```
#!/usr/bin/env python3

# -*- coding: utf-8 -*-



import pandas as pd


# read the dataset into a pandas dataframe object

df = pd.read_csv('/Users/aghazarian/wdbc.data', header=None)


#print some data to make sure the data was properly read into the dataframe

print(df.head(n=3))

rows,columns = df.shape


# get a sense of the size of the dataset

print('# of rows:', rows)

print('# of columns:', columns)

# first column is just an ID number, second column is the diagnostics - M for Malignant and B for Benign

# 3rd to 32nd column include the 30 features of the dataset


# assign the 30 features of the dataset and the target variable to separate Numpy arrays

# same result in the next 2 lines if we replace iloc with loc

X = df.iloc[:, 2:].values

Y = df.iloc[:, 1].values


# print X and Y by typing the variable names in console to make sure you properly extrated the

# independent and dependent variables. You can also directly look at the contents of these variables

# using the variabe explorer on the right


# transform the class labels from their original string representation (M and B) to integers
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(Y)
# now Malignant is 1 and Benign is 0

# devide the dataset into 80% training and 20% separate test data
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

# create a pipeline with the following steps chained to each other
#1. for optimal performance transform all feature values into the same scale - standardize the columns
#   before feeding them to the classifier
#2. Compress the initial 30 dimensional data into a lower 2 dimensional space using PCA
#3. Apply the logistic regression algorithm

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

# Define the steps of the pipeline, each step is a tuple:  a name and either a transformer or estimator
object
classification_pipeline = Pipeline([('standard_scaler', StandardScaler()),
                        ('pca', PCA(n_components=2)),
                        ('classifier',LogisticRegression(random_state=1))])
# fit the training data into the model
classification_pipeline.fit(X_train,Y_train)
```

```
# compute the accuracy of the model on test data
accuracy = classification_pipeline.score(X_test, Y_test)
print('Test Accuracy: %.3f' % accuracy)

# instead of just a single test on test data, let's do a stratified k-fold cross validation on training data
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(estimator=classification_pipeline, X=X_train, y=Y_train, cv=10, n_jobs=1)
import numpy as np
print('CV accuracy %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

# let's take a look at the confusion matrix
from sklearn.metrics import confusion_matrix
y_pred = classification_pipeline.predict(X_test)
confmat = confusion_matrix(y_true=Y_test,y_pred=y_pred)
print(confmat)
# we get [[71,1][5,32]] first list is for class 0 (-) and second list is for class 1 (+)
# to convert this into my convention (a) stack the 2 lists, one in each row to get
# [71, 1]   [TN, FP]   rows are TRUE CLASS, columns are PREDICTED class
# [5, 32]   [FN, TP]
#Then simply switch the elements on the diagonal to get
# [31, 1]    [TP, FP]     rows are PREDICTED CLASS, columns are TRUE CLASS
# [5, 71]    [FN, TN]

from sklearn.metrics import classification_report
print('****************:', classification_report(y_true=Y_test,y_pred=y_pred))

# compute precision, recall, and F measure
# we have already seen this as part of the classification report printed in the above line
# classification report gives these masures for both the positive and negative class
# what we get here is only for the positive class
from sklearn.metrics import precision_score
print('Precision: %.3f' % precision_score(y_true=Y_test, y_pred=y_pred))

from sklearn.metrics import recall_score, f1_score
print('Recall: %.3f' % recall_score(y_true=Y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=Y_test, y_pred=y_pred))



# now let's try the logistic regression model with different parameters and try
# to finetune these hyperpatameters using the Grid Search approach
# Any parameter provided when constructing an estimator may be optimized in this manner.
# Specifically, to find the names and current values for all parameters for a given estimator, use:
# estimator.get_params()
from sklearn.grid_search import GridSearchCV
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
# note that a list of available parameters to tune can be displayed by typing this in the console:
# classification_pipeline.get_params().keys()
print('Paramters available for tuning:', classification_pipeline.get_params().keys())
# also note that the param name is constructed as the string name given to it
# in the pipeline constructor followed by 2 underscores and then the param name
param_grid =  [{'classifier__penalty':['l1', 'l2'], 'classifier__C': param_range}]
# note that in the above line each set of items enclosed in {} defines one grid
# here we only have one grid, but we could have multiple grids to explore
gs = GridSearchCV(estimator=classification_pipeline,
            param_grid=param_grid,
            scoring= 'accuracy',
            cv=10,
            n_jobs=-1)
```

```
gs = gs.fit(X_train, Y_train)

print('Best accuracy score:',gs.best_score_)

print('Best Parameters:',gs.best_params_)
```