Untitled2

March 22, 2020

1 Churn is when a customer stops doing business or ends a relationship with a company.

It's a common problem across a variety of industries, from telecommunications to cable TV to SaaS, and a company that can predict churn can take proactive action to retain valuable customers and get ahead of the competition.

This project will provide a roadmap to create a customer churn models.

Various techniques will be used to explore and visualize the data, preparing it for modeling, making predictions using machine learning, and important, actionable insights to stakeholders will be communicated. Pandas library is used for data analysis and the scikit-learn library for machine learning.

First of all the structure of our customer dataset, which has been pre-loaded into a DataFrame called telco is checked. Being able to check the structure of the data is a fundamental step in the churn modeling process and is often overlooked.

Pandas is used to import, analyze and manipulate data before running any ML algorithm.

[199]:
```
# importing the dataset
file_path = '/Users/MuhammadBilal/Desktop/Data Camp/Marketing Analytics-␣
↪Predicting Customer Churn in Python/churn_data.csv'
```

[200]:
```
import pandas as pd
telco = pd.read_csv(file_path)
```

Pandas .info() method is used to get a sense of datas structure and to observe different columns.

[202]:
```
telco.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 object
Intl_Plan         3333 non-null object
Vmail_Plan        3333 non-null object
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(8), object(5)
memory usage: 546.9+ KB
```

[5]:

```
telco.head()
```

[5]:

```
   Account_Length  Vmail_Message  Day_Mins  Eve_Mins  Night_Mins  Intl_Mins  \
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

   CustServ_Calls Churn Intl_Plan Vmail_Plan  …  Day_Charge  Eve_Calls  \
0               1    no        no        yes  …       45.07         99
1               1    no        no        yes  …       27.47        103
2               0    no        no         no  …       41.38        110
3               2    no       yes         no  …       50.90         88
4               3    no       yes         no  …       28.34        122

   Eve_Charge  Night_Calls  Night_Charge  Intl_Calls  Intl_Charge  State  \
0       16.78           91         11.01           3         2.70     KS
1       16.62          103         11.45           3         3.70     OH
2       10.30          104          7.32           5         3.29     NJ
3        5.26           89          8.86           7         1.78     OH
4       12.61          121          8.41           3         2.73     OK

  Area_Code     Phone
0       415  382-4657
1       415  371-7191
2       415  358-1921
3       408  375-9999
4       415  330-6626
```

```
[5 rows x 21 columns]
```

One feature is of particular interest to us: 'Churn', which can take in two values - yes and no - indicating whether or not the customer has churned. This feature should be explored carefully to conduct an effective analysis. How many churners does the dataset have, and how many non-churners? To easily answer this, the .value_counts() method on telco['Churn'] is used.

[168]:
```python
telco['Churn'].value_counts()
```

[168]:
```
0    2850
1     483
Name: Churn, dtype: int64
```

1.0.1 Churn by State

When dealing with customer data, geographic regions may play an important part in determining whether a customer will cancel their service or not.

Next 'State' and 'Churn' columns will be grouped to count the number of churners by state.

[177]:
```python
# Counting the number of churners and non-churners by State
print(telco.groupby('State')['Churn'].value_counts())
```

```
State  Churn
AK     0        49
       1         3
AL     0        72
       1         8
AR     0        44
                ..
WI     1         7
WV     0        96
       1        10
WY     0        68
       1         9
Name: Churn, Length: 102, dtype: int64
```

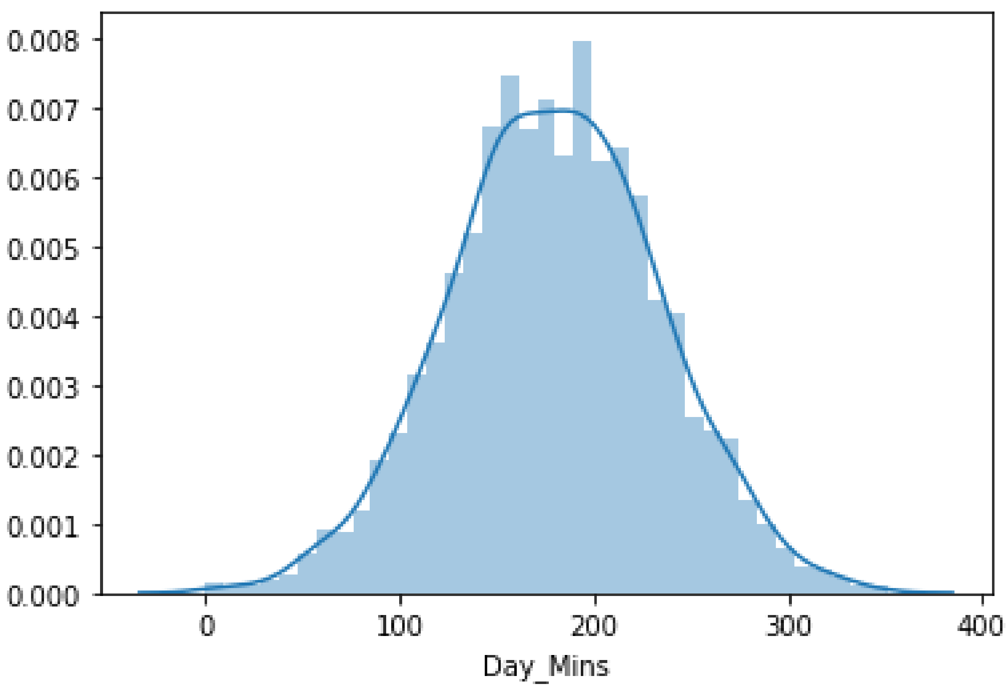
It is indeed useful to see number of Churners in different states. It can be helpful to compare different states.

