61]:

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

5 rows × 21 columns
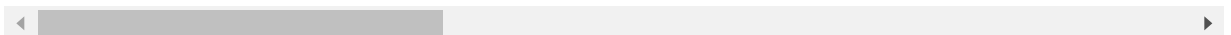
In [16]:
```python
#print info on dataset
churn_data.info()
```

```
<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 float64
Churn               7043 non-null object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

Based on info output, looks like total charges, being a float64 type has either some nulls or non integers values not being recognized

In [20]:
```python
churn_data = churn_data.drop('customerID', axis=1) #not needed for analysis
churn_data['TotalCharges'] = pd.to_numeric(churn_data['TotalCharges'], errors
= 'coerce') #convert to numeric where possible - got an error for " " nulls so
will fill with 0
churn_data[churn_data['TotalCharges'].isna()==True] = 0 # setting nas to zero
 based on finding running to_numeric error
```

In [21]:
```python
#applyign SelectKBest with chi2 fmethod to select best fit features
import matplotlib.pyplot as plt

#extract subset of object data types
obj_churn_data = churn_data.select_dtypes(include=['object']).copy()

# obj_churn_data[obj_churn_data.isnull().any(axis=1)] #no nulls, thats good!

#obj_churn_data.describe()
obj_churn_data_numeric = pd.get_dummies(obj_churn_data)
#obj_churn_data_numeric.info()

int_churn_data = churn_data.select_dtypes(exclude=['object']).copy()

feature_churn = pd.concat([int_churn_data,obj_churn_data_numeric], axis=1)
feature_churn.head()
```
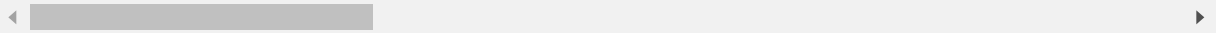
Out[21]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	gender_0	gender_Female	gender_Male
0	0	1	29.85	29.85	0	1	0
1	0	34	56.95	1889.50	0	0	1
2	0	2	53.85	108.15	0	0	1
3	0	45	42.30	1840.75	0	0	1
4	0	2	70.70	151.65	0	1	0

5 rows × 63 columns

In [263]:
```python
#setting variables for feature selection
X = feature_churn.iloc[:,0:-2] #indpendent columns for univariate selection
Y = feature_churn.iloc[:,-1] #target column which is last column


BestUnivariateFeatures = SelectKBest(score_func=chi2, k=10)
fit = BestUnivariateFeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
featurescores = pd.concat([dfcolumns,dfscores],axis=1)
featurescores.columns = ['feature-univariate-chi2','score']
top_10_univariate_feats = featurescores.nlargest(10, columns = 'score')
top_10_univariate_feats['rank'] = top_10_univariate_feats['score'].rank(ascend
ing=False)
print(top_10_univariate_feats)
```

	feature-univariate-chi2	score	rank
3	TotalCharges	624292.003004	1.0
1	tenure	16278.923685	2.0
2	MonthlyCharges	3733.878622	3.0
49	Contract_Month-to-month	519.895311	4.0
51	Contract_Two year	485.007178	5.0
58	PaymentMethod_Electronic check	426.422767	6.0
25	OnlineSecurity_No	416.716774	7.0
37	TechSupport_No	406.645334	8.0
22	InternetService_Fiber optic	374.476216	9.0
29	OnlineBackup_No	284.531747	10.0

We can see that the top three most prominent features that help determine customer churn (value = 1) are Total charges, Tenure, and Monthly Charges

The trailing features that could play an important part in the final stretch of the predicitive model are whether they have: 49 Contract_Month-to-month 519.895311 4.0 51 Contract_Two year 485.007178 5.0 58 PaymentMethod_Electronic check 426.422767 6.0 25 OnlineSecurity_No 416.716774 7.0 37 TechSupport_No 406.645334 8.0 22 InternetService_Fiber optic 374.476216 9.0 29 OnlineBackup_No 284.531747 10.0

In [68]:
```python
#feature detection/selection using extratreesclassifier method
model_churn = ExtraTreesClassifier(n_estimators = 100) #100 trees in forest
model_churn.fit(X,Y)
#print(model_churn.feature_importances_) #inbuilt class from sklearn
feat_importance = pd.Series(model_churn.feature_importances_,index=X.columns)
feat_importance = pd.DataFrame(feat_importance)
feat_importance.reset_index(inplace = True)
feat_importance.columns = ['feature-extra-trees','Classifier Importance']
feat_tree_top10 = feat_importance.nlargest(10, columns = 'Classifier Importanc
e')
plt.barh(feat_tree_top10['feature-extra-trees'], feat_tree_top10['Classifier I
mportance'],align='edge')
plt.gca().invert_yaxis()
plt.show

#curious about less important features? print below by uncommenting and runnin
g
#print(feat_importance.nsmallest(40, columns = 'Classifier Importance'))
```
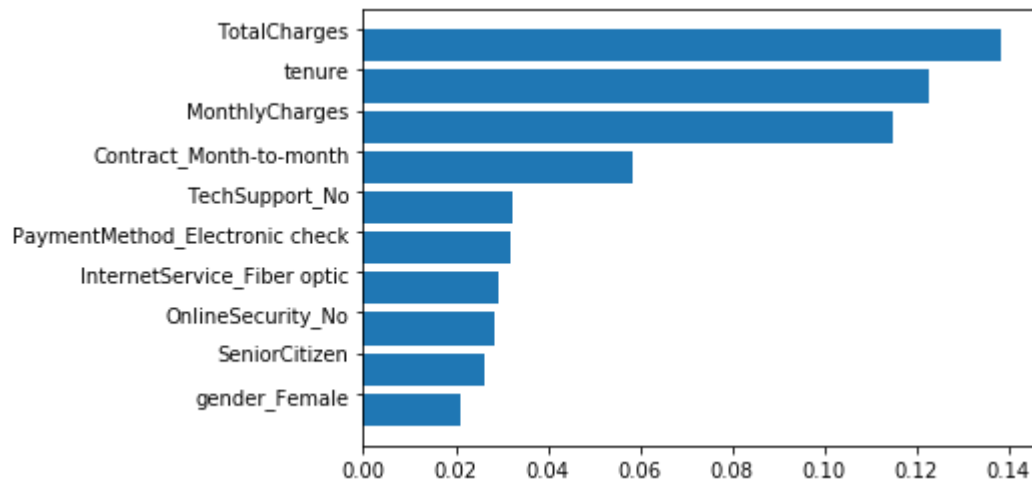
Out[68]: `<function matplotlib.pyplot.show(*args, **kw)>`

We now have the top 12 important features using the ensemble extra trees classifiers within a forest with 100 trees

In [69]:
```python
#compare of univariate vs ensemble method

pd.concat([top_10_univariate_feats,feat_tree_top10], axis = 1).sort_values(by
='Classifier Importance', ascending=False) # allows comparison between univari
ate feature selection vs extratreesclassifier feature detection
```

Out[69]:

|	feature-univariate-chi2	score	rank	feature-extra-trees	Classifier Importance
3	TotalCharges	624292.003004	1.0	TotalCharges	0.138462
1	tenure	16278.923685	2.0	tenure	0.122828
2	MonthlyCharges	3733.878622	3.0	MonthlyCharges	0.114840
49	Contract_Month-to-month	519.895311	4.0	Contract_Month-to-month	0.058625
37	TechSupport_No	406.645334	8.0	TechSupport_No	0.032556
58	PaymentMethod_Electronic check	426.422767	6.0	PaymentMethod_Electronic check	0.031774
22	InternetService_Fiber optic	374.476216	9.0	InternetService_Fiber optic	0.029503
25	OnlineSecurity_No	416.716774	7.0	OnlineSecurity_No	0.028391
0	NaN	NaN	NaN	SeniorCitizen	0.026131
5	NaN	NaN	NaN	gender_Female	0.021046
29	OnlineBackup_No	284.531747	10.0	NaN	NaN
51	Contract_Two year	485.007178	5.0	NaN	NaN

Based on findings above, I will train a model based on the top matching features, which are:

1. TotalCharges 624292.003004 1.0 TotalCharges 0.136892
2. tenure 16278.923685 2.0 tenure 0.123686
3. MonthlyCharges 3733.878622 3.0 MonthlyCharges 0.115721
4. Contract_Month-to-month 519.895311 4.0 Contract_Month-to-month 0.060624
5. OnlineSecurity_No 416.716774 7.0 OnlineSecurity_No 0.031221
6. PaymentMethod_Electronic check 426.422767 6.0 PaymentMethod_Electronic check 0.029601
7. TechSupport_No 406.645334 8.0 TechSupport_No 0.028722
8. InternetService_Fiber optic 374.476216 9.0 InternetService_Fiber optic 0.026591

In [234]:
```python
#As a treat, let's show a correlation heat map to see how those features affec
t the churn rate
import seaborn as sns
```

In [261]:
```python
selected_feats_churn = feature_churn[['Churn_Yes','SeniorCitizen','Contract_Tw
o year', 'TotalCharges','tenure','MonthlyCharges','Contract_Month-to-month','O
nlineSecurity_No','PaymentMethod_Electronic check','TechSupport_No','InternetS
ervice_Fiber optic']]
corrmat = selected_feats_churn.corr()
print(len(corrmat))
```

11

