**Manaranjan Pradhan**
**manaranjan@enablecloud.com**
*This notebook is given as part of **Data Science for everyone** workshop.*
*(Forwarding this document to others is strictly prohibited.)*

# Classification: Building multiple models ¶

- Decision Tree
- Random Forest
- Naive Bayes

In [2]:

```
import pandas as pd
import numpy as np
```

In [3]:

```
churn = pd.read_csv( "churn.csv" )
```

In [4]:

```
churn.head()
```

Out[4]:

| | State | Account Length | Area Code | Phone | Int'l Plan | VMail Plan | VMail Message | Day Mins | Day Calls | Day Charge | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | KS | 128 | 415 | 382-4657 | no | yes | 25 | 265.1 | 110 | 45.07 | ... |
| **1** | OH | 107 | 415 | 371-7191 | no | yes | 26 | 161.6 | 123 | 27.47 | ... |
| **2** | NJ | 137 | 415 | 358-1921 | no | no | 0 | 243.4 | 114 | 41.38 | ... |
| **3** | OH | 84 | 408 | 375-9999 | yes | no | 0 | 299.4 | 71 | 50.90 | ... |
| **4** | OK | 75 | 415 | 330-6626 | yes | no | 0 | 166.7 | 113 | 28.34 | ... |

5 rows × 21 columns

In [5]:

```
churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3333 entries, 0 to 3332
Data columns (total 21 columns):
State             3333 non-null object
Account Length    3333 non-null int64
Area Code         3333 non-null int64
Phone             3333 non-null object
Int'l Plan        3333 non-null object
VMail Plan        3333 non-null object
VMail Message     3333 non-null int64
Day Mins          3333 non-null float64
Day Calls         3333 non-null int64
Day Charge        3333 non-null float64
Eve Mins          3333 non-null float64
Eve Calls         3333 non-null int64
Eve Charge        3333 non-null float64
Night Mins        3333 non-null float64
Night Calls       3333 non-null int64
Night Charge      3333 non-null float64
Intl Mins         3333 non-null float64
Intl Calls        3333 non-null int64
Intl Charge       3333 non-null float64
CustServ Calls    3333 non-null int64
Churn?            3333 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 572.9+ KB
```

In [83]:

```
churn.columns
```

Out[83]:

```
Index(['State', 'Account Length', 'Area Code', 'Phone', 'Int'l Plan',
       'VMail Plan', 'VMail Message', 'Day Mins', 'Day Calls', 'Day C
harge',
       'Eve Mins', 'Eve Calls', 'Eve Charge', 'Night Mins', 'Night Ca
lls',
       'Night Charge', 'Intl Mins', 'Intl Calls', 'Intl Charge',
       'CustServ Calls', 'Churn?'],
      dtype='object')
```

# Data Cleaning

In [7]:

```
drop_columns = ['State','Area Code','Phone','Churn?',
                "Int'l Plan", "VMail Plan"]
```

In [8]:

```
churn_new = churn.drop( drop_columns, axis = 1 )
```

In [9]:

```
churn_new.head()
```

Out[9]:

| | Account Length | VMail Message | Day Mins | Day Calls | Day Charge | Eve Mins | Eve Calls | Eve Charge | Night Mins | Night Calls |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 128 | 25 | 265.1 | 110 | 45.07 | 197.4 | 99 | 16.78 | 244.7 | 91 |
| **1** | 107 | 26 | 161.6 | 123 | 27.47 | 195.5 | 103 | 16.62 | 254.4 | 103 |
| **2** | 137 | 0 | 243.4 | 114 | 41.38 | 121.2 | 110 | 10.30 | 162.6 | 104 |
| **3** | 84 | 0 | 299.4 | 71 | 50.90 | 61.9 | 88 | 5.26 | 196.9 | 89 |
| **4** | 75 | 0 | 166.7 | 113 | 28.34 | 148.3 | 122 | 12.61 | 186.9 | 121 |

