**SOURCE CODE**

Main Analysis:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

#Import the required libraries

import pandas as pd

import numpy as np

get\_ipython().run\_line\_magic('pylab', 'inline')

import seaborn as sns

from sklearn import tree

from sklearn import metrics

# In[2]:

# import the file

df = pd.read\_csv('Churn\_Modelling.csv',error\_bad\_lines=False)

#print len(df) #-- Check the number of rows

#df.columns #-- Check the columns

#Exited : flag indicating customer's churn status.

#0 : Existing customer

#1 : Churned customer

# In[3]:

#Check the customer split at Exited flag level

df['Exited'].value\_counts()

# In[4]:

#Convert categorical variables to string:

df['CustomerId']=df['CustomerId'].astype(str)

df['HasCrCard']=df['HasCrCard'].astype(str)

df['IsActiveMember']=df['IsActiveMember'].astype(str)

# Check the quick summmary of the table

df.describe()

# ##### Finding:

# Level of the table = customerId : checked #rows = #rows when grouped by customerId

# In[5]:

#Check the Age distribution at customer status split

df.boxplot(by='Exited',column=['Age'],grid=False,notch=True, # notch shape

vert=True, # vertical box alignment

patch\_artist=True)

# ##### Findings:

# - Older customers are more likely to churn

# - Credit Score does not seem to be a very distinguishing factors based on the distribution

# In[6]:

#DIstribution at Gender and Churn Flag (Exited) level

df5=df.groupby(by=['Gender','Exited'])['RowNumber'].agg('count')

df4=df5.reset\_index()

df\_churn=df4[df4['Exited']==1][['Gender','RowNumber']]

df\_ret=df4[df4['Exited']==0][['Gender','RowNumber']]

plt.subplot(1, 2, 1)

plt.pie(df\_churn['RowNumber'],labels=df\_churn['Gender'],autopct='%1.1f%%')

plt

plt.subplot(1, 2, 2)

plt.pie(df\_ret['RowNumber'],labels=df\_ret['Gender'],autopct='%1.1f%%')

# ##### Finding:

# - Women are more likely to churn than men

# In[7]:

# Age distribution across countries and churn flag

sns.violinplot(x="Geography", y="Age",hue='Exited', data=df, palette="Pastel1")

# ##### Finding:

# - German and older (by age) customers are more likely to churn

# In[8]:

## number of products vs balance :

prod\_plt=df.groupby(['NumOfProducts','Exited'])[['Balance']].agg('mean')

prod\_plt=prod\_plt.reset\_index()

prod\_plt.pivot("NumOfProducts","Exited","Balance").plot(kind='bar')

# In[9]:

actv\_plt=pd.DataFrame(df.groupby(['IsActiveMember','Exited'])['CustomerId'].count())

actv\_plt=actv\_plt.reset\_index()

actv\_plt.pivot("IsActiveMember","Exited","CustomerId").plot(kind='bar')

# ##### Finding:

# - 37% of non-active members churn vs 17% of active members churn

# In[10]:

card\_plt=pd.DataFrame(df.groupby(['HasCrCard','Exited'])['CustomerId'].count())

card\_plt

# In[11]:

# Check the distribution of various variables across churned and existing customers

plt.subplot(1, 3, 1)

#Check the Estimated Salary distribution at customer status split

sns.violinplot(x="Exited", y="EstimatedSalary", data=df, palette="Pastel1")

plt.subplot(1, 3, 3)

#Check the Tenure distribution at customer status split

sns.violinplot(x="Exited", y="Tenure", data=df, palette="Pastel1")

# ##### Finding:

# - 26% of customers without credit card churn vs 25.2% customers with credit card churn - => might not be a deciding fator

# In[12]:

# Correlation Matrix Heat Map

df2=df[['CreditScore','Tenure','Balance','NumOfProducts','EstimatedSalary']]

corr = df2.corr()

ax = sns.heatmap(

corr,

vmin=-1, vmax=1, center=0,

cmap=sns.diverging\_palette(20, 220, n=200),

square=True

)

ax.set\_xticklabels(

ax.get\_xticklabels(),

rotation=45,

horizontalalignment='right'

);

# In[14]:

##import needed modules

from pandas import Series, DataFrame

import pandas as pd

from patsy import dmatrices

import matplotlib.pyplot as plt

import numpy as np

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report

get\_ipython().run\_line\_magic('pylab', 'inline')

import copy

# import the file

df = pd.read\_csv(r'Churn\_Modelling.csv',error\_bad\_lines=False)

# Drop unnecessary columns

df2= df.drop(['RowNumber', 'CustomerId', 'Surname'], axis = 1)

#create design matrices

Y, X = dmatrices('Exited ~ 0 + CreditScore + Geography + Gender + Age + Tenure + Balance + NumOfProducts + HasCrCard + IsActiveMember + EstimatedSalary', df2, return\_type="dataframe")

# In[15]:

#Varying model paramters

from sklearn import model\_selection

kfold = model\_selection.StratifiedKFold(n\_splits=3).split(X=X[:6], y=[0,0,0,1,1,1])

for train, holdout in kfold:

print('train indices =', train, 'holdout indices =', holdout)

