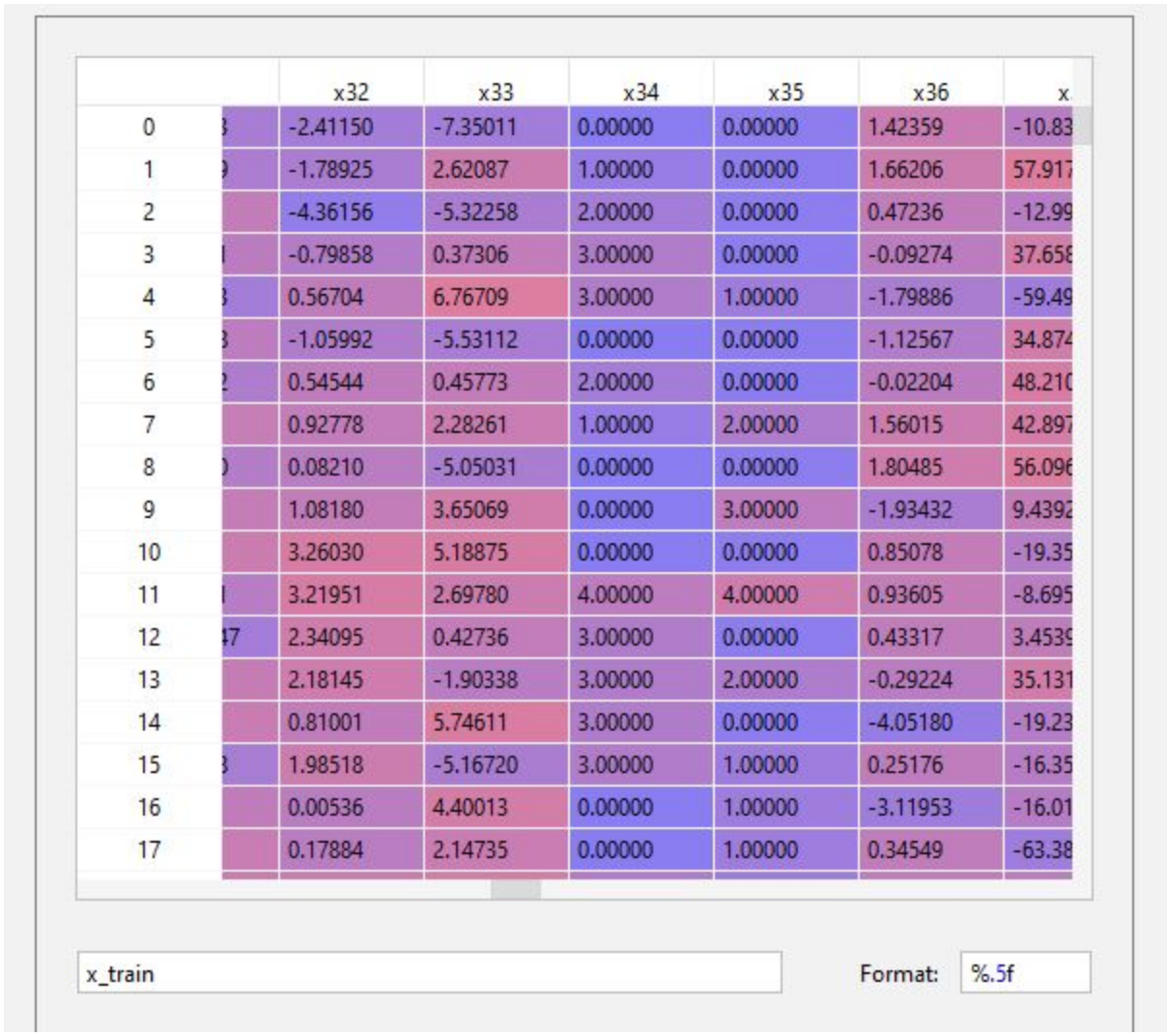
Step 1 - Clean and prepare your data:
There are several entries where values have been deleted to simulate dirty data. Please clean the data with whatever method(s) you believe is best/most suitable. Note that some of the missing values are truly blank (unknown answers). Success in this exercise typically 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)
```



		x32	x33	x34	x35	x36	x
0	3	-2.41150	-7.35011	0.00000	0.00000	1.42359	-10.83
1	9	-1.78925	2.62087	1.00000	0.00000	1.66206	57.917
2		-4.36156	-5.32258	2.00000	0.00000	0.47236	-12.99
3		-0.79858	0.37306	3.00000	0.00000	-0.09274	37.658
4	3	0.56704	6.76709	3.00000	1.00000	-1.79886	-59.49
5	3	-1.05992	-5.53112	0.00000	0.00000	-1.12567	34.874
6	2	0.54544	0.45773	2.00000	0.00000	-0.02204	48.210
7		0.92778	2.28261	1.00000	2.00000	1.56015	42.897
8	0	0.08210	-5.05031	0.00000	0.00000	1.80485	56.096
9		1.08180	3.65069	0.00000	3.00000	-1.93432	9.4392
10		3.26030	5.18875	0.00000	0.00000	0.85078	-19.35
11		3.21951	2.69780	4.00000	4.00000	0.93605	-8.695
12	47	2.34095	0.42736	3.00000	0.00000	0.43317	3.4539
13		2.18145	-1.90338	3.00000	2.00000	-0.29224	35.131
14		0.81001	5.74611	3.00000	0.00000	-4.05180	-19.23
15	3	1.98518	-5.16720	3.00000	1.00000	0.25176	-16.35
16		0.00536	4.40013	0.00000	1.00000	-3.11953	-16.01
17		0.17884	2.14735	0.00000	1.00000	0.34549	-63.38

	x41	x42	x43	x44	x45	x46
0	449.48000	-44.01914	0.89001	-104.38838	0.00000	-11.34186
1	-525.06000	-9.93150	-6.06805	-65.53582	0.00000	6.99182
2	-599.50000	7.81111	0.81098	4.71963	0.01000	11.82007
3	-220.71000	13.44638	-0.46312	26.05379	-0.00000	14.58955
4	-1405.59000	2.20978	-9.01292	52.80877	-0.00000	-5.08095
5	35.23000	-13.62562	3.59970	-75.97839	-0.01000	-11.21372
6	1296.85000	-32.42241	-4.48595	65.42128	-0.02000	14.78258
7	421.92000	-0.26371	5.80291	-164.50190	-0.01000	-16.35871
8	-83.94000	24.02378	20.36781	-140.99844	-0.01000	18.31402
9	1391.32000	-12.92029	-1.11081	-66.85991	0.00000	-4.51601
10	1123.59000	-42.35553	-8.25218	-3.27164	-0.01000	-6.96983
11	112.51000	42.26353	4.41108	-16.52940	0.01000	-8.29569
12	-489.15000	10.14298	-13.30075	128.84556	-0.02000	8.38947
13	388.46000	32.09453	7.08029	-30.61530	0.01000	18.12852
14	-1779.24000	16.75672	14.07744	226.83575	-0.00000	13.24567
15	-482.59000	8.14340	18.68008	-165.57991	0.02000	6.50476
16	-1539.16000	54.18596	-6.59049	-121.07400	-0.00000	3.20589
17	929.05000	-23.49509	13.38723	-8.22416	-0.01000	-8.98774

x_train

Format: %.5f

	x67	x68	x69	x70	x71	x72
0	0.64382	0.00000	-31.39116	41.04021	8.61650	9.52908
1	-3.47381	1.00000	8.68187	36.20478	-25.51541	-3.70372
2	3.43057	2.00000	31.14188	49.53860	17.43889	2.26953
3	-0.03558	0.00000	-13.93432	82.13039	-2.44110	-3.41005
4	-2.17515	3.00000	57.85157	-19.15407	25.60348	-5.13432
5	0.95609	4.00000	-136.36329	29.76984	-18.82712	7.51454
6	2.67225	4.00000	35.47132	-3.27865	7.91575	-0.85219
7	0.86300	4.00000	-86.83974	-58.18051	-80.01264	-4.80825