In [10]:

```
churn_new["Intl Plan"] = np.where( churn["Int'l Plan"] == 'yes',
                                   1, 0 )
```

In [11]:

```
churn_new["VMail Plan"] = np.where( churn["Int'l Plan"] == 'yes',
                                    1, 0 )
```

In [12]:

```
churn_new["churn"] = np.where( churn['Churn?'] == 'True.',1,0)
```

In [13]:

```
churn_new.head()
```

Out[13]:

| | Account Length | VMail Message | Day Mins | Day Calls | Day Charge | Eve Mins | Eve Calls | Eve Charge | Night Mins | Night Calls |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 128 | 25 | 265.1 | 110 | 45.07 | 197.4 | 99 | 16.78 | 244.7 | 91 |
| **1** | 107 | 26 | 161.6 | 123 | 27.47 | 195.5 | 103 | 16.62 | 254.4 | 103 |
| **2** | 137 | 0 | 243.4 | 114 | 41.38 | 121.2 | 110 | 10.30 | 162.6 | 104 |
| **3** | 84 | 0 | 299.4 | 71 | 50.90 | 61.9 | 88 | 5.26 | 196.9 | 89 |
| **4** | 75 | 0 | 166.7 | 113 | 28.34 | 148.3 | 122 | 12.61 | 186.9 | 121 |

In [14]:

```
churn_new.columns = [name.replace(' ', '_') for name in churn_new.columns]
```

In [84]:

```
churn_new.head()
```

Out[84]:

| | Account_Length | VMail_Message | Day_Mins | Day_Calls | Day_Charge | Eve_ |
|---|---|---|---|---|---|---|
| **0** | 128 | 25 | 265.1 | 110 | 45.07 | 197.4 |
| **1** | 107 | 26 | 161.6 | 123 | 27.47 | 195.5 |
| **2** | 137 | 0 | 243.4 | 114 | 41.38 | 121.2 |
| **3** | 84 | 0 | 299.4 | 71 | 50.90 | 61.9 |
| **4** | 75 | 0 | 166.7 | 113 | 28.34 | 148.3 |

## Exploration

In [16]:

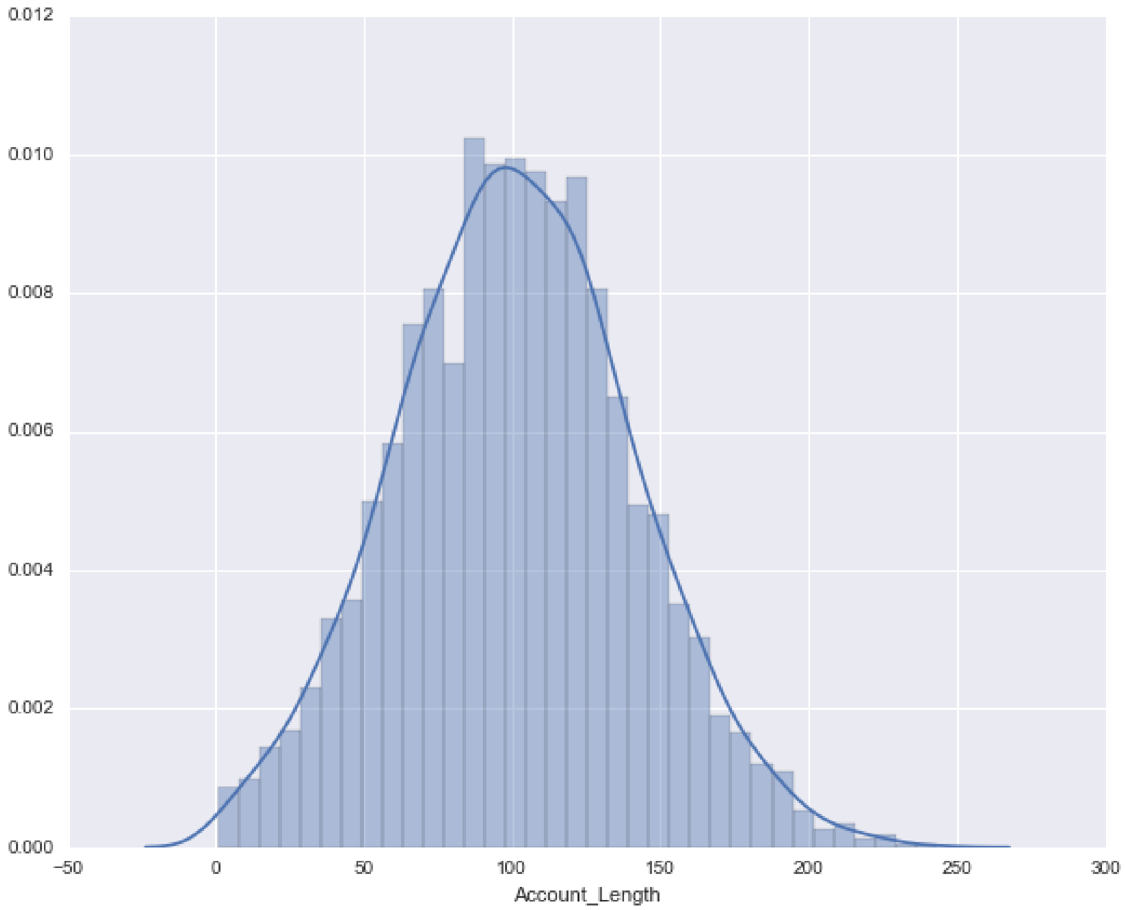
```
import matplotlib as plt
import seaborn as sn
import matplotlib.pyplot as pyplt
%matplotlib inline
```

In [17]:

```
pyplt.figure(figsize=(10, 8))
sn.distplot( churn_new.Account_Length )
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x921b4a8>

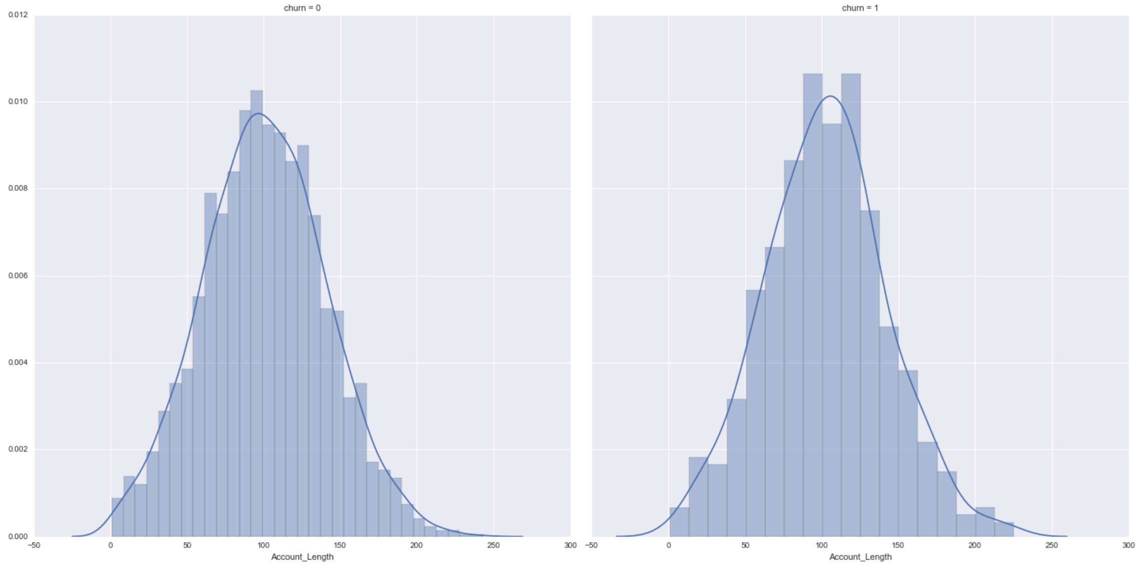


In [18]:

```
g = sn.FacetGrid( churn_new, col="churn", size = 10 )
g.map( sn.distplot, "Account_Length" )
```

Out[18]:

```
<seaborn.axisgrid.FacetGrid at 0x7ee3828>
```




In [19]:

```
churn_by_cs = pd.crosstab( churn_new.churn,
                           churn_new.CustServ_Calls )
```

In [20]:

```
churn_by_cs
```

Out[20]:

| CustServ_Calls | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| churn | | | | | | | | | | |
| 0 | 605 | 1059 | 672 | 385 | 90 | 26 | 8 | 4 | 1 | 0 |
| 1 | 92 | 122 | 87 | 44 | 76 | 40 | 14 | 5 | 1 | 2 |

In [21]:

```
churn_by_cs = pd.DataFrame( churn_by_cs.unstack() ).reset_index()
churn_by_cs.head()
```

Out[21]:

| | CustServ_Calls | churn | 0 |
|---|---|---|---|
| **0** | 0 | 0 | 605 |
| **1** | 0 | 1 | 92 |
| **2** | 1 | 0 | 1059 |
| **3** | 1 | 1 | 122 |
| **4** | 2 | 0 | 672 |

In [22]:

```
churn_by_cs = pd.crosstab( churn_new.churn,
                           churn_new.CustServ_Calls )                      \
    .apply( lambda calls: calls / calls.sum(), axis = 0 ) * 100

churn_by_cs = pd.DataFrame( churn_by_cs.unstack() ).reset_index()

churn_by_cs.head()
```

Out[22]:

| | CustServ_Calls | churn | 0 |
|---|---|---|---|
| **0** | 0 | 0 | 86.800574 |
| **1** | 0 | 1 | 13.199426 |
| **2** | 1 | 0 | 89.669771 |
| **3** | 1 | 1 | 10.330229 |
| **4** | 2 | 0 | 88.537549 |

In [23]:

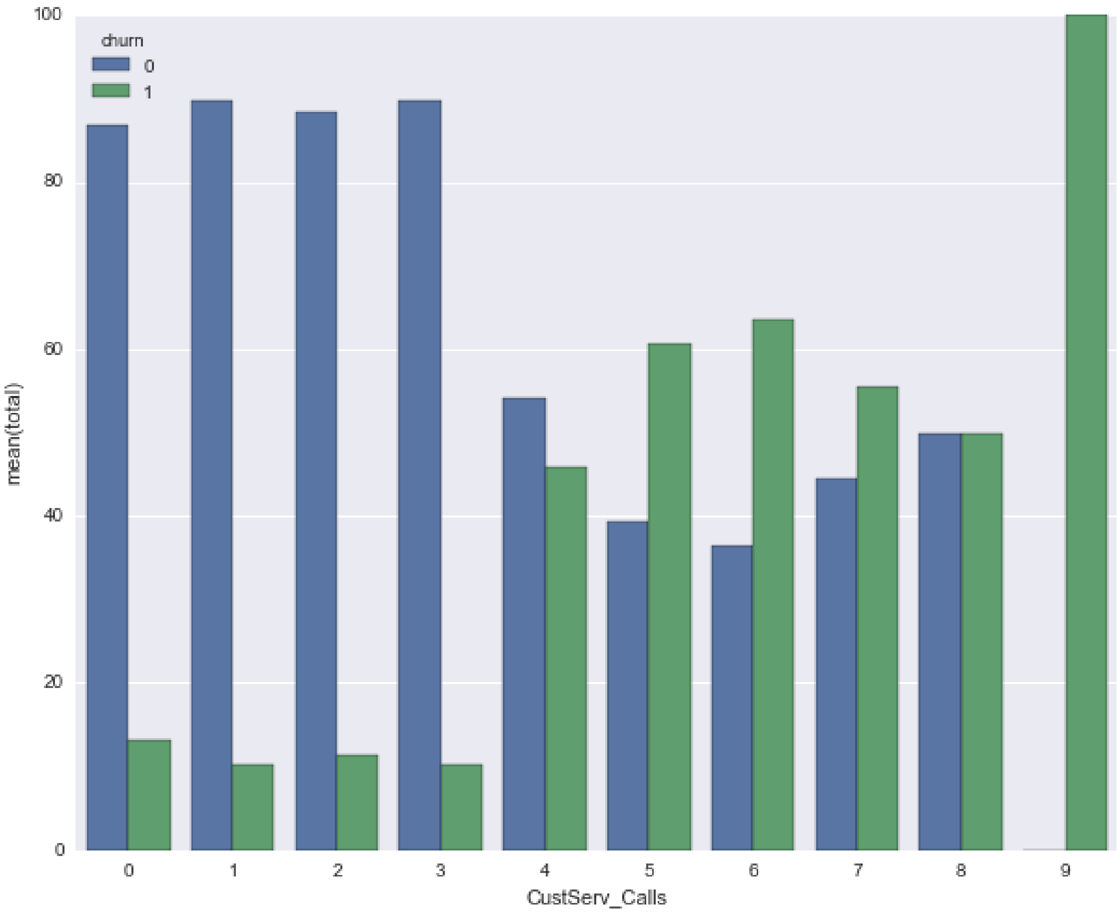
```
churn_by_cs.columns = [ "CustServ_Calls", "churn", "total" ]
```

In [24]:

```
pyplt.figure(figsize=(10, 8))

sn.barplot( churn_by_cs.CustServ_Calls,
           churn_by_cs.total,
           hue = churn_by_cs.churn )
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x943cda0>
```




**The charges and minutes are highly correlated. So, only one variable can be used.**

In [25]:

```
churn_new.corr()
```

Out[25]:

| | Account_Length | VMail_Message | Day_Mins | Day_Calls | Da |
|---|---|---|---|---|---|
| **Account_Length** | 1.000000 | -0.004628 | 0.006216 | 0.038470 | 0.0 |
| **VMail_Message** | -0.004628 | 1.000000 | 0.000778 | -0.009548 | 0.0 |
| **Day_Mins** | 0.006216 | 0.000778 | 1.000000 | 0.006750 | 1.0 |
| **Day_Calls** | 0.038470 | -0.009548 | 0.006750 | 1.000000 | 0.0 |
| **Day_Charge** | 0.006214 | 0.000776 | 1.000000 | 0.006753 | 1.0 |
| **Eve_Mins** | -0.006757 | 0.017562 | 0.007043 | -0.021451 | 0.0 |
| **Eve_Calls** | 0.019260 | -0.005864 | 0.015769 | 0.006462 | 0.0 |
| **Eve_Charge** | -0.006745 | 0.017578 | 0.007029 | -0.021449 | 0.0 |
| **Night_Mins** | -0.008955 | 0.007681 | 0.004323 | 0.022938 | 0.0 |
| **Night_Calls** | -0.013176 | 0.007123 | 0.022972 | -0.019557 | 0.0 |
| **Night_Charge** | -0.008960 | 0.007663 | 0.004300 | 0.022927 | 0.0 |
| **Intl_Mins** | 0.009514 | 0.002856 | -0.010155 | 0.021565 | -0. |
| **Intl_Calls** | 0.020661 | 0.013957 | 0.008033 | 0.004574 | 0.0 |
| **Intl_Charge** | 0.009546 | 0.002884 | -0.010092 | 0.021666 | -0. |
| **CustServ_Calls** | -0.003796 | -0.013263 | -0.013423 | -0.018942 | -0. |
| **Intl_Plan** | 0.024735 | 0.008745 | 0.049396 | 0.003755 | 0.0 |
| **VMail_Plan** | 0.024735 | 0.008745 | 0.049396 | 0.003755 | 0.0 |
| **churn** | 0.016541 | -0.089728 | 0.205151 | 0.018459 | 0.2 |

In [26]:

```
import re
churn_cols = churn_new.columns
```

In [27]:

```
churn_drop_cols = [ x for x in churn_cols if
                    re.search( 'Charge', x ) or
                    re.search( 'Calls', x )]
```

In [28]:

```
churn_drop_cols
```

Out[28]:

```
['Day_Calls',
 'Day_Charge',
 'Eve_Calls',
 'Eve_Charge',
 'Night_Calls',
 'Night_Charge',
 'Intl_Calls',
 'Intl_Charge',
 'CustServ_Calls']
```

In [29]:

```
churn_drop_cols = churn_drop_cols[:-1]
```

In [30]:

```
churn_final = churn_new.drop( churn_drop_cols, axis = 1 )
```

In [31]:

```
churn_final.head()
```

Out[31]:

| | Account_Length | VMail_Message | Day_Mins | Eve_Mins | Night_Mins | Intl_M |
|---|---|---|---|---|---|---|
| **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 |

## Split Dataset into train and test

In [32]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split
```

In [33]:

```
feature_cols = churn_final.columns[:-1].tolist()
```

In [49]:

```
X_train, X_test, y_train, y_test = train_test_split( churn_final[feature_cols],
                                                      churn_final.churn,
                                                      test_size = 0.3,
                                                      random_state = 21 )
```

## Building a Logistic Regression Model

In [35]:

```
logreg = LogisticRegression()
scores = cross_val_score(logreg, X_train, y_train, cv=10, scoring='accuracy')
```

In [36]:

```
scores
```

Out[36]:

```
array([ 0.85897436,  0.84615385,  0.82478632,  0.83333333,  0.8589743
6,
        0.8583691 ,  0.86266094,  0.87553648,  0.87931034,  0.875
])
```

In [37]:

```
scores.mean()
```

Out[37]:

```
0.85730990906549587
```

In [38]:

```
logreg.fit( X_train, y_train )
```

Out[38]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercep
t=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.
0001,
          verbose=0)
```

In [39]:

```
y_predict = logreg.predict( X_test )
```

In [40]:

```
from sklearn import metrics
```

In [41]:

```
cm = metrics.confusion_matrix( y_test, y_predict )
```

In [42]:

```
sn.heatmap(cm, annot=True,  fmt='.2f' );
```


In [43]:

```
metrics.accuracy_score( y_test, y_predict )
```

Out[43]:

```
0.86099999999999999
```

# Building a Decision Tree Model

In [44]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [45]:

```
treeClf = DecisionTreeClassifier()
scores = cross_val_score(treeClf, X_train, y_train, cv=10, scoring='accuracy')
```

In [46]:

```
scores
```

Out[46]:

```
array([ 0.87606838,  0.88461538,  0.86324786,  0.89316239,  0.8931623
9,
        0.90987124,  0.88841202,  0.89270386,  0.86637931,  0.8706896
6])
```

## Verifying the optimal tree depth

In [50]:

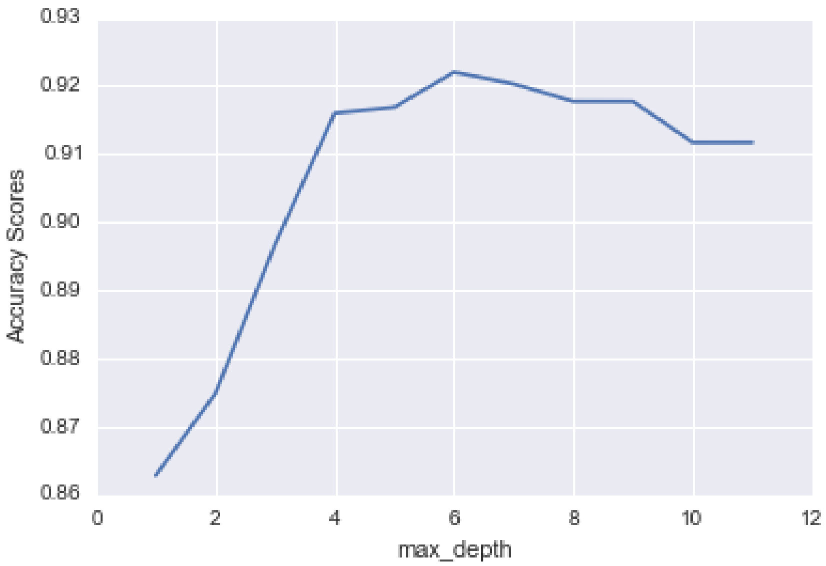
```
max_depth_range = range(1, 12)

# list to store the average RMSE for each value of max_depth
accuracy_scores = []

# use LOOCV with each value of max_depth
for depth in max_depth_range:
    treereg = DecisionTreeClassifier(max_depth=depth, random_state=1)
    scores = cross_val_score(treereg, X_train, y_train, cv=14, scoring='accuracy')
    accuracy_scores.append( scores.mean() )
```

In [51]:

```
# plot max_depth (x-axis) versus RMSE (y-axis)
pyplt.plot( max_depth_range, accuracy_scores )
pyplt.xlabel( 'max_depth' )
pyplt.ylabel('Accuracy Scores')
```

Out[51]:

```
<matplotlib.text.Text at 0xc00e668>
```



In [78]:

```
# max_depth=3 was best, so fit a tree using that parameter
churn_tree_clf = DecisionTreeClassifier(max_depth=6, random_state=1)
churn_tree_clf.fit(X_train, y_train)
```

Out[78]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth
=6,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=
1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=1, splitter='best')
```

In [79]:

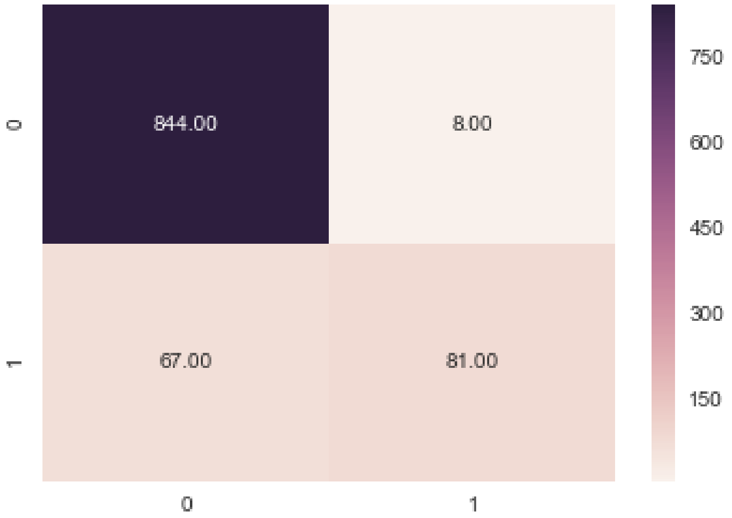
```
pd.DataFrame({'feature':feature_cols,
              'importance':churn_tree_clf.feature_importances_})
```

Out[79]:

| | feature | importance |
|---|---|---|
| **0** | Account_Length | 0.008279 |
| **1** | VMail_Message | 0.069224 |
| **2** | Day_Mins | 0.326225 |
| **3** | Eve_Mins | 0.190501 |
| **4** | Night_Mins | 0.031833 |
| **5** | Intl_Mins | 0.101552 |
| **6** | CustServ_Calls | 0.147903 |
| **7** | Intl_Plan | 0.000000 |
| **8** | VMail_Plan | 0.124483 |

