#!/usr/bin/env python

# coding: utf-8

# # IMPORT LIBRARY

# In[1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

from fancyimpute import KNN

from random import randrange, uniform

from scipy.stats import chi2\_contingency

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

# In[2]:

#set directory according to your location

os.chdir("C:/Users/hp/Desktop/abc satender customer")

# In[3]:

#import data as trans short form of transaction

trans = pd.read\_csv("train.csv")

# # EXPLORATORY DATA ANALYSIS

# ### CHECKOUT THE DATA

# HERE TARGET VALUE 1 MEANS CUSTOMER WILL BUY THE PRODUCT AND 0 MEANS HE WILL NOT BUY THE PRODUCT AND THE TARGET VALUE DEPENDS ON 200 INDEPENDENT VARIABLE

# In[4]:

trans.head()

# In[5]:

trans.shape

# ### THIS SHAPE FUNCTION SHOWS THAT DATA IS VERY LARGE AS IT CONTAIN 2 LAKH OBSERVATION AND 202 VARIABLES.

#

# ### SO IF POSSIBLE WE SHOULD REDUCE THE DATASET VARIABLE INTO NECESSARY VARIABLE

# ### WE USED CORRELATION HEATMAP AND PRINCIPAL COMPONENT ANALYSIS TO DO SO

# In[6]:

trans.info()

# In[7]:

trans.describe()

# ## FINDING OUT THE MISSING VALUES IN TRAIN DATASET

# In[8]:

trans.isnull().sum().head(5)

# In[9]:

#sns.heatmap(trans.isnull(),yticklabels=False,cbar=False,cmap='viridis')

# In[9]:

trans[trans.isna().any(axis=1)]

# In[10]:

#Create dataframe with missing percentage

missing\_val = pd.DataFrame(trans.isnull().sum())

#Reset index

missing\_val = missing\_val.reset\_index()

#Rename variable

missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})

#Calculate percentage

missing\_val['Missing\_percentage'] = (missing\_val['Missing\_percentage']/len(trans))\*100

#descending order

missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).reset\_index(drop = True)

#save output results

missing\_val.to\_csv("Missing\_perc.csv", index = False)

# In[11]:

missing\_val.head()

# ### UNIVARIATE ANALYSIS

# In[12]:

sns.set\_style('whitegrid')

sns.countplot(x='target',data=trans,palette='RdBu\_r')

# In[13]:

trans['var\_0'].hist(bins=30,color='darkred',alpha=0.7)

# In[14]:

trans['var\_2'].hist(bins=30,color='darkred',alpha=0.7)

# ### BIVARIATE ANALYSIS

# In[15]:

#HERE WE HAVE DRAWN PAIRPLOT OF INITIAL TEN VARIABLES TO GIVE AN ANLYSIS HOW THE VARIABLES ARE

#DEPENDING WITH EACH OTHER

sns.pairplot(trans.iloc[:,0:10],hue='target',palette='Set1')

# ### MULTIVARIATE ANALYSIS

# ### FINDING OUT CORRELATION HEAT MAP SO THAT EXTRA VARIABLE CAN BE REMOVED OUT AND GET DATASET CONTAIN LESSER VARIABLE WHICH ARE EASY TO HANDEL

# In[16]:

df\_corr = trans.iloc[:,2:202]

# In[17]:

#Set the width and hieght of the plot

f, ax = plt.subplots(figsize=(20, 10))

#Generate correlation matrix

corr = df\_corr.corr()

#Plot using seaborn library

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

square=True, ax=ax)

# #### WHOLE HEATMAP IS SHOWING BLUE COLOR EXCEPT FOR DIAGONAL SO HERE HEAT MAP IS SHOWING THAT NO TWO INDEPENDENT VARIABLES ARE DEPENDING WITH EACH OTHER SO WE CAN NOT REMOVE ANY VARIABLE