8	2.79516	5.00000	-100.11868	-19.80410	-33.64863	-0.14864
9	-3.48676	0.00000	-130.77972	-25.27809	33.42012	-16.38679
10	-0.59237	3.00000	-24.18178	53.71243	49.21743	-16.19356
11	-2.58443	6.00000	-120.67128	-0.36379	-5.94330	-10.23274
12	-0.14833	1.00000	-22.80026	22.04956	26.61229	5.04090
13	-3.35372	6.00000	-117.92581	-40.78003	-30.99721	11.13308
14	-0.37735	6.00000	31.89081	-2.80577	50.43232	13.03285
15	4.07092	5.00000	57.88079	12.05385	54.27748	-3.45031
16	2.20990	5.00000	-117.10002	-60.88918	-1.09647	9.71980
17	1.13358	3.00000	29.14226	11.66263	37.89883	-15.18119

x_train

Format: %.5f

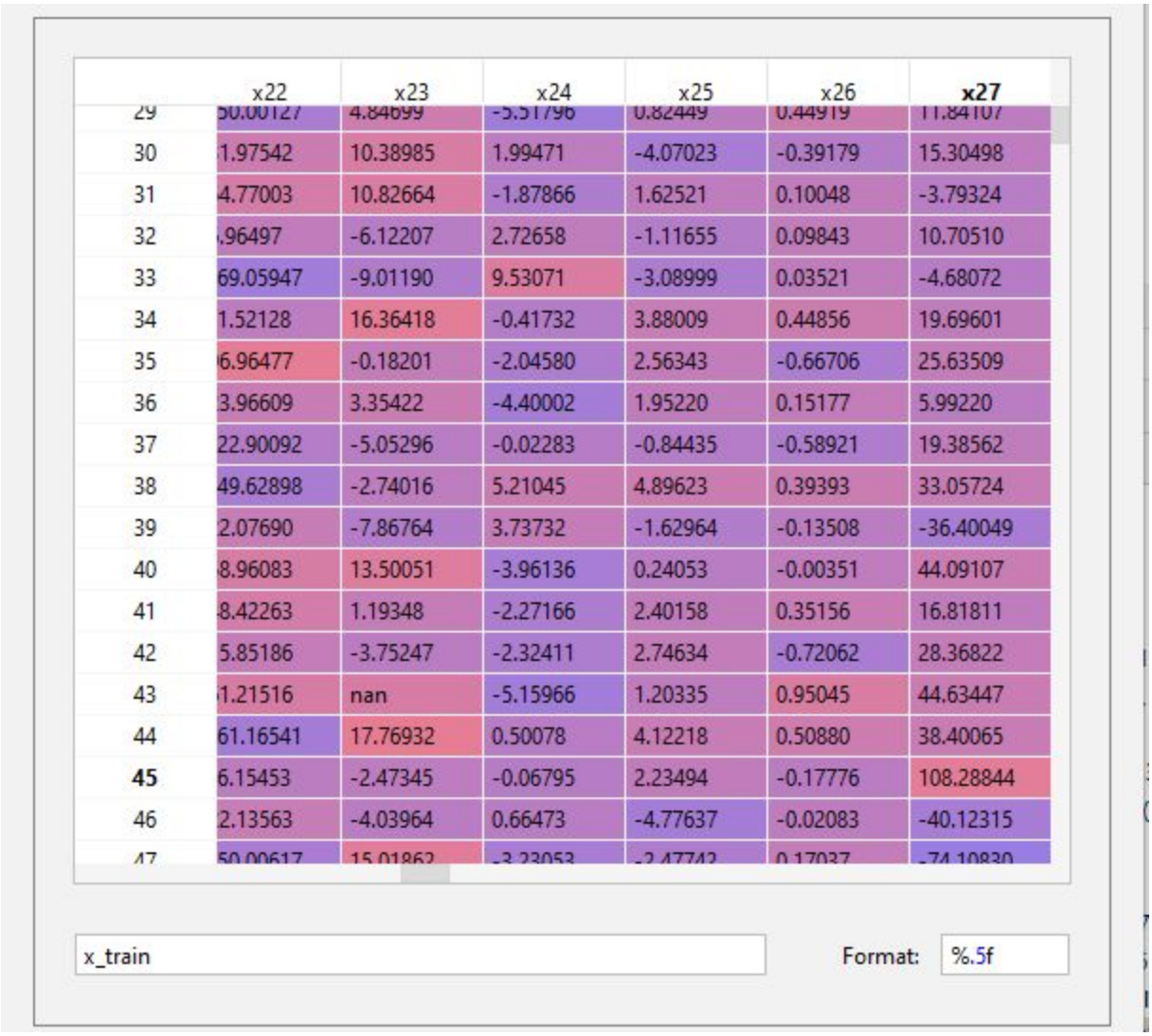
	x93	x94	x95	x96	x97	x98
0	0.00000	-1.09393	16.20256	26.23859	-2.12557	9.64447
1	0.00000	-3.65954	29.67426	-15.14165	-36.03060	5.82038
2	1.00000	1.29914	33.01809	-19.91489	26.21274	2.37269
3	0.00000	0.52140	9.66409	-27.19764	19.22113	13.38271
4	0.00000	1.52607	-25.60833	33.38380	-5.70327	-11.0237
5	0.00000	-3.05176	-1.83833	-21.41521	25.10123	6.33365
6	0.00000	0.00943	35.61398	-1.94740	7.31750	-0.17612
7	0.00000	1.06608	15.54567	6.31450	6.43395	6.22698
8	0.00000	-3.21457	-5.33034	-8.08884	-4.69867	-7.81113
9	0.00000	0.15832	-53.49463	-28.47449	10.08108	-1.05312
10	0.00000	2.51981	55.17320	-24.88805	-12.19623	8.37332
11	0.00000	-0.11819	9.79803	-32.58796	-27.39402	0.54768
12	0.00000	1.20146	-42.79377	-7.98548	22.24393	-13.4643
13	0.00000	0.81467	37.06758	-22.89846	6.91548	-8.25845
14	0.00000	-0.60623	46.37118	6.17578	-9.65843	-6.00248
15	0.00000	2.47745	31.61207	-3.79570	17.84408	14.24222
16	0.00000	4.41809	-62.64170	-5.14270	18.74056	4.29790
17	0.00000	-1.83242	-29.71622	-10.61789	-3.03747	9.50518

The same is for x_test.

Cleaning Datasets

Nearest neighbors imputation used

Some values are missing. For example [row= 43, col = 'x23']



	x22	x23	x24	x25	x26	x27
29	50.00127	4.84699	-5.51796	0.82449	0.44919	11.84107
30	1.97542	10.38985	1.99471	-4.07023	-0.39179	15.30498
31	4.77003	10.82664	-1.87866	1.62521	0.10048	-3.79324
32	.96497	-6.12207	2.72658	-1.11655	0.09843	10.70510
33	69.05947	-9.01190	9.53071	-3.08999	0.03521	-4.68072
34	1.52128	16.36418	-0.41732	3.88009	0.44856	19.69601
35	6.96477	-0.18201	-2.04580	2.56343	-0.66706	25.63509
36	3.96609	3.35422	-4.40002	1.95220	0.15177	5.99220
37	22.90092	-5.05296	-0.02283	-0.84435	-0.58921	19.38562
38	49.62898	-2.74016	5.21045	4.89623	0.39393	33.05724
39	2.07690	-7.86764	3.73732	-1.62964	-0.13508	-36.40049
40	8.96083	13.50051	-3.96136	0.24053	-0.00351	44.09107
41	8.42263	1.19348	-2.27166	2.40158	0.35156	16.81811
42	5.85186	-3.75247	-2.32411	2.74634	-0.72062	28.36822
43	1.21516	nan	-5.15966	1.20335	0.95045	44.63447
44	61.16541	17.76932	0.50078	4.12218	0.50880	38.40065
45	6.15453	-2.47345	-0.06795	2.23494	-0.17776	108.28844
46	2.13563	-4.03964	0.66473	-4.77637	-0.02083	-40.12315
47	50.00617	15.01862	-3.23053	-2.47742	0.17037	-74.10830

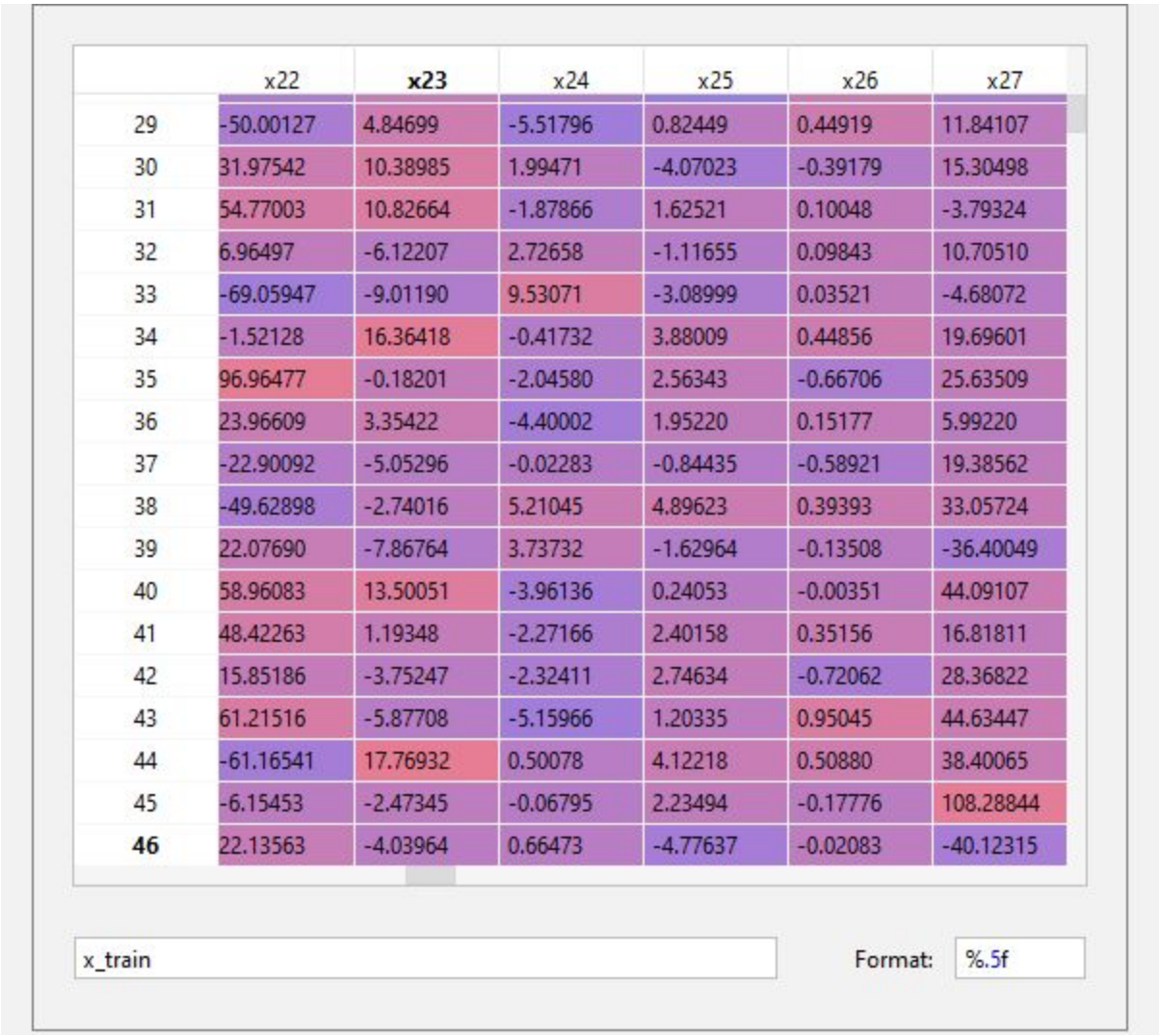
x_train Format: %.5f

