Explorations in Churn

This file utilizes a data set regarding voluntary customer churn in the telecom industry where customers are on some sort of contract involving international phone calling plans. The data set will be prepared for data analysis, exploratory data analysis will be performed and different types of models will be created including decision tree analysis, logistic regression, support vector classifier, and a random forest with optimized hyperparameters. The most influential features of the data set will be revealed.

You can find the churn data set that I used [here (https://data.world/earino/churn)](https://data.world/earino/churn).

Below is a summary of columns, size and data types of the intital data set.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Account_Length  3333 non-null   int64
 1   Vmail_Message   3333 non-null   int64
 2   Day_Mins        3333 non-null   float64
 3   Eve_Mins        3333 non-null   float64
 4   Night_Mins      3333 non-null   float64
 5   Intl_Mins       3333 non-null   float64
 6   CustServ_Calls  3333 non-null   int64
 7   Churn           3333 non-null   object
 8   Intl_Plan       3333 non-null   object
 9   Vmail_Plan      3333 non-null   object
 10  Day_Calls       3333 non-null   int64
 11  Day_Charge      3333 non-null   float64
 12  Eve_Calls       3333 non-null   int64
 13  Eve_Charge      3333 non-null   float64
 14  Night_Calls     3333 non-null   int64
 15  Night_Charge    3333 non-null   float64
 16  Intl_Calls      3333 non-null   int64
 17  Intl_Charge     3333 non-null   float64
 18  State           3333 non-null   object
 19  Area_Code       3333 non-null   int64
 20  Phone           3333 non-null   object
dtypes: float64(8), int64(8), object(5)
memory usage: 546.9+ KB
```

Below is the first five rows of the intital data set showing all of the intital features.

Out[5]:

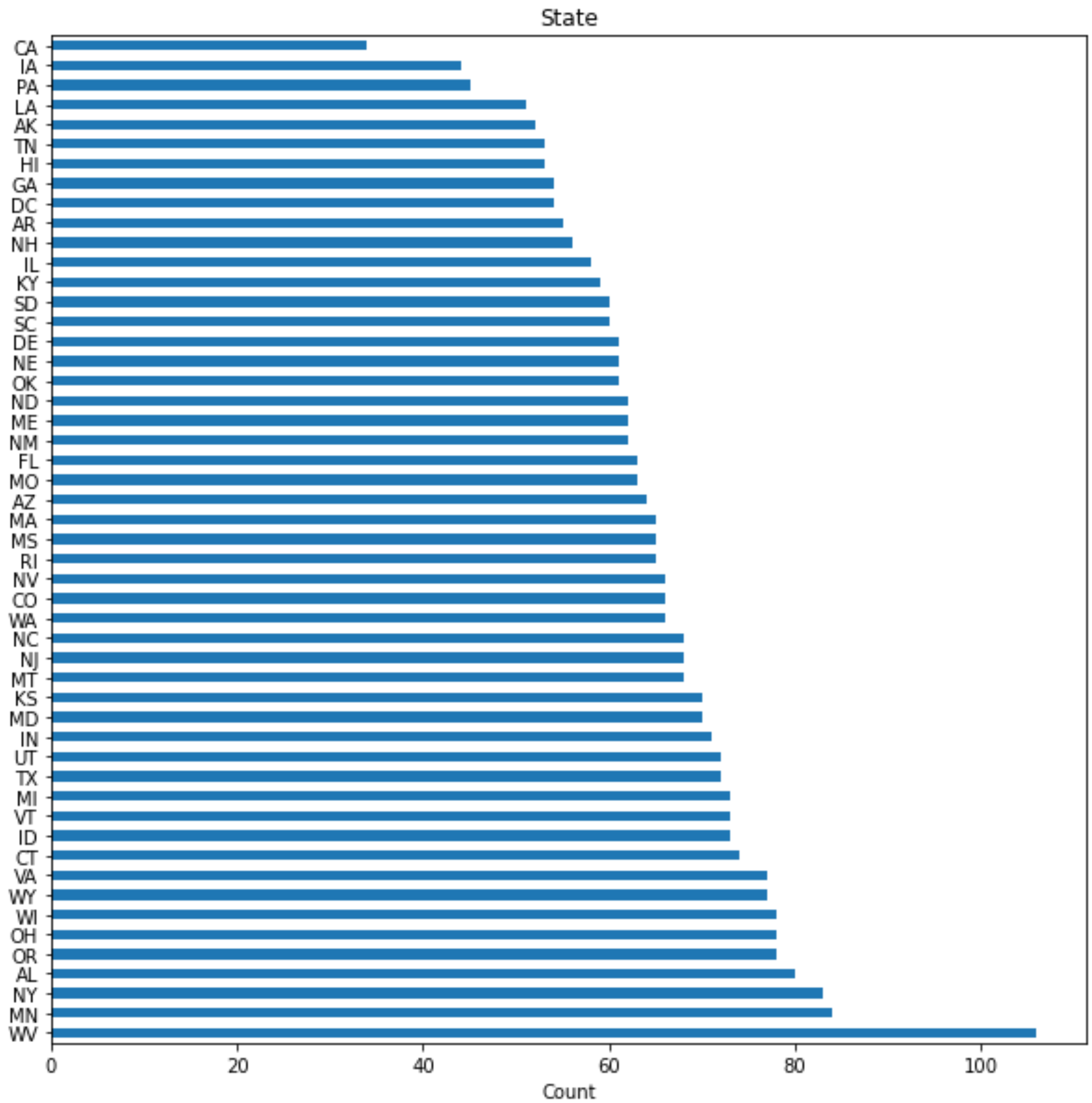
	Account_Length	Vmail_Message	Day_Mins	Eve_Mins	Night_Mins	Intl_Mins	CustServ_Cal
0	128	25	265.1	197.4	244.7	10.0	
1	107	26	161.6	195.5	254.4	13.7	
2	137	0	243.4	121.2	162.6	12.2	
3	84	0	299.4	61.9	196.9	6.6	
4	75	0	166.7	148.3	186.9	10.1	

5 rows × 21 columns

This tells us a great deal already as there are no missing values, the target is Churn and there are some object variables that need to be converted. Phone and Area Code can be dropped so let's do that now.