# ## USING PCA FOR REMOVING AN EXTRA VARIABLE

# In[18]:

#IMPORT LIBRARY TO STANDARDIZE THE DATA

from sklearn.preprocessing import StandardScaler

# In[19]:

scaler = StandardScaler()

scaler.fit(trans.iloc[:,2:202])

# In[20]:

scaled\_data = scaler.transform(trans.iloc[:,2:202])

# In[21]:

#IMPORTING THE PCA LIBRARY

from sklearn.decomposition import PCA

pca = PCA(n\_components=100)

# #### HERE USING DIFFERENT n\_component WE CAN SEE THE VARIATION COVERED BY THE N VARIABLE IT SHOULD COMES OUT ATLEAST GREATER THAN 90% PERCENT HERE WE HAVE TAKEN n VALUE EQUAL TO 100 TO JUST CHECK THE TOTAL VARIATION THEY HAVE COVERED

# In[22]:

pca.fit(scaled\_data)

my\_model = pca.transform(scaled\_data)

# In[23]:

#HERE WE ARE USING FUCTION TO GET VARIATION COVERED BY TOP 100 VARIABLES

print(pca.explained\_variance\_)

#HERE WE ARE USING FUCTION TO GET VARIATION COVERED BY TOP 100 VARIABLES IN FORM OF RATIO

print(pca.explained\_variance\_ratio\_)

#HERE WE ARE USING FUCTION TO GET VARIATION COVERED BY TOP 100 VARIABLES IN FORM OF SUM OF RATIO

print(pca.explained\_variance\_ratio\_.cumsum())

# #### SO WE CAN SEE THAT ONLY 50% OF VARIATIONS IS COVERED BY 100 VARIABLE WHICH IS ALSO A 50 % OF TOTAL VARIABLE.SO IT SHOWS WE SHOULD USE ALL THE VARIABLES TO FORM THE MODEL AS THEY ALL ARE CONTRIBUTING EQUALLY TO FIND TARGET VARIABLE

# # DATA PREPROCESSING

# #### FOUR METHOD USED IN PREPROCESSING ARE. 1) MISSING VALUE ANALYSIS 2) OUTLIER ANALYSIS 3) DATA SELECTION 4) DATA SCALING

# #### DATA DOES NOT HAVE ANY MISSING VALUE SO WE ARE NOT USE MISSING VALUE IMPUTATION

# ## OUTLIER ANALYSIS

# #### BY USING TWO WAYS OUTLIER CAN BE HANDLE.

# #### BY REMOVING THE OUTLIER VALUES.

# #### OR BY PUTTING OUTLIER AS NULL VALUE AND THE IMPUTE USING KNN IMPUTATION.

# #### SO WE ARE USING FIRST METHOD AS IS GIVES LESSER ERROR

# In[24]:

# #Detect and delete outliers from data

for i in range(2,202):

#print(i)

q75, q25 = np.percentile(trans.iloc[:,i], [75 ,25])

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

#print(min)

#print(max)

trans = trans.drop(trans[trans.iloc[:,i] < min].index)

trans = trans.drop(trans[trans.iloc[:,i] > max].index)

# #Replace with NA

#trans.iloc[:,i] = np.where(trans.iloc[:,i] >max, np.NaN,trans.iloc[:,i])

#trans.iloc[:,i] = np.where(trans.iloc[:,i] <min, np.NaN,trans.iloc[:,i])

# #Calculate missing value

#missing\_val = pd.DataFrame(trans.isnull().sum())

#Reset index

#missing\_val = missing\_val.reset\_index()

#Rename variable

#missing\_val = missing\_val.rename(columns = {'index': 'Variables', 0: 'Missing\_percentage'})

#Calculate percentage

#missing\_val['Missing\_percentage'] = (missing\_val['Missing\_percentage']/len(fare))\*100

#descending order

#missing\_val = missing\_val.sort\_values('Missing\_percentage', ascending = False).reset\_index(drop = True)

#Impute with KNN

#fare= pd.DataFrame(KNN(k = 3).complete(fare), columns = fare.columns)

