Predicting Churn with Machine Learning (Python Code)

By Margaret Catherman, May 2023

copy of Predicting Churn with Machine Learning.Rmd + format: ph2_join_w_model.v8_approach_B_w_eda

Prepare the environment

In [93]:
```
import pandas as pd
import numpy as np
import requests
import math
import geopandas as gpd
from shapely import wkt
from shapely.geometry import Point
import ast
import re
from dataprep.clean import clean_address
import seaborn as sns
from datetime import datetime
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn import ensemble
from sklearn.preprocessing import OrdinalEncoder
from sklearn import preprocessing
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import cohen_kappa_score
from sklearn.preprocessing import LabelEncoder
import lazypredict
from lazypredict.Supervised import LazyClassifier
import matplotlib.pyplot as plt
from datetime import date
%matplotlib inline
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import random
from matplotlib import pyplot as plt #plotting


pd.options.display.max_columns = None
pd.options.display.max_rows = None
```

In [94]:
```
pd.set_option('display.float_format', '{:.5f}'.format)
```

Churn Data

Data prepared from "Predicting Churn with Machine Learning" in Rmb,
file:///Users/margaretcatherman/Downloads/Predicting%20Churn%20with%20Machine%20Learning.nb.html#

In [3]:

```
result_3 = pd.read_csv('Churn_prepped.csv')
result_3.info()
result_3.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3774 entries, 0 to 3773
Data columns (total 36 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   CustomerID                            3774 non-null   int64  
 1   Churn                                 3774 non-null   int64  
 2   Tenure                                3774 non-null   int64  
 3   CityTier                              3774 non-null   int64  
 4   WarehouseToHome                       3774 non-null   int64  
 5   HourSpendOnApp                        3774 non-null   int64  
 6   NumberOfDeviceRegistered              3774 non-null   int64  
 7   SatisfactionScore                     3774 non-null   int64  
 8   NumberOfAddress                       3774 non-null   int64  
 9   Complain                              3774 non-null   int64  
 10  OrderAmountHikeFromlastYear           3774 non-null   int64  
 11  CouponUsed                            3774 non-null   int64  
 12  OrderCount                            3774 non-null   int64  
 13  DaySinceLastOrder                     3774 non-null   int64  
 14  CashbackAmount                        3774 non-null   float64
 15  PreferredLoginDeviceComputer          3774 non-null   int64  
 16  PreferredLoginDeviceMobile.Phone      3774 non-null   int64  
 17  PreferredLoginDevicePhone             3774 non-null   int64  
 18  PreferredPaymentModeCash.on.Delivery  3774 non-null   int64  
 19  PreferredPaymentModeCC                3774 non-null   int64  
 20  PreferredPaymentModeCOD               3774 non-null   int64  
 21  PreferredPaymentModeCredit.Card       3774 non-null   int64  
 22  PreferredPaymentModeDebit.Card        3774 non-null   int64  
 23  PreferredPaymentModeE.wallet          3774 non-null   int64  
 24  PreferredPaymentModeUPI               3774 non-null   int64  
 25  GenderFemale                          3774 non-null   int64  
 26  GenderMale                            3774 non-null   int64  
 27  PreferedOrderCatFashion               3774 non-null   int64  
 28  PreferedOrderCatGrocery               3774 non-null   int64  
 29  PreferedOrderCatLaptop...Accessory    3774 non-null   int64  
 30  PreferedOrderCatMobile                3774 non-null   int64  
 31  PreferedOrderCatMobile.Phone          3774 non-null   int64  
 32  PreferedOrderCatOthers                3774 non-null   int64  
 33  MaritalStatusDivorced                 3774 non-null   int64  
 34  MaritalStatusMarried                  3774 non-null   int64  
 35  MaritalStatusSingle                   3774 non-null   int64  
dtypes: float64(1), int64(35)
memory usage: 1.0 MB
```

Out[3]:

	CustomerID	Churn	Tenure	CityTier	WarehouseToHome	HourSpendOnApp	NumberOfDeviceRegistered	Satis
0	50001	1	4	3	6	3	3	
1	50004	1	0	3	15	2	4	
2	50006	1	0	1	22	3	5	
3	50012	1	11	1	6	3	4	
4	50013	1	0	1	11	2	3	