```
In [262]: top_corr_features = corrmat.index
          g =sns.heatmap(feature_churn[top_corr_features].corr(),annot=True,cmap="RdYlG
          n")
          #top and bottom row cut off due to unstable seaborn and matplotlib with latest
          update. could revert to older version of matplotlib, but we'll roll with this
           for now.
```
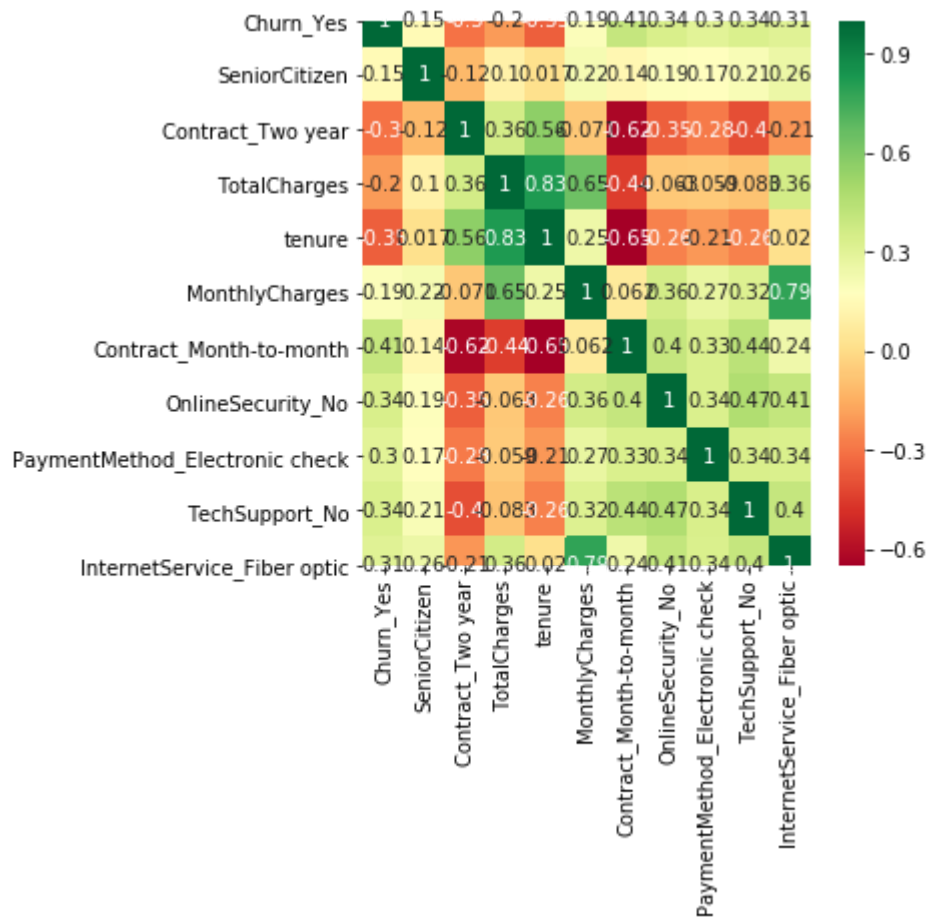

Of the top 10 features identified, 7 have a positive correlation to a customer churning and 3 have a negative correlation

Summary of Feature Selection

Based on findings, an analyst would find most value building visualizations or reports that target the 8 features identified above, and when there is a deviation above the median for numerical measurements, then a flag or warning should appear next to that customer record. This would include TotalCharges, Tenure, MonthlyCharges. For the categorical variables, having a contract month-to-month and a combination of one ore more of the above breaching a median, would flag the customer as a potential churn, and allow sales rep to reach out and attempt a conversion to contract vs month-to-month. The same would go for all other categorical features such as tech support and online security. Improving the online security of a user, and tech support are simple steps that the company can take to retain more customers as shown below on the correlation map. (having online security and tech support reduces churn)

