Code 1: data_preprocessing.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 22 17:14:19 2018

@author: rickerish_nah
"""

import numpy as np
import pandas as pd
#from sklearn.model_selection import train_test_split #split
from sklearn.cross_validation import train_test_split #shuffle & split
from sklearn.preprocessing import MinMaxScaler

import handle_missing_data as handle
from imblearn.over_sampling import SMOTE

import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns

def data_case_create(data):
   input_data = data.copy()
   #replacing Unknown with NaN_____________________________________________________
   inp_data = input_data.replace(to_replace = ['unknown'], value = np.NaN , regex = True)
   inp_data.pdays = input_data.pdays.replace(to_replace = 999, value = 0 , regex = True)
   #print(inp_data.pdays)
   #print("Ori Size:\n",inp_data.shape)

   # splitting TRAIN and TEST data_______________________________________________
   ip_data,ip_test_data = train_test_split(inp_data, test_size=0.20,
random_state=42,stratify=inp_data.y)

   #print("before:",ip_test_data.shape)
   ip_test_data = ip_test_data.dropna()
   #print("before:",ip_test_data.shape)
   ip_test_data = pd.get_dummies(ip_test_data)
   #print("before:",ip_test_data.shape)
   label_test = ip_test_data.y_yes #.........................................................................Label_IGN
   #print(list(ip_test_data))
   ip_test_data =
ip_test_data.drop(['housing_no','loan_no','default_no','y_no','y_yes'],axis=1)#................DATA_I
GN
   #print("after:",ip_test_data.shape,"\t lab:",label_test.shape)
   #print("Split Shape:\n",ip_data.shape,ip_test_data.shape)

   print("Decription of Numerical data present in the Bank Data Set:\n",ip_data.describe())
   print("Decription of Non-Numeric data present in the Bank Data
Set:\n",ip_data.describe(include = ["object"]))

   pd.DataFrame.hist(ip_data,layout=(3,3),grid = True,bins = 50, color= 'blue') #change color
   his1 = ip_data.select_dtypes(include = "object")
   lio = list(his1)
   for i in lio:
      plt.figure()
      his1[i].value_counts().plot(kind = 'bar',figsize = (10,10))
```

```
        plt.xlabel(i)
        plt.ylabel("Frequency")
        plt.title("Bank Data Set")
    lb = ip_data['y']
    lb = lb.replace(to_replace = ['yes'], value = True , regex = True)
    lb = lb.replace(to_replace = ['no'], value = False , regex = True)
    vb = ip_data.describe().columns
    pd.plotting.scatter_matrix(ip_data[vb],c = lb, figsize = (12,12),cmap = "viridis", alpha=0.3)

    f, ax = plt.subplots(figsize=(10, 8))
    corr = ip_data.corr()
    #sns.heatmap(corr, xticklabels=corr.columns.values,  yticklabels=corr.columns.values)
    sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=True), square=True, ax=ax)


    #Find the columns that have Missing values___________________________________
    col_miss_label = ip_data.columns[ip_data.isna().any()].tolist()
    print("Columns having unknown data in our Bank Dataset:")
    print(col_miss_label,"\n")

    #number of missing values in each column_____________________________________
    col_miss_val = ip_data.isna().sum().tolist()
    print("\nno of unknowns in each column:")
    print(col_miss_val)
    print("End of main")
    #Percentage of missing values
    per_unk=np.zeros(np.size(col_miss_val))
    for i in range(0,np.size(col_miss_val)):
        per_unk[i] = col_miss_val[i]/ip_data.shape[0]
    print("\n% of unknowns present in each column:")
    print(per_unk*100)
    #Since %of Unknown is lesser than expected, we do not remove any feature

    #seperating df based on label C1 and C2______________________________________
    c1_data = ip_data[ip_data.y == 'yes']
    c2_data = ip_data[ip_data.y == 'no']
    #seperating this sub dataset based on known and unknown fature points______
    data_C1_k = c1_data.dropna();data_C1_k = data_C1_k.drop(['y'],axis=1)
#............................#known class 1
    data_C1_unk = c1_data[pd.isnull(c1_data).any(axis=1)];data_C1_unk =
data_C1_unk.drop(['y'],axis=1) #unknown class 1
    data_C2_k = c2_data.dropna();data_C2_k = data_C2_k.drop(['y'],axis=1)
#............................#known class 2
    data_C2_unk = c2_data[pd.isnull(c2_data).any(axis=1)];data_C2_unk =
data_C2_unk.drop(['y'],axis=1) #unknown class 2

    ###___handling Missing data__________________________________________________
    # Type 1 ignoring data with unknown features_______________________________1

    #print("Ignore:::")
    data_ign = ip_data.dropna()
    data_ign = pd.get_dummies(data_ign)
    label_ign = data_ign.y_yes #......................................................................Label_IGN
    data_ign =
data_ign.drop(['housing_no','loan_no','default_no','y_no','y_yes'],axis=1)#...............DATA_IGN
    #print("Size:\nOriginal Train Data set:\t",ip_data.shape[0])
    #print("IGNORE_DATASET:\t",data_ign.shape[0])
    #  Type 2 Handling using MODE impute__________________________________________2
    mode_c1,mode_c2 = handle.handle_using_mode(c1_data,c2_data)

    lc1 = mode_c1.y
    lc2 = mode_c2.y

    data_mode = mode_c1.copy()
    data_mode = data_mode.append([mode_c2])
    #print(list(data_mode))
    label_mode = lc1.copy()
    label_mode = label_mode.append([lc2])
    label_mode = label_mode.replace(to_replace = ['yes'], value = 1 , regex = True)
    label_mode = label_mode.replace(to_replace = ['no'], value = 0 , regex = True)

    data_mode = data_mode.drop(['y'],axis=1)

    data_mode = pd.get_dummies(data_mode)
    data_mode = data_mode.drop(['housing_no','loan_no','default_no'],axis=1)


    ###  Type 3 Handling using SVM impute_________________________________________3
    svm_1,svm_2=handle.handle_using_SVM(data_C1_k,data_C2_k,data_C1_unk,data_C2_unk)

    label1 = pd.DataFrame(0,columns=['y_yes'],index=np.arange(svm_1.shape[0]))
    label2 = pd.DataFrame(1,columns=['y_yes'],index=np.arange(svm_2.shape[0]))

    data_svm = svm_1.append(svm_2)
    label_svm = label1.append(label2)
    data_svm = data_svm.drop(['housing_no','loan_no','default_no'],axis=1)
    print("Size:\nOriginal Train Data set:\t",ip_data.shape[0])
    print("PREDICTIVE MODEL BASED IMPUTATION_DATASET:\t",data_mode.shape[0])
    #feature dimension matching
    l1 = list(data_ign)
    l2 = list(data_mode)
    l3 = list(data_svm)
    lt = list(ip_test_data)
    intr1 = list(set(lt).symmetric_difference(l1))
    intr2 = list(set(lt).symmetric_difference(l2))
    intr3 = list(set(lt).symmetric_difference(l3))
    #print("1:",intr1,"\n2:",intr2,"\n3:",intr3)
    data_ign,data_mode,data_svm = drop_1(data_ign,data_mode,data_svm)


    return
data_ign,label_ign,data_mode,label_mode,data_svm,label_svm,ip_test_data,label_test

def normalize(train,tsst):
    ob=train.copy()
    test=tsst.copy()
    l1 = list(ob)
    #print(l1)
    ob = ob.reset_index()
    ob = ob.drop(['index'],axis=1)
    #print(ob)
    min_max_sc = MinMaxScaler()
    min_max_sc.fit(ob)
    b1 = min_max_sc.transform(ob)
    b2 = min_max_sc.transform(test)
    ob=pd.DataFrame(b1)
    tst=pd.DataFrame(b2)
    #ob = ob.rename(columns=[l1])
    ob.columns = l1
    tst.columns = l1
    #print(ob)
    return ob,tst


def drop_1(ob1,ob2,ob3):
    ## based on lines 105-108
    ob1 = ob1.drop(['education_illiterate','default_yes'],axis =1)
    ob2 = ob2.drop(['education_illiterate','default_yes'],axis =1)
    ob3 = ob3.drop(['education_illiterate','default_yes'],axis =1)
    return ob1,ob2,ob3


def resampling(ob,label):
    obj = ob.copy()
    lab = label.copy()
    sm = SMOTE(random_state=42)
    X_res, y_res = sm.fit_sample(obj,lab)
    #print("IGN res Dataset:\n",X_res.shape,"\nlabl:",len(y_res))
    return X_res,y_res
```

Code 2: handle_missing_data.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 20 16:30:06 2018

@author: rickerish_nah
"""

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from collections import defaultdict
from  sklearn import svm


def handle_using_mode(c1,c2):
    c1_unk = c1.copy()
    c2_unk = c2.copy()

    col1=c1_unk.columns[c1_unk.isna().any()].tolist()
    col2=c2_unk.columns[c2_unk.isna().any()].tolist()
    #print(col1,col2)
    #replace with mode
    c1_unk[col1] = c1_unk[col1].fillna(c1_unk.mode().iloc[0])
    c2_unk[col2] = c2_unk[col2].fillna(c2_unk.mode().iloc[0])


    return c1_unk,c2_unk

def handle_using_SVM(c1,c2,c1_u,c2_u):
    #variable initialization
    c1_unk = c1_u.copy()
    c2_unk = c2_u.copy()
    c1_k = c1.copy()
    c2_k = c2.copy()

    l_op = pd.DataFrame({'A' : []})
    l_op2 = pd.DataFrame({'A' : []})# empty dataframe
    #labels
    label1 = c1_unk.columns[c1_unk.isna().any()].tolist()
    label2 = c2_unk.columns[c2_unk.isna().any()].tolist() #here same as label 1
    #lables with the entire column of data
    l_val=c1[label1].copy();l_v = list(l_val)
#__________________________________________LABEL
    l_val2=c2[label2].copy();l_v2 = list(l_val2)
    l_op = pd.DataFrame({'A' : []})
    l_op2 = pd.DataFrame({'A' : []})# empty dataframe
    ###################################################
    #make copy before deletion:                       #
    c1_k1 = c1_k.copy();c1_k1 = pd.get_dummies(c1_k1)   #
    c2_k2 = c2_k.copy();c2_k2 = pd.get_dummies(c2_k2)   #
    ###################################################

    c1_k = c1_k.drop(label1,axis=1); c1_k = pd.get_dummies(c1_k)
#______________________TRAIN
    c2_k = c2_k.drop(label2,axis=1); c2_k = pd.get_dummies(c2_k)
    c1_unk = c1_unk.drop(label1,axis=1); c1_unk = pd.get_dummies(c1_unk)
#_____________TEST
    c2_unk = c2_unk.drop(label2,axis=1); c2_unk = pd.get_dummies(c2_unk)
    #creating dictionary
    d = defaultdict(LabelEncoder)
    #label encoding
    fit = l_val.apply(lambda x: d[x.name].fit_transform(x));l1=list(l_val) #label encoding
    fit2 = l_val2.apply(lambda x: d[x.name].fit_transform(x));l2=list(l_val2)
    for i in range(0,fit.shape[1]):
        name = l1[i]
        label = fit.loc[:,l1[i]]
        equal = all(label == 0)
        if(equal==False):
            kernel1 = svm.SVC(C=50,gamma=1,kernel="rbf")
            kernel1.fit(c1_k,label)
            y_test1 = kernel1.predict(c1_unk)
            y_test1 = pd.DataFrame(y_test1)
            y_test1.columns=[name]
            l_op = pd.concat([l_op,y_test1],axis = 1)
        else:
            y_test1 = np.zeros(c1_unk.shape[0])
            y_test1 = pd.DataFrame(y_test1)
            y_test1.columns=[name]
            y_test1 = y_test1.astype(int)
            l_op = pd.concat([l_op,y_test1],axis = 1)
    l_op = l_op.drop(['A'],axis=1)
    l_op=l_op.apply(lambda x: d[x.name].inverse_transform(x))
    c2_k = c2_k.drop(['month_dec'],axis=1)
    for i in range(0,fit2.shape[1]):
        name = l2[i]
        label2 = fit2.loc[:,l2[i]]
        equal2 = all(label2 == 0)
        if(equal2==False):
            kernel2 = svm.SVC(C=50,gamma=1,kernel="rbf")
            kernel2.fit(c2_k,label2)
            y_test2 = kernel2.predict(c2_unk)
            y_test2 = pd.DataFrame(y_test2)
            y_test2.columns=[name]
            l_op2 = pd.concat([l_op2,y_test2],axis = 1)
        else:
            y_test2 = np.zeros(c2_unk.shape[0])
            y_test2 = pd.DataFrame(y_test2)
            y_test2.columns=[name]
            y_test2 = y_test2.astype(int)
            l_op2 = pd.concat([l_op2,y_test2],axis = 1)
    l_op2 = l_op2.drop(['A'],axis=1)
    l_op2=l_op2.apply(lambda x: d[x.name].inverse_transform(x))

    l1 = pd.get_dummies(l_op)
    l2 = pd.get_dummies(l_op2)

    c2_TST=combine_col(c2_unk,l2)
    c1_TST=combine_col(c1_unk,l1)


    c1_k1=column_eq(c1_k1,c2_k2)
    cC1_TEST = column_eq(c1_k1,c1_TST)
    cC2_TEST = column_eq(c2_k2,c2_TST)
    C1 = c1_k1.append(cC1_TEST)
    C2 = c2_k2.append(cC2_TEST)
    return C1,C2


def column_eq(ob1,ob2):
    #print("inside C-eq")
    c1=ob1.shape[1]
    c2=ob2.shape[1]
    obj1=ob1.copy()
    obj2=ob2.copy()
    if(c2>c1):
        temp = obj1.copy()
        obj1 = obj2.copy()
        obj2 = temp.copy()
    c1=obj1.shape[1]
    c2=obj2.shape[1]
    lis1=list(obj1);lis2=list(obj2)
    intr = list(set(lis1).symmetric_difference(lis2))
    #print(intr)
    temp = pd.DataFrame(0,columns=intr,index=np.arange(obj2.shape[0]))
    obj2 = obj2.reset_index()
    obj2=obj2.drop(['index'],axis=1)
    oob1 = pd.concat([obj2,temp],axis=1)

    return oob1

def combine_col(ob1,ob2):
    obj1=ob1.copy()
    obj2=ob2.copy()

    lis=list(obj1)
    obj1=obj1.reset_index()
    obj1=obj1.drop(['index'],axis=1)
    ob = pd.concat([obj1,obj2],axis=1)
    return ob
```

Code 3: classification.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 22 20:39:29 2018

@author: rickerish_nah
"""

import numpy as np
import pandas as pd

import handle_missing_data

from sklearn.model_selection import StratifiedKFold


from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

from sklearn.metrics import auc
import matplotlib.pyplot as plt


def classifier(train,label,test,tl):

    train_data = train.copy()
    train_labels = label.copy()
    data_test = test.copy()
    label_test = tl.copy()

    #train_data = train_data.as_matrix()
    train_labels = np.ravel(train_labels)#train_labels.values#as_matrix()


    #data_test = data_test.as_matrix()
    label_test = np.ravel(label_test)#label_test.values#as_matrix()
    #Used for cross validation
    '''
    #max_it = get_Perc(train_data,train_labels) #Perceptron
    #pp, n_nB = get_KNN(train_data,train_labels) #KNN
    #cC,gaMMa = get_SVM(train_data,train_labels) #SVM-rbf
    #cC,gaMMa = get_SVM(train_data,train_labels) #SVM-linear: for here remove one loop
    #max_ite,hl_n = get_NN(train_data,train_labels) #MLP
    #n_est = get_RF(train_data,train_labels) #Random Forest

    #default to run
    clsr_names=["Perceptron","KNN", "SVM gausian",
     "Decision Tree", "Random Forest", "Neural Net:MLP",
     ]
    classifiers = 
[Perceptron(class_weight='balanced',shuffle=True),KNeighborsClassifier(n_neighbors=n_nB,p=p
p),
    SVC(kernel="poly",C=cC,gamma=gaMMa,probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(n_estimators = n_est),
    MLPClassifier(hidden_layer_sizes=(hl_n,),max_iter=max_ite)]
    '''

    # used after parameter extraction
    clsr_names=["Perceptron","KNN", "SVM Linear","SVM gausian", "SVM Poly","Decision Tree", 
"Random Forest", "Neural Net:MLP","Naive Bayes"]


    classifiers = 
[Perceptron(class_weight='balanced',shuffle=True),KNeighborsClassifier(n_neighbors=6,p=1),
    SVC(kernel="linear",probability=True, C = 13.73),
    SVC(kernel="rbf",probability=True, C=0.021544, gamma = 0.004641),
    SVC(kernel="poly",probability=True, C = 215, gamma = 4.356, degree = 5),
    DecisionTreeClassifier(),
    RandomForestClassifier(n_estimators = 14),
    MLPClassifier(hidden_layer_sizes=(160,),max_iter=200),
    GaussianNB()]
    fpr = {}
    tpr = {}
    r_a = {}

    print("Comparing different models:\n")

    for name, clf in zip(clsr_names, classifiers):
        if (name == 'Perceptron'):
            model=clf.fit(train_data,train_labels)
            y_pred=model.predict(data_test)