In [54]:

```
result_3.Churn.value_counts()
```

Out[54]:

```
0    3143
1     631
Name: Churn, dtype: int64
```

In [4]:
```
# Labels are the values we want to predict
y = np.array(result_3['Churn'])
# Remove the labels from the features
X= result_3.drop(columns=['Churn'], axis = 1)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
X_train_clean=X_train.drop(columns=['CustomerID'])  #FireIndicator_fi
X_test_clean=X_test.drop(columns='CustomerID')
print('Training Features Shape:', X_train_clean.shape)
print('Training Labels Shape:', y_train.shape)
print('Testing Features Shape:', X_test_clean.shape)
print('Testing Labels Shape:', y_test.shape)
feature_list = list(X_train_clean.columns)
```

```
Training Features Shape: (2830, 34)
Training Labels Shape: (2830,)
Testing Features Shape: (944, 34)
Testing Labels Shape: (944,)
```

Optional: See recomended models from LazyClassifier

In [5]:
```
'''
For this classification task we are going to use random forest, as it was used in pase 1,
and a second mode to compare.  We use LazyPredict module to pick the best performing mode

'''

clf = LazyClassifier(verbose=0, ignore_warnings=False, custom_metric=None)
models, predictions = clf.fit(X_train_clean, X_test_clean, y_train, y_test)
model_dictionary = clf.provide_models(X_train_clean, X_test_clean, y_train, y_test)
models
```

```
 28%|███       | 8/29 [00:02<00:04,  4.64it/s]
CategoricalNB model failed to execute
Negative values in data passed to CategoricalNB (input X)
 66%|███████   | 19/29 [00:04<00:01,  7.15it/s]
NuSVC model failed to execute
specified nu is infeasible
 90%|█████████ | 26/29 [00:05<00:00,  6.45it/s]
StackingClassifier model failed to execute
__init__() missing 1 required positional argument: 'estimators'
100%|██████████| 29/29 [00:06<00:00,  4.32it/s]
```

Out[5]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
XGBClassifier	0.97	0.95	0.95	0.97	0.85
LGBMClassifier	0.97	0.94	0.94	0.97	0.37
DecisionTreeClassifier	0.96	0.93	0.93	0.96	0.04
RandomForestClassifier	0.96	0.91	0.91	0.96	0.49
BaggingClassifier	0.95	0.90	0.90	0.95	0.16
LabelPropagation	0.94	0.88	0.88	0.94	0.42
LabelSpreading	0.94	0.88	0.88	0.94	0.58
ExtraTreesClassifier	0.95	0.86	0.86	0.95	0.44
ExtraTreeClassifier	0.90	0.81	0.81	0.90	0.02

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
AdaBoostClassifier	0.89	0.76	0.76	0.89	0.34
NearestCentroid	0.76	0.75	0.75	0.78	0.02
SVC	0.90	0.72	0.72	0.88	0.34
LogisticRegression	0.88	0.72	0.72	0.87	0.06
Perceptron	0.83	0.71	0.71	0.83	0.04
LinearSVC	0.88	0.70	0.70	0.87	0.29
CalibratedClassifierCV	0.88	0.70	0.70	0.87	1.42
BernoulliNB	0.85	0.70	0.70	0.85	0.04
LinearDiscriminantAnalysis	0.88	0.70	0.70	0.86	0.10
KNeighborsClassifier	0.87	0.67	0.67	0.85	0.29
PassiveAggressiveClassifier	0.81	0.66	0.66	0.81	0.03
SGDClassifier	0.86	0.65	0.65	0.84	0.09
RidgeClassifier	0.87	0.62	0.62	0.83	0.06
RidgeClassifierCV	0.87	0.62	0.62	0.83	0.04
GaussianNB	0.24	0.54	0.54	0.20	0.03
QuadraticDiscriminantAnalysis	0.19	0.51	0.51	0.09	0.03
DummyClassifier	0.72	0.49	0.49	0.72	0.03

1. AdaBoostClassifier

In [55]:

```
np.any(np.isnan(X_train_clean))
```

Out[55]:

```
False
```

In [56]:

```
from sklearn.tree import DecisionTreeClassifier
dtclf = DecisionTreeClassifier(max_depth=1, criterion='gini',
                               random_state=1)
dtclf.fit(X_train_clean, y_train)
#from sklearn.metrics import accuracy_score
dtclf_train_sc = accuracy_score(y_train, dtclf.predict(X_train_clean))
dtclf_test_sc = accuracy_score(y_test, dtclf.predict(X_test_clean))
print('Decision tree train/test accuracies %.3f/%.3f' % (dtclf_train_sc, dtclf_test_sc))
```

```
Decision tree train/test accuracies 0.839/0.832
```