```
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:

```
C:\Users\Inna\Anaconda3\envs\tensorflow\python.exe "C:\Program Files\
pydev debugger: process 5404 is connecting

Connected to pydev debugger (build 193.6015.41)
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
Backend TkAgg is interactive backend. Turning interactive mode on.
Before removing impurities =  96
Before removing impurities =  87
After removing impurities =  0
Before removing impurities =  0
```



	x22	x23	x24	x25	x26	x27
29	-50.00127	4.84699	-5.51796	0.82449	0.44919	11.84107
30	31.97542	10.38985	1.99471	-4.07023	-0.39179	15.30498
31	54.77003	10.82664	-1.87866	1.62521	0.10048	-3.79324
32	6.96497	-6.12207	2.72658	-1.11655	0.09843	10.70510
33	-69.05947	-9.01190	9.53071	-3.08999	0.03521	-4.68072
34	-1.52128	16.36418	-0.41732	3.88009	0.44856	19.69601
35	96.96477	-0.18201	-2.04580	2.56343	-0.66706	25.63509
36	23.96609	3.35422	-4.40002	1.95220	0.15177	5.99220
37	-22.90092	-5.05296	-0.02283	-0.84435	-0.58921	19.38562
38	-49.62898	-2.74016	5.21045	4.89623	0.39393	33.05724
39	22.07690	-7.86764	3.73732	-1.62964	-0.13508	-36.40049
40	58.96083	13.50051	-3.96136	0.24053	-0.00351	44.09107
41	48.42263	1.19348	-2.27166	2.40158	0.35156	16.81811
42	15.85186	-3.75247	-2.32411	2.74634	-0.72062	28.36822
43	61.21516	-5.87708	-5.15966	1.20335	0.95045	44.63447
44	-61.16541	17.76932	0.50078	4.12218	0.50880	38.40065
45	-6.15453	-2.47345	-0.06795	2.23494	-0.17776	108.28844
46	22.13563	-4.03964	0.66473	-4.77637	-0.02083	-40.12315