            #F score report
            fr = classification_report(label_test,y_pred,)
            #using ROC_AUC score
            roc_auc = roc_auc_score(label_test,y_pred)

            print("-",name,"-")
            print("F Measure Report:\n",fr)
            print("roc_auc:\t",roc_auc)
            print("\n\n")

        else:
            model=clf.fit(train_data,train_labels)
```

```
      y_pred=model.predict(data_test)
      #
      fr = classification_report(label_test,y_pred,)
      #using ROC_AUC score
      prob_ = model.predict_proba(data_test)
      fpr_, tpr_, thresholds = roc_curve(label_test, prob_[:,1])
      fpr[name] = fpr_
      tpr[name] = tpr_
      #fpr = np.append(fpr,fpr_)
      #tpr = np.append(tpr,tpr_)
      r_a_ = auc(fpr_, tpr_)
      r_a[name] = r_a_
      print("Using-"+str(name),"-")
      print("F measure Report:\n",fr)
      print("ROC_Curve:\t",r_a_)
      plt.figure()
      plt.plot(fpr_, tpr_,lw=2, color = 'red',label='ROC curve (area = %0.2f)' % r_a_)
      plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver operating characteristic: '+str(name))
      plt.legend()
      plt.show()
      print("\n\n")
  plt.figure()
  clsr_names1=["KNN", "SVM Linear","SVM gausian", "SVM Poly","Decision Tree", "Random Forest", "Neural Net:MLP","Naive Bayes"]
  for i in clsr_names1:
    plt.plot(fpr[i], tpr[i],lw=2,label='%s ROC curve (area = %0.2f)' % (i,r_a[i]))
  plt.plot([0, 1], [0, 1], lw=lw, linestyle='--')
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.title('Receiver operating characteristic: MODE_IMPUTED_DATASET')
  plt.legend()
  plt.show()



  return 0

def get_SVM(train_data,train_labels):

  iter = 50
  ACC=np.zeros((iter))

  cC=np.logspace(-3,3, num=50,endpoint = True)
  gGamma=np.logspace(-3,3,num=50,endpoint = True)

  ACC = np.zeros((iter,iter))
  DEV = np.zeros((iter,iter))
  max_ACC=0
  max_C=0
  max_Gamma=0
  max_STD=0

  for i in range(0,iter):
    print("#",i)
```

```
        param_accuracy=[]
        for j in range(0,iter):
            c=cC[i]
            g=gGamma[j]
            num_splits=5
            s_kFold = StratifiedKFold(n_splits = num_splits,shuffle = True )
            #s_kFold.get_n_splits(train,label)

            fold_accuracy = []

            for train_index, test_index in s_kFold.split(train_data,train_labels):
                X_Train, X_Test = train_data[train_index], train_data[test_index]
                Y_Train, Y_Test = train_labels[train_index], train_labels[test_index]

                kernel = SVC(C=c,gamma=g,kernel="linear")
                kernel.fit(X_Train,Y_Train)
                #Accuracy Score
                prediction =  kernel.predict(X_Test)
                fold_accuracy = np.append(fold_accuracy,accuracy_score(Y_Test,prediction))

            ACC[i,j]=np.mean(fold_accuracy)

            DEV[i,j]=np.std(fold_accuracy)
            #print("Size:",accuracy.shape[0])
            if(max_ACC<ACC[i,j]):
                max_ACC=ACC[i,j]
                max_C=c
                max_Gamma=g
                max_STD=DEV[i,j]

    print("Maximum Training Accuracy:",max_ACC*100,"%\nfor C:",max_C,"\nfor
Gamma:",max_Gamma,"\nStd.Dev:",max_STD)
    return max_C,max_Gamma

def get_KNN(train_data,train_labels):
    iter_n = 20 #-gamma
    iter_p = 2  #-C
    N = range(1,21)
    P = range(1,3)
    ACC = np.zeros((iter_p,iter_n))
    DEV = np.zeros((iter_p,iter_n))
    max_ACC=0
    max_C=0
    max_Gamma=0
    max_STD=0
    for i in range(0,iter_p):
        print("#",i)
        for j in range(0,iter_n):
            print("#*",j)
            pp=P[i]
            n=N[j]
            num_splits=5
            s_kFold = StratifiedKFold(n_splits = num_splits,shuffle = True )
            #s_kFold.get_n_splits(train,label)