```
Churn_Yes            1.000000
OnlineSecurity_Yes  -0.170573
TechSupport_Yes     -0.164016
```

In [309]:
```python
#plot data
from pylab import rcParams
import warnings
warnings.filterwarnings("ignore")

sizes = feature_churn['Churn_Yes'].value_counts(sort=True)
colors = ["grey","pink"]
rcParams['figure.figsize'] = 5,5

#Pie Chart
plt.pie(sizes, explode = (0.05,0.05),labels = ["Yes","No"], colors = colors, a
utopct = '%1.1f%%',startangle = 180)

plt.title('Percentage Churn')
plt.show()
```


Percentage Churn
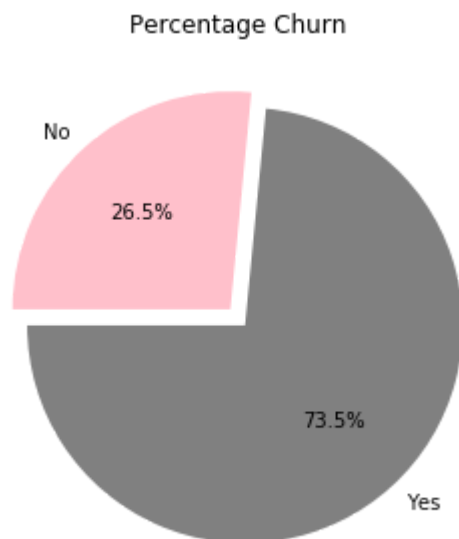
Logistic Regression Model to test features (Baseline)

Model shows 80% accuracy in predicting outcomes of Churn based on inputs. Running optimzation statistical logistics regression model with X_feat = selected_feats_churn.loc[:,1:] X_train, X_test, Y_train, Y_test = train_test_split(X_feat,Y,test_size = 0.30, random_state = 101) import statsmodels.api as sm logit_model=sm.Logit(Y_train,X_train) result=logit_model.fit() print(result.summary2()) #we get the below summaryRemoving Month to Month as it is not statistically significant Optimization terminated successfully. Current function value: inf Iterations 8 Results: Logit === Model: Logit Pseudo R-squared: inf Dependent Variable: Churn_Yes AIC: inf Date: 2019-11-22 17:52 BIC: inf No. Observations: 4930 Log-Likelihood: -inf Df Model: 9 LL-Null: 0.0000 Df Residuals: 4920 LLR p-value: 1.0000 Converged: 1.0000 Scale: 1.0000 No. Iterations: 8.0000 --- Coef. Std.Err. z P>|z| [0.025 0.975] --- SeniorCitizen 0.2916 0.0982 2.9687 0.0030 0.0991 0.4841 Contract_Two year -1.2530 0.2057 -6.0911 0.0000 -1.6562 -0.8498 TotalCharges 0.0007 0.0001 9.6265 0.0000 0.0006 0.0008 tenure -0.0932 0.0064 -14.5685 0.0000 -0.1057 -0.0806 MonthlyCharges -0.0173 0.0023 -7.3935 0.0000 -0.0219 -0.0127 Contract_Month-to-month -0.0605 0.0964 -0.6272 0.5305 -0.2494 0.1285 OnlineSecurity_No 0.4384 0.0932 4.7031 0.0000 0.2557 0.6211 PaymentMethod_Electronic check 0.4555 0.0812 5.6073 0.0000 0.2963 0.6147 TechSupport_No 0.4018 0.0929 4.3242 0.0000 0.2197 0.5839 InternetService_Fiber optic 1.2088 0.1330 9.0912 0.0000 0.9482 1.4694

In [271]:
```python
#model with selected features:
feat_lin = feature_churn[['Churn_Yes','SeniorCitizen','Contract_Two year', 'To
talCharges','tenure','MonthlyCharges','OnlineSecurity_No','PaymentMethod_Elect
ronic check','TechSupport_No','InternetService_Fiber optic']]
X_feat = feat_lin.iloc[:,1:]
X_train, X_test, Y_train, Y_test = train_test_split(X_feat,Y,test_size = 0.30,
random_state = 101)

import statsmodels.api as sm
logit_model=sm.Logit(Y_train,X_train)
result=logit_model.fit()
print(result.summary2())
#we get the below summary