```
Index(['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins', 'Ni
ght_Mins',
       'Intl_Mins', 'CustServ_Calls', 'Churn', 'Intl_Plan', 'Vmail_Pl
an',
       'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge', 'Night_C
alls',
       'Night_Charge', 'Intl_Calls', 'Intl_Charge', 'State'],
      dtype='object')
```

There are object columns that have to be converted to numeric for modeling unless we want to use a decision tree. Single trees are great for EDA and visualization but have poor predictive power. One of these values is the State variable. Let's take a look at it to see if this may be a relative variable. One-hot encoding may be needed here but it will create 51 additonal variables and require deep learning. We will drop the State variable and move forward.



Let's see about the percentage of churn vs. no churn. If the percentage is below 5% then we may have an overfitting problem.

```
Churn
no       85.508551
yes      14.491449
dtype: float64
```

Only around 14% of the samples have churned which is not ideal but we should be able to model the data. We may have to stratify the train and test split here. Before we convert the categorical variables, let's do some feature engineering because there are some columns that may need to be combined or eliminated. The International, daytime, night time and evening call minutes can be divided by their respective total calls for average call minutes per call and let's check the columns/indexes.

```
Index(['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins', 'Ni
ght_Mins',
       'Intl_Mins', 'CustServ_Calls', 'Churn', 'Intl_Plan', 'Vmail_Pl
an',
       'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge', 'Night_C
alls',
