```python
"""
Version: 1.0
Created on: Sat Mar 20 18:55:27 2021
Author: Balaji Kannan
Description: Lower Back Pain Dataset from Kaggle
    1. https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset
    2. This pipeline is custom built for a single target variable with Datatype
    Object.
    3. Binary classification problem.

"""

#%% Library

import pandas as pd
pd.set_option('display.max_columns', 20)
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#%% IO Path & Dataframe Definitions

PATH = r"C:\DSML_Case_Studies\02_Logistic_Regression\Input"
FNAME = r"\Dataset_Lower_Back_Pain.csv" # Prefix LrBkPn

OUTPATH = r"C:\DSML_Case_Studies\02_Logistic_Regression\Output"
PREFIX = r"\LrBkPn_"

df = pd.read_csv(f"{PATH}{FNAME}")
df = df.round(decimals=3) # rounding the decimals

targetvar = 0
while targetvar <= 0:
    targetvar = int(input("Enter # of Target Variables: ")) # user specifies # of
    targets

collst = []
for columns in df.columns:
    collst.append(columns)

featlst = collst[0:len(collst)-targetvar]
targlst = collst[-targetvar:]

# Sanity Checks

print(df.head(), sep='\n')
print("List of Features:", featlst, sep='\n')
print("List of Targets:", targlst, sep='\n')

#%% Exploratory Data Analysis

OFNAME1 = r"01_Descriptive_Stats.txt"

desc_stat = df.describe() # Univariate analyses
print(desc_stat)
FOUT1 = open(f"{OUTPATH}{PREFIX}{OFNAME1}", 'w+')
desc_stat.to_string(FOUT1)
FOUT1.close()

# Generating n*4 matrix of box plots

n_rows = len(collst)//4
fig, axes = plt.subplots(n_rows, 4, figsize = (15,15))
axes = axes.flatten()

FIG1 = r"_FIG01_Boxplot"
for i in range(0,len(df.columns)-targetvar):
    sns.boxplot(x=targlst[0], y=df.iloc[:,i], data=df, orient='v', ax=axes[i])
    plt.tight_layout()
    plt.savefig(f"{OUTPATH}{PREFIX}{FIG1}")

# Linear Correlation Heatmap
```

```python

cormethod = {0:'pearson', 1:'kendall', 2:'spearman'}
for i in range(0, 3, 1):
    temp = 'linear_cor' + str(i)
    temp = df.corr(method=cormethod[i])
    ftemp = cormethod[i].title()
    FIG2 = r"_FIG02_Corr_"
    mask = np.zeros(temp.shape, dtype=bool)
    mask[np.tril_indices(len(mask))] = False
    plt.subplots(figsize=(20,15))
    plt.title(f"{ftemp} Corrleation")
    sns.heatmap(temp, annot=True, vmin=-1, vmax=1, center=0,
                cmap='coolwarm', square=True, mask=mask)
    plt.savefig(f"{OUTPATH}{PREFIX}{FIG2}{ftemp}")

# Non-Linear Correlation Predictive Power Score - Heatmap

import ppscore as pps

FIG3 = r"_FIG03_Predictive_Power_Score"

ppscorr = pps.matrix(df) # Predictive Power Score - PPS
matrix_df = pps.matrix(df)[['x', 'y', 'ppscore']].pivot(columns='x', index='y',
                                                        values='ppscore')
plt.subplots(figsize=(20,15))
sns.heatmap(matrix_df, cmap="Greens", annot=True, linewidth=0,
annot_kws={"size":12}, fmt='.2g')
plt.savefig(f"{OUTPATH}{PREFIX}{FIG3}")

# Scatter Plot with Hue

FIG4 = r"_FIG04_Scatter_Plot"

grid1 = sns.PairGrid(df, hue=targlst[0])
grid1.map(plt.scatter)
grid1.map_diag(sns.kdeplot)
grid1.add_legend()
grid1.fig.suptitle("Scatter Plot", y=1.01)
grid1.savefig(f"{OUTPATH}{PREFIX}{FIG4}")

#%% Pandas Profile Report

from pandas_profiling import ProfileReport

OFNAME2 = r"02_Descriptive_Stats.html"
report = ProfileReport(df) # Descriptive statistics report
report.to_file(f"{OUTPATH}{PREFIX}{OFNAME2}") # Rendering to HTML

#%% High Dimensional Interactive Plot - HD Plot

import hiplot as hip

OFNAME3 = r"03_Parallel_Plot.html"
parplot = hip.Experiment.from_dataframe(df)
parplot.to_html(f"{OUTPATH}{PREFIX}{OFNAME3}")

#%% Machine Learning Model - Pre-Processing

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

import statsmodels.api as sm

df.loc[df.Class_att=='Abnormal',targlst[0]] = 1
df.loc[df.Class_att=='Normal', targlst[0]] = 0
X = df.drop(columns=targlst)
y = df.filter(targlst, axis=1)


```

```python
def data_split(X,y):
    """
    Parameters
    ----------
    X : Feature Variables
    y : Target Variables
    Returns
    -------
    Split dataframe into train and test
    """
    X_train, X_test, y_train, y_test = train_test_split(X, y.values.ravel(), test_size=0.3,
                                                                             random_state=39)
    scaler = StandardScaler(copy=True, with_mean=True, with_std=True)
    scaler.fit(X_train)
    train_scaled = scaler.transform(X_train) # Only feature variables are scaled
    test_scaled = scaler.transform(X_test) # only features varaibels are scaled
    y_train=y_train.astype('int')
    y_test=y_test.astype('int')
    return(train_scaled, test_scaled, y_train, y_test)

#%% Logistic Regression Model

def logistic_regression(x,y):
    """
    Parameters
    ----------
    x : Feature Variables
    y : Target Variables

    Returns
    -------
    Logistic Regression Fit

    """
    logreg = LogisticRegression().fit(x, y)
    return(logreg)

X_train_scaled, X_test_scaled, y_train, y_test = data_split(X,y)
logreg_result = logistic_regression(X_train_scaled, y_train)

print("Training set score: {:.3f}".format(logreg_result.score(X_train_scaled,y_train)))
print("Test set score: {:.3f}".format(logreg_result.score(X_test_scaled,y_test)))

#%% Logit with Stats Model
"""
1. Scikit LogisticRegression is good in predicting target variable on a test set.
2. It did not interpret anything about the individual features.
3. Which variable(s) influence the Target variable more?
4. Use logit from stats model to answer questions 2 & 3
"""

logit_model = sm.Logit(y_train, X_train_scaled)
result = logit_model.fit()
print(result.summary2())

"""
1. Output such as:
    Maximum Likelihood optimization failed to converge -
    indicates that there is multicoliniarity.
    Removing those variable would improve the convergence.
"""

#%% Feature Reduction Tryouts

features_to_drop = ['pelvic_incidence', 'lumbar_lordosis_angle'] # From Correlation Heatmap

new_lst = targlst + features_to_drop
X = df.drop(columns=new_lst)
y = df.filter(targlst, axis=1)
```

```python
X_train_scaled, X_test_scaled, y_train, y_test = data_split(X,y)
logreg_result = logistic_regression(X_train_scaled, y_train)

print("Training set score: {:.3f}".format(logreg_result.score(X_train_scaled,y_train)))
print("Test set score: {:.3f}".format(logreg_result.score(X_test_scaled,y_test)))

logit_model = sm.Logit(y_train, X_train_scaled)
result = logit_model.fit()
print(result.summary2())

#%% Consider only Variables for which P-value is < 0.05

features_to_drop1 = ['sacral_slope',  'pelvic_slope',  'Direct_tilt',
                      'thoracic_slope',  'cervical_tilt',  'sacrum_angle',
                      'scoliosis_slope', 'pelvic_tilt'] #
new_lst1 = targlst + features_to_drop + features_to_drop1
X = df.drop(columns=new_lst1)
y = df.filter(targlst, axis=1)
print(X.head())

X_train_scaled, X_test_scaled, y_train, y_test = data_split(X,y)
logreg_result = logistic_regression(X_train_scaled, y_train)

print("Training set score: {:.3f}".format(logreg_result.score(X_train_scaled,y_train)))
print("Test set score: {:.3f}".format(logreg_result.score(X_test_scaled,y_test)))

logit_model = sm.Logit(y_train, X_train_scaled)
result = logit_model.fit()
print(result.summary2())

#%% Model Metrics

# assigning the model predicted values to y_pred
y_pred = logreg_result.predict(X_test_scaled)


# assigning the string Normal and Abnormal to the 0 and 1 values respectively. This is useful in plotting
# the confusion matrix
y_pred_string = y_pred.astype(str)
y_pred_string[np.where(y_pred_string == '0')] = 'Normal'
y_pred_string[np.where(y_pred_string == '1')] = 'Abnormal'

y_test_string = y_test.astype(str)
y_test_string[np.where(y_test_string == '0')] = 'Normal'
y_test_string[np.where(y_test_string == '1')] = 'Abnormal'

from sklearn.metrics import confusion_matrix

FIG5 = r"_FIG05_Confusion_Matrix"
ax= plt.subplot()
labels = ['Abnormal','Normal']
cm = confusion_matrix(y_test_string, y_pred_string, labels)
sns.heatmap(cm, annot=True, ax = ax, cmap='coolwarm'); #annot=True to annotate cells
plt.savefig(f"{OUTPATH}{PREFIX}{FIG5}")

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Abnormal', 'Normal']); ax.yaxis.set_ticklabels(['Abnormal', 'Normal']);
plt.show()

print(classification_report(y_test, y_pred, target_names=labels))
```