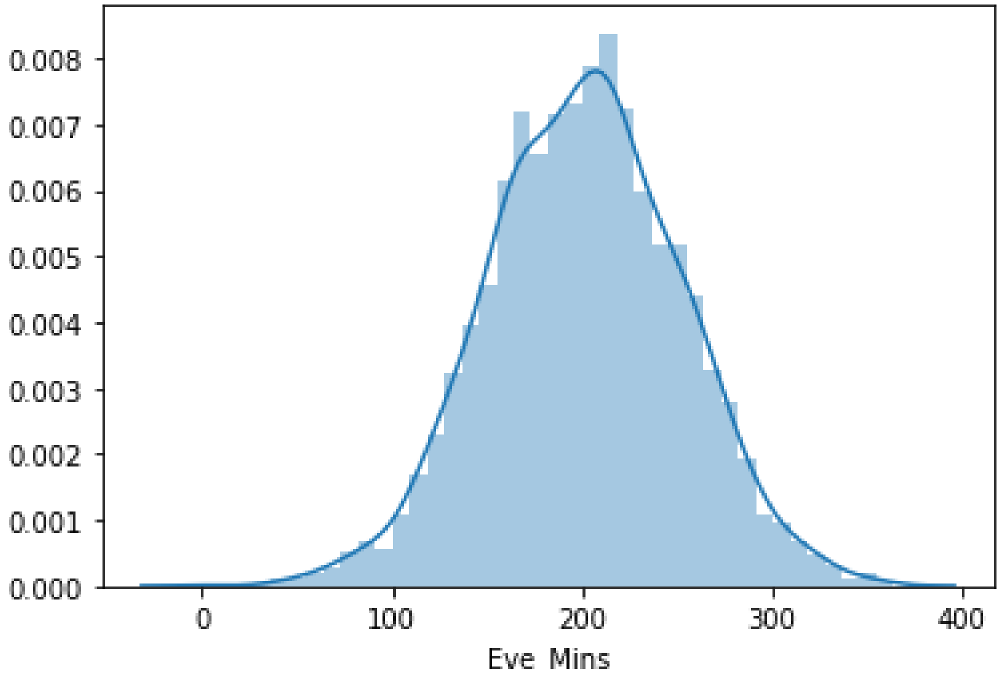
1.0.2 Exploring feature distributions

Features are explored to see if they are normally distributed.

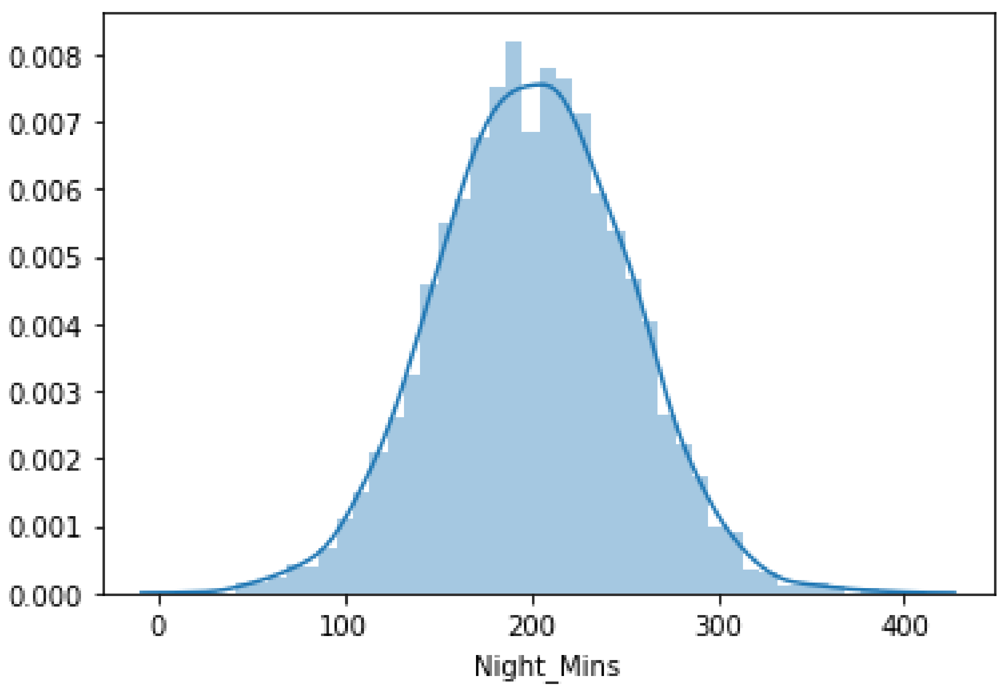
Different featuers are explored and visualized to see their distribution

[180]:
```python
# Import matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns

# Visualize the distribution of 'Day_Mins'
sns.distplot(telco['Day_Mins'])

# Display the plot
plt.show()

# Visualize the distribution of 'Eve_Mins'
sns.distplot(telco['Eve_Mins'])
plt.show()

# Visualize the distribution of 'Night_Mins'
sns.distplot(telco['Night_Mins'])
plt.show()

# Visualize the distribution of 'Intl_Mins'
sns.distplot(telco['Intl_Mins'])
plt.show()
```

