**Assessment**

**Created By**

**Inna Williams**

**April 12, 2020**

**Step 1 - Clean and prepare your data:**

There are several entries where values have been deleted to simulate dirty data. Please clean the data with whatever method(s) you believe is best/most suitable. Note that some of the missing values are truly blank (unknown answers). Success in this exercise typically involves feature engineering and avoiding data leakage.

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**import statsmodels.api as sm**

**import matplotlib.pyplot as plt**

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

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import classification\_report**

**from sklearn import tree**

**from sklearn import linear\_model**

Reads the training and testing datasets , Factorizes Categorical Data and assigns a numerical value to it, Converts $ and % data into flot data.

**def get\_float\_rows(dataset) :**

**col\_float = []**

**col\_object = []**

**for i in dataset.columns:**

**if dataset[i].dtypes == np.float:**

**col\_float.append(i)**

**elif dataset[i].dtypes == np.object:**

**col\_object.append(i)**

**return col\_float,col\_object**

***# Load in data***

**df\_train = pd.read\_csv('exercise\_02\_train.csv')**

**x\_test\_set = pd.read\_csv('exercise\_02\_test.csv')**

***# check data types before training***

**df\_train\_types = df\_train.dtypes**

**x\_test\_types = df\_train.dtypes**

**y\_train = df\_train['y']**

**x\_train\_set = df\_train.iloc[:,0:len(df\_train.columns)-1]**

**col\_float,col\_object = get\_float\_rows(x\_train\_set)**

**x\_train\_set[col\_object[2]] =x\_train\_set[col\_object[2]].replace('[\$,]', '', regex=True).astype(float)**

**x\_train\_set[col\_object[3]] =x\_train\_set[col\_object[3]].replace('[\%,]', '', regex=True).astype(float)**

**categorized = [col\_object[0],col\_object[1],col\_object[4],col\_object[5]]**

**for i in range(0,len(categorized)):**

**codes, uniques = pd.factorize(x\_train\_set[categorized[i]])**

**print(type(list(codes)))**

**x\_train\_set[categorized[i]] = list(codes)**

**x\_test\_set[col\_object[2]] =x\_test\_set[col\_object[2]].replace('[\$,]', '', regex=True).astype(float)**

**x\_test\_set[col\_object[3]] =x\_test\_set[col\_object[3]].replace('[\%,]', '', regex=True).astype(float)**

**categorized = [col\_object[0],col\_object[1],col\_object[4],col\_object[5]]**

**for i in range(0,len(categorized)):**

**codes, uniques = pd.factorize(x\_test\_set[categorized[i]])**

**print(type(list(codes)))**

**x\_test\_set[categorized[i]] = list(codes)**

**x\_train = x\_train\_set**

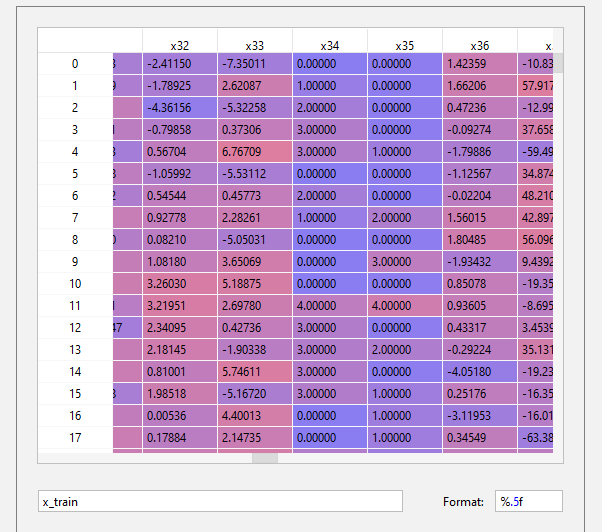
**x\_test = x\_test\_set**

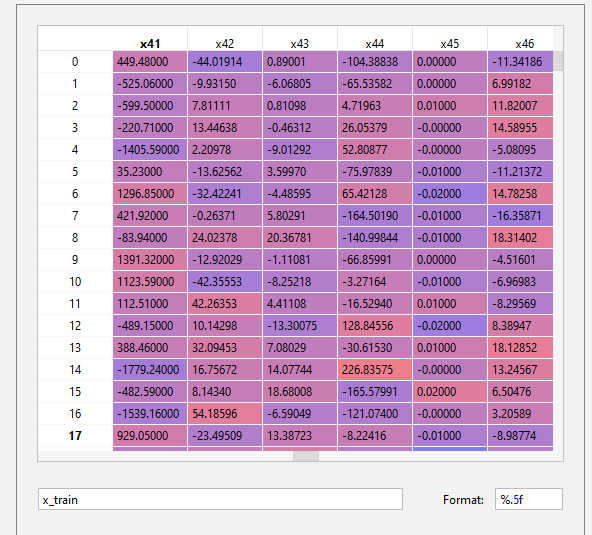
**miss\_number\_train = (x\_train.isnull().sum() > 0).astype(np.int64).sum()**