       'Night_Charge', 'Intl_Calls', 'Intl_Charge', 'State', 'Avg_Day
_Calls',
       'Avg_Eve_Calls', 'Avg_Intl_Calls', 'Avg_Night_Calls'],
      dtype='object')
```

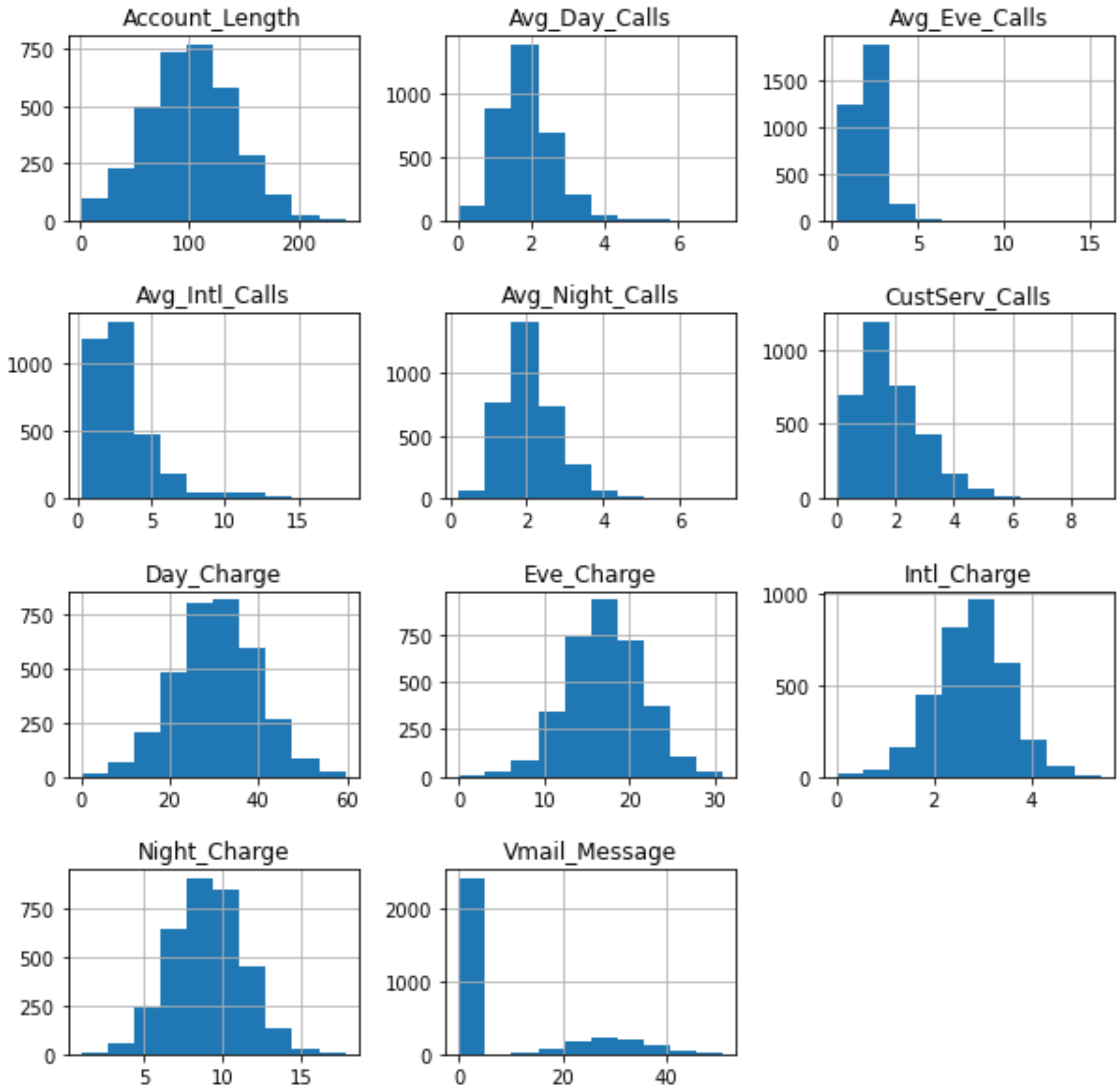
Now, we can create a new dataframe with the previous features dropped by subetting. That way if there is an error, we can revert back to the original df.

Out[11]:

	Account_Length	Vmail_Message	CustServ_Calls	Churn	Intl_Plan	Vmail_Plan	Day_Charge
0	128	25	1	no	no	yes	45.07
1	107	26	1	no	no	yes	27.47
2	137	0	0	no	no	no	41.38
3	84	0	2	no	yes	no	50.90
4	75	0	3	no	yes	no	28.34

```
State  Churn
AK     no       49
       yes       3
AL     no       72
       yes       8
AR     no       44
       yes      11
AZ     no       60
       yes       4
CA     no       25
       yes       9
CO     no       57
       yes       9
CT     no       62
       yes      12
DC     no       49
       yes       5
DE     no       52
       yes       9
FL     no       55
       yes       8
Name: Churn, dtype: int64
State  Churn
SD     no       52
       yes       8
TN     no       48
       yes       5
TX     no       54
       yes      18
UT     no       62
       yes      10
VA     no       72
       yes       5
VT     no       65
       yes       8
WA     no       52
       yes      14
WI     no       71
       yes       7
WV     no       96
       yes      10
WY     no       68
       yes       9
Name: Churn, dtype: int64
```

Exploratory data analysis will now be performed on the final features.



Some of these values have a normal distribution but some are skewed, namely our new features. Voicemail messages appear more categorical but let's check if they are relevant to the target first. We will need to standardize the numerical variabes.