In [57]:

```
#Grid to get best parameters; time consuming, does not to be re-run each time
random.seed(1299)
hyperparameter_space = {'n_estimators':list(range(2, 102, 2)),
                        'learning_rate':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
                       'algorithm': ['SAMME', 'SAMME.R']}
from sklearn.model_selection import GridSearchCV
gs = GridSearchCV(AdaBoostClassifier(base_estimator=dtclf,
                                     algorithm='SAMME.R',
                                      random_state=1),
                param_grid=hyperparameter_space,
```

```
                scoring="accuracy", n_jobs=-1, cv=5)

gs.fit(X_train_clean, y_train)
print("Optimal hyperparameter combination:", gs.best_params_)
```

```
Optimal hyperparameter combination: {'algorithm': 'SAMME.R', 'learning_rate': 0.5, 'n_esti
mators': 32}
```

In [68]:

```
random.seed(1299)
classifier = AdaBoostClassifier(algorithm='SAMME.R',base_estimator=dtclf,
    learning_rate = 0.5,
    n_estimators=32, random_state=42
)
classifier.fit(X_train_clean, y_train)
```

Out[68]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         random_state=1),
                   learning_rate=0.5, n_estimators=32, random_state=42)
```

In [69]:

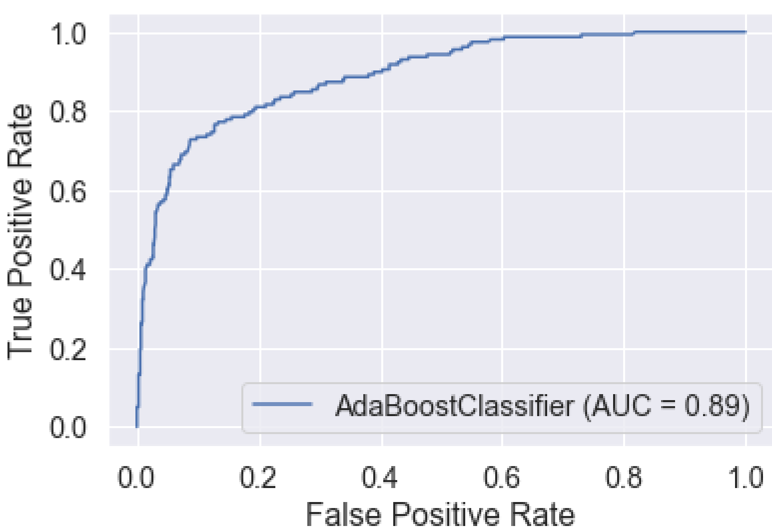
```
predictions = classifier.predict(X_test_clean)
confusion_matrix(y_test, predictions)
```

Out[69]:

```
array([[763,  23],
       [ 81,  77]])
```

In [70]:

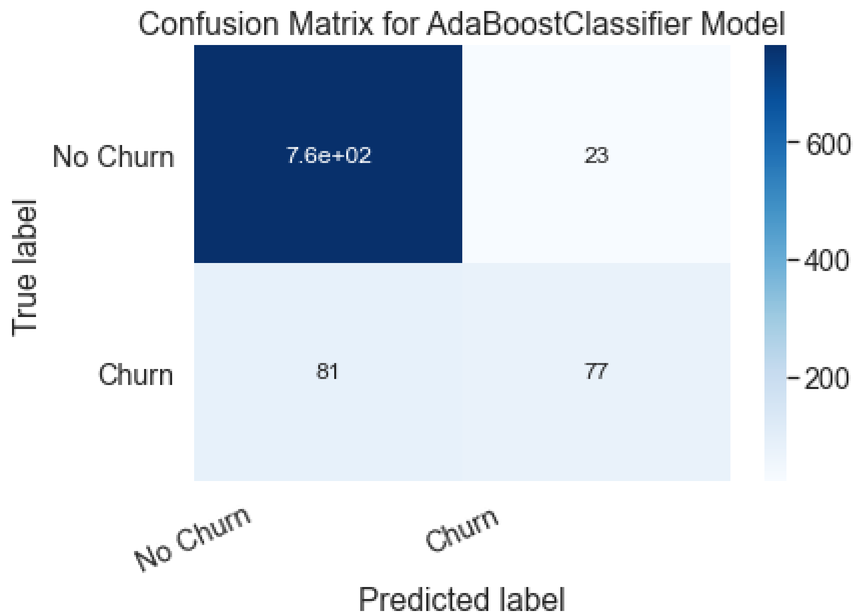
```
#plot AUC  #MESSED UP?
metrics.plot_roc_curve(clf, X_test_clean, y_test)
plt.show() #AUC .89, .90, .92, .94, .95, vif: .95 .95


plot_rf_confusion(rf_model)
```



In [71]:

```
# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, predictions)
# Build the plot
plt.figure(figsize=(6,4))
sns.set(font_scale=1.3)
sns.heatmap(matrix, annot=True, annot_kws={'size':12},
            cmap=plt.cm.Blues)
# Add labels to the plot
class_names = ['No Churn','Churn']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
```

```
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for AdaBoostClassifier Model')
plt.show()
```



In [72]:

```
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, predictions))
#87
```

```
ACCURACY OF THE MODEL:  0.8898305084745762
```

In [73]:

```
print(classification_report(y_test, predictions)) #model 2
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.94       786
           1       0.77      0.49      0.60       158

    accuracy                           0.89       944
   macro avg       0.84      0.73      0.77       944
weighted avg       0.88      0.89      0.88       944
```

In [74]:

```
#Cohen_kappa_score .46
cohen_kappa_score(y_test, predictions)
```

Out[74]:

```
0.5368007850834151
```

2. Random Forest

In [30]:

```
# Keep; just run until best parameters establised.
# Parameter Grid: Used to get best parameters for best fit/best results.
# Best param manually eneterd in next code chunk,
# therefore no need to rerun once "best" selected, as it slows down run, unless new data u
# Create the parameter grid based on the results of random search
#R: rfb <- randomForest(as.factor(Churn) ~., data=train, ntree= 500,
                   #mtry = 19, nodesize = 1,  importance = TRUE)
param_grid = {
    'bootstrap': [False,True],  #,True], #[True, False],
```

```
    'max_depth': [9],    #,5,6],    #10, 15],   #30
    'max_features': [10],    #,9,10],   #     5, 6, 7, 8],    #5
   'min_samples_leaf': [4],   #,4],    # 5, 6, 7, 8],   #4
    'min_samples_split': [10],   #,8],    #   5,6,7,8],   #2  3,4
    'n_estimators': [25]}  #  , 48, 50] }  #,55,60,65]   #60 #[50,100,150]
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 5, n_jobs = -1, verbose = 2)
# Fit the grid search to the data
grid_search.fit(X_train_clean,y_train)
```

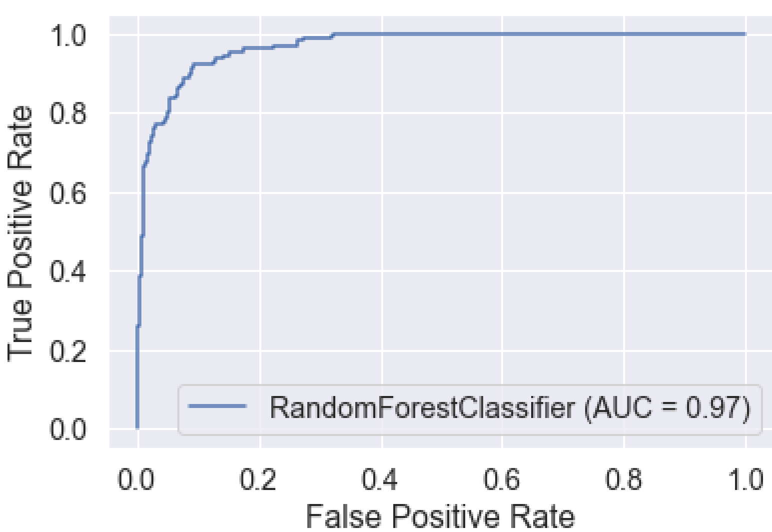
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    2.4s finished
```