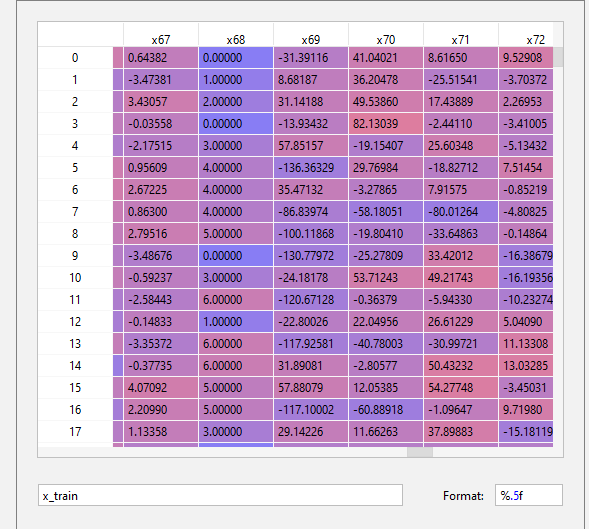
**print("After removing impurities = ",miss\_number\_train)**

**miss\_number\_test = (x\_test.isnull().sum() > 0).astype(np.int64).sum()**

**print("Before removing impurities = ",miss\_number\_test)**

****

****

****

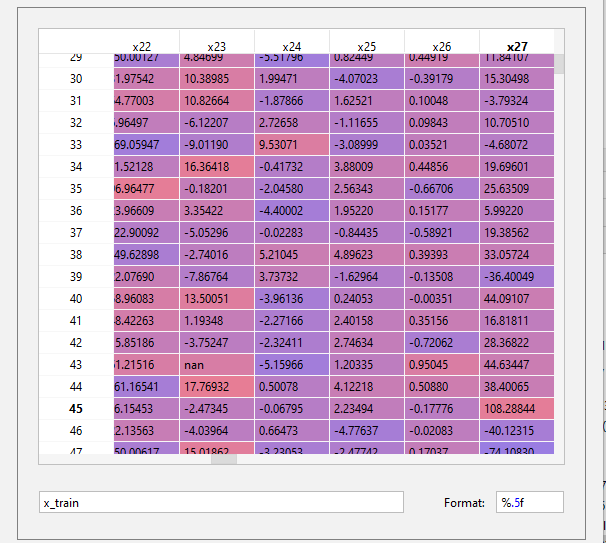
****

The same is for x\_test.

**Cleaning Datasets**

**Nearest neighbors imputation used**

**Some values are missing. For example [row= 43, col = ‘x23’]**

****

**import numpy as np**

**from sklearn.impute import KNNImputer**

**x\_train = x\_train.replace(x\_train.isnull(), np.nan)**

**x\_test = x\_test.replace(x\_train.isnull(), np.nan)**

**miss\_number\_train = (x\_train.isnull().sum() > 0).astype(np.int64).sum()**

**print("Before removing impurities = ",miss\_number\_train)**

**miss\_number\_test = (x\_test.isnull().sum() > 0).astype(np.int64).sum()**

**print("Before removing impurities = ",miss\_number\_test)**

**imputer = KNNImputer(n\_neighbors=3)**

**xtrain\_filled = imputer.fit\_transform(x\_train)**

**xtest\_filled = imputer.fit\_transform(x\_test)**

**x\_train = pd.DataFrame(data=xtrain\_filled, columns=x\_train.columns)**

**x\_test = pd.DataFrame(data=xtest\_filled, columns=x\_test.columns)**

**x\_train\_ridge = x\_train**

**x\_test\_ridge = x\_test**

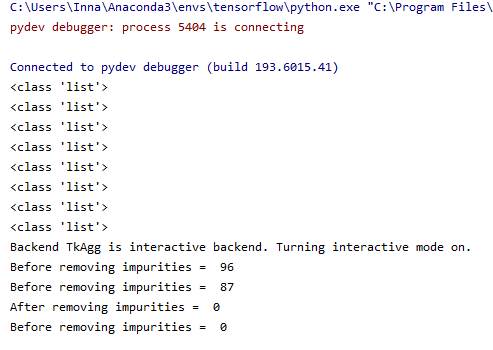
**miss\_number\_train = (x\_train.isnull().sum() > 0).astype(np.int64).sum()**

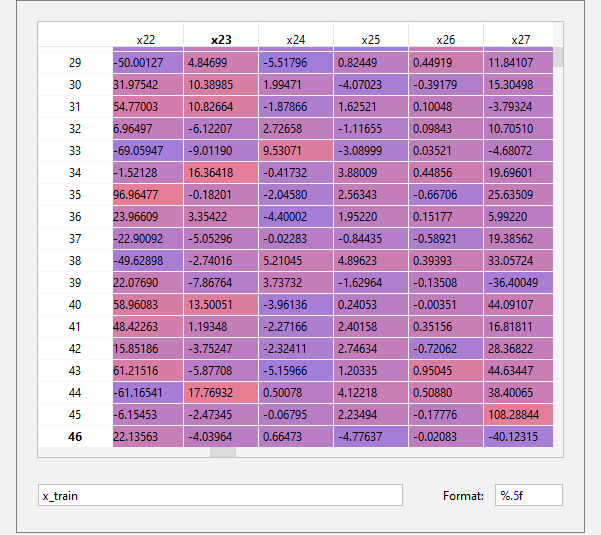
**print("After removing impurities = ",miss\_number\_train)**

**miss\_number\_test = (x\_test.isnull().sum() > 0).astype(np.int64).sum()**

**print("Before removing impurities = ",miss\_number\_test)**

**Output:**





Value for [row= 43, col = ‘x23’] =

Has been calculated using the cosine distance. The most closest column vector found to the column vector where the missing value found and the missing value was replaced by the value from the same row of the closest vector.

**Step 2 - Build your models:**

Please use two different machine learning/statistical algorithms to develop a total of two models. Please include comments that document choices you make (such as those for feature engineering and for model tuning).

