In [86]:

```
from sklearn import model_selection
from sklearn.model_selection import cross_validate
from sklearn import tree
from sklearn import svm
from sklearn import ensemble
from sklearn import neighbors
from sklearn import linear_model
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import StratifiedKFold
```

In [41]:

```
%matplotlib inline

from IPython.display import Image
import matplotlib as mlp
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import sklearn
import seaborn as sns
```

In [43]:

```
#df = pd.read_csv('../input/mytest.csv')
df = pd.read_csv('/content/bigml_59c28831336c6604c800002a.csv')

print (df.shape)

#df.dtypes
```

```
(3333, 21)
```

In [44]:

```
# Load data
df.head(3)
```

Out[44]:

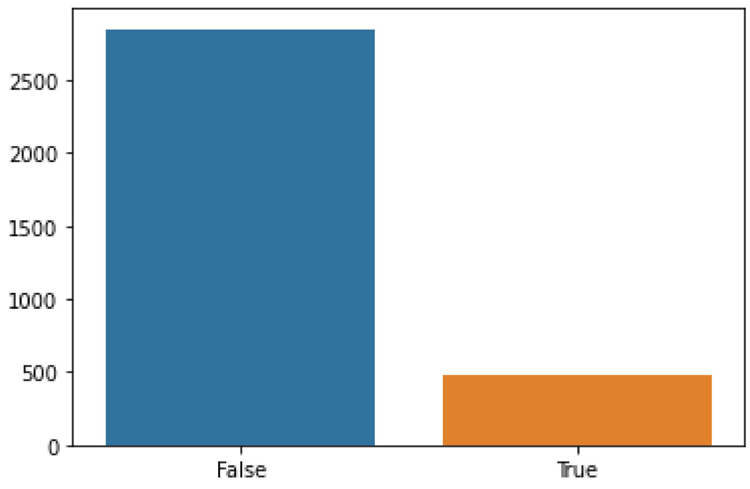
	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110

3 rows × 21 columns

In [45]:
```
y = df["churn"].value_counts()
#print (y)
sns.barplot(y.index, y.values)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c000dcd10>
```



In [46]:
```
y_True = df["churn"][df["churn"] == True]
print ("Churn Percentage = "+str( (y_True.shape[0] / df["churn"].shape[0]) * 100
```

```
Churn Percentage = 14.491449144914492
```

Conclusion 1 = Imbalanced data - Lesser datapoints in True Churn category

Descriptive Analysis

In [47]:
```
df.describe()
```

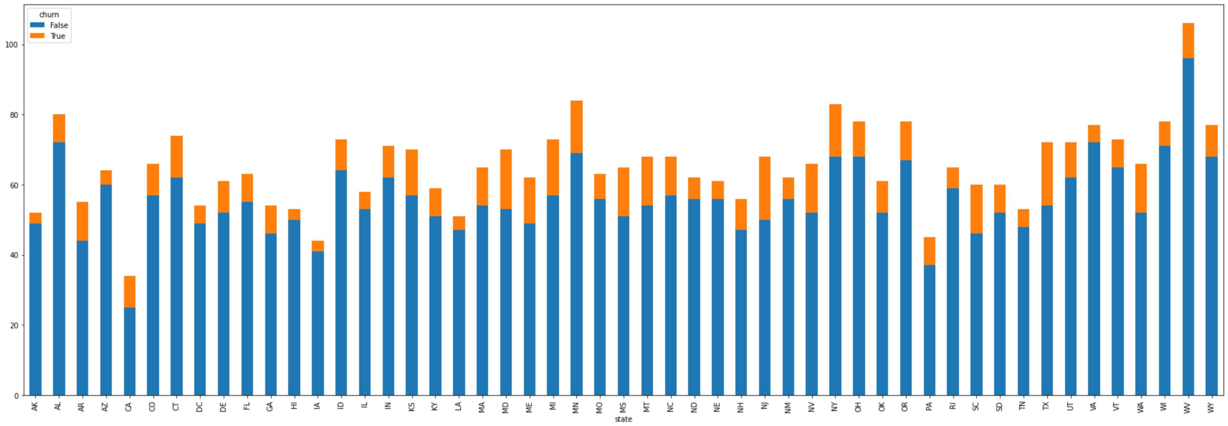
Out[47]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000

Churn By State

In [48]:
```
df.groupby(["state", "churn"]).size().unstack().plot(kind='bar', stacked=True, fi
```