#import model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result = model.fit(X_train, Y_train)

from sklearn import metrics
prediction_test = model.predict(X_test)

#print accuracy
print(str(metrics.accuracy_score(Y_test, prediction_test)*100)+'%')
```

```
Optimization terminated successfully.
        Current function value: inf
        Iterations 8
                              Results: Logit
================================================================================
==
Model:                  Logit              Pseudo R-squared:    inf
Dependent Variable:     Churn_Yes          AIC:                 inf
Date:                   2019-11-22 18:11   BIC:                 inf
No. Observations:       4930               Log-Likelihood:      -inf
Df Model:               8                  LL-Null:             0.00
00
Df Residuals:           4921               LLR p-value:         1.00
00
Converged:              1.0000             Scale:               1.00
00
No. Iterations:         8.0000
--------------------------------------------------------------------------------
--
                              Coef.  Std.Err.   z     P>|z|   [0.025  0.97
5]
--------------------------------------------------------------------------------
--
SeniorCitizen                 0.2883   0.0981   2.9399 0.0033  0.0961  0.48
05
Contract_Two year            -1.2321   0.2032  -6.0640 0.0000 -1.6304 -0.83
39
TotalCharges                  0.0007   0.0001  10.3846 0.0000  0.0006  0.00
08
tenure                       -0.0938   0.0063 -14.8132 0.0000 -0.1062 -0.08
14
MonthlyCharges               -0.0181   0.0020  -9.0806 0.0000 -0.0220 -0.01
42
OnlineSecurity_No             0.4323   0.0927   4.6616 0.0000  0.2505  0.61
40
PaymentMethod_Electronic check  0.4507   0.0809   5.5725 0.0000  0.2922  0.60
92
TechSupport_No                0.3935   0.0920   4.2770 0.0000  0.2132  0.57
39
InternetService_Fiber optic   1.2233   0.1312   9.3271 0.0000  0.9662  1.48
04
================================================================================
==

80.88026502602933%
```

We will now compare two other popular methods: Decision Tree, and K-Nearest-Neighbor to gauge best model to predict

In [272]:
```python
#Print weights ov variables
weights = pd.Series(model.coef_[0], index = X_feat.columns.values)
weights.sort_values(ascending = False)
weights.sort_values(ascending = True)
```

Out[272]:
```
Contract_Two year              -0.696976
tenure                         -0.063308
TotalCharges                    0.000307
MonthlyCharges                  0.000496
SeniorCitizen                   0.329954
PaymentMethod_Electronic check  0.427774
OnlineSecurity_No               0.433929
TechSupport_No                  0.436380
InternetService_Fiber optic     0.897562
dtype: float64
```

Exploring other Machine Learning Predictive Models

Using decision trees then k-nearest-neight

In [304]:
```python
from sklearn import tree

model = tree.DecisionTreeClassifier(random_state=0, max_depth=5)


from sklearn import metrics

results = []
for i in range(1,10): # replicating model to get best average
    result = model.fit(X_train, Y_train)
    prediction_test = model.predict(X_test)
#print accuracy
    results.append(float(metrics.accuracy_score(Y_test, prediction_test)*100))

#print((results))
#another method to get accuracy:
#from sklearn.metrics import accuracy_score
print(str(sum((results))/len(results))+'%')
#accuracy_score(Y_test, prediction_test)
```

```
78.98722195929957%
```

title

Not as good as our logistics regression model, but did not hurt to check

```python
In [275]: from sklearn.neighbors import KNeighborsClassifier

          score_knn = 0
          neighborcount = 3
          max_value = -1

          while score_knn > max_value:
              model = KNeighborsClassifier(n_neighbors=neighborcount)
              result = model.fit(X_train, Y_train)
              prediction_test = model.predict(X_test)
              max_value = float(score_knn)
              score_knn = float(metrics.accuracy_score(Y_test, prediction_test))
              neighborcount +=1
          print(str(max_value*100)+'% using '+str(neighborcount) + ' neighbors')
          #print((results))
          #another method to get accuracy:
          #from sklearn.metrics import accuracy_score

          #accuracy_score(Y_test, prediction_test)
```

```
76.99952673923332% using 8 neighbors
```

A little better than our decision tree model, but still not as good as our linear regression.

End of Case Study

Personal Website (https://technerdhelp.com)

```
In [ ]:
```