**1 st algorithm : PCA/Logistic Regression**

***Training logistic regression classifier using the first Optimal principal components.***

***from sklearn.linear\_model import LogisticRegression***

***from sklearn.decomposition import PCA***

***from sklearn.preprocessing import StandardScaler***

***from sklearn import linear\_model***

***from sklearn import model\_selection***

***from sklearn.model\_selection import cross\_val\_predict***

***from sklearn.metrics import mean\_squared\_error, r2\_score***

***from sklearn.model\_selection import GridSearchCV***

***print("Start PCA/PCR ML Algorithm")***

***scaler = StandardScaler()***

***X = scaler.fit\_transform(x\_train)***

***X\_test = scaler.fit\_transform(x\_test)***

***pca = PCA(0.99)***

***pca.fit(X)***

***num\_components = pca.n\_components\_***

***print("Optimal number of Principal Components = ",num\_components)***

***pca = PCA(n\_components=num\_components)***

***x\_train\_pca = pca.fit\_transform(X)***

***x\_test\_pca = pca.fit\_transform(X\_test)***

***columns = []***

***for i in range(0,num\_components):***

***pca\_numner = i+1***

***columns.append('PC'+ str(pca\_numner))***

***# X = dataset reduced to 92 Principal Components***

***# x\_test = test dataset reduced to 92 Principal Components***

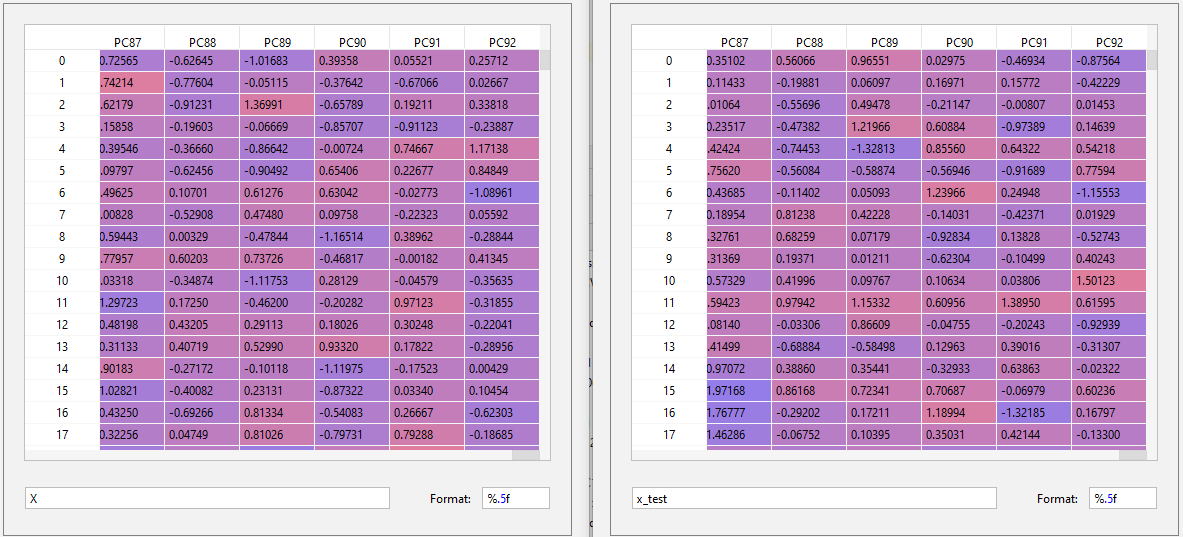
***X = pd.DataFrame(data = x\_train\_pca, columns = columns)***

***x\_test = pd.DataFrame(data = x\_test\_pca, columns = columns)***

***y = df\_train['y']***

**Optimal Value Of Principal Components found = 92**

**Train with PCA = 92**

****

**Algorithm 1: PCA/Logistic Regression**

**Regression Performed On the Principal Components.**

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

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)**

**regr = linear\_model.LogisticRegression()**

**regr.fit(X\_train, y\_train)**

**regr\_predict = regr.predict(X\_test)**

**Statistics Results**

**###############################################################**

**from sklearn.model\_selection import KFold**

**from sklearn.model\_selection import cross\_val\_score**

**kf = KFold(shuffle=True, n\_splits=5)**

**regr\_cv\_score = cross\_val\_score(regr, X, y, cv=kf, scoring='accuracy')**

**from sklearn import metrics**

**print('=============== Accuracy Score ===============')**

**print(metrics.accuracy\_score(y\_test, regr\_predict))**

**print('=============== Total Counts ================')**

**print(y\_test.value\_counts())**

**print('=============== y\_test.mean ===============')**

**print(y\_test.mean())**

**print('=============== 1 - y\_test.mean ===============')**

**print(1 - y\_test.mean())**

**print('=============== the first 25 true and predicted responses ===============')**

**print('True:', y\_test.values[0:25])**

**print('False:',regr\_predict[0:25])**

**###############################################################**

**confusion = metrics.confusion\_matrix(y\_test, regr\_predict)**

**print('=============== Confusion Matrix ===============')**

**print(confusion)**

**TP = confusion[1, 1]**

**TN = confusion[0, 0]**

**FP = confusion[0, 1]**

**FN = confusion[1, 0]**

***# use float to perform true division, not integer division***

**print((TP + TN) / float(TP + TN + FP + FN))**

**print(metrics.accuracy\_score(y\_test, regr\_predict))**

**classification\_error = (FP + FN) / float(TP + TN + FP + FN)**

**print('=============== classification error ===============')**

**print(classification\_error)**

**print(1 - metrics.accuracy\_score(y\_test, regr\_predict))**

**sensitivity = TP / float(FN + TP)**

**print('=============== sensitivity =============== ')**

**print(sensitivity)**

**print(metrics.recall\_score(y\_test, regr\_predict))**

**specificity = TN / (TN + FP)**

**print("=============== specificity =============== ")**

**print(specificity)**

**false\_positive\_rate = FP / float(TN + FP)**

**print("=============== false positive ===============")**

**print(false\_positive\_rate)**

**print(1 - specificity)**

**false\_negative\_rate = FN / float(FN + TP)**

**print("=============== false Negative ===============")**

**print(false\_negative\_rate)**

**print(1 - sensitivity)**

**precision = TP / float(TP + FP)**

**print("=============== precision ===============")**

**print(precision)**

**print(metrics.precision\_score(y\_test, regr\_predict))**

**print("================== Classification Report ===============")**

**print(classification\_report(y\_test, regr\_predict))**

**print('Cross Validation Score = ')**

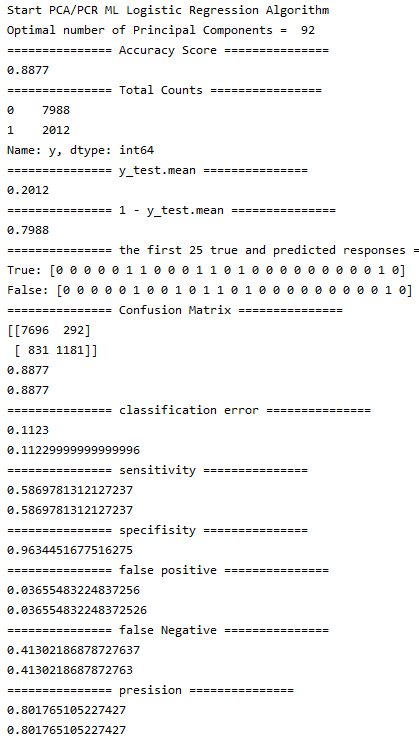
**print(regr\_cv\_score)**

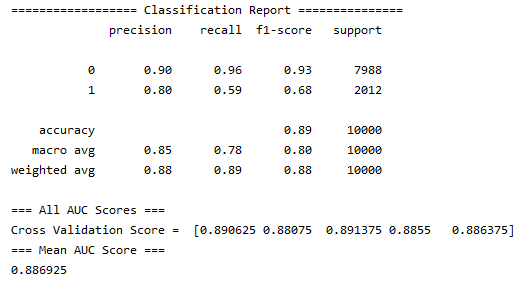
**print("=== Mean Of Cross Validation Score ===")**

**print( regr\_cv\_score.mean())**

**###############################################################**

**Statistics Results Output**

****

****

**Confusion Matrix**

**from sklearn.metrics import plot\_confusion\_matrix**

***# Plot non-normalized confusion matrix***

**titles\_options = [("Confusion matrix, without normalization", None),**

**("Normalized confusion matrix", 'true')]**

**for title, normalize in titles\_options:**

**disp = plot\_confusion\_matrix(regr, X\_test, y\_test,**

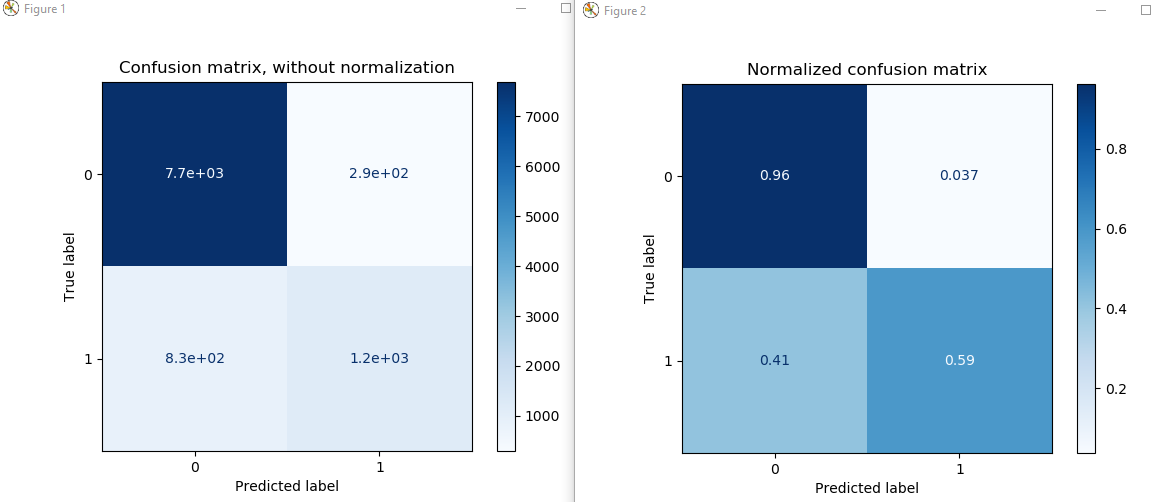
**display\_labels= ['0','1'],**

**cmap=plt.cm.Blues,**

**normalize=normalize)**

**disp.ax\_.set\_title(title)**

**plt.show()**

****

**ROC Curve**

**Receiver Operating characteristic curve is a plot of true positive rate against the**

**False positive rate.It shows the tradeoff between sensitivity and specificity**

**y\_pred\_prob = regr.predict\_proba(X\_test)[:, 1]**

**import matplotlib.pyplot as plt**

**plt.hist(y\_pred\_prob, bins=8)**

***# x-axis limit from 0 to 1***

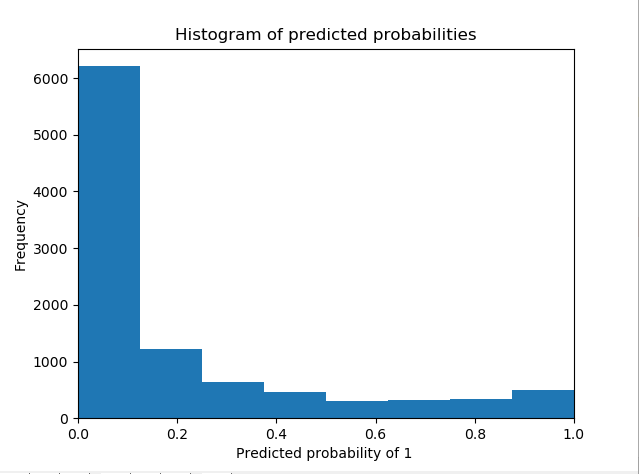
**plt.xlim(0,1)**

**plt.title('Histogram of predicted probabilities on Cross Validation test')**

**plt.xlabel('Predicted probability of 1 ')**

**plt.ylabel('Frequency')**

**plt.show()**

****

**Receiver Operating Characteristic**

y\_pred\_proba = regr.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)

auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)

plt.plot(fpr,tpr,label=**"data 1, auc="**+str(auc))

plt.legend(loc=4)

plt.show()

**import** matplotlib.pyplot **as** plt

plt.hist(y\_pred\_proba, bins=8)

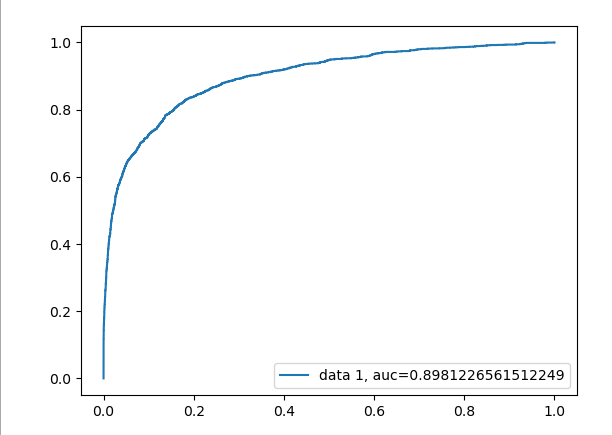
plt.xlim(0,1)

plt.title(**'Histogram of predicted probabilities'**)

plt.xlabel(**'Predicted probability of 1 '**)

plt.ylabel(**'Frequency'**)

plt.show()



**Calculate and Output predicted values on exersize\_02\_test.csv**

**y\_pred\_class = np.trunc(regr.predict(x\_test))**

**y\_pred\_class = y\_pred\_class.astype(int)**

**import numpy**

**numpy.savetxt('results1.csv', y\_pred\_class, delimiter=',', header="y predicted PCA/ Logistic Regression", comments="",fmt="%d")**

**Algorithm 2: Random Forest With best Feature Estimator**

**Find Best Feature n\_estimators and max\_depth**

**X=x\_train**

**y = df\_train['y']**

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

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)**

**from sklearn import model\_selection**

**from sklearn.model\_selection import RandomizedSearchCV**

***# number of trees in random forest***

**n\_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]**

***# number of features at every split***

**max\_features = ['auto', 'sqrt']**

***# max depth***

**max\_depth = [int(x) for x in np.linspace(100, 500, num = 11)]**

**max\_depth.append(None)**

***# create random grid***

**random\_grid = {**

**'n\_estimators': n\_estimators,**

**'max\_features': max\_features,**

**'max\_depth': max\_depth**

**}**

***# Random search of parameters***

**rfc = RandomForestClassifier()**

**rfc\_random = RandomizedSearchCV(estimator = rfc, param\_distributions = random\_grid, n\_iter = 100, cv = 3, verbose=2, random\_state=42, n\_jobs = -1)**

