Churn Rate Analysis

A common business problem is customer churn. Being able to predict if and when a customer is likely to leave and have incentives in place to offer them, can lead to huge business savings. Creating attrition models that help make churn predictions can benefit businesses with a niche audience.

I am using a publicly available [dataset (https://www.iainpardoe.com/teaching/dsc433/data.htm)](https://www.iainpardoe.com/teaching/dsc433/data.htm) from an anonymous telecom company. The columns provide the following details:

- Account Length
- Voicemail message
- Day minutes used
- Evening minutes used
- Night minutes
- International minutes
- Number of Customer Service Calls
- Churn
- International Plan
- Voicemail Plan
- Day Calls
- Day Charge
- Eve calls
- Eve charge
- Night calls
- Night charge
- International calls
- International charge
- State
- Area code
- Phone

We work hard at gaining customers, models like the one here can help us keep them. In addition to the use of Decision Trees, this model can also be built using logistic regression or ensemble models. Prior to presenting to the team, I would test and train a portion of the data through each of these models and evaluate which would be the best to work with.

-**Sarah Aristil**, Marketing Analyst Applicant

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import display, HTML
%matplotlib inline

from pandas import ExcelWriter
from pandas import ExcelFile
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
churn = pd.read_excel('Churn.xls', sheetname='churn')
```

Exploring Dataset

In [3]:
```
churn.head()
```

Out[3]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Churn	Int'l Plan	VMail Plan	...	Day Charge	Eve Calls	Eve Charge	Night Calls	Night Charge	Intl Calls	Intl Charge
0	128	25	265.1	197.4	244.7	10.0	1	0	0	1	...	45.07	99	16.78	91	11.01	3	2.70
1	107	26	161.6	195.5	254.4	13.7	1	0	0	1	...	27.47	103	16.62	103	11.45	3	3.70
2	137	0	243.4	121.2	162.6	12.2	0	0	0	0	...	41.38	110	10.30	104	7.32	5	3.29
3	84	0	299.4	61.9	196.9	6.6	2	0	1	0	...	50.90	88	5.26	89	8.86	7	1.78
4	75	0	166.7	148.3	186.9	10.1	3	0	1	0	...	28.34	122	12.61	121	8.41	3	2.73

5 rows × 21 columns

In [4]:
```
churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
Account Length    3333 non-null int64
VMail Message     3333 non-null int64
Day Mins          3333 non-null float64
Eve Mins          3333 non-null float64
Night Mins        3333 non-null float64
Intl Mins         3333 non-null float64
CustServ Calls    3333 non-null int64
Churn             3333 non-null int64
Int'l Plan        3333 non-null int64
VMail Plan        3333 non-null int64
Day Calls         3333 non-null int64
Day Charge        3333 non-null float64
Eve Calls         3333 non-null int64
Eve Charge        3333 non-null float64
Night Calls       3333 non-null int64
Night Charge      3333 non-null float64
Intl Calls        3333 non-null int64
Intl Charge       3333 non-null float64
State             3333 non-null object
Area Code         3333 non-null int64
Phone             3333 non-null object
dtypes: float64(8), int64(11), object(2)
memory usage: 546.9+ KB
```

In [5]:
```
#Checking for missing values
print("Number of rows: ", churn.shape[0])
counts = churn.describe().iloc[0]
display(
    pd.DataFrame(
        counts.tolist(),
        columns=["Count of values"],
        index=counts.index.values
    ).transpose()
)
```

```
Number of rows:  3333
```

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Churn	Int'l Plan	VMail Plan	Day Calls	Day Charge	Eve Calls	Eve Charge	Night Calls	Nig Char
Count of values	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	3333.0	333

In [6]:
```
churn.describe()
```

Out[6]:

	Account Length	VMail Message	Day Mins	Eve Mins	Night Mins	Intl Mins	CustServ Calls	Churn	Int'l Plan	VMail Plan
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	8.099010	179.775098	200.980348	200.872037	10.237294	1.562856	0.144914	0.096910	0.276628
std	39.822106	13.688365	54.467389	50.713844	50.573847	2.791840	1.315491	0.352067	0.295879	0.447398
min	1.000000	0.000000	0.000000	0.000000	23.200000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	0.000000	143.700000	166.600000	167.000000	8.500000	1.000000	0.000000	0.000000	0.000000
50%	101.000000	0.000000	179.400000	201.400000	201.200000	10.300000	1.000000	0.000000	0.000000	0.000000
75%	127.000000	20.000000	216.400000	235.300000	235.300000	12.100000	2.000000	0.000000	0.000000	1.000000
max	243.000000	51.000000	350.800000	363.700000	395.000000	20.000000	9.000000	1.000000	1.000000	1.000000

Setting Up the Model

Feature Selection

In [7]:
```
#Dropping the columns that won't be used
churn = churn.drop(["Phone", "Area Code", "State"], axis=1)
features = churn.drop(["Churn"], axis=1).columns
```

In [8]:
```
#Splitting data into training/test set
churn_train, churn_test = train_test_split(churn, test_size=0.25)
```

In [9]:

```python
#Set up the RandomForestClassifier and fit to data
clf = RandomForestClassifier(n_estimators=30)
clf.fit(churn_train[features], churn_train["Churn"])

# Make predictions
predictions = clf.predict(churn_test[features])
probs = clf.predict_proba(churn_test[features])
display(predictions)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

The results show a list of 0's and 1's representing whether or not the model thinks a customer has churned or not. This can be compared to whether or not the customer actually churned in order to evaluate the model. With an accuracy rate of 90% (as shown below), this model could help drive business decisions around which customers to target for retention.

In [10]:

```python
score = clf.score(churn_test[features], churn_test["Churn"])
print("Accuracy: ", score)
```

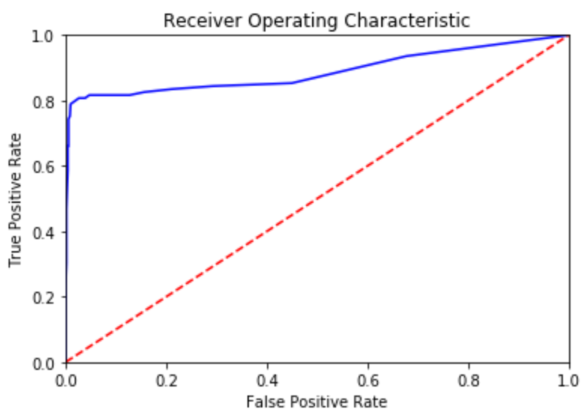
```
Accuracy:  0.9616306954436451
```

For additional support of the model's utility, and to showcase an understanding in statistical-based model evaluations, I will construct a confusion matix, or error matric, and a ROC curve, a graphical plot that illustrates the ability of a binary classifier system.