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

	PC87	PC88	PC89	PC90	PC91	PC92
0	0.72565	-0.62645	-1.01683	0.39358	0.05521	0.25712
1	.74214	-0.77604	-0.05115	-0.37642	-0.67066	0.02667
2	.62179	-0.91231	1.36991	-0.65789	0.19211	0.33818
3	.15858	-0.19603	-0.06669	-0.85707	-0.91123	-0.23887
4	0.39546	-0.36660	-0.86642	-0.00724	0.74667	1.17138
5	.09797	-0.62456	-0.90492	0.65406	0.22677	0.84849
6	.49625	0.10701	0.61276	0.63042	-0.02773	-1.08961
7	.00828	-0.52908	0.47480	0.09758	-0.22323	0.05592
8	0.59443	0.00329	-0.47844	-1.16514	0.38962	-0.28844
9	.77957	0.60203	0.73726	-0.46817	-0.00182	0.41345
10	.03318	-0.34874	-1.11753	0.28129	-0.04579	-0.35635
11	1.29723	0.17250	-0.46200	-0.20282	0.97123	-0.31855
12	0.48198	0.43205	0.29113	0.18026	0.30248	-0.22041
13	0.31133	0.40719	0.52990	0.93320	0.17822	-0.28956
14	.90183	-0.27172	-0.10118	-1.11975	-0.17523	0.00429
15	1.02821	-0.40082	0.23131	-0.87322	0.03340	0.10454
16	0.43250	-0.69266	0.81334	-0.54083	0.26667	-0.62303
17	0.32256	0.04749	0.81026	-0.79731	0.79288	-0.18685

X Format: %.5f

	PC87	PC88	PC89	PC90	PC91	PC92
0	0.35102	0.56066	0.96551	0.02975	-0.46934	-0.87564
1	0.11433	-0.19881	0.06097	0.16971	0.15772	-0.42229
2	.01064	-0.55696	0.49478	-0.21147	-0.00807	0.01453
3	0.23517	-0.47382	1.21966	0.60884	-0.97389	0.14639
4	.42424	-0.74453	-1.32813	0.85560	0.64322	0.54218
5	.75620	-0.56084	-0.58874	-0.56946	-0.91689	0.77594
6	0.43685	-0.11402	0.05093	1.23966	0.24948	-1.15553
7	0.18954	0.81238	0.42228	-0.14031	-0.42371	0.01929
8	.32761	0.68259	0.07179	-0.92834	0.13828	-0.52743
9	.31369	0.19371	0.01211	-0.62304	-0.10499	0.40243
10	0.57329	0.41996	0.09767	0.10634	0.03806	1.50123
11	.59423	0.97942	1.15332	0.60956	1.38950	0.61595
12	.08140	-0.03306	0.86609	-0.04755	-0.20243	-0.92939
13	.41499	-0.68884	-0.58498	0.12963	0.39016	-0.31307
14	0.97072	0.38860	0.35441	-0.32933	0.63863	-0.02322
15	1.97168	0.86168	0.72341	0.70687	-0.06979	0.60236
16	1.76777	-0.29202	0.17211	1.18994	-1.32185	0.16797
17	1.46286	-0.06752	0.10395	0.35031	0.42144	-0.13300

x_test Format: %.5f

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())
```

##

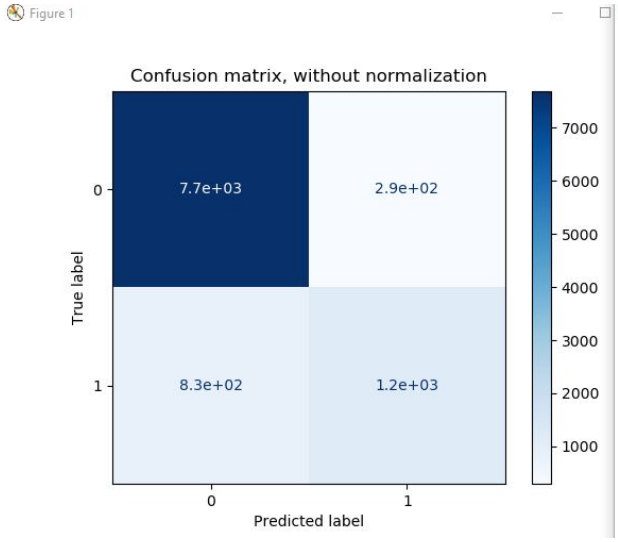
Statistics Results Output

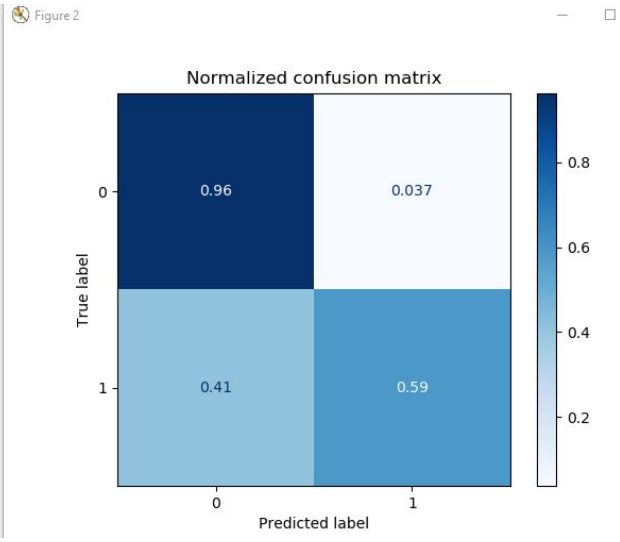
```
Start PCA/PCR ML Logistic Regression Algorithm
Optimal number of Principal Components =  92
=============== Accuracy Score ===============
0.8877
=============== Total Counts ================
0    7988
1    2012
Name: y, dtype: int64
=============== y_test.mean ===============
0.2012
=============== 1 - y_test.mean ===============
0.7988
=============== the first 25 true and predicted responses =
True: [0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0]
False: [0 0 0 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0]
=============== Confusion Matrix ===============
[[7696  292]
 [ 831 1181]]
0.8877
0.8877
=============== classification error ===============
0.1123
0.11229999999999996
=============== sensitivity ===============
0.5869781312127237
0.5869781312127237
=============== specifisity ===============
0.9634451677516275
=============== false positive ===============
0.03655483224837256
0.036554832248372526
=============== false Negative ===============
0.41302186878727637
0.4130218687872763
=============== presision ===============
0.801765105227427
0.801765105227427
```