***# Fit the model***

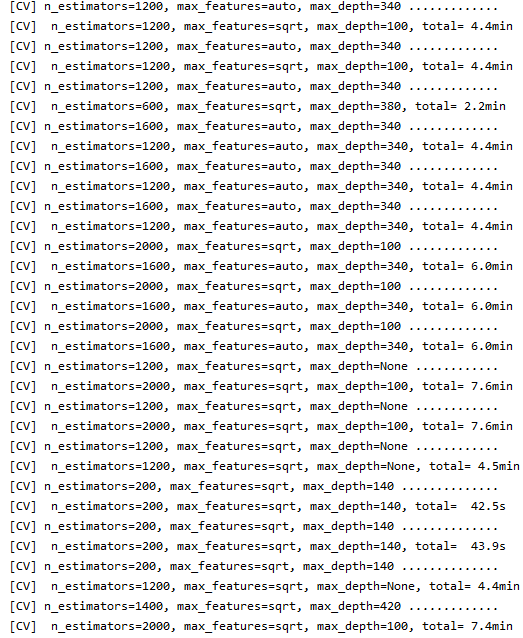
**rfc\_random.fit(X\_train, y\_train)**

***# print results***

**best\_param = rfc\_random.best\_params\_**

**print(best\_param)**

**Output**

****

**Optimal Number Of Estimator = 200**

**Optimal Number of Depth = 100**

**Run Random Forest Algorithm**

**X=x\_train**

**y = df\_train['y']**

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

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)**

**rfc = RandomForestClassifier(n\_estimators=200, max\_depth=100, max\_features='sqrt')**

**rfc.fit(X\_train,y\_train)**

**rfc\_predict = rfc.predict(X\_test)**

**Random Forest Statistics**

**from sklearn.model\_selection import KFold**

**from sklearn.model\_selection import cross\_val\_score**

**kf = KFold(shuffle=True, n\_splits=5)**

**rfc\_cv\_score = cross\_val\_score(rfc, X, y, cv=kf, scoring='accuracy')**

**from sklearn import metrics**

**print('=============== Accuracy Score ===============')**

**print(metrics.accuracy\_score(y\_test, rfc\_predict))**

**print('=============== Total Counts ================')**

**print(y\_test.value\_counts())**

**print('=============== y\_test.mean ===============')**

**print(y\_test.mean())**

**print('=============== 1 - y\_test.mean ===============')**

**print(1 - y\_test.mean())**

**print('=============== the first 25 true and predicted responses ===============')**

**print('True:', y\_test.values[0:25])**

**print('False:',rfc\_predict[0:25])**

***########################################################################***

**confusion = metrics.confusion\_matrix(y\_test, rfc\_predict)**

**TP = confusion[1, 1]**

**TN = confusion[0, 0]**

**FP = confusion[0, 1]**

**FN = confusion[1, 0]**

***# use float to perform true division, not integer division***

**print((TP + TN) / float(TP + TN + FP + FN))**

**print(metrics.accuracy\_score(y\_test, rfc\_predict))**

**classification\_error = (FP + FN) / float(TP + TN + FP + FN)**

**print('=============== classification error ===============')**

**print(classification\_error)**

**print(1 - metrics.accuracy\_score(y\_test, rfc\_predict))**

**sensitivity = TP / float(FN + TP)**

**print('=============== sensitivity =============== ')**

**print(sensitivity)**

**print(metrics.recall\_score(y\_test, rfc\_predict))**

**specificity = TN / (TN + FP)**

**print("=============== specificity =============== ")**

**print(specificity)**

**false\_positive\_rate = FP / float(TN + FP)**

**print("=============== false positive ===============")**

**print(false\_positive\_rate)**

**print(1 - specificity)**

**false\_negative\_rate = FN / float(FN + TP)**

**print("=============== false Negative ===============")**

**print(false\_negative\_rate)**

**print(1 - sensitivity)**

**precision = TP / float(TP + FP)**

**print("=============== precision ===============")**

**print(precision)**

**print(metrics.precision\_score(y\_test, rfc\_predict))**

**print("================== Classification Report ===============")**

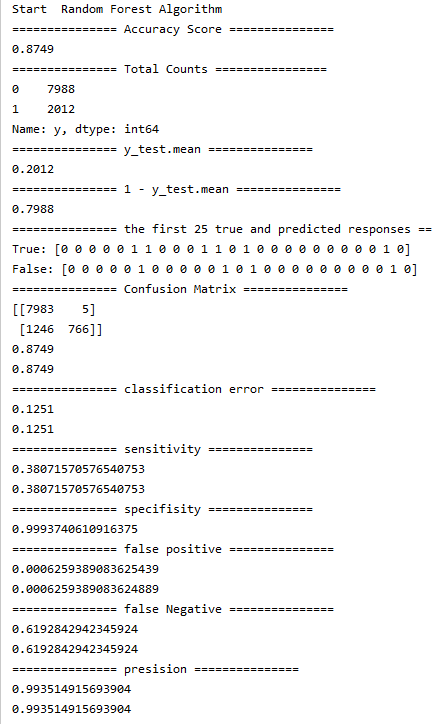
**print(classification\_report(y\_test, rfc\_predict))**