In [11]:
```
get_ipython().magic('matplotlib inline')
confusion_matrix = pd.DataFrame(
    confusion_matrix(churn_test["Churn"], predictions),
    columns=["Predicted False", "Predicted True"],
    index=["Actual False", "Actual True"]
)
display(confusion_matrix)

# Calculate the fpr and tpr
fpr, tpr, threshold = roc_curve(churn_test["Churn"], probs[:,1])
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

	Predicted False	Predicted True
Actual False	721	4
Actual True	28	81



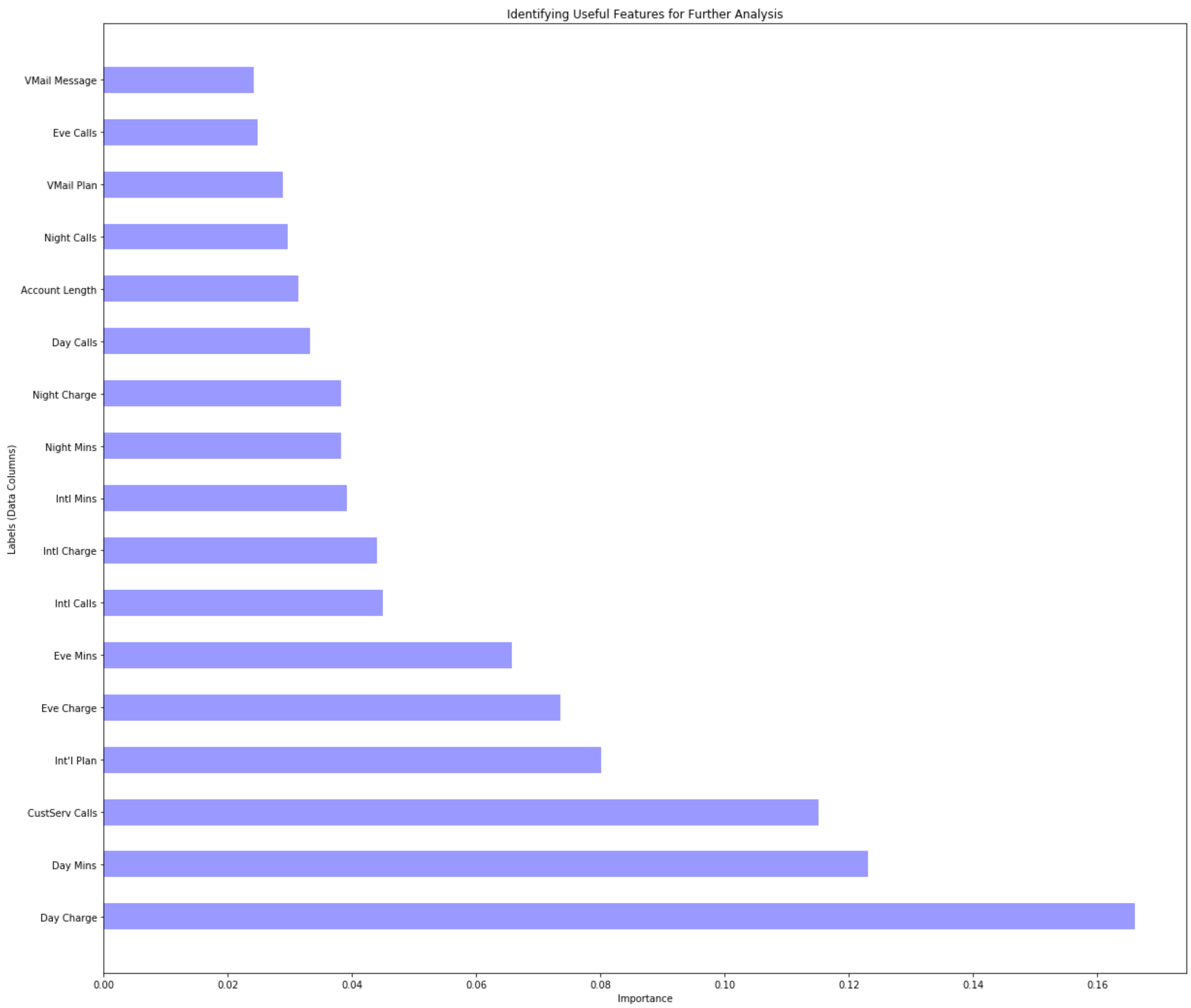
In [12]:

```
fig = plt.figure(figsize=(20, 18))
ax = fig.add_subplot(111)

churn_f = pd.DataFrame(clf.feature_importances_, columns=["importance"])
churn_f["labels"] = features
churn_f.sort_values("importance", inplace=True, ascending=False)
display(churn_f.head(5))

index = np.arange(len(clf.feature_importances_))
bar_width = 0.5
rects = plt.barh(index , churn_f["importance"], bar_width, alpha=0.4, color='b', label='Main')
plt.yticks(index, churn_f["labels"])
plt.xlabel("Importance", fontsize=10)
plt.ylabel("Labels (Data Columns)", fontsize=10)
plt.title("Identifying Useful Features for Further Analysis")
plt.show()
```

	importance	labels
10	0.166103	Day Charge
2	0.123113	Day Mins
6	0.115053	CustServ Calls
7	0.080136	Int'l Plan
12	0.073550	Eve Charge



In [13]:

```
churn_test["prob_true"] = probs[:, 1]
df_risky = churn_test[churn_test["prob_true"] > 0.9]
display(df_risky.head(5)[["prob_true"]])
```

	prob_true
2536	0.933333
3205	1.000000
2038	1.000000
894	0.966667
3072	1.000000

The above shows five individual customers who have a high risk for churning. Further analysis would give us insights into why and help navigate the incentives we could offer to retain them. Before presenting this model to the marketing team, I would also test the data against a neural network model and a support vector to ensure the highest accuracy rate for the test data.