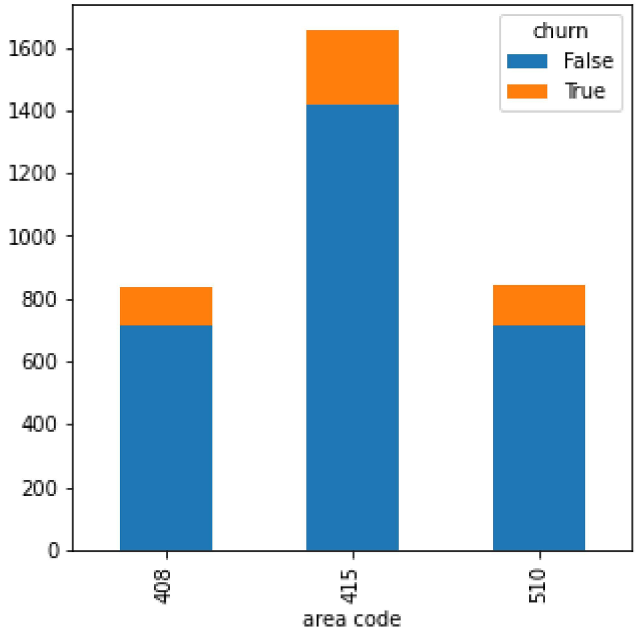
Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c000d0450>
```



Churn By Area Code

```
In [49]: df.groupby(["area code", "churn"]).size().unstack().plot(kind='bar', stacked=True
```

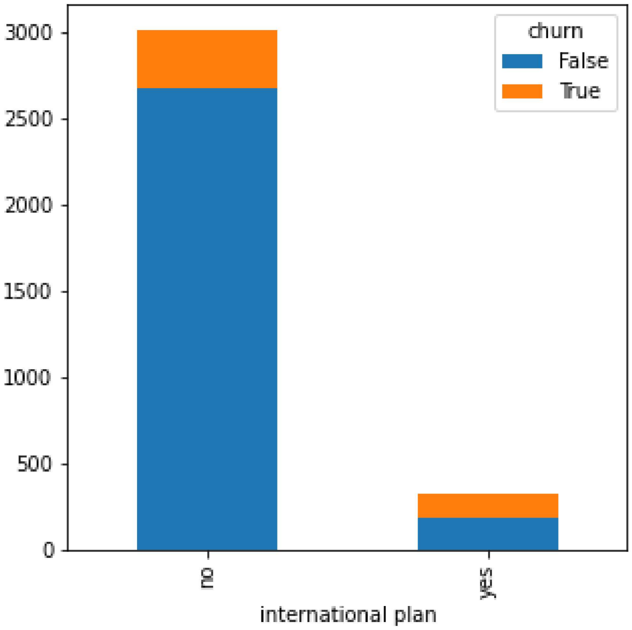
```
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7bffe20850>
```



Churn By Customers with International plan

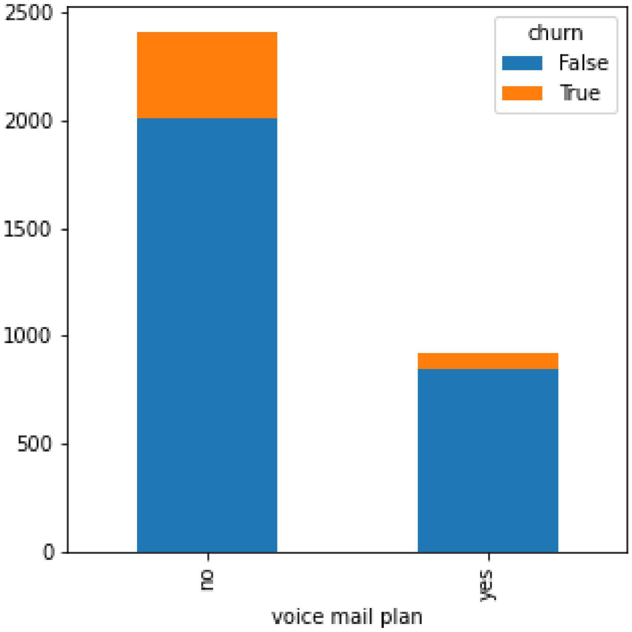
```
In [50]: df.groupby(["international plan", "churn"]).size().unstack().plot(kind='bar', sta
```

```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7bffe17810>
```



In [51]:
```
#Churn By Customers with Voice mail plan
df.groupby(["voice mail plan", "churn"]).size().unstack().plot(kind='bar', stacke
```

Out[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7bffd6e910>
```



Handle Categorical Cols - Label Encode

In [52]:
```
# Discreet value integer encoder
label_encoder = preprocessing.LabelEncoder()
```

In [53]:

```
# State is string and we want discreet integer values
df['state'] = label_encoder.fit_transform(df['state'])
df['international plan'] = label_encoder.fit_transform(df['international plan'])
df['voice mail plan'] = label_encoder.fit_transform(df['voice mail plan'])

#print (df['Voice mail plan'][:4])
print (df.dtypes)
```

```
state                       int64
account length              int64
area code                   int64
phone number               object
international plan          int64
voice mail plan             int64
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

In [54]:

```
df.shape
```

Out[54]:

```
(3333, 21)
```

In [55]:

```
df.head()
```

Out[55]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota eve calls
0	16	128	415	382-4657	0	1	25	265.1	110	45.07	...	99
1	35	107	415	371-7191	0	1	26	161.6	123	27.47	...	103
2	31	137	415	358-1921	0	0	0	243.4	114	41.38	...	110
3	35	84	408	375-9999	1	0	0	299.4	71	50.90	...	88
4	36	75	415	330-6626	1	0	0	166.7	113	28.34	...	122

5 rows × 21 columns

Strip of Response value

In [57]:

```
y = df['churn'].to_numpy().astype(np.int)
y.size
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarn
ing: `np.int` is a deprecated alias for the builtin `int`. To silence this warn
ing, use `int` by itself. Doing this will not modify any behavior and is safe.
When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to s
pecify the precision. If you wish to review your current use, check the release
note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  """Entry point for launching an IPython kernel.
```

Out[57]:

```
3333
```

Strip off Redundant cols

In [58]:

```
# df = df.drop(["Id","Churn"], axis = 1, inplace=True)
df.drop(["phone number","churn"], axis = 1, inplace=True)
```

In [59]:
```
df.head(3)
```

Out[59]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	c
0	16	128	415	0	1	25	265.1	110	45.07	197.4	99	
1	35	107	415	0	1	26	161.6	123	27.47	195.5	103	
2	31	137	415	0	0	0	243.4	114	41.38	121.2	110	

Build Feature Matrix

In [61]:
```
X = df.to_numpy().astype(np.float)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarn
ing: `np.float` is a deprecated alias for the builtin `float`. To silence this
warning, use `float` by itself. Doing this will not modify any behavior and is
safe. If you specifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  """Entry point for launching an IPython kernel.
```

In [62]:
```
X
```

Out[62]:
```
array([[ 16.  , 128.  , 415.  , ...,   3.  ,   2.7 ,   1.  ],
       [ 35.  , 107.  , 415.  , ...,   3.  ,   3.7 ,   1.  ],
       [ 31.  , 137.  , 415.  , ...,   5.  ,   3.29,   0.  ],
       ...,
       [ 39.  ,  28.  , 510.  , ...,   6.  ,   3.81,   2.  ],
       [  6.  , 184.  , 510.  , ...,  10.  ,   1.35,   2.  ],
       [ 42.  ,  74.  , 415.  , ...,   4.  ,   3.7 ,   0.  ]])
```

In [63]:
```
X.shape
```

Out[63]:
```
(3333, 19)
```

Standardize Feature Matrix values

In [64]:
```
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)
```

In [65]:
```
X
```

Out[65]:
```
array([[-0.6786493 ,  0.67648946, -0.52360328, ..., -0.60119509,
        -0.0856905 , -0.42793202],
       [ 0.6031696 ,  0.14906505, -0.52360328, ..., -0.60119509,
         1.2411686 , -0.42793202],
       [ 0.33331299,  0.9025285 , -0.52360328, ...,  0.21153386,
         0.69715637, -1.1882185 ],
       ...,
       [ 0.87302621, -1.83505538,  1.71881732, ...,  0.61789834,
         1.3871231 ,  0.33235445],
       [-1.35329082,  2.08295458,  1.71881732, ...,  2.24335625,
        -1.87695028,  0.33235445],
       [ 1.07541867, -0.67974475, -0.52360328, ..., -0.19483061,
         1.2411686 , -1.1882185 ]])
```

Build Models and Train

In [89]:
```
# Create Train & Test Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
```

In [90]:
```
# Running logistic regression model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
result = model.fit(X_train, y_train)
```

In [91]:
```
from sklearn import metrics
prediction_test = model.predict(X_test)
# Print the prediction accuracy
print (metrics.accuracy_score(y_test, prediction_test))
```

```
0.859
```

In [92]:
```
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
model_rf = RandomForestClassifier(n_estimators=1000 , oob_score = True, n_jobs =
                                  random_state =50, max_features = "auto",
                                  max_leaf_nodes = 30)
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))
```

```
0.9250374812593704
```

In [93]:

```
#SUPPORT VECTOR MACHINE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
from sklearn.svm import SVC
model.svm = SVC(kernel='linear')
model.svm.fit(X_train,y_train)
preds = model.svm.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

Out[93]:

```
0.8860569715142429
```

In [94]:

```
# Create the Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,preds))
```

```
[[591   0]
 [ 76   0]]
```

In [97]:

```
# AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier()
# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)
model.fit(X_train,y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

Out[97]:

```
0.8995502248875562
```

In [98]:

```
#XGBOOST
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(X_train, y_train)
preds = model.predict(X_test)
metrics.accuracy_score(y_test, preds)
```

Out[98]:

```
0.967016491754123
```

with XG Boost I was able to increase the accuracy on test data to almost 96%

Clearly, XG Boost is a winner among all other techniques

XG Boost is a slow learning model and is based on the concept of Boosting