```
Account_Length      212
Vmail_Message        46
CustServ_Calls       10
Churn                 2
Intl_Plan             2
Vmail_Plan            2
Day_Charge         1667
Eve_Charge         1440
Night_Charge        933
Intl_Charge         162
State                51
Avg_Day_Calls      3259
Avg_Eve_Calls      3253
Avg_Intl_Calls      781
Avg_Night_Calls    3244
dtype: int64
```

Essentially, what we see here is that Intl_Plan, Vmail_Plan and State are our categorical variables. Churn is our target and should be removed before modeling. First, the numerical fields will be standardized or scaled using StandardScaler by removing the categorical variables. For now, the target will remain so we can run a Decision Tree model.

We will merge categorical and numerical variables back together in a final dataframe and make categorical variables binary. The state data will be omitted as the previous graph shows only limited variability and one-hot encoding introduced 51 new variables making a decision tree unreadable. We can re-add this data later if necessary. The data set is ready for modeling.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Churn            3333 non-null   object 
 1   Intl_Plan        3333 non-null   object 
 2   Vmail_Plan       3333 non-null   object 
 3   Account_Length   3333 non-null   float64
 4   Vmail_Message    3333 non-null   float64
 5   CustServ_Calls   3333 non-null   float64
 6   Day_Charge       3333 non-null   float64
 7   Eve_Charge       3333 non-null   float64
 8   Night_Charge     3333 non-null   float64
 9   Intl_Charge      3333 non-null   float64
 10  Avg_Day_Calls    3331 non-null   float64
 11  Avg_Eve_Calls    3332 non-null   float64
 12  Avg_Intl_Calls   3315 non-null   float64
 13  Avg_Night_Calls  3333 non-null   float64
dtypes: float64(11), object(3)
memory usage: 364.7+ KB
```

Out[50]:

	Churn	Intl_Plan	Vmail_Plan	Account_Length	Vmail_Message	CustServ_Calls	Day_Charge
0	0	0	1	0.676489	1.234883	-0.427932	1.56703€
1	0	0	1	0.149065	1.307948	-0.427932	-0.33401:
2	0	0	0	0.902529	-0.591760	-1.188218	1.16846
3	0	1	0	-0.428590	-0.591760	0.332354	2.19675
4	0	1	0	-0.654629	-0.591760	1.092641	-0.240041

Now, we will define our target as X and the rest of dff as y and run a decision tree. This will be used for plotting and visualization as opposed to assessing model performance. We will try a test size of 1/4 and check the split. Wow, that is a ridiculous test accuracy. I almost wonder if I am doing something wrong. Let's check and print out the tree after optimizing the hyperparameter for maximum depth to prevent overfitting. We can see below that somewhere between 3 and 8 is advantageous.

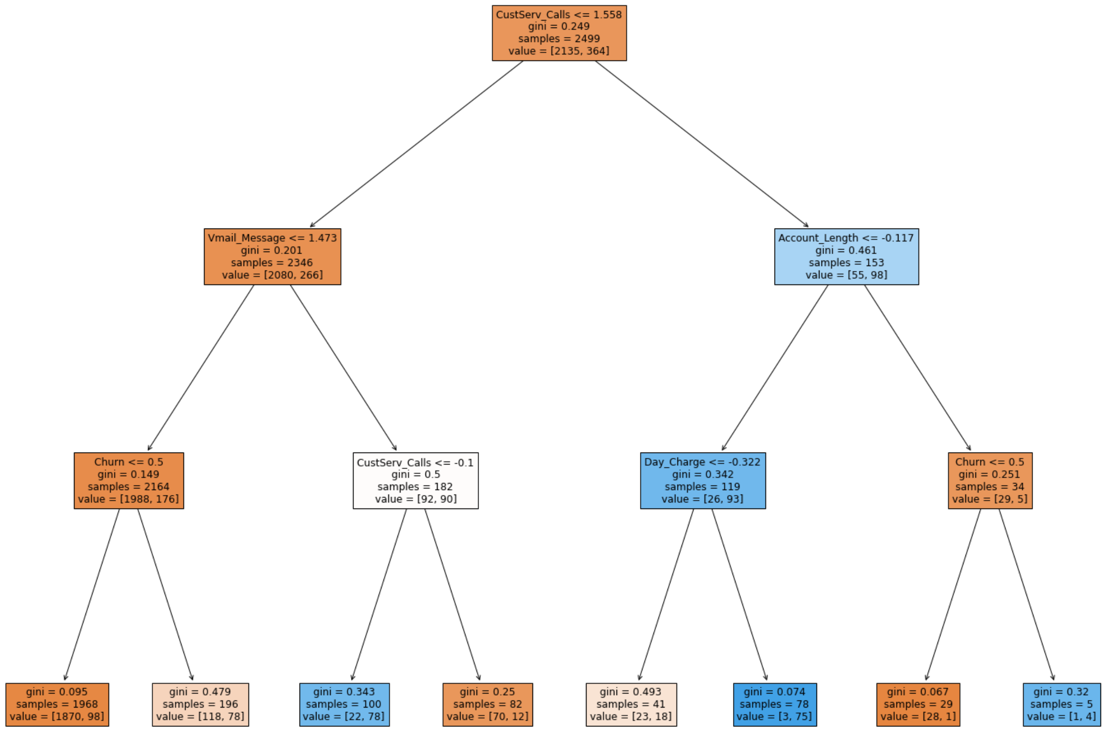
```
0.7497749774977498
0.2502250225022502


Training accuracy:  0.969
Test accuracy:  0.928


Test precision: 0.8105, Test recall: 0.6471
```