```
================== Classification Report ===============
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      7988
           1       0.80      0.59      0.68      2012

    accuracy                           0.89     10000
   macro avg       0.85      0.78      0.80     10000
weighted avg       0.88      0.89      0.88     10000

=== All AUC Scores ===
Cross Validation Score =  [0.890625 0.88075  0.891375 0.8855   0.886375]
=== Mean AUC Score ===
0.886925
```

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()
```

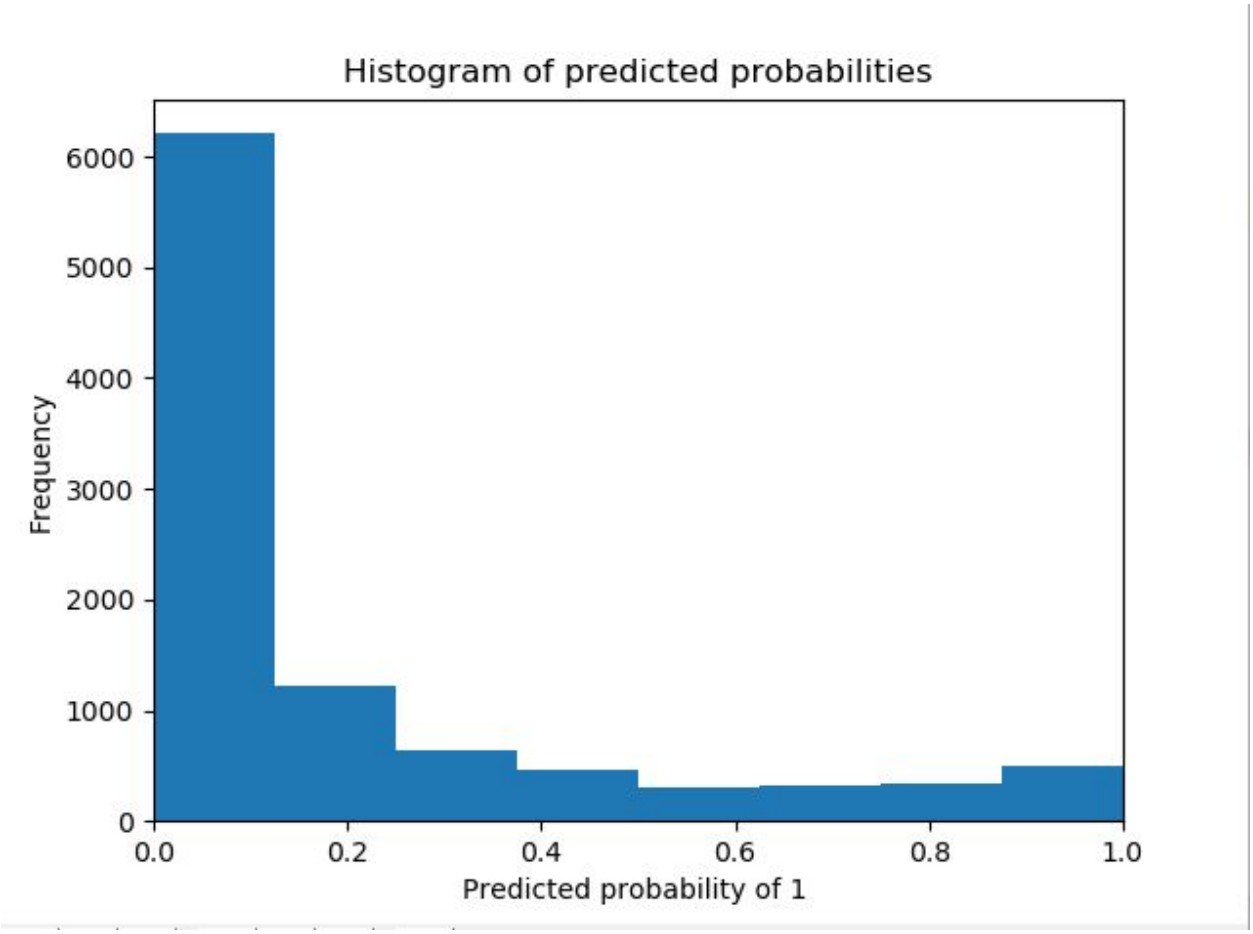




ROC Curve

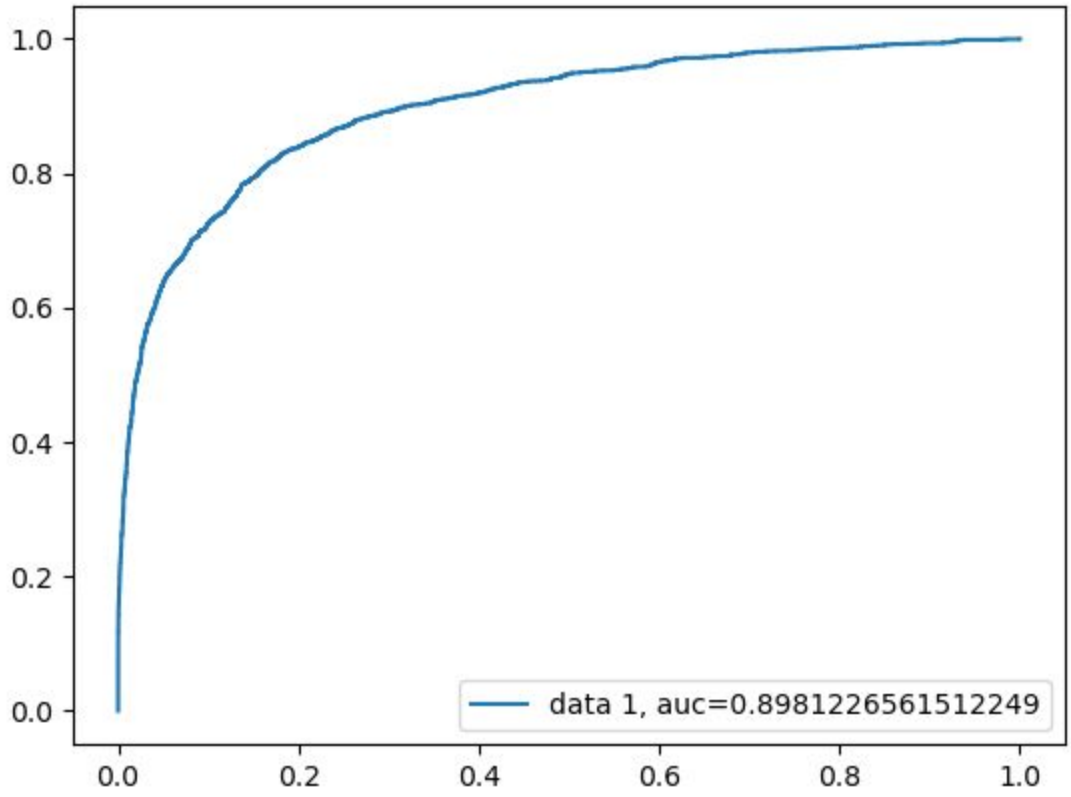
Receiver Operating characteristic curve is a plot of true positive rate against the False positive rate.It shows the tradeoff between sensitivity and specificity

```
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