# In[ ]:

missing\_val.to\_csv("Outliers percentages in each column.csv")

# In[25]:

trans.to\_csv("outlier values.csv")

# In[26]:

trans = pd.read\_csv("outlier values.csv")

# #### AFTER REMOVING OUTLIER WE GET 175073 OBSERVATION FROM 200000 OBSERVATIONS SO 24927 OBSERVATION ARE EXIST AS AN OUTLIER IN TRANS DATASET

# In[27]:

trans.shape

# In[28]:

# #Plot boxplot to visualize Outliers

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

plt.boxplot(trans['var\_0'] ,vert=True,patch\_artist=True)

# In[31]:

trans = trans.drop(trans.columns[0], axis = 1)

# In[32]:

trans.head()

# ## FEATURE SELECTION

# #### IN DATA EXPLORATORY ANALYSIS WE HAVE SHOWN EARLIER THAT NO VARIABLE IS DEPENDENT WITH EACH OTHER CORRELATION VALUE IS COMES OUT TO BE 0

# #### WE HAVE FETCH OUT INITIAL 10 VARIABLES JUST FOR EXAMPLE TO SHOW BIGGER HEATMAP AND SHOW THAT NO TWO VARIABLE ARE DEPEND ON EACH OTHER

# In[33]:

trans\_corr = trans.iloc[:,2:10]

# In[34]:

#Set the width and hieght of the plot

f, ax = plt.subplots(figsize=(20, 10))

#Generate correlation matrix

corr = trans\_corr.corr()

#Plot using seaborn library

sns.heatmap(corr, mask=np.zeros\_like(corr, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

square=True, ax=ax)

# #### DATA SCALING WILL BE DONE ACCORDING TO MACHINE LEARNING ALGORITHM FORM. IF THE ALGORITHM ASSOCIATED WITH DISTANCE THAN WE DO SCALING OF NUMERIC VARIBALE FIRST

# # MODEL FORMATION

# ## Building a Logistic Regression model

# ### TRAIN TEST SPLIT

# Let's start by splitting our data into a training set and test set

# In[35]:

#THIS VARIABLE IS OF STRING DATA TYPE AND IT HAS NO USE IN PREDICTING TARGET VALUE

trans = trans.drop("ID\_code" , axis=1)

# In[36]:

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

# In[37]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(trans.drop('target',axis=1),

trans['target'], test\_size=0.30,

random\_state=101)

# ### TRAINING AND PREDICTING

# In[44]:

from sklearn.linear\_model import LogisticRegression

# In[45]:

logmodel = LogisticRegression()

logmodel.fit(x\_train,y\_train)

# In[47]:

predictions = logmodel.predict(x\_test)

# ### EVALUATION

# HERE WE TAKE THE PRECISION AND RECALL VALUE FOR TARGET VALUE 1 WHICH MEANS HE BUY THE PRODUCT AS IT IS A POSITIVE IN NATURE

# In[41]:

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

# In[49]:

# precision and recall for both the values are given

print(classification\_report(y\_test,predictions))

# In[57]:

#build confusion matrix

from sklearn.metrics import confusion\_matrix

CM = confusion\_matrix(y\_test, predictions)

CM = pd.crosstab(y\_test, predictions)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#recall(y\_test, y\_pred)\*100

print("recall", "=", (TP\*100)/(TP+FN))

#precision

print("precision", "=" ,(TP\*100)/(TP+FP))

#Results

#recall = 26.34356068008599

#precision = 69.05737704918033

#AUC =62.53%

#accuracy =92%

# In[52]:

#CONFUSION MATRIX

CM

# In[27]:

from sklearn import metrics

# In[59]:

metrics.roc\_auc\_score(y\_test, predictions)

# #### SUMMARY:Results

# #### recall = 26.34356068008599

# #### precision = 69.05737704918033

# ##### AUC =62.53%

# #### accuracy =92%

# ## DECISION TREES

# In[15]:

from sklearn.tree import DecisionTreeClassifier

# In[16]:

dtree = DecisionTreeClassifier(criterion="entropy")