	Max_Depth	Accuracy	Precision	Recall
0	2.0	0.876499	0.710526	0.226891
1	3.0	0.902878	0.806452	0.420168
2	4.0	0.919664	0.893939	0.495798
3	5.0	0.934053	0.863636	0.638655
4	6.0	0.931655	0.844444	0.638655
5	7.0	0.928058	0.817204	0.638655
6	8.0	0.925659	0.806452	0.630252
7	9.0	0.928058	0.817204	0.638655
8	10.0	0.930456	0.790476	0.697479
9	11.0	0.926859	0.773585	0.689076
10	12.0	0.932854	0.811881	0.689076
11	13.0	0.924460	0.754545	0.697479
12	14.0	0.924460	0.750000	0.705882

Out[59]:
```
[Text(697.5, 951.3000000000001, 'CustServ_Calls <= 1.558\ngini = 0.24
9\nsamples = 2499\nvalue = [2135, 364]'),
 Text(348.75, 679.5, 'Vmail_Message <= 1.473\ngini = 0.201\nsamples =
2346\nvalue = [2080, 266]'),
 Text(174.375, 407.70000000000005, 'Churn <= 0.5\ngini = 0.149\nsampl
es = 2164\nvalue = [1988, 176]'),
 Text(87.1875, 135.89999999999998, 'gini = 0.095\nsamples = 1968\nval
ue = [1870, 98]'),
 Text(261.5625, 135.89999999999998, 'gini = 0.479\nsamples = 196\nval
ue = [118, 78]'),
 Text(523.125, 407.70000000000005, 'CustServ_Calls <= -0.1\ngini = 0.
5\nsamples = 182\nvalue = [92, 90]'),
 Text(435.9375, 135.89999999999998, 'gini = 0.343\nsamples = 100\nval
ue = [22, 78]'),
 Text(610.3125, 135.89999999999998, 'gini = 0.25\nsamples = 82\nvalue
= [70, 12]'),
 Text(1046.25, 679.5, 'Account_Length <= -0.117\ngini = 0.461\nsample
s = 153\nvalue = [55, 98]'),
 Text(871.875, 407.70000000000005, 'Day_Charge <= -0.322\ngini = 0.34
2\nsamples = 119\nvalue = [26, 93]'),
 Text(784.6875, 135.89999999999998, 'gini = 0.493\nsamples = 41\nvalu
e = [23, 18]'),
 Text(959.0625, 135.89999999999998, 'gini = 0.074\nsamples = 78\nvalu
e = [3, 75]'),
 Text(1220.625, 407.70000000000005, 'Churn <= 0.5\ngini = 0.251\nsamp
les = 34\nvalue = [29, 5]'),
 Text(1133.4375, 135.89999999999998, 'gini = 0.067\nsamples = 29\nval
ue = [28, 1]'),
 Text(1307.8125, 135.89999999999998, 'gini = 0.32\nsamples = 5\nvalue
= [1, 4]')]
```



It looks like the tree split on customer service calls first and then on voice mail messages and account length. This is only one tree though so other models will be explored but it is good to take a first look. On to logistic regression where we can use information to determine best features.

```
Test accuracy for logistic regression:  0.9245
```

The accuracy is not quite as good as the decision tree but we cannot rely on just one tree. Let's see if we can optimize the logistic regression model with lasso regularization that helps with overfitting and maybe some feature selection.