```
[CV] n_estimators=1200, max_features=auto, max_depth=340 .............
[CV]  n_estimators=1200, max_features=sqrt, max_depth=100, total= 4.4min
[CV] n_estimators=1200, max_features=auto, max_depth=340 .............
[CV]  n_estimators=1200, max_features=sqrt, max_depth=100, total= 4.4min
[CV] n_estimators=1200, max_features=auto, max_depth=340 .............
[CV]  n_estimators=600, max_features=sqrt, max_depth=380, total= 2.2min
[CV] n_estimators=1600, max_features=auto, max_depth=340 .............
[CV]  n_estimators=1200, max_features=auto, max_depth=340, total= 4.4min
[CV] n_estimators=1600, max_features=auto, max_depth=340 .............
[CV]  n_estimators=1200, max_features=auto, max_depth=340, total= 4.4min
[CV] n_estimators=1600, max_features=auto, max_depth=340 .............
[CV]  n_estimators=1200, max_features=auto, max_depth=340, total= 4.4min
[CV] n_estimators=2000, max_features=sqrt, max_depth=100 .............
[CV]  n_estimators=1600, max_features=auto, max_depth=340, total= 6.0min
[CV] n_estimators=2000, max_features=sqrt, max_depth=100 .............
[CV]  n_estimators=1600, max_features=auto, max_depth=340, total= 6.0min
[CV] n_estimators=2000, max_features=sqrt, max_depth=100 .............
[CV]  n_estimators=1600, max_features=auto, max_depth=340, total= 6.0min
[CV] n_estimators=1200, max_features=sqrt, max_depth=None ............
[CV]  n_estimators=2000, max_features=sqrt, max_depth=100, total= 7.6min
[CV] n_estimators=1200, max_features=sqrt, max_depth=None ............
[CV]  n_estimators=2000, max_features=sqrt, max_depth=100, total= 7.6min
[CV] n_estimators=1200, max_features=sqrt, max_depth=None ............
[CV]  n_estimators=1200, max_features=sqrt, max_depth=None, total= 4.5min
[CV] n_estimators=200, max_features=sqrt, max_depth=140 ..............
[CV]  n_estimators=200, max_features=sqrt, max_depth=140, total=  42.5s
[CV] n_estimators=200, max_features=sqrt, max_depth=140 ..............
[CV]  n_estimators=200, max_features=sqrt, max_depth=140, total=  43.9s
[CV] n_estimators=200, max_features=sqrt, max_depth=140 ..............
[CV]  n_estimators=1200, max_features=sqrt, max_depth=None, total= 4.4min
[CV] n_estimators=1400, max_features=sqrt, max_depth=420 .............
[CV]  n_estimators=2000, max_features=sqrt, max_depth=100, total= 7.4min
```

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

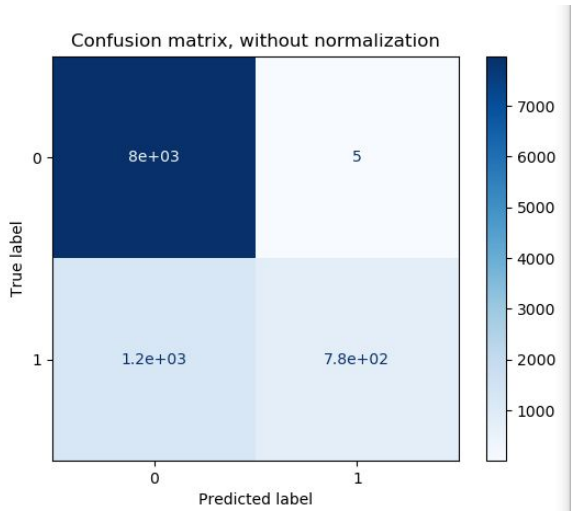
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kf = KFold(shuffle=True, n_splits=5)
rfc_cv_score = cross_val_score(rfc, X, y, cv=kf, scoring='accuracy')

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)
```

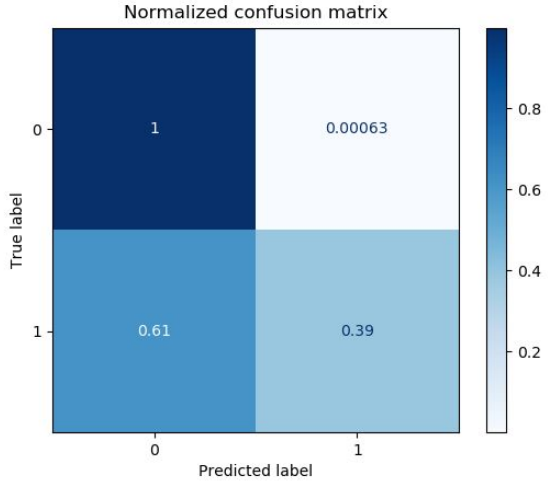
```
print("================ false Negative ================")
print(false_negative_rate)
print(1 - sensitivity)
precision = TP / float(TP + FP)
print("================ precision ================")
print(precision)
print(metrics.precision_score(y_test, rfc_predict))
print("=================== Classification 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())
```

```
Start  Random Forest Algorithm
=============== Accuracy Score ===============
0.8749
=============== Total Counts ================
0    7988
1    2012
Name: y, dtype: int64
=============== y_test.mean ===============
0.2012
=============== 1 - y_test.mean ===============
0.7988
=============== the first 25 true and predicted responses ==
True: [0 0 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0]
False: [0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0]
=============== Confusion Matrix ===============
[[7983    5]
 [1246  766]]
0.8749
0.8749
=============== classification error ===============
0.1251
0.1251
=============== sensitivity ===============
0.38071570576540753
0.38071570576540753
=============== specifisity ===============
0.9993740610916375
=============== false positive ===============
0.0006259389083625439
0.0006259389083624889
=============== false Negative ===============
0.6192842942345924
0.6192842942345924
=============== presision ===============
0.993514915693904
0.993514915693904
```

```
================== Classification Report ===============
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      7988
           1       0.99      0.39      0.56      2012

    accuracy                           0.88     10000
   macro avg       0.93      0.69      0.74     10000
weighted avg       0.89      0.88      0.85     10000

=== All AUC Scores ===
Cross Validation Score =  [0.878125 0.884875 0.888125 0.87475  0.879125]
=== Mean AUC Score ===
0.881
END  Random Forest Algorithm
```