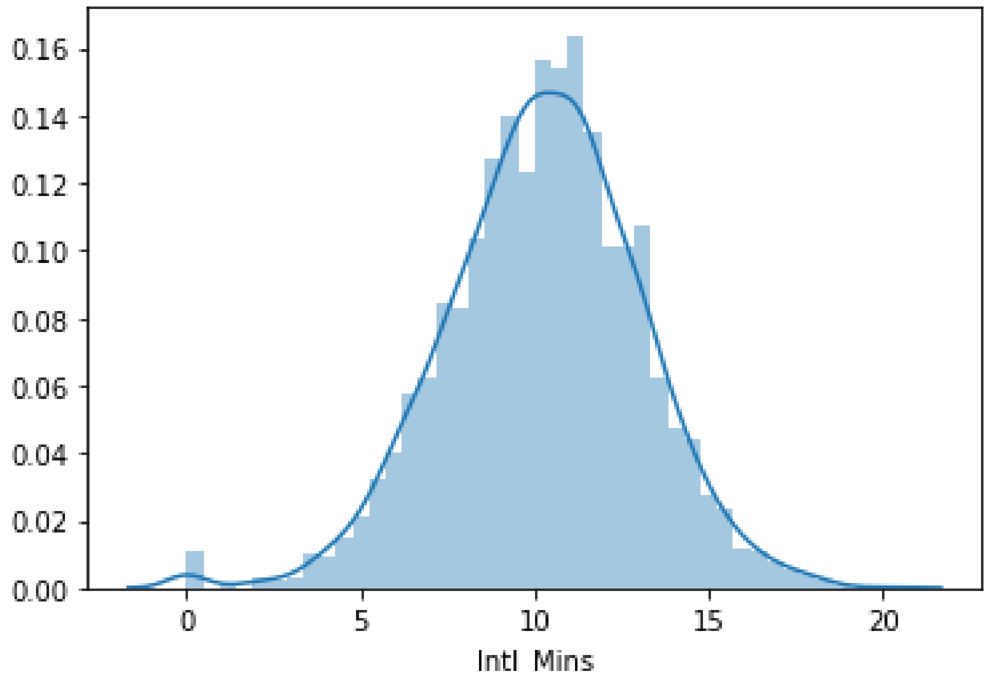




0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
Eve_Mins



0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
Night_Mins



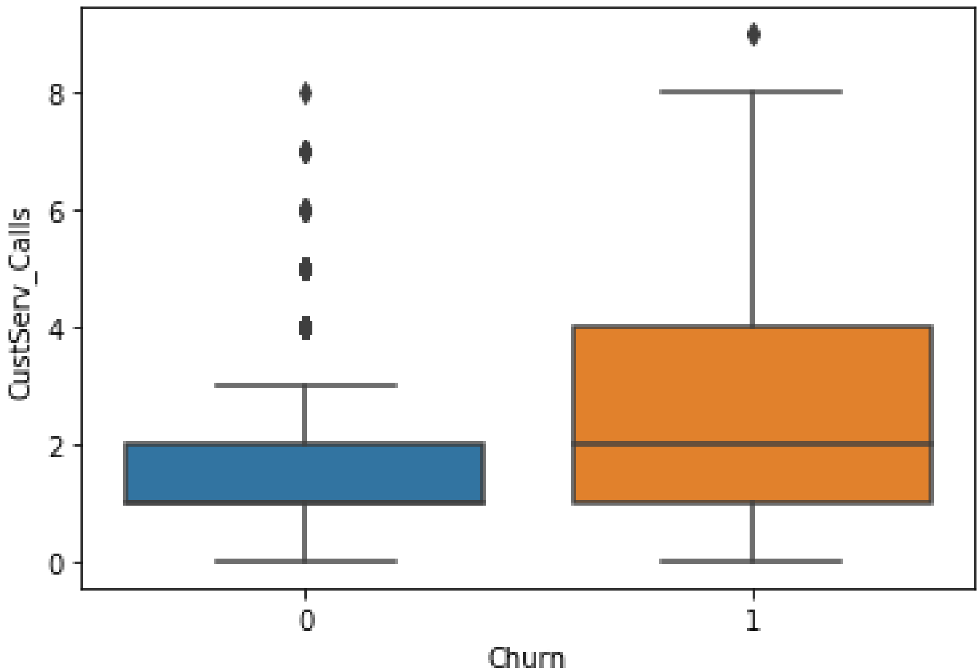
All of the above features appear to be well approximated by the normal distribution. If this were not the case, we would have to consider applying a feature transformation of some kind.

1.0.3 Customer service calls and churn

It can be seen that there's not much of a difference in account lengths between churners and non-churners, but that there is a difference in the number of customer service calls left by churners.

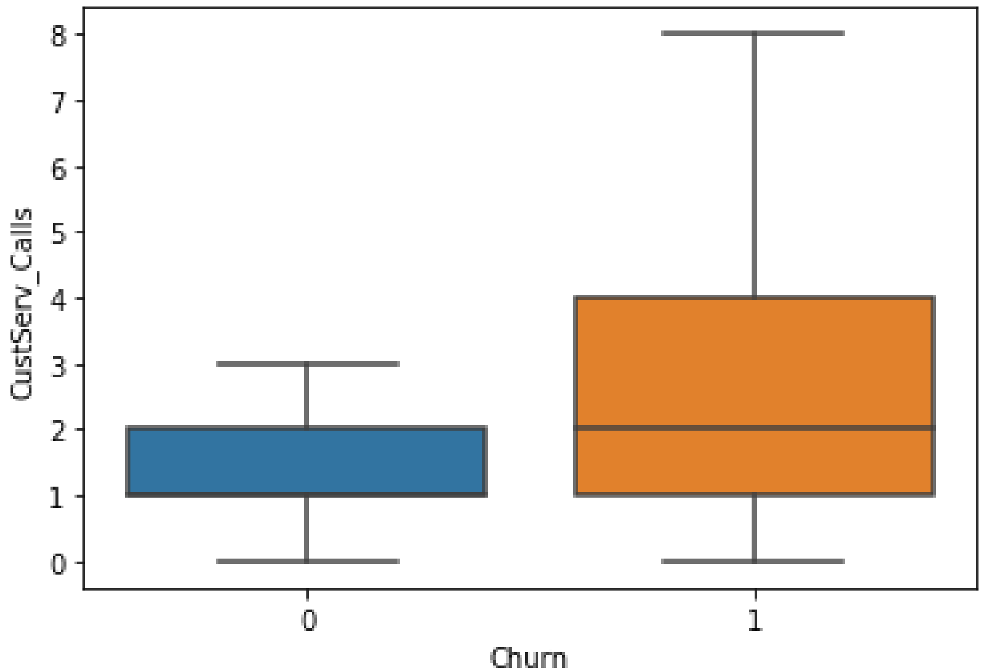
Its time to visualize this difference using a box plot and incorporate other features of interest - do customers who have international plans make more customer service calls? Or do they tend to churn more? How about voicemail plans? Let's find out!

[184]:
```
# Import matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns

# Create the box plot
sns.boxplot(x = 'Churn',
            y = 'CustServ_Calls',
            data = telco)

# Display the plot
plt.show()
```



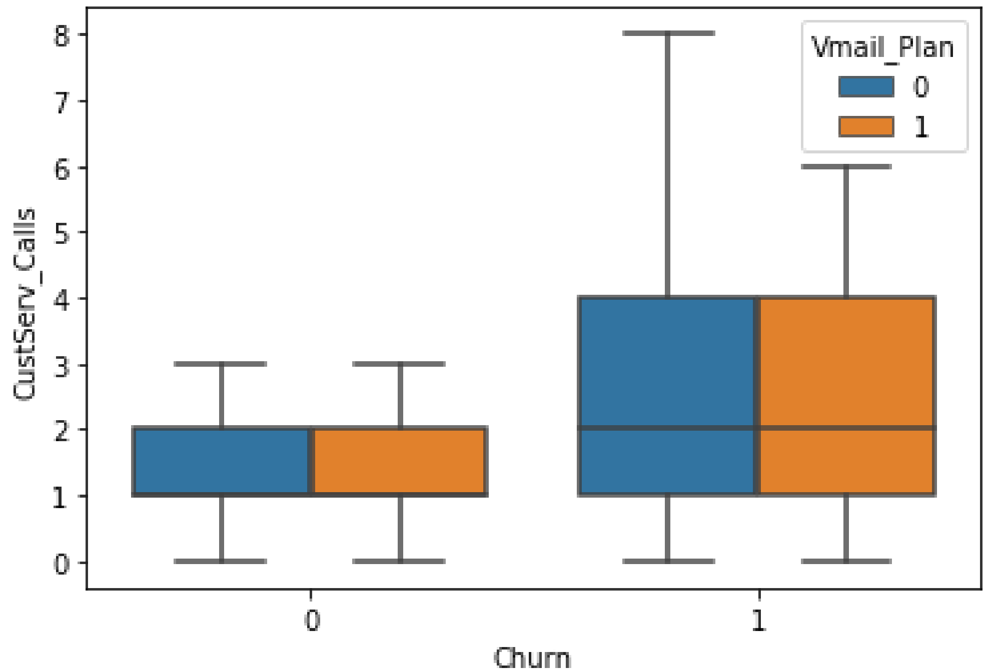
There is a very noticeable difference here between churners and non-churners! Now, outliers from the box plot are removed in the next step.