	C	Non-Zero Coefficients	Accuracy	Precision	Recall
0	1.000	13.0	0.862110	0.535714	0.252101
1	0.500	13.0	0.860911	0.527273	0.243697
2	0.250	11.0	0.862110	0.540000	0.226891
3	0.100	10.0	0.859712	0.521739	0.201681
4	0.050	10.0	0.859712	0.531250	0.142857
5	0.025	8.0	0.862110	0.833333	0.042017
6	0.010	2.0	0.857314	0.000000	0.000000
7	0.005	0.0	0.857314	0.000000	0.000000

```
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classific
ation.py:1221: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 due to no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Owner\anaconda3\lib\site-packages\sklearn\metrics\_classific
ation.py:1221: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 due to no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Well, not much can be said with such low recall numbers. At 8 coefficients, precision is high but recall is really low. A value of 10 non-zero coefficients may be good choice. There must be lot of false negatives in this model.

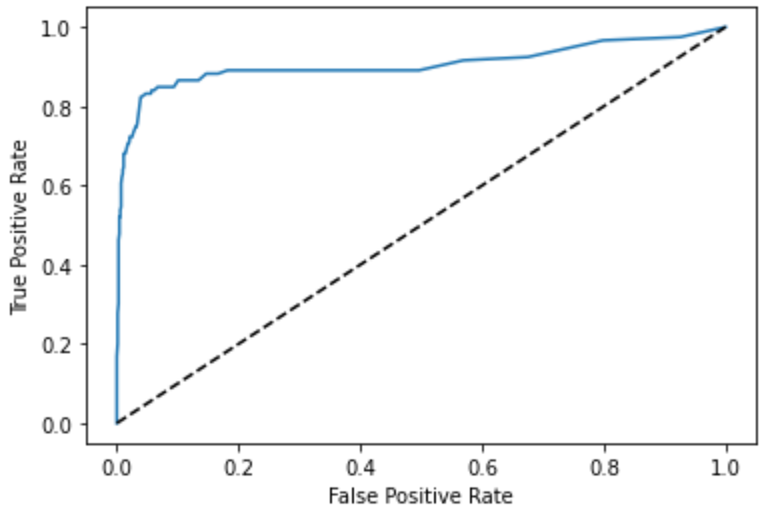
	Feature	Coefficient	Exp_Coefficient
1	Vmail_Plan	-2.024492	0.132061
10	Avg_Eve_Calls	-0.062619	0.939302
9	Avg_Day_Calls	-0.017728	0.982428
2	Account_Length	0.048755	1.049964
12	Avg_Night_Calls	0.061457	1.063384
7	Night_Charge	0.160371	1.173946
8	Intl_Charge	0.184835	1.203020
11	Avg_Intl_Calls	0.201229	1.222905
6	Eve_Charge	0.418178	1.519190
3	Vmail_Message	0.519393	1.681007
4	CustServ_Calls	0.623876	1.866148
5	Day_Charge	0.760530	2.139409
0	Intl_Plan	2.123355	8.359135

```
0.9400479616306955
```

```
0.9400479616306955
```

A support vector classifier and random forest have the same exact. That is quite interesting and maybe the data has been overfit. Let's go with the Random Forest and find out. Let's take a look at a confusion matrix which will give us a sense of true and false positives and negatives and the ROC curve.

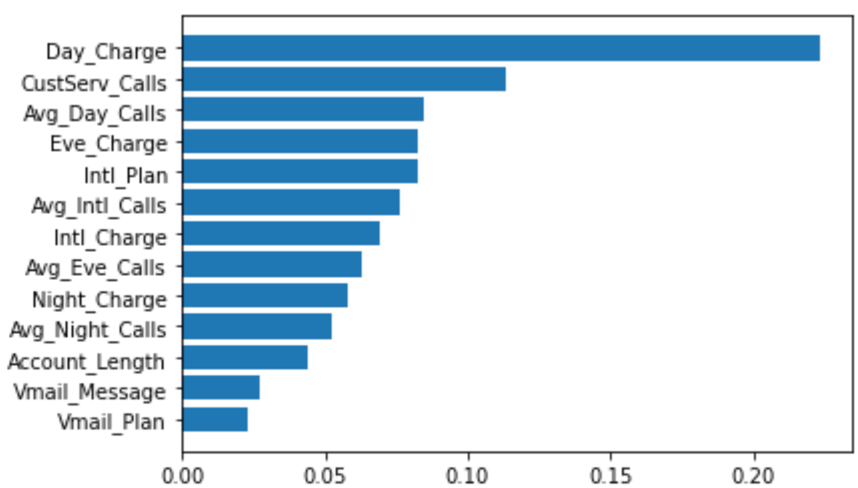
```
Training accuracy for random forest:  1.0
Test accuracy for random forest:  0.94
Test precision for random forest: 0.9059, Test recall for random fore
st: 0.6471
Confusion Matrix for random forest:
[[707   8]
 [ 42  77]]
```




```
Area under the ROC curve for random forest:  0.9057
F1 score for random forest:  0.7549
```

The Random Forest Classifier seems to be a pretty good model. Now, we are going to use a cross-validation method to optimize the model called GridSearchCV.