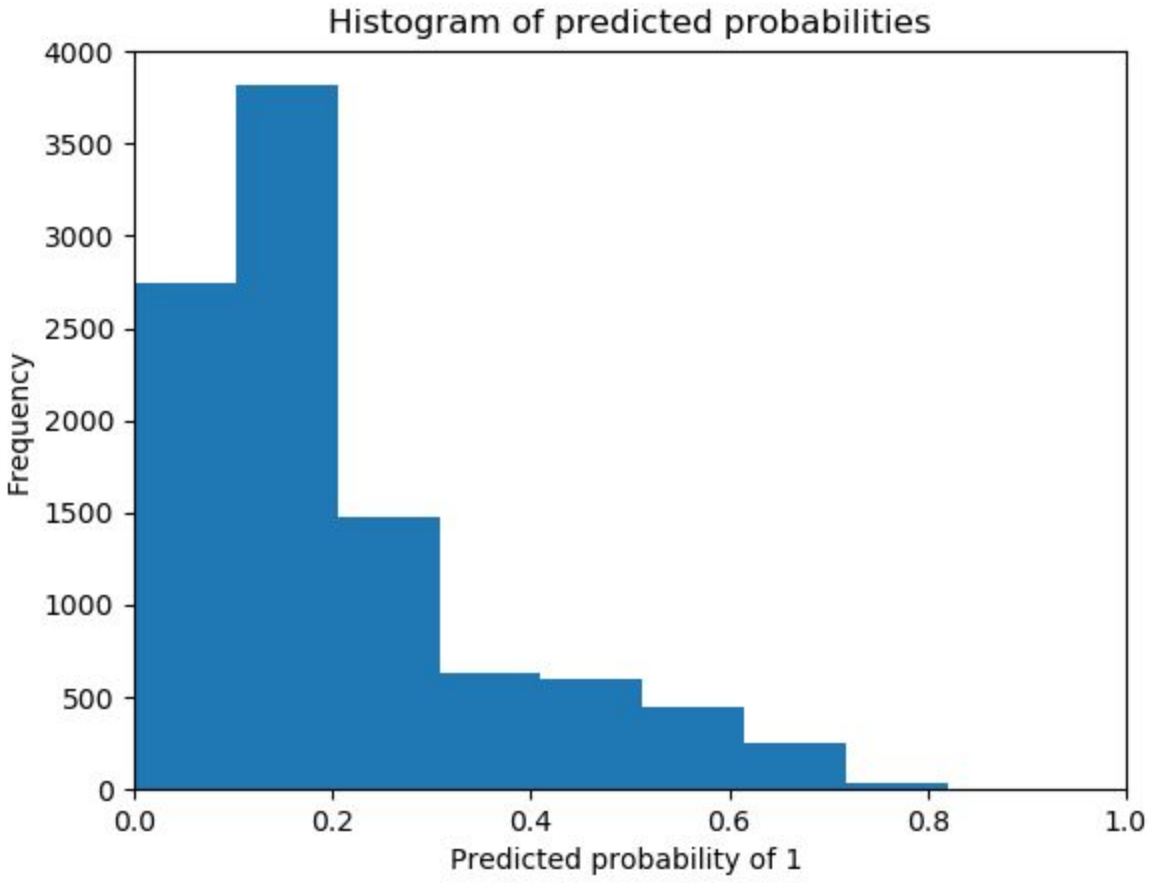
```
from sklearn.metrics import plot_confusion_matrix
# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
            ("Normalized confusion matrix", 'true')]
for title, normalize in titles_options:
  disp = plot_confusion_matrix(rfc, X_test, y_test,
                       display_labels= ['0','1'],
                       cmap=plt.cm.Blues,
                       normalize=normalize)
  disp.ax_.set_title(title)
plt.show()
```