            fold_accuracy = []

            for train_index, test_index in s_kFold.split(train_data,train_labels):
                X_Train, X_Test = train_data[train_index], train_data[test_index]
                Y_Train, Y_Test = train_labels[train_index], train_labels[test_index]

                kernel = KNeighborsClassifier(n_neighbors=n,p=pp)
                kernel.fit(X_Train,Y_Train)
                #Accuracy Score
                prediction =  kernel.predict(X_Test)
                fold_accuracy = np.append(fold_accuracy,accuracy_score(Y_Test,prediction))

            ACC[i,j]=np.mean(fold_accuracy)

            DEV[i,j]=np.std(fold_accuracy)
            #print("Size:",accuracy.shape[0])
            if(max_ACC<ACC[i,j]):
                max_ACC=ACC[i,j]
                max_C=pp
                max_Gamma=n
                max_STD=DEV[i,j]
    print("Maximum Training Accuracy:",max_ACC*100,"%\nfor norm (p):",max_C,"\nfor
#neighbors (n):",max_Gamma,"\nStd.Dev:",max_STD)

    return max_C, max_Gamma
def get_NN(train_data,train_labels):


    hl1 = range(10,101)
    hl = [i*10 for i in hl1]
    ite1 = range(2,5)
    ite = [i*100 for i in ite1]
    iter_p = len(ite)  #-C
    iter_n = len(hl)

    ACC = np.zeros((iter_p,iter_n))
    DEV = np.zeros((iter_p,iter_n))
    max_ACC=0
    max_C=0
    max_Gamma=0
    max_STD=0
     #-gamma
    for i in range(0,iter_p):
        print("#",i)
        for j in range(0,iter_n):
            print("#*",j)
            pp=ite[i]
            n=hl[j]
            num_splits=5
            s_kFold = StratifiedKFold(n_splits = num_splits,shuffle = True )
            #s_kFold.get_n_splits(train,label)

            fold_accuracy = []

            for train_index, test_index in s_kFold.split(train_data,train_labels):
                X_Train, X_Test = train_data[train_index], train_data[test_index]
                Y_Train, Y_Test = train_labels[train_index], train_labels[test_index]

                kernel = MLPClassifier(hidden_layer_sizes=(n,), max_iter=pp )
                kernel.fit(X_Train,Y_Train)
                #Accuracy Score
            prediction =  kernel.predict(X_Test)
            fold_accuracy = np.append(fold_accuracy,accuracy_score(Y_Test,prediction))

         ACC[i,j]=np.mean(fold_accuracy)

         DEV[i,j]=np.std(fold_accuracy)
         #print("Size:",accuracy.shape[0])
         if(max_ACC<ACC[i,j]):
            max_ACC=ACC[i,j]
            max_C=pp
            max_Gamma=n
            max_STD=DEV[i,j]
   print("Maximum Training Accuracy:",max_ACC*100,"%\nfor #max_iter (p):",max_C,"\nfor
#hidden_layers (n):",max_Gamma,"\nStd.Dev:",max_STD)


   return max_C,max_Gamma
def get_RF(train_data,train_labels):

   n_o = range(1,21)
   itera = len(n_o)  #-C

   ACC = np.zeros(itera)
   DEV = np.zeros(itera)
   max_ACC=0
   max_C=0
   max_STD=0
    #-gamma
   for i in range(0,itera):
      print("#",i)
      n=n_o[i]
      num_splits=5
      s_kFold = StratifiedKFold(n_splits = num_splits,shuffle = True )
      #s_kFold.get_n_splits(train,label)
      fold_accuracy = []

      for train_index, test_index in s_kFold.split(train_data,train_labels):
         X_Train, X_Test = train_data[train_index], train_data[test_index]
         Y_Train, Y_Test = train_labels[train_index], train_labels[test_index]

         kernel = RandomForestClassifier(n_estimators = n)
         kernel.fit(X_Train,Y_Train)
         #Accuracy Score
         prediction =  kernel.predict(X_Test)
         fold_accuracy = np.append(fold_accuracy,accuracy_score(Y_Test,prediction))

      ACC[i]=np.mean(fold_accuracy)