```
{'bootstrap': True, 'criterion': 'gini', 'max_depth': None, 'max_feat
ures': 10}
```

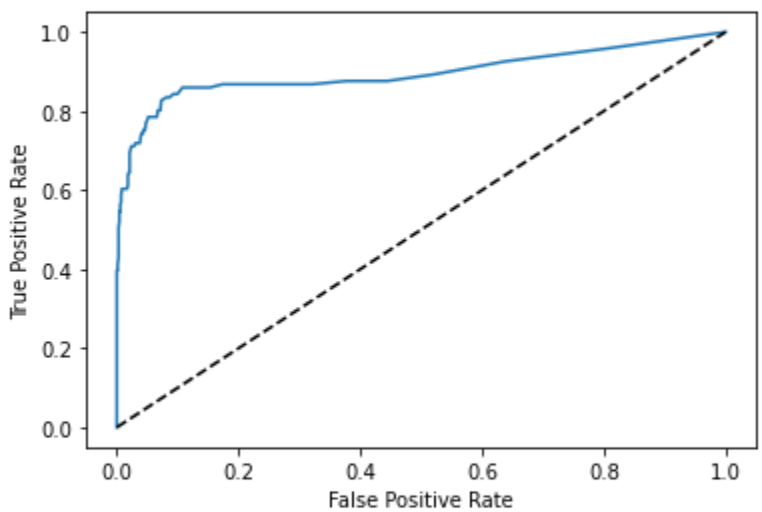


We will run it all again with the final dataframe and see what the results are after we create a new partition. This time, we will use a stratified split which means that we can guarantee about 14% of the test set will also have churn = 1. We will also set the random state to ensure the same set is used for all models.

```
0    2137
1     362
Name: Churn, dtype: int64
0    713
1    121
Name: Churn, dtype: int64
0    0.855142
1    0.144858
Name: Churn, dtype: float64
0    0.854916
1    0.145084
Name: Churn, dtype: float64
0.7497749774977498
0.2502250225022502
```

As you can see, the training and test splits have roughly the same ratio of churn customers as the original data set. Also, We have preserved a train test split of 75/25. Let's see what our metrics look like now with optimal parameters:

```
Training accuracy for random forest:  1.0
Test accuracy for random forest:  0.932
Test precision for random forest: 0.8478, Test recall for random fore
st: 0.6446
Confusion Matrix for random forest:
[[699  14]
 [ 43  78]]
```



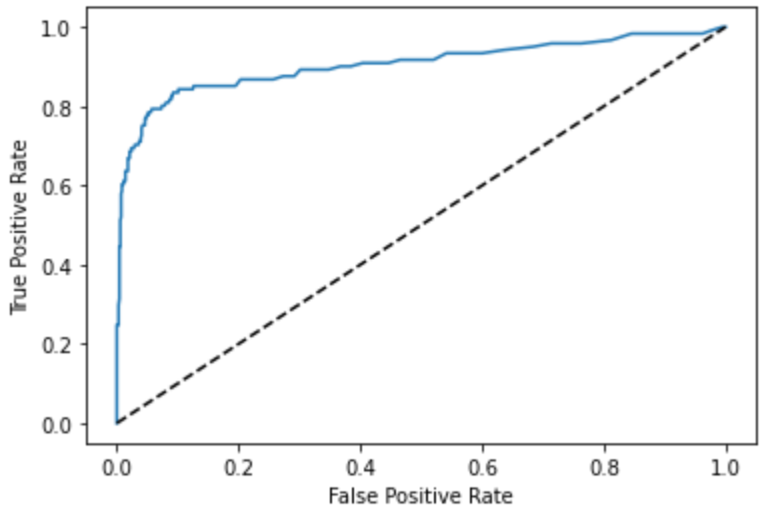

```
Area under the ROC curve for random forest:  0.8952
F1 score for random forest:  0.7324
```

The precision went down as there are more false positives. Good thing we used a stratified split and parameter tuning. The numbers are relatively similar so let's look at the best parameters and best features. Now, we are going to use GridSearchCV for some final tuning.