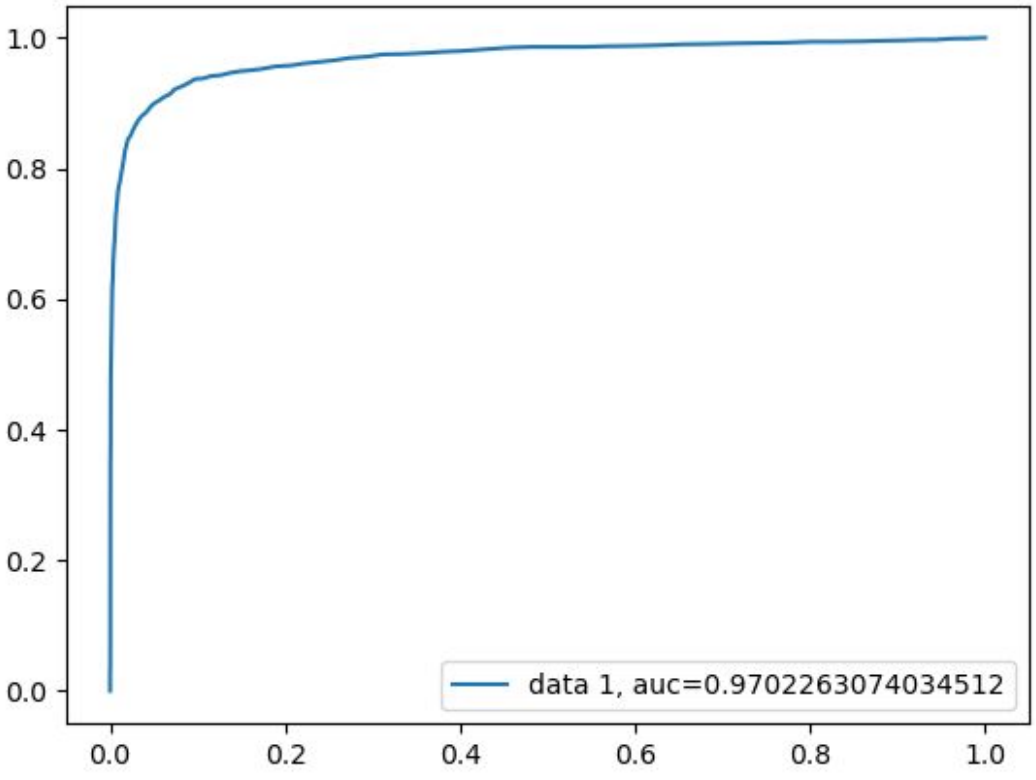

```
y_pred_prob = rfc.predict_proba(X_test)[:, 1]
import matplotlib.pyplot as plt
plt.hist(y_pred_prob, bins=8)
plt.xlim(0,1)
plt.title('Histogram of predicted probabilities')
```

```
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()
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
data 1, auc=0.9702263074034512

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 =

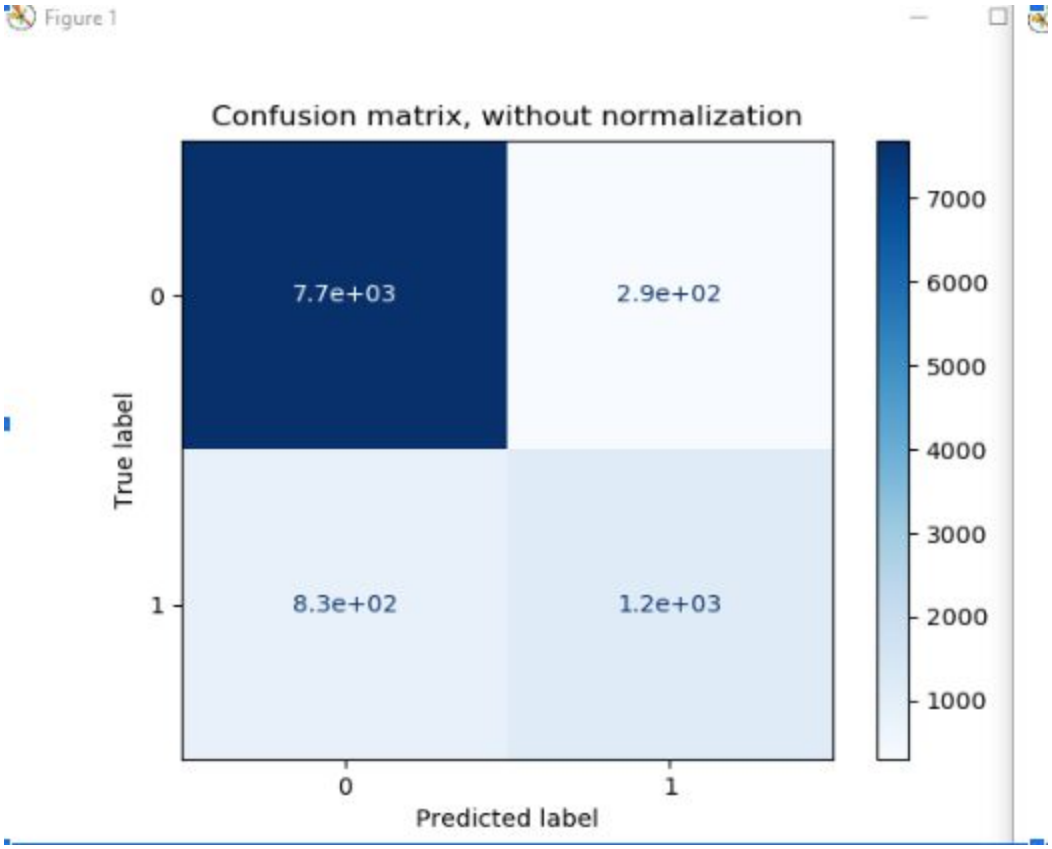
PCA/Logistic Regression = **0.5900** > **0.3900** = Random Forest

F1 score =

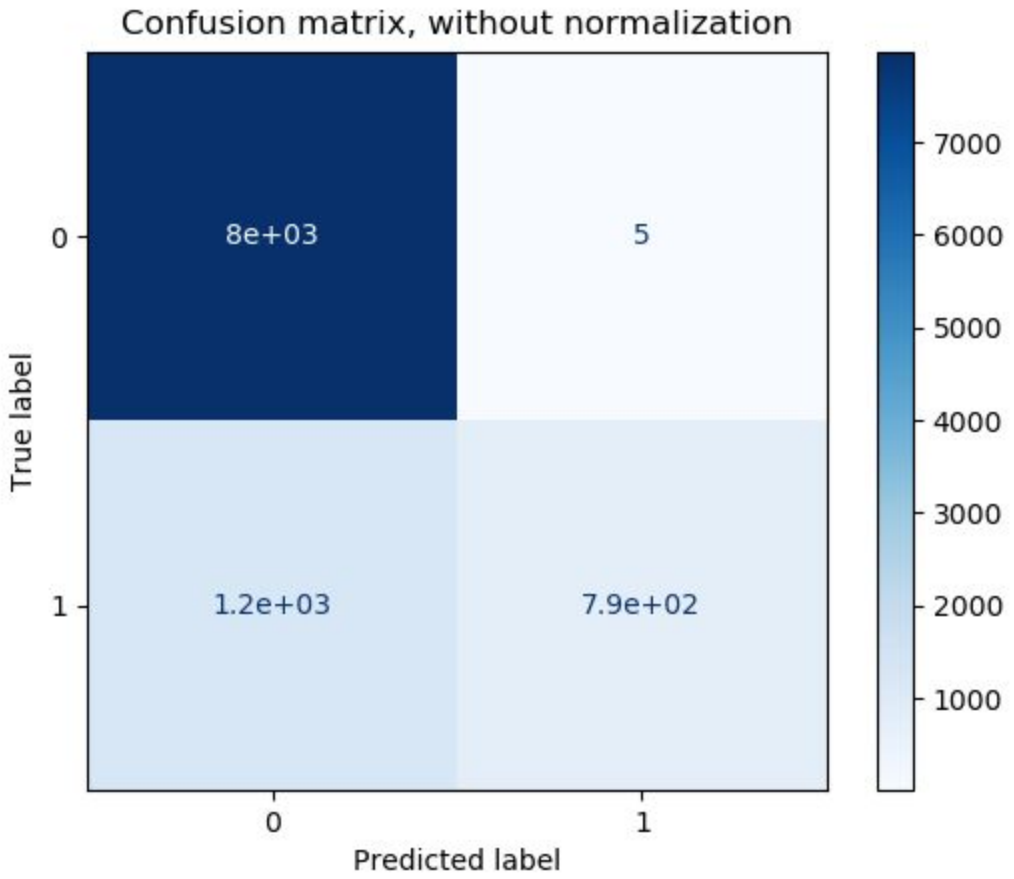
PCA/Logistic for Regression '1' = **0.68** > **0.56** = Random Forest

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 to evaluate it would be F1 score vs ROC AUC. In our case the 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.