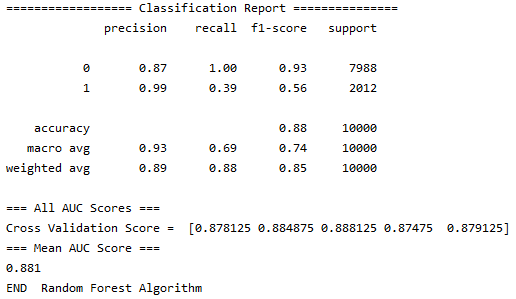
**print("=== All AUC Scores ===")**

**print('Cross Validation Score = ',rfc\_cv\_score)**

**print("=== Mean AUC Score ===")**

**print( rfc\_cv\_score.mean())**

****

****

**from sklearn.metrics import plot\_confusion\_matrix**

***# Plot non-normalized confusion matrix***

**titles\_options = [("Confusion matrix, without normalization", None),**

**("Normalized confusion matrix", 'true')]**

**for title, normalize in titles\_options:**

**disp = plot\_confusion\_matrix(rfc, X\_test, y\_test,**

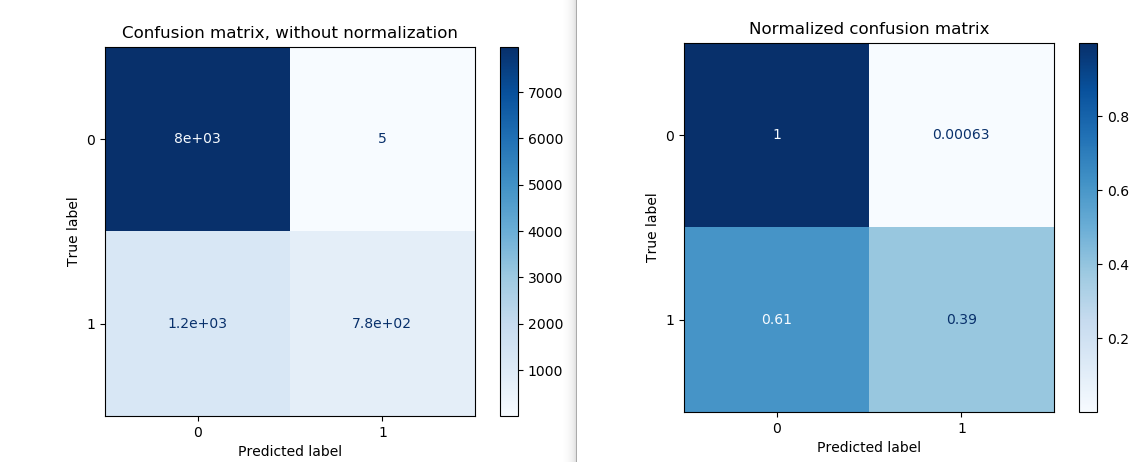
**display\_labels= ['0','1'],**

**cmap=plt.cm.Blues,**

**normalize=normalize)**

**disp.ax\_.set\_title(title)**

**plt.show()**

****

**y\_pred\_prob = rfc.predict\_proba(X\_test)[:, 1]**

**import matplotlib.pyplot as plt**

**plt.hist(y\_pred\_prob, bins=8)**

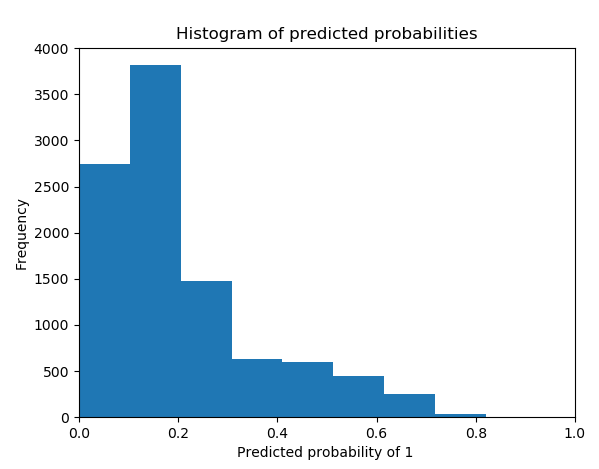
**plt.xlim(0,1)**

**plt.title('Histogram of predicted probabilities')**

**plt.xlabel('Predicted probability of 1 ')**

**plt.ylabel('Frequency')**

**plt.show()**

****

**Receiver Operating Characteristic**

y\_pred\_proba = rfc.predict\_proba(X\_test)[:, 1]

fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)

auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)

plt.plot(fpr,tpr,label=**"data 1, auc="**+str(auc))

plt.legend(loc=4)

plt.show()

**import** matplotlib.pyplot **as** plt

plt.hist(y\_pred\_proba, bins=8)

plt.xlim(0,1)

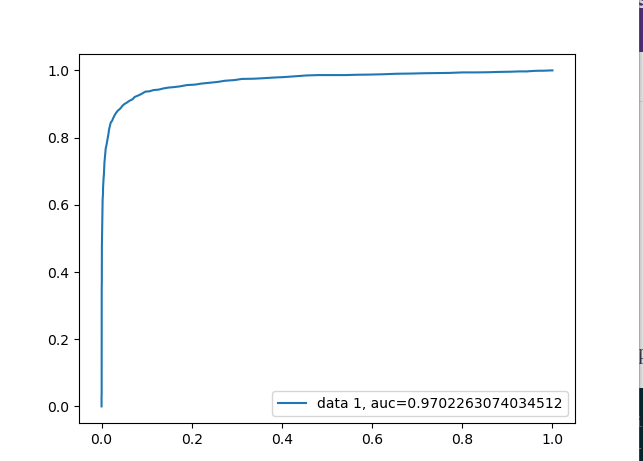
plt.title(**'Histogram of predicted probabilities'**)

plt.xlabel(**'Predicted probability of 1 '**)

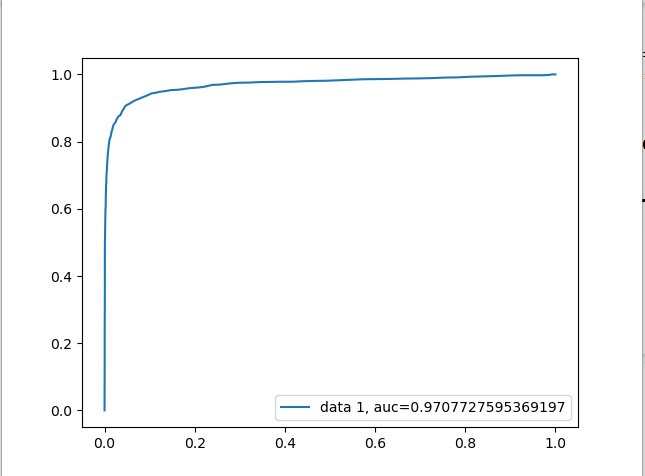
plt.ylabel(**'Frequency'**)

plt.show()

**rfc\_cv\_score = cross\_val\_score(rfc, X, y, cv=kf, scoring='accuracy')**

****

**rfc\_cv\_score = cross\_val\_score(rfc, X, y, cv=kf, scoring='roc\_auc')**

****

**Step 3 - Generate predictions:**

Create predictions on the data in **test.csv** using each of your trained models. The predictions should be the class probabilities for belonging to the positive class (labeled '1').