      DEV[i]=np.std(fold_accuracy)
      #print("Size:",accuracy.shape[0])
      if(max_ACC<ACC[i]):
         max_ACC=ACC[i]
         max_C=n
         max_STD=DEV[i]
   print("Maximum Training Accuracy:",max_ACC*100,"%\nfor #estimators
(p):",max_C,"\nStd.Dev:",max_STD)
   return max_C #n_est
```

Code 4: project_tr1.py (Main Program)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 14 13:10:50 2018

@author: rickerish_nah
Name: Harikrishna Prabhu
USC ID: 3333077042
email ID: hrapbhu@usc.edu

"""

import numpy as np
import pandas as pd
import handle_missing_data as handle
import data_preprocessing as dP
import classification as classify

from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler


print("Initiating the System.....\nPlease Wait...")
#########################______________MAIN__Program_________################
#############
#read data
input_read_data=pd.read_csv("/Users/rickerish_nah/Documents/trial/final project/bank-
additional.csv", delimiter=',')
input_data = pd.DataFrame(data=input_read_data)
#input_data = input_data.drop(['default'],axis=1)

###Preprocessing input data___________________________________________________1
print("Stage 1: Data Preprocessing...")
data_ign,label_ign,data_mode,label_mode,data_svm,label_svm,data_test,label_test =
dP.data_case_create(input_data)



#now all missing values are deduced and few features are dropped

###Normalizing features_______________________________________________________2
data_ign,test_ign = dP.normalize(data_ign,data_test)
data_mode,test__mode = dP.normalize(data_mode,data_test)
data_svm,test_svm = dP.normalize(data_svm,data_test)

label_ign = label_ign.reset_index()
label_ign = label_ign.drop(['index'],axis=1)

label_mode = label_mode.reset_index()
label_mode = label_mode.drop(['index'],axis=1)

label_svm = label_svm.reset_index()
label_svm = label_svm.drop(['index'],axis=1)

label_test = label_test.reset_index()
label_test = label_test.drop(['index'],axis=1)

#data_mode,test__mode = dP.normalize(data_mode,data_test)
#data_svm,test_svm = dP.normalize(data_svm,data_test)


print("Done...\nData sizes:\nIGNORE_DATASET:",data_ign.shape,"\t LABEL:",label_ign.shape)
print("MODE_DATASET:",data_mode.shape,"\t LABEL:",label_mode.shape)
print("SVM_DATASET:",data_svm.shape,"\t LABEL:",label_svm.shape)
print("TEST_DATASET:",data_test.shape,"\t LABEL:",label_test.shape)

# SET 1:  Original Datasets

print("Ignore Dataset:")
f=classify.classifier(data_ign,label_ign,test_ign,label_test)

#print("Mode Impute Dataset:")
f=classify.classifier(data_mode,label_mode,test__mode,label_test)


print("SVM Imputed Dataset:")
f=classify.classifier(data_svm,label_svm,test_svm,label_test)

print("Before Balancing the Data:")
print(data_mode.shape,test__mode.shape)
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_sample(data_mode,label_mode)
X_res_tst, y_res_tst = sm.fit_sample(test__mode,label_test)
print("After Balancing the Data Over-Sampling:")
print(X_res.shape,X_res_tst.shape)
#print("Mode Impute Dataset:")
f=classify.classifier(X_res, y_res,X_res_tst, y_res_tst)

rus = RandomUnderSampler()

X_resampled, y_resampled = rus.fit_sample(data_mode,label_mode)
X_resampled_tst, y_resampled_tst = rus.fit_sample(test__mode,label_test)
print("After Balancing the Data Under-Sampling:")
print(X_resampled.shape,X_resampled_tst.shape)
f=classify.classifier(X_resampled, y_resampled,X_resampled_tst, y_resampled_tst)

#to check if feature dimenaionality works
'''
print("After Feature reduction: ")

for i in range(1,20):
    f_s = SelectKBest(k=i)
    f_s.fit(data_mode,label_mode)
    train = f_s.transform(data_mode)
    test = f_s.transform(test__mode)
    kernel = RandomForestClassifier(n_estimators = 14)
    kernel.fit(train,label_mode)
    prob_ = kernel.predict_proba(test)
    fpr_, tpr_, thresholds = roc_curve(label_test, prob_[:,1])
    r_a_ = auc(fpr_, tpr_)
```

```
print("For ",i," reduced features, The AUC is: ",r_a_)
'''
```