```
{'bootstrap': False, 'criterion': 'entropy', 'max_depth': None, 'max_
features': 3}
```

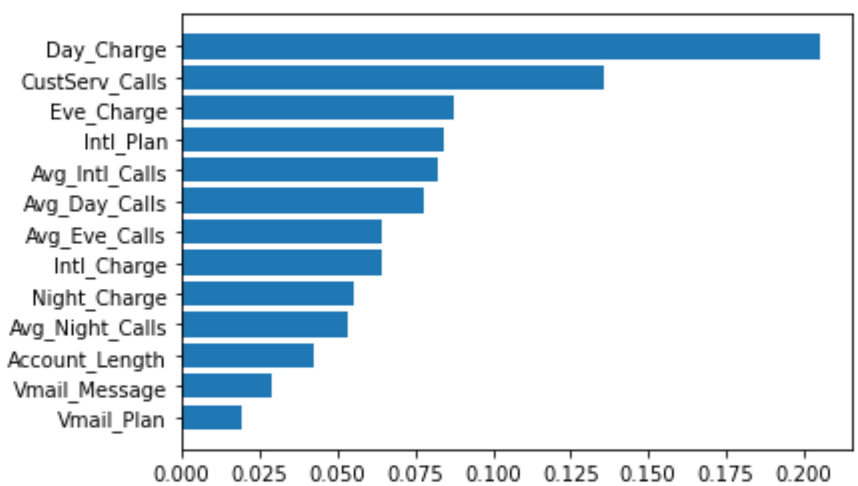
Interesting, the criterion changed to entropy! Additionally, max features has been reduced to 3 and bootstrapping must be set to false. Let's also increase the number of trees to 400.

```
Training accuracy for random forest:  1.0
Test accuracy for random forest:  0.934
Test precision for random forest: 0.8587, Test recall for random fore
st: 0.6529
Confusion Matrix for random forest:
[[700  13]
 [ 42  79]]
```




```
Area under the ROC curve for random forest:  0.9052
F1 score for random forest:  0.7418
```

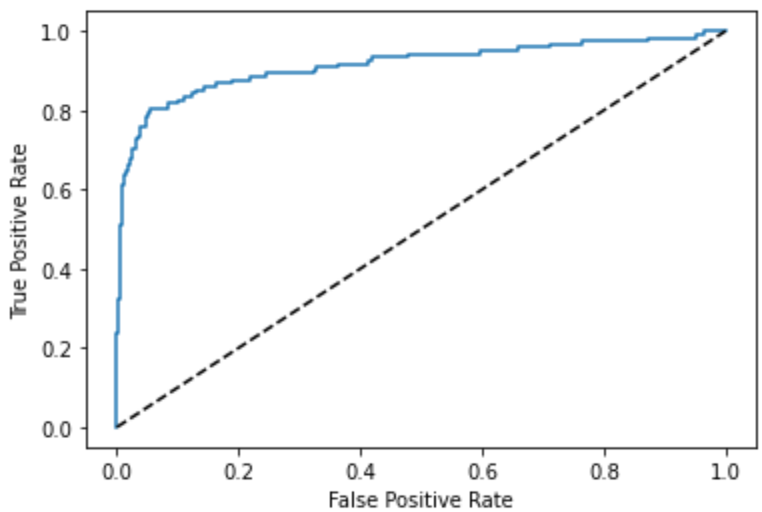
A slight improvement so I will take it. Finally, we will observe the most predictive features of the Random Forest Model.



Average day calls drop down and Day charge looks to be the consistent major predictive feature. Customer service calls are second on the list. It turns out that this is different than our decision tree so it was a good idea to use ensemble methods for prediction. Now, we will try randomized search since we have 4 hyperparameters and a large amount of data. We will enter these hyperparameters into the model to see what kind of metrics we get.

```
{'max_features': 'sqrt', 'max_depth': 12, 'criterion': 'entropy', 'bo
otstrap': False}
```

```
Training accuracy for random forest:  0.987
Test accuracy for random forest:  0.934
Test precision for random forest: 0.8667, Test recall for random fore
st: 0.6446
Confusion Matrix for random forest:
[[701  12]
 [ 43  78]]
```




```
Area under the ROC curve for random forest:  0.9144
F1 score for random forest:  0.7393
```

Conclusion

The numbers for Random Search cross validation and Grid Search cross validation are very similar. There may not be much more hyperparameter tuning that can be done but it would be interesting to see if Bayesian or Genetic methods could be used to determine the most optimum model or if a combination of randomized and grid search can be used to narrow in on certain spaces of hyperparameters and use grid search on those spaces to surely find the best model. Still, the area under the ROC curve is above 0.91 and the F1 score, which is a relation of the precision and recall for the model is quite high at almost 0.74 although the test accuracy did not change much after raising the number of trees to 400 from the default.