Be sure to output a prediction for each of the rows in the test dataset (**10K rows**). Save the results of each of your models in a separate CSV file. Title the two files **'results1.csv'** and **'results2.csv'**. A result file should each have a single column representing the output from one model (no header label or index column is needed).

**Files are attached.**

**Step 4 - Compare your modeling approaches:**

Please prepare a relatively short write-up comparing the pros and cons of the two algorithms you used (PDF preferred). As part of the write-up, please identify which algorithm you think will perform the best. For the best performing model, are there choices you made in the context of the exercise that might be different in a business context? How would explain to a business partner the concept that one model is better than the other?

Accuracy Score =

PCA/Logistic Regression = **0.8877 > 0.8766** = Random Forest

Classification Error =

PCA/Logistic Regression = **0.1123 < 0.1234** = Random Forest

Sensitivity =

PCA/Logistic Regression = **0.5870 < 0.3892** = Random Forest

Specificity =

PCA/Logistic Regression = **0.9635 < 0.9994** = Random Forest

False Positive =

PCA/Logistic Regression = **0.0366 > 0.0006** = Random Forest

False Negative =

PCA/Logistic Regression = **0.4130 < 0.6108** = Random Forest

Precision =

PCA/Logistic Regression = **0.8918 < 0.9937** = Random Forest

Cross Validation Score =

PCA/Logistic Regression = **0.8869 > 0.8810** = Random Forest

RECALL =

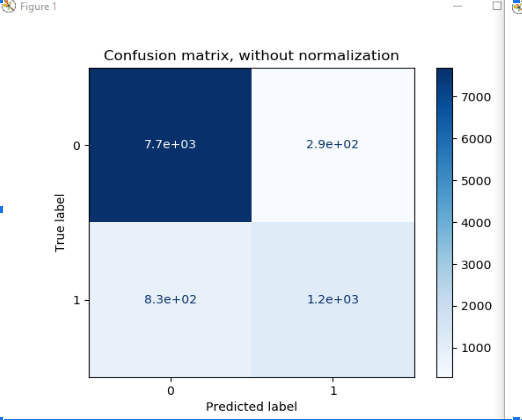
PCA/Logistic Regression = **0.5900 > 0.3900** = Random Forest

F1 score =

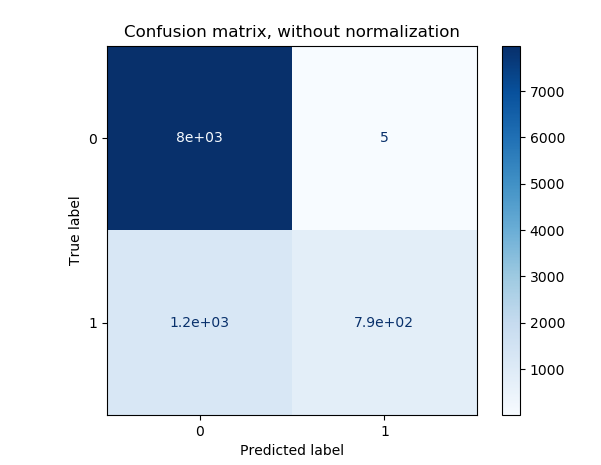
PCA/Logistic for Regression ‘1’ = **0.68 > 0.56** = Random Forest

Confusion Matrix evaluates performance of a classification model.

**Confusion Matrix for Logistic Regression**

****

**Confusion matrix for Random Forest**

****

Performance for Logistic Regression => **0.8877** accuracy

Accurate prediction = 7696 + 1181 = 8877

Inaccurate predictions = 831 + 292 = 1123

Performance forRandom Forest => 0.8754

Accurate prediction = 7982 + 772 = 8754

Inaccurate predictions = 1240 +6 = 1246

**From the above results Accuracy, Precision, Recall,Cross Validation Score we can see that Logistic regression has slightly better scores. Classification Error is less for Logistic Regression Model. F1 score shows that Logistic regression vs Random Forest is a better model.**

**The model has**

**0 7988**

**1 2012**

**It is not a balanced dataset and therefore the best criterion between**

**AUC vs Accuracy will be Accuracy.**

**AUC Random Forest = 0.97 > 0.89 = Logistic Regression**

**But Accuracy Of Random Forest = 0.8766 < 0.8877 = Logistic Regression**

**AUC is, I think, a more comprehensive measure, although applicable in fewer situations. It's not strictly better than accuracy; it's different. It depends in part on whether you care more about true positives, false negatives, etc.**

**When you have a data imbalance between positive and negative samples, you should always use F1-score because of ROC averages over all possible thresholds.**

**In our case**

**F1 score =**

**PCA/Logistic for Regression ‘1’ = 0.68 > 0.56 = Random Forest**

**Therefore a logistic regression model would be a better choice.**

**Step 5 - Submit your work:**

Your submission should consist of all the code used for exploratory data analysis, cleaning, prepping, and modeling (text, html, or pdf preferred), the two result files (.csv format - each containing 10,000 decimal probabilities), and your write-up comparing the pros and cons of the two modeling techniques used (text, html, or pdf preferred). Note:

Code written in python ,attached in assignment.py

The results files should not include the original data, only the probabilities.