# In[17]:

# Import the required libraries

import pandas as pd

import numpy as np

import patsy as pt

get\_ipython().run\_line\_magic('pylab', 'inline')

from sklearn.linear\_model import LogisticRegression

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import itertools

from scipy import stats

from sklearn import metrics

#Import the data :

df = pd.read\_csv('Churn\_Modelling.csv',error\_bad\_lines=False)

# Data Processing

formula = 'Exited ~ 0 + EstimatedSalary+CreditScore+ Age+HasCrCard+IsActiveMember + Tenure+ Balance + NumOfProducts + C(Geography) + C(Gender)'

Y, X = pt.dmatrices(formula, df, return\_type='dataframe')

y = Y['Exited'].values

#Split the data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

# In[18]:

#split up data into training and testing set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=1)

#set up the classifier

#from sklearn import neighabors

from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n\_neighbors=30,

weights='uniform',

p=2)

#fit the classifier

model.fit(X\_train, y\_train)

#run on training data

from sklearn import metrics

prediction\_on\_training = model.predict(X\_train)

print ("Model accuracy of training dataset is: ",metrics.accuracy\_score(y\_train, prediction\_on\_training))

# Predict accuracy on test dataset

predicted\_classes = model.predict(X\_test)

print ("Model accuracy of test dataset is: ",metrics.accuracy\_score(y\_test, predicted\_classes))

# In[19]:

#Varying model paramters

from sklearn import model\_selection

kfold = model\_selection.StratifiedKFold(n\_splits=3).split(X=X[:6], y=[0,0,0,1,1,1])

for train, holdout in kfold:

print('train indices =', train, 'holdout indices =', holdout)

# In[20]:

# Train the logistic regression model

model = LogisticRegression()

result = model.fit(X\_train, y\_train)

model.fit(X\_train, y\_train)

# Predict on training dataset

prediction\_train = model.predict(X\_train)

print ("Model accuracy on test dataset: ",metrics.accuracy\_score(y\_train, prediction\_train))

#Predict on test dataset

prediction = model.predict(X\_test)

print ("Model accuracy on test dataset: ",metrics.accuracy\_score(y\_test, prediction))

conf\_mat=confusion\_matrix(y\_test, prediction, labels=None, sample\_weight=None)

print ("Confustion Matrix: ")

print ("conf\_mat")

# In[21]:

# Coefficients :

weights = pd.Series(model.coef\_[0],

index=X.columns.values)

weights.sort\_values()

# In[22]:

# Build the model

model2 = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=2)

result = model2.fit(X\_train, y\_train)

# Predict on training dataset

prediction\_train = model2.predict(X\_train)

print ("Model accuracy on training dataset: ",metrics.accuracy\_score(y\_train, prediction\_train))

# Predict on Tese dataset

prediction = model2.predict(X\_test)

print ("Model accuracy on test dataset: ",metrics.accuracy\_score(y\_test, prediction))

# In[23]:

import pandas as pd

import numpy as np

import patsy as pt

get\_ipython().run\_line\_magic('pylab', 'inline')

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import itertools

from patsy import dmatrices

from subprocess import check\_call

#Import the data :

df = pd.read\_csv('Churn\_Modelling.csv',error\_bad\_lines=False)

# One Hot Encoding

df1 = pd.get\_dummies(df.iloc[:,3:14])

# Extract features and labels

labels=df1['Exited']

# Training and Testing Sets

df1 = df1.drop('Exited', axis = 1)

train, test, train\_labels, test\_labels = train\_test\_split(df1,labels, test\_size = 0.3, random\_state = 11)

#create test data set :

test\_df=test.merge(test\_labels, right\_index=True,left\_index=True)

#test\_df[:10]

# In[24]:

# Build random forest model:

rf\_exp = RandomForestClassifier(n\_estimators= 1000,bootstrap = True, random\_state=100)

rf\_exp.fit(train, train\_labels)

# Make predictions on test data

predictions = rf\_exp.predict(test)

train\_rf\_probs = rf\_exp.predict\_proba(test)[:, 1]

# In[25]:

## Confusion Matrix

cm = confusion\_matrix(test\_labels, predictions)

# Calculate the model accuracy

accuracy = (cm[0,0]+cm[1,1])\*1.0/(cm[0,0]+cm[0,1]+cm[1,0]+cm[1,1])

print ("The model accuracy is :", round((accuracy\*100),2), " %")

print ("Confustion Matrix: ")

print (cm)

# In[26]:

# Metrics importance:

importances = list(rf\_exp.feature\_importances\_)

col\_list = list(df1.columns)

feat\_importances = pd.Series(rf\_exp.feature\_importances\_, index=train.columns)

feat\_importances.sort\_values(ascending=True).plot(kind='barh')

# In[27]:

# Get numerical feature importances

importances = list(rf\_exp.feature\_importances\_)

# List of tuples with variable and importance

feature\_importances = [(df1, round(importance, 2)) for df1, importance in zip(col\_list, importances)]

feature\_importances

# In[ ]:

# In[ ]: