```
import pandas
import missingno as mno
import matplotlib.pyplot as plt
from sklearn import linear_model
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.mixture import BayesianGaussianMixture
import joblib
from sklearn.preprocessing import PolynomialFeatures

f_handle = pandas.read_csv('credit_train.csv')
N = f_handle.shape[0]
D = f_handle.shape[1]




                                        ### Split the data
# split the data into train and test first    ---> 10% test and 90% training
f_handle.drop(f_handle.tail(514).index, inplace=True)
X_pre = f_handle.iloc[:N//20, :]
X_test = f_handle.iloc[N//20:int((1.5*N)//10), :]
X_test.to_csv('X_test_data.csv')
X_train = f_handle.iloc[int((1.5*N)//10):, :]
name_lst = f_handle.columns






                                         ### PREPROCESSING
# inspect the training data
print(len(f_handle))
# get the feature names
name_lst = f_handle.columns
#print(name_lst)
# get rid of irrelevant information such as Loan ID and Customer ID
X_train = X_train.iloc[:, 2:]
# Construct y_train
```

```
y_train = X_train.iloc[:, 0]
X_train = X_train.iloc[:, 1:]
print(X_train.dtypes)


# check for missing data number
#print(X_train.isnull().sum())
#print(y_train.isnull().sum()) # there is no missing data in y label
#mno.matrix(X_train, figsize=(20, 20))
#plt.show()


# ways to deal with missing data
# convert categorical data to numeric
# count each category
# categorical column = ['Term', 'Years in current job?', 'Home Ownership', 'Purpose']
#print(X_train['Term'].value_counts(), X_train['Years in current job'].value_counts(),
      #X_train['Home Ownership'].value_counts(), X_train['Purpose'].value_counts())
cleanup_data = {'Term': {'Short Term': -1, 'Long Term': 1},
                'Years in current job': {'< 1 year': 0, '1 year': 1, '2 years': 2, '3 years': 3, '4 years': 4,
                                         '5 years': 5, '6 years': 6, '7 years': 7, '8 years': 8, '9 years': 9, '10+ years': 10},
                'Home Ownership': {'Rent': 1, 'Own Home': 2, 'HaveMortgage': 3, 'Home Mortgage': 4}}
X_train.replace(cleanup_data, inplace=True)
#print(X_train.dtypes)

# drop purpose columns, two many categories, hard to convert
#X_train['Months since last delinquent'] = X_train['Months since last delinquent'].replace('np.nan', 0)
X_train = X_train.drop(['Purpose', 'Months since last delinquent'], axis=1)
#X_train['Years in current job'] = X_train['Years in current job'].astype(float)
#print(X_train.dtypes)
# NA in months since last delinquent means not late for every payment
missing_columns = ['Credit Score', 'Annual Income', 'Years in current job', 'Bankruptcies', 'Maximum Open Credit',
                   'Tax Liens']
# deterministic regression imputation


'''def random_imputation(df, feature):
    number_missing = df[feature].isnull().sum()
    observed_values = df.loc[df[feature].notnull(), feature]
    df.loc[df[feature].isnull(), feature + '_imp'] = np.random.choice(observed_values,
number_missing, replace=True)
    return df



for feature in missing_columns:
    X_train[feature + '_imp'] = X_train[feature]
    X_train = random_imputation(X_train, feature)'''
X_train_para = {}
deter_data = pandas.DataFrame(columns=['Det' + name for name in missing_columns])
for feature in missing_columns:
    deter_data['Det'+feature] = X_train[feature]
    parameters = list(set(X_train.columns) - set(missing_columns) - {feature})
    X_train_para[feature] = parameters
    model = linear_model.LinearRegression()
    X_train_drop =
X_train[parameters].drop(X_train[feature].index[X_train[feature].apply(np.isnan)])
    model.fit(X=X_train_drop,
y=X_train[feature].drop(X_train[feature].index[X_train[feature].apply(np.isnan)]))
    filename = '{}_data.sav'.format(feature)
    joblib.dump(model, filename) # save prediction model for test data
    deter_data.loc[X_train[feature].isnull(), 'Det' + feature] = \
        model.predict(X_train[parameters])[X_train[feature].isnull()]
# finish filling missing data put predict values back to X_train
print(X_train_para)
for feature in missing_columns:
    X_train[feature] = deter_data['Det' + feature]
#print(X_train['Credit Score'])
#mno.matrix(X_train, figsize = (20,16))
#plt.show()

# assign Charged Off as 0, Fully Paid as 1
label_cleanup = {'Fully Paid': 1, 'Charged Off': 0}
y_train.replace(label_cleanup, inplace=True)

# normalization
# normalize all numeric data
col_min = {}
col_max = {}
cols_to_norm = ['Current Loan Amount', 'Credit Score', 'Annual Income', 'Monthly Debt',
'Current Credit Balance',
                'Maximum Open Credit']
# store max and min for columns that need to be normalized for test data
for col in cols_to_norm:
    col_min[col] = X_train[col].min()
    col_max[col] = X_train[col].max()
X_train[cols_to_norm] = X_train[cols_to_norm].apply(lambda x: (x - x.min())/(x.max()-x.min()))
print(col_min)
print(col_max)
                                        ### Training
                    ### Now we are ready to test out different machine learning algorithms

# in mind: logistic regression, random forest, EM, KNN
# cross validation
kf = StratifiedKFold(n_splits=2)
EM_error = 0
iteration = 1
lr_para = ['none', 'l2']
for lr in lr_para:
    lr_error = 0
    for train_index, test_index in kf.split(X_train, y_train):
        X_cv_train, X_cv_test = X_train.iloc[train_index], X_train.iloc[test_index]
        y_cv_train, y_cv_test = y_train.iloc[train_index], y_train.iloc[test_index]

        # logistic regression
        lr_clf = LogisticRegression(penalty=lr, max_iter=2500)
        lr_clf.fit(X_cv_train, y_cv_train)
        y_pred = lr_clf.predict(X_cv_test)
        y_cv_test_lst = y_cv_test.values.tolist()

        error = 0
        for i in range(len(y_pred)):
            if y_pred[i] != y_cv_test_lst[i]:
                error += 1
        #print('logistic regression accuracy {}%'.format((1-error/len(y_pred))*100))
        lr_error += error
    print('parameter {}, logistic regression average error {}%, accuracy is {}%'.format(lr, lr_error / 2 / len(y_pred) * 100, (1 - (lr_error / 2 / len(y_pred))) * 100))

#rf_max_depth = [500, 250, 100, 50]
rf_max_depth = [50]
for depth in rf_max_depth:
    rf_error = 0
    for train_index, test_index in kf.split(X_train, y_train):
        X_cv_train, X_cv_test = X_train.iloc[train_index], X_train.iloc[test_index]
        y_cv_train, y_cv_test = y_train.iloc[train_index], y_train.iloc[test_index]
        # random forest
        error = 0
        rf_clf = RandomForestClassifier(n_estimators=depth)
        rf_clf.fit(X_cv_train, y_cv_train)
        y_pred = rf_clf.predict(X_cv_test)
        y_cv_test_lst = y_cv_test.values.tolist()
        for i in range(len(y_pred)):
            if y_pred[i] != y_cv_test_lst[i]:
                error += 1
        rf_error += error
        #print('random forest accuracy {}%'.format((1-error/len(y_pred))*100))
    print('parameter {}, random forest average error {}%, accuracy is {}%'.format(depth, rf_error
/ 2 / len(y_pred) * 100, (1 - (rf_error / 2 / len(y_pred))) * 100))
plt.figure()
plt.scatter(X_cv_train['Credit Score'], lr_clf.predict_proba(X_cv_train)[:, 1])
plt.scatter(X_cv_train['Credit Score'], rf_clf.predict_proba(X_cv_train)[:, 1])
plt.legend(('logistic regression', 'ramdom forest'))
plt.show()
knn_para = [5, 10, 15, 20, 25, 50, 100]
for knn in knn_para:
    knn_error = 0
    for train_index, test_index in kf.split(X_train, y_train):
        X_cv_train, X_cv_test = X_train.iloc[train_index], X_train.iloc[test_index]
        y_cv_train, y_cv_test = y_train.iloc[train_index], y_train.iloc[test_index]
        # k nearest neighbor
        error_knn = 0
        neigh = KNeighborsClassifier(n_neighbors=knn)
        neigh.fit(X_cv_train, y_cv_train)
        y_pred = neigh.predict(X_cv_test)
        y_cv_test_lst = y_cv_test.values.tolist()
        for i in range(len(y_pred)):
            if y_pred[i] != y_cv_test_lst[i]:
                error_knn += 1
        knn_error += error_knn
        #print('k nearest neighbor accuracy {}%'.format((1-error_knn/len(y_pred))*100))
    print('parameter {}, k nearest neighbor average error {}%, accuracy is {}%'.format(knn,
knn_error / 2 / len(y_pred) * 100, (1-(knn_error/2/len(y_pred)))*100 ))


for train_index, test_index in kf.split(X_train, y_train):
    X_cv_train, X_cv_test = X_train.iloc[train_index], X_train.iloc[test_index]
    y_cv_train, y_cv_test = y_train.iloc[train_index], y_train.iloc[test_index]
    # Gaussian Mixture Model
    error_GM = 0
    g = BayesianGaussianMixture(n_components=2)
    g.fit(X_cv_train, y_cv_train)
    y_pred = g.predict(X_cv_test)
    y_cv_test_lst = y_cv_test.values.tolist()
    for i in range(len(y_pred)):
        if y_pred[i] != y_cv_test_lst[i]:
            error_GM += 1
    EM_error += error_GM
    #print('EM estimator accuracy {}%'.format((1-error_GM/len(y_pred))*100))

#print('\n\nlogisitc regression average error {}%, accuracy is
{}%'.format(lr_error/5/len(y_pred)*100, (1-(lr_error/5/len(y_pred)))*100 ))
#print('random forest average error {}%, accuracy is {}%'.format(rf_error/5/len(y_pred)*100, (1-
(rf_error/5/len(y_pred)))*100 ))
#print('k nearest neighbor average error {}%, accuracy is
{}%'.format(knn_error/5/len(y_pred)*100, (1-(knn_error/5/len(y_pred)))*100 ))
print('EM estimator average error {}%, accuracy is {}%'.format(EM_error/2/len(y_pred)*100, (1-
(EM_error/2/len(y_pred)))*100 ))
joblib.dump(lr_clf, filename='logistic regression_model.sav')
joblib.dump(rf_clf, filename='random forest_model.sav')
joblib.dump(neigh, filename='k nearest neighbor_model.sav')
joblib.dump(g, filename='EM estimation_model.sav')


                                        ### TEST
# random forest and logistic regression has the comparable accuracy, but logistic regression
runs faster
# Preprocessing using parameters from x_train
X_test = pandas.read_csv('X_test_data.csv')
X_test = X_test.iloc[:, 3:]
X_test = X_test.drop(['Purpose', 'Months since last delinquent'], axis=1)
# Construct y_train
y_test = X_test.iloc[:, 0]
X_test = X_test.iloc[:, 1:]

# replace categorical data with numeric values
X_test.replace(cleanup_data, inplace=True)

# fill in missing data according to the parameters from training
missing_columns = ['Credit Score', 'Annual Income', 'Years in current job', 'Bankruptcies',
'Maximum Open Credit',
                   'Tax Liens']
for feature in missing_columns:
    filesaved = '{}_data.sav'.format(feature)
    loaded_model = joblib.load(filesaved)
    X_test.loc[X_test[feature].isnull(), feature] = \
        loaded_model.predict(X_test[X_train_para[feature]])[X_test[feature].isnull()]

# normalize test according to training parameters
cols_to_norm = ['Current Loan Amount', 'Credit Score', 'Annual Income', 'Monthly Debt',
'Current Credit Balance',
                'Maximum Open Credit']
for col in cols_to_norm:
    X_test[col] = X_test[col].apply(lambda x: (x - col_min[col])/(col_max[col]-col_min[col]))

# assign y_test to 0 and 1 for comparison
label_cleanup = {'Fully Paid': 1, 'Charged Off': 0}
y_test.replace(label_cleanup, inplace=True)

## load prediction model: logistic regression
y_pred = lr_clf.predict(X_test)
y_test_lst = y_test.values.tolist()
error = 0
for i in range(len(y_pred)):
    if y_pred[i] != y_test_lst[i]:
        error += 1
#print(error)
print('\nlogistic regression accuracy on test set {}%'.format((1 - error / len(y_pred)) * 100))

                            ### predict the test set data
X_test = pandas.read_csv('credit_test.csv')
X_test.drop(X_test.tail(353).index, inplace=True)
X_test = X_test.iloc[:, 2:18]
X_test = X_test.drop(['Purpose', 'Months since last delinquent'], axis=1)
cleanup_data = {'Term': {'Short Term': -1, 'Long Term': 1},
                'Years in current job': {'< 1 year': 0, '1 year': 1, '2 years': 2, '3 years': 3, '4
years': 4,
                                         '5 years': 5, '6 years': 6, '7 years': 7, '8 years':
8, '9 years': 9, '10+ years': 10},
                'Home Ownership': {'Rent': 1, 'Own Home': 2, 'HaveMortgage': 3, 'Home
Mortgage': 4}}
X_test.replace(cleanup_data, inplace=True)
missing_columns = ['Credit Score', 'Annual Income', 'Years in current job', 'Bankruptcies',
'Maximum Open Credit',
                   'Tax Liens']
for feature in missing_columns:
    filesaved = '{}_data.sav'.format(feature)
    loaded_model = joblib.load(filesaved)
    X_test.loc[X_test[feature].isnull(), feature] = \
            loaded_model.predict(X_test[X_train_para[feature]])[X_test[feature].isnull()]

# normalize test according to training parameters
cols_to_norm = ['Current Loan Amount', 'Credit Score', 'Annual Income', 'Monthly Debt',
'Current Credit Balance',
                        'Maximum Open Credit']
for col in cols_to_norm:
    X_test[col] = X_test[col].apply(lambda x: (x - col_min[col])/(col_max[col]-col_min[col]))
y_pred = lr_clf.predict(X_test)
y_pred = pandas.DataFrame(y_pred)
X_test_write = pandas.read_csv('credit_test.csv')
X_test_write['Loan Status'] = y_pred
X_test_write.to_csv('credit_test.csv', index=False)




######################Main.py
import pandas
import missingno as mno
import matplotlib.pyplot as plt
from sklearn import linear_model
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.mixture import BayesianGaussianMixture
import joblib
from sklearn.preprocessing import PolynomialFeatures

X_test = pandas.read_csv('credit_test.csv')
X_test.drop(X_test.tail(353).index, inplace=True)
X_test = X_test.iloc[:, 2:18]
X_test = X_test.drop(['Purpose', 'Months since last delinquent'], axis=1)
cleanup_data = {'Term': {'Short Term': -1, 'Long Term': 1},
                        'Years in current job': {'< 1 year': 0, '1 year': 1, '2 years': 2, '3 years': 3, '4
years': 4,
                                                             '5 years': 5, '6 years': 6, '7 years': 7, '8 years':
8, '9 years': 9, '10+ years': 10},
                        'Home Ownership': {'Rent': 1, 'Own Home': 2, 'HaveMortgage': 3, 'Home
Mortgage': 4}}
X_test.replace(cleanup_data, inplace=True)
```

```
missing_columns = ['Credit Score', 'Annual Income', 'Years in current job', 'Bankruptcies',
'Maximum Open Credit',
                         'Tax Liens']
X_train_para = {'Credit Score': ['Current Credit Balance', 'Number of Credit Problems',
'Monthly Debt', 'Term', 'Number of Open Accounts', 'Home Ownership', 'Years of Credit
History', 'Current Loan Amount'], 'Annual Income': ['Current Credit Balance', 'Number of
Credit Problems', 'Monthly Debt', 'Term', 'Number of Open Accounts', 'Home Ownership',
'Years of Credit History', 'Current Loan Amount'], 'Years in current job': ['Current Credit
Balance', 'Number of Credit Problems', 'Monthly Debt', 'Term', 'Number of Open Accounts',
'Home Ownership', 'Years of Credit History', 'Current Loan Amount'], 'Bankruptcies': ['Current
Credit Balance', 'Number of Credit Problems', 'Monthly Debt', 'Term', 'Number of Open
Accounts', 'Home Ownership', 'Years of Credit History', 'Current Loan Amount'], 'Maximum
Open Credit': ['Current Credit Balance', 'Number of Credit Problems', 'Monthly Debt', 'Term',
'Number of Open Accounts', 'Home Ownership', 'Years of Credit History', 'Current Loan
Amount'], 'Tax Liens': ['Current Credit Balance', 'Number of Credit Problems', 'Monthly Debt',
'Term', 'Number of Open Accounts', 'Home Ownership', 'Years of Credit History', 'Current
Loan Amount']}
col_min = {'Current Loan Amount': 10802.0, 'Credit Score': 585.0, 'Annual Income': 76627.0,
'Monthly Debt': 0.0, 'Current Credit Balance': 0.0, 'Maximum Open Credit': -
756539.8799273572}
col_max = {'Current Loan Amount': 99999999.0, 'Credit Score': 7510.0, 'Annual Income':
165557393.0, 'Monthly Debt': 435843.28, 'Current Credit Balance': 32878968.0, 'Maximum
Open Credit': 1539737892.0}
for feature in missing_columns:
    filesaved = '{}_data.sav'.format(feature)
    loaded_model = joblib.load(filesaved)
    X_test.loc[X_test[feature].isnull(), feature] = \
        loaded_model.predict(X_test[X_train_para[feature]])[X_test[feature].isnull()]

# normalize test according to training parameters
cols_to_norm = ['Current Loan Amount', 'Credit Score', 'Annual Income', 'Monthly Debt',
'Current Credit Balance',
                      'Maximum Open Credit']
for col in cols_to_norm:
    X_test[col] = X_test[col].apply(lambda x: (x - col_min[col])/(col_max[col]-col_min[col]))
lr_clf = joblib.load('logistic regression_model.sav')
y_pred = lr_clf.predict(X_test)
y_pred = pandas.DataFrame(y_pred)
X_test_write = pandas.read_csv('credit_test.csv')
X_test_write['Loan Status'] = y_pred
X_test_write.to_csv('credit_test.csv', index=False)
```

### TEST created from credit_train

```
# random forest and logistic regression has the comparable accuracy, but logistic regression runs faster
# Preprocessing using parameters from x_train
X_test = pandas.read_csv('X_test_data.csv')
X_test = X_test.iloc[:, 3:]
X_test = X_test.drop(['Purpose', 'Months since last delinquent'], axis=1)
# Construct y_train
y_test = X_test.iloc[:, 0]
X_test = X_test.iloc[:, 1:]

# replace categorical data with numeric values
X_test.replace(cleanup_data, inplace=True)

# fill in missing data according to the parameters from training
missing_columns = ['Credit Score', 'Annual Income', 'Years in current job', 'Bankruptcies', 'Maximum Open Credit',
                   'Tax Liens']
for feature in missing_columns:
    filesaved = '{}_data.sav'.format(feature)
    loaded_model = joblib.load(filesaved)
    X_test.loc[X_test[feature].isnull(), feature] = \
        loaded_model.predict(X_test[X_train_para[feature]])[X_test[feature].isnull()]

# normalize test according to training parameters
cols_to_norm = ['Current Loan Amount', 'Credit Score', 'Annual Income', 'Monthly Debt', 'Current Credit Balance',
                'Maximum Open Credit']
for col in cols_to_norm:
    X_test[col] = X_test[col].apply(lambda x: (x - col_min[col])/(col_max[col]-col_min[col]))

# assign y_test to 0 and 1 for comparison
label_cleanup = {'Fully Paid': 1, 'Charged Off': 0}
y_test.replace(label_cleanup, inplace=True)

## load prediction model: logistic regression
y_pred = lr_clf.predict(X_test)
y_test_lst = y_test.values.tolist()
error = 0
for i in range(len(y_pred)):
    if y_pred[i] != y_test_lst[i]:
        error += 1
#print(error)
print('\nlogistic regression accuracy on test set {}%'.format((1 - error / len(y_pred)) * 100))
print(lr_clf.coef_)
print('results for credit test is added to the last column, 0 is paid
off, 1 is charged off.')
```