# In[17]:

dtree.fit(x\_train,y\_train)

# ### PREDICTION AND EVALUATION

# In[21]:

predictions\_dt = dtree.predict(x\_test)

# In[22]:

print(classification\_report(y\_test,predictions\_dt))

# In[23]:

#build confusion matrix

from sklearn.metrics import confusion\_matrix

CM = confusion\_matrix(y\_test, predictions\_dt)

CM = pd.crosstab(y\_test, predictions\_dt)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#recall(y\_test, y\_pred)\*100

print("recall", "=", (TP\*100)/(TP+FN))

#precision

print("precision", "=" ,(TP\*100)/(TP+FP))

# In[74]:

#CONFUSION MATRIX

CM

# In[70]:

metrics.roc\_auc\_score(y\_test, predictions\_dt)

# #### SUMMARY:Results

# #### recall = 19.53488372093023%

# #### precision = 19.174955160606554%

# ##### AUC =54.89%

# #### accuracy =84%

# ## RANDOM FOREST CLASSIFIER

# In[75]:

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=100)

rfc.fit(x\_train, y\_train)

# In[76]:

predictions\_rfc = rfc.predict(x\_test)

# In[77]:

print(classification\_report(y\_test,predictions\_rfc))

# In[78]:

#build confusion matrix

CM = confusion\_matrix(y\_test, predictions\_rfc)

CM = pd.crosstab(y\_test, predictions\_rfc)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#recall(y\_test, y\_pred)\*100

print("recall", "=", (TP\*100)/(TP+FN))

#precision

print("precision", "=" ,(TP\*100)/(TP+FP))

# In[79]:

#CONFUSION MATRIX

CM

# In simple terms, high precision means that an algorithm returned substantially more relevant results than irrelevant ones, while high recall means that an algorithm returned most of the relevant results.

# In[80]:

metrics.roc\_auc\_score(y\_test, predictions\_rfc)

# #### SUMMARY

# #### n\_estimators=100

# #### recall = 0.019542700801250732

# #### precision = 100.0

# #### AUC =50%

# #### accuracy =90%

# ## NAIVE BAYES

# In[38]:

#Naive Bayes

from sklearn.naive\_bayes import GaussianNB

#Naive Bayes implementation

NB\_model = GaussianNB().fit(x\_train, y\_train)

# ### PREDICTION AND EVALUATION

# In[39]:

#predict test cases

predictions\_nb = NB\_model.predict(x\_test)

# In[42]:

print(classification\_report(y\_test,predictions\_nb))

# In[43]:

#build confusion matrix

CM = confusion\_matrix(y\_test, predictions\_nb)

CM = pd.crosstab(y\_test, predictions\_nb)

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

#check accuracy of model

#recall(y\_test, y\_pred)\*100

print("recall", "=", (TP\*100)/(TP+FN))

#precision

print("precision", "=" ,(TP\*100)/(TP+FP))

#Results

#Accuracy: 84.49

#FNR: 63

# In[25]:

#confusion matrix

CM

# In[29]:

metrics.roc\_auc\_score(y\_test, predictions\_nb)

# #### SUMMARY:Results

# #### recall = 35.86085597029509%

# #### precision = 72.04554377699255%

# #### AUC =67.08%

# #### accuracy =92%

# ## K-NEAREST NEIGHBOUR

# In[44]:

#SCALING IS REQUIRED AS KNN USED DISTANCE BETWEEN OBSERVATION TO GET THE TARGET VALUES

from sklearn.preprocessing import StandardScaler

# In[45]:

scaler = StandardScaler()

# In[46]:

scaler.fit(trans.drop('target',axis=1))

# In[47]:

scaled\_features = scaler.transform(trans.drop('target',axis=1))

# In[48]:

df\_feat = pd.DataFrame(scaled\_features,columns=trans.columns[:-1])

df\_feat.head()