[186]:
```
# Create the box plot
sns.boxplot(x = 'Churn',
            y = 'CustServ_Calls',
            data = telco,
            sym = "")

# Display the plot
plt.show()
```



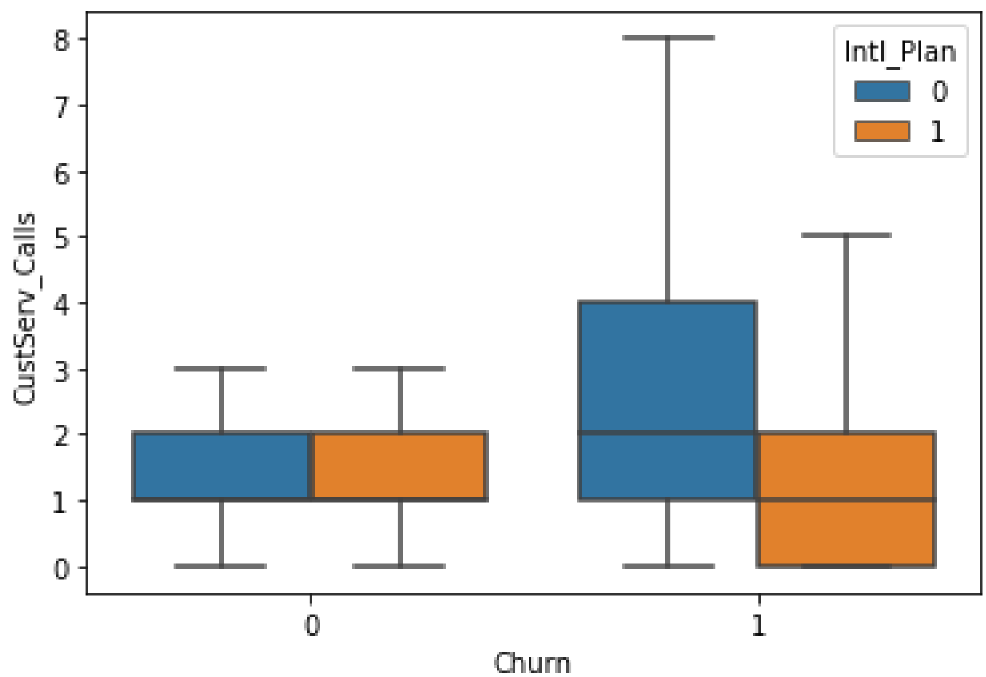
Adding a third variable to this plot - 'Vmail_Plan' - to visualize whether or not having a voice mail plan affects the number of customer service calls or churn.

[188]:
```
# Adding "Vmail_Plan" as a third variable
sns.boxplot(x = 'Churn',
            y = 'CustServ_Calls',
            data = telco,
            sym = "",
            hue = "Vmail_Plan")

# Display the plot
plt.show()
```



Not much of a difference there. Updating the code so that the third variable is 'Intl_Plan' instead.

[190]:
```
# Add "Intl_Plan" as a third variable
sns.boxplot(x = 'Churn',
            y = 'CustServ_Calls',
            data = telco,
            sym = "",
            hue = "Intl_Plan")

# Display the plot
plt.show()
```



It looks like customers who do churn end up leaving more customer service calls, unless these customers also have an international plan, in which case they leave fewer customer service calls. This type of information is really useful in better understanding the drivers of churn. It's now time to learn about how to preprocess the data prior to modeling.

Identifying features to convert

It is preferable to have features like 'Churn' encoded as 0 and 1 instead of no and yes, so that they can be fed into machine learning algorithms that only accept numeric values.

Besides 'Churn', other features that are of type object can be converted into 0s and 1s. In the following codes, different data types of telco will be explored to identify the ones that are of type object.

[194]:
```
telco.dtypes
```

[194]:
```
Account_Length      int64
Vmail_Message       int64
Day_Mins          float64
Eve_Mins          float64
Night_Mins        float64
Intl_Mins         float64
CustServ_Calls      int64
Churn               int64
Intl_Plan           int64
Vmail_Plan          int64
```

```
Day_Calls            int64
Day_Charge         float64
Eve_Calls            int64
Eve_Charge         float64
Night_Calls          int64
Night_Charge       float64
Intl_Calls           int64
Intl_Charge        float64
State               object
Area_Code            int64
Phone               object
dtype: object
```

Indeed! Churn, Vmail_Plan, and Intl_Plan, in particular, are binary features that can easily be converted into 0s and 1s.

Encoding binary features

Recasting data types is an important part of data preprocessing. In the following codes the values 1 to 'yes' and 0 to 'no' will be assigned to the 'Vmail_Plan' and 'Churn' features, respectively.

[203]:
```
# Replacing 'no' with 0 and 'yes' with 1 in 'Vmail_Plan'
telco['Vmail_Plan'] = telco['Vmail_Plan'].replace({'no':0, 'yes':1})
```

[204]:
```
# Replacing 'no' with 0 and 'yes' with 1 in 'Churn'
telco['Churn'] = telco['Churn'].replace({'no':0, 'yes':1})
```

[205]:
```
# Replace 'no' with 0 and 'yes' with 1 in 'Intl_Plan'
telco['Intl_Plan'] = telco['Intl_Plan'].replace({'no':0, 'yes':1})
```

With these features encoded as 0 and 1, these can be used in machine learning algorithms.

One hot encoding

The 'State' feature can be encoded numerically using the technique of one hot encoding.

Doing this manually would be quite tedious, especially when there are 50 states and over 3000 customers! Fortunately, pandas has a get_dummies() function which automatically applies one hot encoding over the selected feature.

Using the pd.get_dummies() function to apply one hot encoding on the 'State' feature of telco.

[206]:
```
# Performing one hot encoding on 'State'
telco_state = pd.get_dummies(telco['State'])

# Printing the head of telco_state
print(telco_state.head())
```

```
   AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL  …  SD  TN  TX  UT  VA  VT  WA  \
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
```

```
2  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0
4  0  0  0  0  0  0  0  0  0  0  …  0  0  0  0  0  0  0

   WI  WV  WY
0   0   0   0
1   0   0   0
2   0   0   0
3   0   0   0
4   0   0   0

[5 rows x 51 columns]
```

[ ]:
```
# It can be noticed that this creates an entirely new DataFrame. Once this
↪dataframe is merged back into the original telco DataFrame, we can begin
↪using these state features in the models. Do note, however, there are now
↪many more features in the dataset, so we should consider dropping any that
↪are unnecessary.
```

Feature scaling

Different features of 'Intl_Calls' and 'Night_Mins' features can be seen.

[207]:
```
telco['Intl_Calls'].describe()
```

[207]:
```
count    3333.000000
mean        4.479448
std         2.461214
min         0.000000
25%         3.000000
50%         4.000000
75%         6.000000
max        20.000000
Name: Intl_Calls, dtype: float64
```

[208]:
```
telco['Night_Mins'].describe()
```

[208]:
```
count    3333.000000
mean      200.872037
std        50.573847
min        23.200000
25%       167.000000
50%       201.200000
75%       235.300000
max       395.000000
Name: Night_Mins, dtype: float64
```

The telco DataFrame has been subset to only include the features that have to be rescaled: 'Intl_Calls' and 'Night_Mins'. To apply StandardScaler, it should be instantiated first by using StandardScaler(), and then applying the fit_transform() method, passing in the DataFrame that has to be rescaled.

[209]:
```
telco_subset = telco[['Intl_Calls', 'Night_Mins']]
print(telco_subset)
```

```
      Intl_Calls  Night_Mins
0              3       244.7
1              3       254.4
2              5       162.6
3              7       196.9
4              3       186.9
…            …          …
3328           6       279.1
3329           4       191.3
3330           6       191.9
3331          10       139.2
3332           4       241.4

[3333 rows x 2 columns]
```

[210]:
```
# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Scale telco using StandardScaler
telco_scaled = StandardScaler().fit_transform(telco_subset)

# Add column names back for readability
telco_scaled_df = pd.DataFrame(telco_scaled, columns=['Intl_Calls',␣
↪'Night_Mins'])

# Print summary statistics
print(telco_scaled_df.describe())
```

```
         Intl_Calls    Night_Mins
count  3.333000e+03  3.333000e+03
mean  -1.264615e-16  6.602046e-17
std    1.000150e+00  1.000150e+00
min   -1.820289e+00 -3.513648e+00
25%   -6.011951e-01 -6.698545e-01
50%   -1.948306e-01  6.485803e-03
75%    6.178983e-01  6.808485e-01
max    6.307001e+00  3.839081e+00
```

Dropping unnecessary features

Some features such as 'Area_Code' and 'Phone' are not useful when it comes to predicting customer churn, and they need to be dropped prior to modeling. The easiest way to do so in Python is using the .drop() method of pandas DataFrames.

[35]:
```python
# Dropping the unnecessary features
telco = telco.drop(['Area_Code', 'Phone'], axis=1)

# Verifying dropped features
print(telco.columns)
```

```
Index(['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins', 'Night_Mins',
       'Intl_Mins', 'CustServ_Calls', 'Churn', 'Intl_Plan', 'Vmail_Plan',
       'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge', 'Night_Calls',
       'Night_Charge', 'Intl_Calls', 'Intl_Charge', 'State'],
      dtype='object')
```

Engineering a new column

Leveraging domain knowledge to engineer new features is an essential part of modeling. This quote from Andrew Ng summarizes the importance of feature engineering:

"Coming up with features is difficult, time-consuming, requires expert knowledge."Applied machine learning" is basically feature engineering".

Next a new feature is created that contains information about the average length of night calls made by customers.

[211]:
```python
# Creating the new feature
telco['Avg_Night_Calls'] = telco['Night_Mins']/telco['Night_Calls']

# Printing the first five rows of 'Avg_Night_Calls'
print(telco['Avg_Night_Calls'].head())
```

```
0    2.689011
1    2.469903
2    1.563462
3    2.212360
4    1.544628
Name: Avg_Night_Calls, dtype: float64
```

[212]:
```python
telco
```

[212]:
```
      Account_Length  Vmail_Message  Day_Mins  Eve_Mins  Night_Mins  \
0                128             25     265.1     197.4       244.7
1                107             26     161.6     195.5       254.4
2                137              0     243.4     121.2       162.6
3                 84              0     299.4      61.9       196.9
4                 75              0     166.7     148.3       186.9
…                  …              …       …         …           …
3328             192             36     156.2     215.5       279.1
3329              68              0     231.1     153.4       191.3
3330              28              0     180.8     288.8       191.9
3331             184              0     213.8     159.6       139.2
```

```
3332             74              25      234.4      265.9       241.4

      Intl_Mins  CustServ_Calls  Churn  Intl_Plan  Vmail_Plan  …  Eve_Calls  \
0          10.0               1      0          0           1  …         99
1          13.7               1      0          0           1  …        103
2          12.2               0      0          0           0  …        110
3           6.6               2      0          1           0  …         88
4          10.1               3      0          1           0  …        122
…           …             …     …         …          … …        …
3328        9.9               2      0          0           1  …        126
3329        9.6               3      0          0           0  …         55
3330       14.1               2      0          0           0  …         58
3331        5.0               2      0          1           0  …         84
3332       13.7               0      0          0           1  …         82

      Eve_Charge  Night_Calls  Night_Charge  Intl_Calls  Intl_Charge  State  \
0          16.78           91         11.01           3         2.70     KS
1          16.62          103         11.45           3         3.70     OH
2          10.30          104          7.32           5         3.29     NJ
3           5.26           89          8.86           7         1.78     OH
4          12.61          121          8.41           3         2.73     OK
…           …          …          …         …          …    …
3328       18.32           83         12.56           6         2.67     AZ
3329       13.04          123          8.61           4         2.59     WV
3330       24.55           91          8.64           6         3.81     RI
3331       13.57          137          6.26          10         1.35     CT
3332       22.60           77         10.86           4         3.70     TN

      Area_Code     Phone  Avg_Night_Calls
0           415  382-4657         2.689011
1           415  371-7191         2.469903
2           415  358-1921         1.563462
3           408  375-9999         2.212360
4           415  330-6626         1.544628
…           …       …              …
3328        415  414-4276         3.362651
3329        415  370-3271         1.555285
3330        510  328-8230         2.108791
3331        510  364-6381         1.016058
3332        415  400-4344         3.135065

[3333 rows x 22 columns]
```

[213]:
```
# Creating feature variable
```

```
X = telco[['Account_Length', 'Vmail_Message', 'Day_Mins', 'Eve_Mins',␣
↪'Night_Mins', 'Intl_Mins', 'CustServ_Calls', 'Intl_Plan', 'Vmail_Plan',␣
↪'Day_Calls', 'Day_Charge', 'Eve_Calls', 'Eve_Charge', 'Night_Calls',␣
↪'Night_Charge', 'Intl_Calls', 'Intl_Charge']]
y = telco['Churn']
```

[214]:
```
X.dtypes
```

[214]:
```
Account_Length      int64
Vmail_Message       int64
Day_Mins          float64
Eve_Mins          float64
Night_Mins        float64
Intl_Mins         float64
CustServ_Calls      int64
Intl_Plan           int64
Vmail_Plan          int64
Day_Calls           int64
Day_Charge        float64
Eve_Calls           int64
Eve_Charge        float64
Night_Calls         int64
Night_Charge      float64
Intl_Calls          int64
Intl_Charge       float64
dtype: object
```

Predicting whether a new customer will churn

The first argument consists of the features, while the second argument is the label that will be predicted - whether or not the customer will churn. After the model is fitted, we can see the model's .predict() method to predict the label of a new customer.

[215]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
↪random_state = 0)
```

[217]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[269]:
```
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression

# Instantiate the classifier
```

```
clf = LogisticRegression(random_state = 0)


# Fitting the classifier
clf.fit(X_train, y_train)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[269]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

[270]:
```
# Predicting the Test set results
y_pred = clf.predict(X_test)
```

[271]:
```
# Compute accuracy
print(clf.score(X_test, y_test))
```

```
0.8717026378896883
```

[272]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[707  12]
 [ 95  20]]
```

Based on the model we can predict whether a new customer will Churn or not.

[274]:
```
new_customer_file_path = '/Users/MuhammadBilal/Desktop/Data Camp/Marketing␣
 ↪Analytics- Predicting Customer Churn in Python/new_customer.csv'
```

[275]:
```
new_customer = pd.read_csv(new_customer_file_path)
```

[276]:
```
print(new_customer)
```

```
   Account_Length  Vmail_Message  Day_Mins  Eve_Mins  Night_Mins  Intl_Mins  \
0             128             25     265.1     197.4       244.7         10

   CustServ_Calls  Intl_Plan  Vmail_Plan  Day_Calls  Day_Charge  Eve_Calls  \
0               1          1           0        110          77         99
```

```
   Eve_Charge  Night_Calls  Night_Charge  Intl_Calls  Intl_Charge
0       16.78           91         11.01           3          2.7
```

[277]:
```
# Predicting the label of new_customer
print(clf.predict(new_customer))
```

```
[1]
```

[284]:
```
# Applying Decision Tree Classifier

# Importing DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

# Instantiating the classifier
classifier = DecisionTreeClassifier()

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state = 0)

# Fitting the classifier
classifier.fit(X_train, y_train)
```

[284]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

[285]:
```
# Predicting the label of new_customer
print(classifier.predict(new_customer))
```

```
[1]
```

[286]:
```
# Printing the confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[707  12]
 [ 95  20]]
```

[287]:
```
# Computing accuracy
print(classifier.score(X_test, y_test))
```

```
0.9136690647482014
```

Applying Random Forest Classifier

[288]:
```python
# Import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Instantiate the classifier
clf = RandomForestClassifier()

# Fit to the training data
clf.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = clf.predict(X_test)

# Import confusion_matrix
from sklearn.metrics import confusion_matrix

# Print confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[710   9]
 [ 30  85]]
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:245:
FutureWarning: The default value of n_estimators will change from 10 in version
0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[290]:
```python
# Computing precision and recall
# Importing precision_score
from sklearn.metrics import precision_score

# Printing the precision of the classifier.
# Printing the precision
print(precision_score(y_test, y_pred))
```

```
0.9042553191489362
```

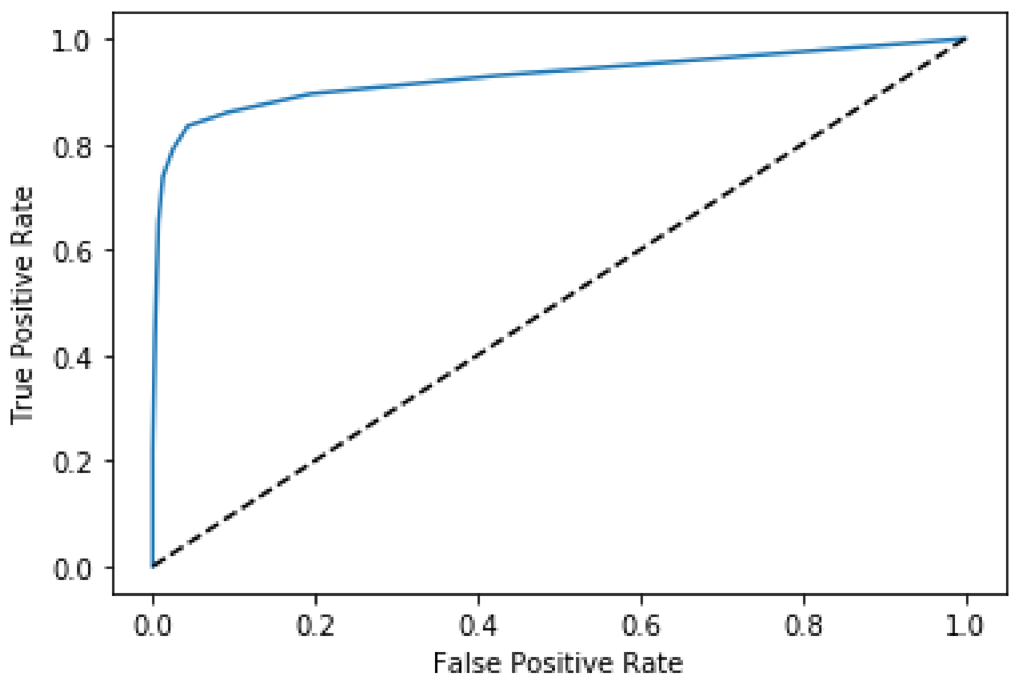
[292]:
```python
# Importing recall_score
from sklearn.metrics import recall_score

# Printing the recall
print(recall_score(y_test, y_pred))
```

```
0.7391304347826086
```

1.0.4 ROC curve

An ROC curve for our random forest classifier is created. The first step is to calculate the predicted probabilities output by the classifier for each label using its .predict_proba() method. Then, we can use the roc_curve function from sklearn.metrics to compute the false positive rate and true positive rate, which can then be plotted using matplotlib.

[293]:
```
# Generating the probabilities
y_pred_prob = clf.predict_proba(X_test)[:, 1]

# Importing roc_curve
from sklearn.metrics import roc_curve

# Calculating the roc metrics
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Plotting the ROC curve
plt.plot(fpr, tpr)

# Adding labels and diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.plot([0, 1], [0, 1], "k--")
plt.show()
```



1.0.5 Area under the curve

Visually, it looks like a well-performing model. It can be quantified by computing the area under the curve.

[294]:
```
# Importing roc_auc_score
from sklearn.metrics import roc_auc_score

# Printing the AUC
print(roc_auc_score(y_test, y_pred_prob))
```

```
0.9282034226280462
```

1.0.6 F1 score

There's a tradeoff between precision and recall. Both are important metrics, and depending on how the business is trying to model churn, we may want to focus on optimizing one over the other. Often, stakeholders are interested in a single metric that can quantify model performance. The AUC is one metric that can be used in these cases, and another is the F1 score, which is calculated as below:

2 * (precision * recall) / (precision + recall)

The advantage of the F1 score is it incorporates both precision and recall into a single metric, and a high F1 score is a sign of a well-performing model, even in situations where we might have imbalanced classes. In scikit-learn, we can compute the f-1 score using using the f1_score function.

[295]:
```
# Importing f1_score
from sklearn.metrics import f1_score

# Printing the F1 score
print(f1_score(y_pred, y_test))
```

```
0.8133971291866029
```

Among all the used machine algorithms, Random Forest showing the best results as it predicted highest number of TPs and TNs.

1.0.7 Tuning the number of features

The default hyperparameters used by the models are not optimized for the data. The goal of grid search cross-validation is to identify those hyperparameters that lead to optimal model performance. Here, we will practice tuning the max_features hyperparameter.

Purpose of hyperparameters is to select a number of featuers for best split.

A random forest is an ensemble of many decision trees. The n_estimators hyperparameter controls the number of trees to use in the forest, while the max_features hyperparameter controls the number of features the random forest should consider when looking for the best split at decision tree.

[296]:
```
# Importing GridSearchCV
from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid
param_grid = {'max_features': ['auto', 'sqrt', 'log2']}
```

```python
# Calling GridSearchCV
grid_search = GridSearchCV(clf, param_grid)

# Fitting the model
grid_search.fit(X, y)

# Printing the optimal parameters
print(grid_search.best_params_)
```

```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default
value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
{'max_features': 'auto'}
```

1.0.8 Tuning other hyperparameters

The power of GridSearchCV really comes into play when multiple hyperparameters are tuned, as then the algorithm tries out all possible combinations of hyperparameters to identify the best combination. Here, following random forest hyperparameters will be tuned:

Hyperparameter - Purpose

criterion - Quality of Split

max_features - Number of features for best split

max_depth - Max depth of tree

bootstrap - Whether Bootstrap samples are used

[298]:
```python
# Importing GridSearchCV
from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid
param_grid = {"max_depth": [3, None],
              "max_features": [1, 3, 10],
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}

# Instantiating the GridSearchCV object using clf and param_grid.
# Call GridSearchCV
grid_search = GridSearchCV(clf, param_grid)

# Fitting the model
grid_search.fit(X, y)
```

```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default
value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

[298]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=10, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'bootstrap': [True, False],
                         'criterion': ['gini', 'entropy'],
                         'max_depth': [3, None], 'max_features': [1, 3, 10]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In the above chunk of code, the first line of code did not take much time to run, while the call to .fit() took several seconds to execute.

This is because .fit() is what actually performs the grid search, and in our case, it was grid with many different combinations. As the hyperparameter grid gets larger, grid search becomes slower. In order to solve this problem, instead of trying out every single combination of values, we could randomly jump around the grid and try different combinations. There's a small possibility we may miss the best combination, but we would save a lot of time, or be able to tune more hyperparameters in the same amount of time.

In scikit-learn, we can do this using RandomizedSearchCV. It has the same API as GridSearchCV, except that we may need to specify a parameter distribution that it can sample from instead of specific hyperparameter values.

[301]:
```
# Importing RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV

# Creating the hyperparameter grid
param_dist = {"max_depth": [3, None],
              "max_features": (1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}
```

```
# Calling RandomizedSearchCV
random_search = RandomizedSearchCV(clf, param_dist)

# Fitting the model
random_search.fit(X, y)

# Printing best parameters
print(random_search.best_params_)
```

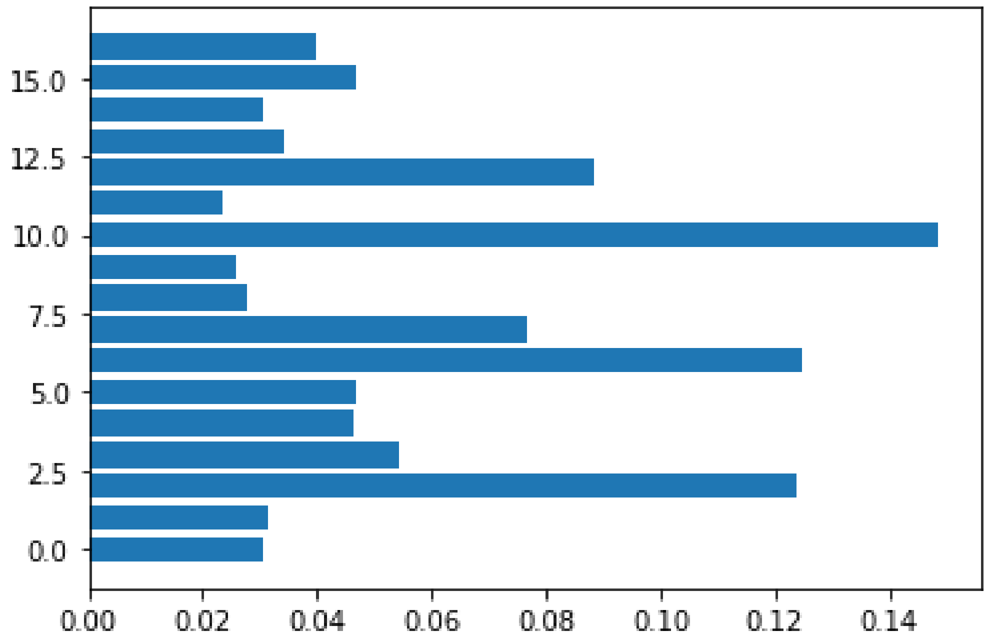
```
/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default
value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
{'max_features': 11, 'max_depth': None, 'criterion': 'entropy', 'bootstrap':
True}
```

1.0.9 Visualizing feature importances

The random forest classifier from earlier exercises has been fit to the telco data and is available as clf. Let's visualize the feature importances and get a sense for what the drivers of churn are, using matplotlib's barh to create a horizontal bar plot of feature importances.

[302]:
```
# Calculating feature importances
importances = clf.feature_importances_