Out[30]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [False, True], 'max_depth': [9, 10, 11],
                         'max_features': [10], 'min_samples_leaf': [4],
                         'min_samples_split': [10], 'n_estimators': [25]},
             verbose=2)
```

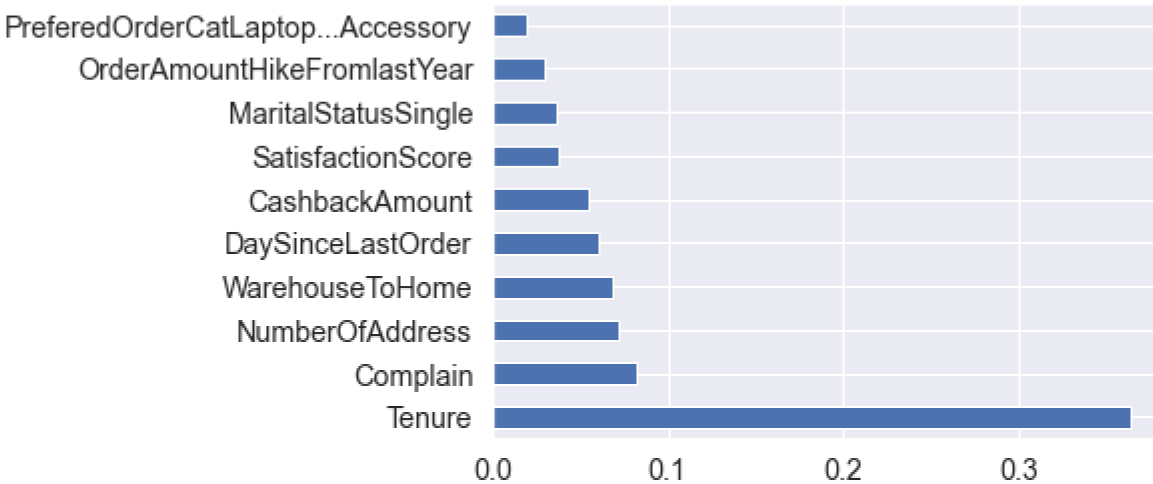
In [31]:
```
grid_search.best_params_
```

Out[31]:
```
{'bootstrap': False,
 'max_depth': 9,
 'max_features': 10,
 'min_samples_leaf': 4,
 'min_samples_split': 10,
 'n_estimators': 25}
```

In [32]:
```
# Creating the RF classifier
random.seed(1299)
#Use best features from above
clf = RandomForestClassifier(bootstrap=False,max_depth=9,max_features=10,min_samples_leaf=

#clf = RandomForestClassifier(bootstrap=False,max_depth=4,max_features=9,min_samples_leaf=
# Training the model on the training dataset;
clf.fit(X_train_clean, y_train)
# performing predictions on the test dataset
y_pred = clf.predict(X_test_clean)
#plot AUC
metrics.plot_roc_curve(clf, X_test_clean, y_test)
plt.show() #AUC .89, .90, .92, .94, .95, vif: .95 .95
```

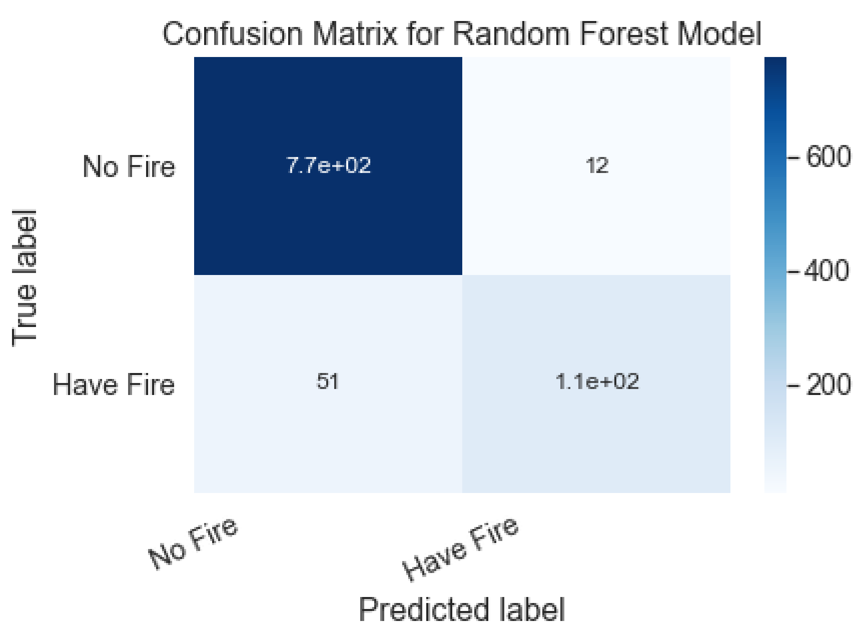


In [33]:
```
#Select top 10 predictors:
random.seed(1299)

feature_imp = pd.Series(clf.feature_importances_, index = feature_list).sort_values(ascend
feature_imp_sel = feature_imp.iloc[0:10] #33
feature_imp_sel_s = feature_imp_sel.sort_values(ascending = False)
feature_plot_1 = feature_imp_sel_s.plot.barh()
#
```



In [91]:
```
feature_imp
```

Out[91]:
```
Tenure                                 0.36
Complain                               0.08
NumberOfAddress                        0.07
WarehouseToHome                        0.07
DaySinceLastOrder                      0.06
CashbackAmount                         0.06
SatisfactionScore                      0.04
MaritalStatusSingle                    0.04
OrderAmountHikeFromlastYear            0.03
PreferedOrderCatLaptop...Accessory     0.02
OrderCount                             0.02
NumberOfDeviceRegistered               0.02
CityTier                               0.02
PreferedOrderCatMobile.Phone           0.01
CouponUsed                             0.01
PreferredLoginDeviceComputer           0.01
PreferredPaymentModeCredit.Card        0.01
MaritalStatusMarried                   0.01
GenderFemale                           0.01
GenderMale                             0.01
PreferredPaymentModeDebit.Card         0.01
PreferredLoginDeviceMobile.Phone       0.01
PreferedOrderCatFashion                0.01
PreferredPaymentModeE.wallet           0.01
PreferredPaymentModeCOD                0.01
HourSpendOnApp                         0.01
MaritalStatusDivorced                  0.00
PreferredLoginDevicePhone              0.00
PreferredPaymentModeUPI                0.00
PreferedOrderCatOthers                 0.00
PreferedOrderCatMobile                 0.00
PreferredPaymentModeCash.on.Delivery   0.00
PreferedOrderCatGrocery                0.00
PreferredPaymentModeCC                 0.00
dtype: float64
```

In [35]:
```
feature_imp.shape
```
Out[35]:
```
(34,)
```
In [36]:
```
feature_imp.to_csv('feature_imp_a.csv',index=True, na_rep='NA')   #A
```
In [37]:
```
#For viz in Tableau
feature_imp_sel_df = pd.DataFrame(feature_imp_sel)
#feature_imp_sel_df.to_csv('feature_imp_sel_sum.csv',index=True, na_rep='NA' )
random.seed(1299)

confusion_matrix(y_test, y_pred)
```
Out[37]:
```
array([[774,  12],
       [ 51, 107]])
```
In [38]:
```
# Get and reshape confusion matrix data
random.seed(1299)

matrix = confusion_matrix(y_test, y_pred)
# Build the plot
plt.figure(figsize=(6,4))
sns.set(font_scale=1.3)
sns.heatmap(matrix, annot=True, annot_kws={'size':12},
            cmap=plt.cm.Blues)
# Add labels to the plot
class_names = ['No Fire','Have Fire']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```


In [39]:
```
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test, y_pred))
#.828, .84, .835, .839, .847, .866, .88 w only non VIF .862, .87
```
```
ACCURACY OF THE MODEL:  0.9332627118644068
```

In [40]:

```
random.seed(1299)

print(classification_report(y_test, y_pred)) #model 2
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       786
           1       0.90      0.68      0.77       158

    accuracy                           0.93       944
   macro avg       0.92      0.83      0.87       944
weighted avg       0.93      0.93      0.93       944
```

In [41]:

```
#cohen_kappa_score  .69, .73. .68, 10: .58  .62 76 VIF: .72, .74
random.seed(1299)

from sklearn.metrics import cohen_kappa_score
cohen_kappa_score(y_test, y_pred)
```

Out[41]:

```
0.7343624376909471
```

Churn Prediction Tables

In [43]:

```
#Table 1:  For client: w/ address & goecoordinates
predprob =clf.predict_proba(result_3.drop(['Churn', 'CustomerID'], axis = 1))
pd.DataFrame(predprob).shape
result_data = pd.concat([result_3[['CustomerID']].reset_index(drop=True),
                         pd.DataFrame(predprob[:,1],columns=['Predicted Churn Probability'
                         axis=1)
# function to classify risk based on probability
def churn_risk(prob):
  return np.where(prob<1/3,'Low Risk',
            np.where(prob<2/3, 'Moderate Risk',
              np.where(prob<=1, 'High Risk', np.nan)))
```

In [42]:

```
result_3.head()
```

Out[42]:

	Churn	CustomerID	Tenure	CityTier	WarehouseToHome	HourSpendOnApp	NumberOfDeviceRegistered	Satis
0	1	50001	4	3	6	3	3	
1	1	50004	0	3	15	2	4	
2	1	50006	0	1	22	3	5	
3	1	50012	11	1	6	3	4	
4	1	50013	0	1	11	2	3	

In [95]:

```
#W/ Google name
#Cols for chart   result_r2B.columns))  #AptName_ys
data = {'CustomerID' :result_3.CustomerID,
        'Tenure': result_3.Tenure,
        'DaySinceLastOrder':result_3.DaySinceLastOrder,
        'CashbackAmount':result_3.CashbackAmount,
        'DaySinceLastOrder':result_3.DaySinceLastOrder, #result_r2
        'Predicted Churn Risk': churn_risk(predprob[:,1]),
```

```
        'Predicted Churn Probability':predprob[:,1],
        'CurrentDate':date.isoformat(date.today())
        }
pred_addr_g=pd.DataFrame(data=data)
#pred_addr_g['District'] = pred_addr_g['District'].fillna(' ')
pred_addr_g_sort = pred_addr_g.sort_values(by='Predicted Churn Probability',ascending=Fals

#pred_addr_g_se = pred_addr_g.loc[pred_addr_g['Predicted Fire Risk'] == 'Low Risk']
pred_addr_g_sort.head()
#for internal use
```

Out[95]:

	CustomerID	Tenure	DaySinceLastOrder	CashbackAmount	Predicted Churn Risk	Predicted Churn Probability	CurrentDate
0	53748	1	1	182.65000	High Risk	0.99214	2023-05-15
1	53402	1	1	149.04000	High Risk	0.99111	2023-05-15
2	54872	1	1	149.04000	High Risk	0.99111	2023-05-15
3	54175	1	4	145.28000	High Risk	0.99083	2023-05-15
4	52818	1	4	145.28000	High Risk	0.99083	2023-05-15

In [75]:

```
#pred_addr_g.to_csv('fire_prediction_table_google_name.csv' ,index=True, na_rep='NA')
pred_addr_g.shape
```

Out[75]:

```
(3774, 7)
```

In [76]:

```
pred_addr_g.CustomerID.nunique()
```

Out[76]:

```
3774
```

In [77]:

```
pred_addr_g['Predicted Churn Risk'].value_counts()
```

Out[77]:

```
Low Risk          3127
High Risk          400
Moderate Risk      247
Name: Predicted Churn Risk, dtype: int64
```

In [78]:

```
# For viz, internal use
#pred_addr.to_csv('fire_prediction_table_1.csv' ,index=True, na_rep='NA')
#pred_addr_2.to_csv('fire_prediction_table_2.csv',index=True, na_rep='NA')
###  pred_addr.to_csv('fire_prediction_table_1_client.csv' ,index=True, na_rep='NA')
#pred_addr.to_csv('fire_prediction_table_1_client_v2.csv' ,index=True, na_rep='NA')
```

Part 3: Prepare Findings for Visualization in Tableau, Power BI or RShiny

In [85]:

```
#discription <- read_excel("E Commerce Dataset.xlsx", sheet = "Data Dict")
#df <- read_excel("E Commerce Dataset.xlsx", sheet = "E Comm")
# data from excel
#import xlwings as xw
ws = pd.read_excel("E Commerce Dataset.xlsx", sheet_name="E Comm")  #, *, header=0, names=

ws.head()
#PreferredLoginDevice, PreferredPaymentMode  Gender  PreferedOrderCat  MaritalStatus
```

Out[85]:

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gend
0	50001	1	4.00	Mobile Phone	3	6.00	Debit Card	Fema
1	50002	1	NaN	Phone	1	8.00	UPI	Ma
2	50003	1	NaN	Phone	1	30.00	Debit Card	Ma
3	50004	1	0.00	Phone	3	15.00	Debit Card	Ma
4	50005	1	0.00	Phone	1	12.00	CC	Ma

In [86]:

```
ws.shape
```

Out[86]:

```
(5630, 20)
```

In [87]:

```
ws_nonan = ws. dropna()
ws_nonan.shape
```

Out[87]:

```
(3774, 20)
```

In [88]:

```
#Select cols & join
pred_addr_g_sel = pred_addr_g.loc[:, ('CustomerID','Predicted Churn Risk', 'Predicted Chu

round_1 = pd.merge(pred_addr_g_sel, result_3,left_on = ['CustomerID'],right_on = ['Custome

#sel values from ws_nonan #PreferredLoginDevice, PreferredPaymentMode  Gender  PreferedOr
ws_nonan_sel = ws_nonan.loc[:, ('CustomerID','PreferredLoginDevice', 'PreferredPaymentMode

churn_viz_raw = pd.merge(round_1, ws_nonan_sel,left_on = ['CustomerID'],right_on = ['Custo
churn_viz_ra = churn_viz_raw.drop_duplicates()

churn_viz = churn_viz_ra.sort_index(axis=1)#sorts alphabeticaly by column name
churn_viz.head()
```

Out[88]:

	CashbackAmount	Churn	CityTier	Complain	CouponUsed	CustomerID	DaySinceLastOrder	Gender	GenderF
0	159.93	1	3	1	1	50001	5	Female	
1	134.07	1	3	0	0	50004	3	Male	
2	139.19	1	1	1	4	50006	7	Female	
3	153.81	1	1	1	0	50012	0	Male	
4	134.41	1	1	1	2	50013	2	Male	

In [89]:

```
churn_viz.shape    #3774, 38
```

Out[89]:

```
(3774, 43)
```

In [90]:

```
churn_viz.to_csv('churn_viz.csv',index=True, na_rep='NA')
```

Pearson's Correlation Matrix

In [6]:

```
cols_to_move = ['Churn']
result_3 = result_3[ cols_to_move + [ col for col in result_3.columns if col not in cols_t
```

In [52]:

```
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline
```

In [97]:

```
def top_entries(df):
    mat = df.corr(method = 'pearson').abs()
    # Remove duplicate and identity entries
    mat.loc[:,:] = np.tril(mat.values, k=-1)
    mat = mat[mat>0]
    # Unstack, sort ascending, and reset the index, so features are in columns
    # instead of indexes (allowing e.g. a pretty print in Jupyter).
    # Also rename these it for good measure.
    return (mat.unstack()
             .sort_values(ascending=False)
             .reset_index()
             .rename(columns={
                 "level_0": "feature_a",
                 "level_1": "feature_b",
                 0: "correlation"
             }))
#TRY SPEARMAN'S W ALL
top_spear_all = top_entries(result_3)  #result_3_SEL2?
top_spear_all.head(12)
```

Out[97]:

	feature_a	feature_b	correlation
0	GenderFemale	GenderMale	1.00000
1	MaritalStatusMarried	MaritalStatusSingle	0.73789
2	CouponUsed	OrderCount	0.73247
3	PreferedOrderCatLaptop...Accessory	PreferedOrderCatMobile.Phone	0.72185
4	PreferredLoginDeviceComputer	PreferredLoginDeviceMobile.Phone	0.66291
5	CustomerID	HourSpendOnApp	0.59425
6	PreferredPaymentModeCredit.Card	PreferredPaymentModeDebit.Card	0.54014
7	PreferredPaymentModeCC	PreferedOrderCatMobile	0.50456
8	CityTier	PreferredPaymentModeE.wallet	0.50317
9	PreferredLoginDeviceMobile.Phone	PreferredLoginDevicePhone	0.50132
10	OrderCount	DaySinceLastOrder	0.45734
11	CashbackAmount	PreferedOrderCatFashion	0.45547

In [98]:

```
def filter_rows_by_values(df, col, values):
    return df[~df[col].isin(values)]
top_spear_all_1=top_spear_all.loc[top_spear_all['feature_a'] == 'Churn']
top_spear_all_1.reset_index(drop=True, inplace=True)
top_spear_all_1.head(12)
```

Out[98]:

	feature_a	feature_b	correlation
0	Churn	Tenure	0.34001
1	Churn	Complain	0.23814
2	Churn	PreferedOrderCatLaptop...Accessory	0.18458
3	Churn	PreferedOrderCatMobile.Phone	0.18168
4	Churn	MaritalStatusSingle	0.17948
5	Churn	MaritalStatusMarried	0.15981
6	Churn	NumberOfDeviceRegistered	0.14904
7	Churn	DaySinceLastOrder	0.13925
8	Churn	SatisfactionScore	0.09576
9	Churn	WarehouseToHome	0.08732
10	Churn	PreferredLoginDeviceMobile.Phone	0.07912
11	Churn	NumberOfAddress	0.07634

In [99]:

```
#Check influencers on Tenure
top_spear_all_1=top_spear_all.loc[top_spear_all['feature_a'] == 'Tenure']
top_spear_all_1.reset_index(drop=True, inplace=True)

top_spear_all_1.head(12)
```

Out[99]:

	feature_a	feature_b	correlation
0	Tenure	CashbackAmount	0.21385
1	Tenure	PreferedOrderCatMobile.Phone	0.21148
2	Tenure	NumberOfAddress	0.19655
3	Tenure	PreferedOrderCatFashion	0.13026
4	Tenure	DaySinceLastOrder	0.11893
5	Tenure	OrderCount	0.11234
6	Tenure	PreferedOrderCatLaptop...Accessory	0.10448
7	Tenure	PreferedOrderCatOthers	0.10370
8	Tenure	MaritalStatusSingle	0.10315
9	Tenure	PreferredLoginDevicePhone	0.09677
10	Tenure	PreferedOrderCatGrocery	0.09125
11	Tenure	MaritalStatusMarried	0.08599