# ### TRAIN AND SPLIT THE DATA

# In[49]:

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

# In[50]:

x\_train, x\_test, y\_train, y\_test = train\_test\_split(scaled\_features,trans['target'],

test\_size=0.30)

# In[51]:

from sklearn.neighbors import KNeighborsClassifier

# In[52]:

knn = KNeighborsClassifier(n\_neighbors=1)

# In[53]:

knn.fit(x\_train,y\_train)

# #### HERE WE CAN NOT PREDICT THE DATA USING KNN AS KNN IS NOT ABLE TO PREDICT THIS DATASET BECAUSE DATASET CONTAIN VERY LARGE NUMBER OF VARIABLES AS IT USES DISTANCE SO IT TAKES INFINITE TIME TO PREDICT THIS DATASET

#

predictions\_knn = knn.predict(x\_test)print(classification\_report(y\_test,predictions\_knn))error\_rate = []

# Will take some time

for i in range(1,40):

knn = KNeighborsClassifier(n\_neighbors=i)

knn.fit(x\_train,y\_train)

pred\_i = knn.predict(x\_test)

error\_rate.append(np.mean(pred\_i != y\_test))

plt.figure(figsize=(10,6))

plt.plot(range(1,40),error\_rate,color='blue', linestyle='dashed', marker='o',

markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

# # SUMMARY

PRECISION CRITERION AS FOLLOW:

RF> NAÏVE BAYES>LOGISTIC REGRESSION>DECISION TREE

RECALL CRITERION AS FOLLOW:

NAÏVE BAYES>LOGISTIC REGRESSION>DECISION TREE>RF

AUC CRITERION AS FOLLOW:

NAÏVE BAYES>LOGISTIC REGRESSION> DECISION TREE>RF

ERRORS

LOGISTIC REGRESSION:Results

recall = 26.34356068008599

precision = 69.05737704918033

AUC =62.53%

accuracy =92%

DECISION TREES:Results

recall = 19.53488372093023%

precision = 19.174955160606554%

AUC =54.89%

accuracy =84%

RANDOM FORREST:Results

n\_estimators=100

recall = 0.019542700801250732

precision = 100.0

AUC =50%

accuracy =90%

NAIVE BAYES:Results

recall = 35.86085597029509%

precision = 72.04554377699255%

AUC =67.08%

accuracy =92%

KNN:Results

taking a very very long time can not use it to predict the result.

"To get the most accurate model values of recall, precision, AUC should be high".

According to the question we have to predict the result based on recall , precision and AUC. so after analysing the results of all four machine learning algorithm we can see that "Naive Bayes" is giving all the three parameters equally good. In Random Forest precision is high but recal is very low.

# # FINDING THE TARGET VALUE OF TEST DATA

# In[80]:

#IMPORT DATA

test =pd.read\_csv("test.csv")

# In[81]:

test.head()

# In[82]:

id\_code = test.iloc[:,0]

# In[83]:

#REMOVE ID\_code column as it is string datatype and has no use.

test = test.drop("ID\_code" ,axis=1)

# ## PREDICTIONS

# In[84]:

predictions\_test = NB\_model.predict(test)

# In[85]:

df = pd.DataFrame({"ID\_code" :id\_code ,"target": predictions\_test})

df.head()

# In[86]:

test = df.join(test)

# In[87]:

test.head()

# In[88]:

test.shape

# In[89]:

sns.set\_style('whitegrid')

sns.countplot(x='target',data=test,palette='RdBu\_r')

# In[90]:

test["target"].value\_counts()

# ##### CONVERT TARGET VARIABLE "1" AS BUY THE PRODUCT AND "0" AS NOT BUY THE PRODUCT

# In[91]:

test["target"] =test["target"].apply( lambda x: "buy" if x==1 else "not buy")

# In[92]:

test.head()

# #### EXPORTING THE FINEL PREDICTIONS AS test\_prediction result.csv

# In[93]:

test.to\_csv("test\_prediction result.csv")

# ### THANK YOU