# Creating plot
plt.barh(range(X.shape[1]), importances)
plt.show()
```



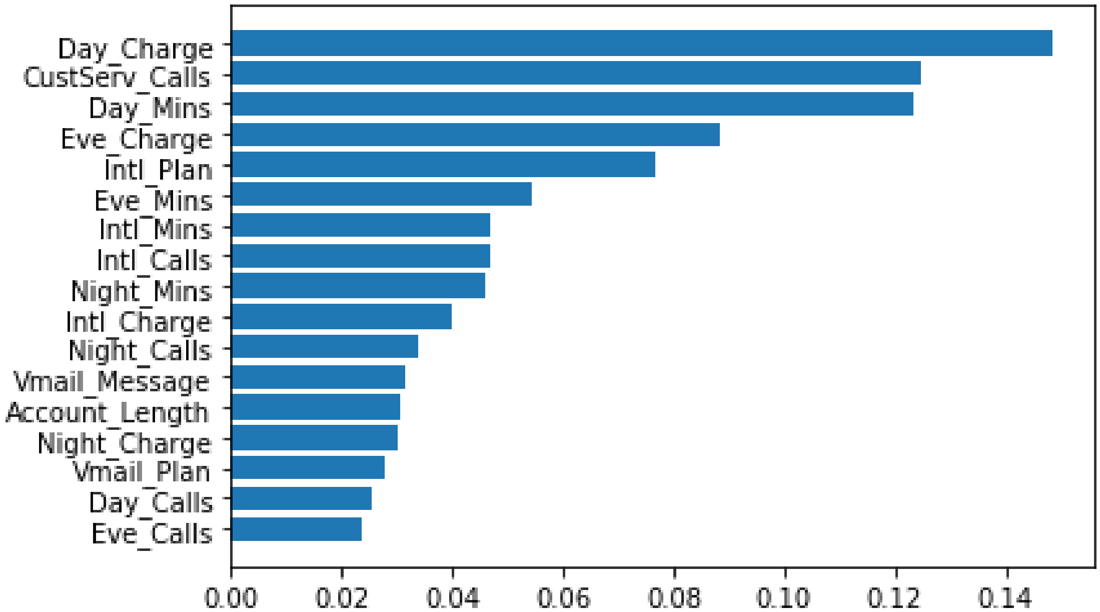
1.0.10 Improving the plot

In order to make the plot more readable, we need to do achieve two goals:

Re-order the bars in ascending order.

Add labels to the plot that correspond to the feature names.

[303]:
```
import numpy as np
# Sort importances
sorted_index = np.argsort(importances)

# Creating labels
labels = X.columns[sorted_index]

# Clearing current plot
plt.clf()

# Creating plot
plt.barh(range(X.shape[1]), importances[sorted_index], tick_label=labels)
plt.show()
```



The plot tells us that CustServ_Calls, Day_Mins and Day_Charge are the most important drivers of churn.

Here, the 'Churn' column is further explored to see if there are differences between churners and non-churners. A subset version of the telco DataFrame, consisting of the columns 'Churn', 'CustServ_Calls', and 'Vmail_Message' is available in the workspace.

[364]:
```
# Groupping telco by 'Churn' and computing the mean
print(telco.groupby(['Churn']).mean())
```

```
       Account_Length  Vmail_Message    Day_Mins    Eve_Mins  Night_Mins  \
Churn
0          100.793684       8.604561  175.175754  199.043298  200.133193
1          102.664596       5.115942  206.914079  212.410145  205.231677

       Intl_Mins  CustServ_Calls  Intl_Plan  Vmail_Plan   Day_Calls  \
Churn
0      10.158877        1.449825   0.065263    0.295439  100.283158
1      10.700000        2.229814   0.283644    0.165631  101.335404

       Day_Charge   Eve_Calls  Eve_Charge  Night_Calls  Night_Charge  \
Churn
0       29.780421  100.038596   16.918909   100.058246      9.006074
1       35.175921  100.561077   18.054969   100.399586      9.235528

       Intl_Calls  Intl_Charge   Area_Code  Avg_Night_Calls
Churn
```

```
0        4.532982     2.743404  437.074737          2.084904
1        4.163561     2.889545  437.817805          2.132133
```

The code is adapted to compute the standard deviation instead of the mean.

[365]:
```python
# Adapting the code to compute the standard deviation
print(telco.groupby(['Churn']).std())
```

```
       Account_Length  Vmail_Message   Day_Mins   Eve_Mins  Night_Mins  \
Churn
0            39.88235      13.913125  50.181655  50.292175   51.105032
1            39.46782      11.860138  68.997792  51.728910   47.132825

       Intl_Mins  CustServ_Calls  Intl_Plan  Vmail_Plan  Day_Calls  \
Churn
0       2.784489        1.163883   0.247033    0.456320  19.801157
1       2.793190        1.853275   0.451233    0.372135  21.582307

       Day_Charge  Eve_Calls  Eve_Charge  Night_Calls  Night_Charge  \
Churn
0        8.530835  19.958414    4.274863    19.506246      2.299768
1       11.729710  19.724711    4.396762    19.950659      2.121081

       Intl_Calls  Intl_Charge  Area_Code  Avg_Night_Calls
Churn
0        2.441984     0.751784  42.306156         0.714886
1        2.551575     0.754152  42.792270         0.676741
```

Based on the results it can be seen that Churners make more customer service calls than non-churners.

1.0.11 Conclusion and Recommendations

From above summary statistics it is evident that churning customers are using more Day_Mins and Day_Calls. This gives us a hint that customers using more day minutes are not satisfied with the service or charges and they tend to churn.

Customers who are using Intl_Plan and make Intl_Calls have a tendency to churn and it can also be deduced from the box plots that customers using Intl_Plan are making more CustServ_Calls.

From Above figure it can also be seen that Day_Charge, Day_Mins and CustServ_Calls are one of the most important factors for churn.

In order to reduce the churn, telco should redesign its Intl_Plan and Day_Call plans and should mitigate the reasons for customers to make frequent customer service calls.

[ ]:
```

```