In [80]:

```
y_predict = churn_tree_clf.predict( X_test )
cm = metrics.confusion_matrix( y_test, y_predict )
sn.heatmap(cm, annot=True,  fmt='.2f' );
```




In [81]:

```
metrics.accuracy_score( y_test, y_predict )
```

Out[81]:

0.9250000000000004

## Create tree graph

Download and Install **Graphviz** from http://www.graphviz.org/Download..php (http://www.graphviz.org/Download..php)

Add the path (for example C:\Program Files (x86)\Graphviz2.38\bin) to your PATH variable.

In [82]:

```
from sklearn.tree import export_graphviz

export_graphviz(churn_tree_clf, out_file='churn_tree.odt',
                feature_names=feature_cols)
```

*Now convert the dot file to png file*

**dot -Tpng filename.dot -o outfile.png**

Now you can open and see the decision tree diagram in any editor.

decision tree

# Random Forest Classifier

In [72]:

```
from sklearn.ensemble import RandomForestClassifier
rfClf = RandomForestClassifier( n_estimators= 50 )
scores = cross_val_score(rfClf,
                         X_train,
                         y_train,
                         cv=10,
                         scoring='accuracy')

scores
```

Out[72]:

```
array([ 0.92735043,  0.92307692,  0.94017094,  0.92735043,  0.9230769
2,
        0.9527897 ,  0.93562232,  0.92274678,  0.90517241,  0.9310344
8])
```

In [73]:

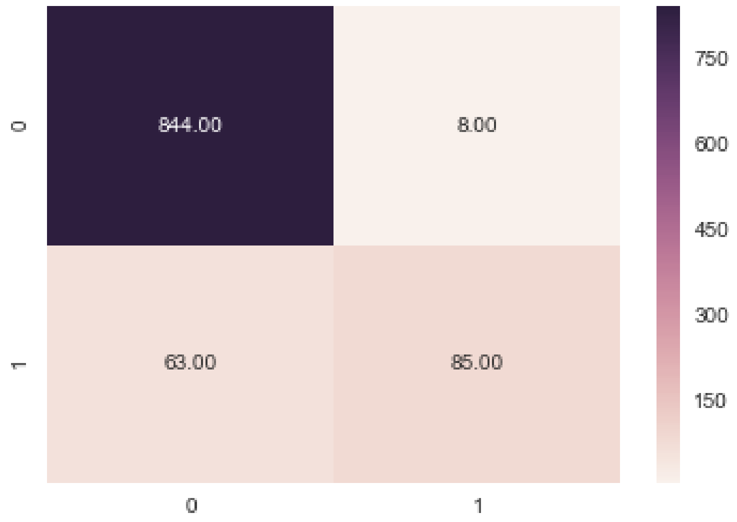
```
scores.mean()
```

Out[73]:

```
0.92883913358606274
```

In [74]:

```
rfClf.fit( X_train, y_train )
y_predict = rfClf.predict( X_test )
cm = metrics.confusion_matrix( y_test, y_predict )
sn.heatmap(cm, annot=True,  fmt='.2f' );
```




In [85]:

```
metrics.accuracy_score( y_test, y_predict )
```

Out[85]:

```
0.92500000000000004
```

# Naive Bayes Model

In [76]:

```
from sklearn.naive_bayes  import GaussianNB
nbClf = GaussianNB()
scores = cross_val_score(nbClf,
                         X_train,
                         y_train,
                         cv=10,
                         scoring='accuracy')

scores
```

Out[76]:

```
array([ 0.83760684,  0.84188034,  0.8034188 ,  0.81196581,  0.8547008
5,
        0.86266094,  0.84978541,  0.87982833,  0.87068966,  0.8362069
])
```

In [86]:

```
scores.mean()
```

Out[86]:

```
0.84487438794083736
```