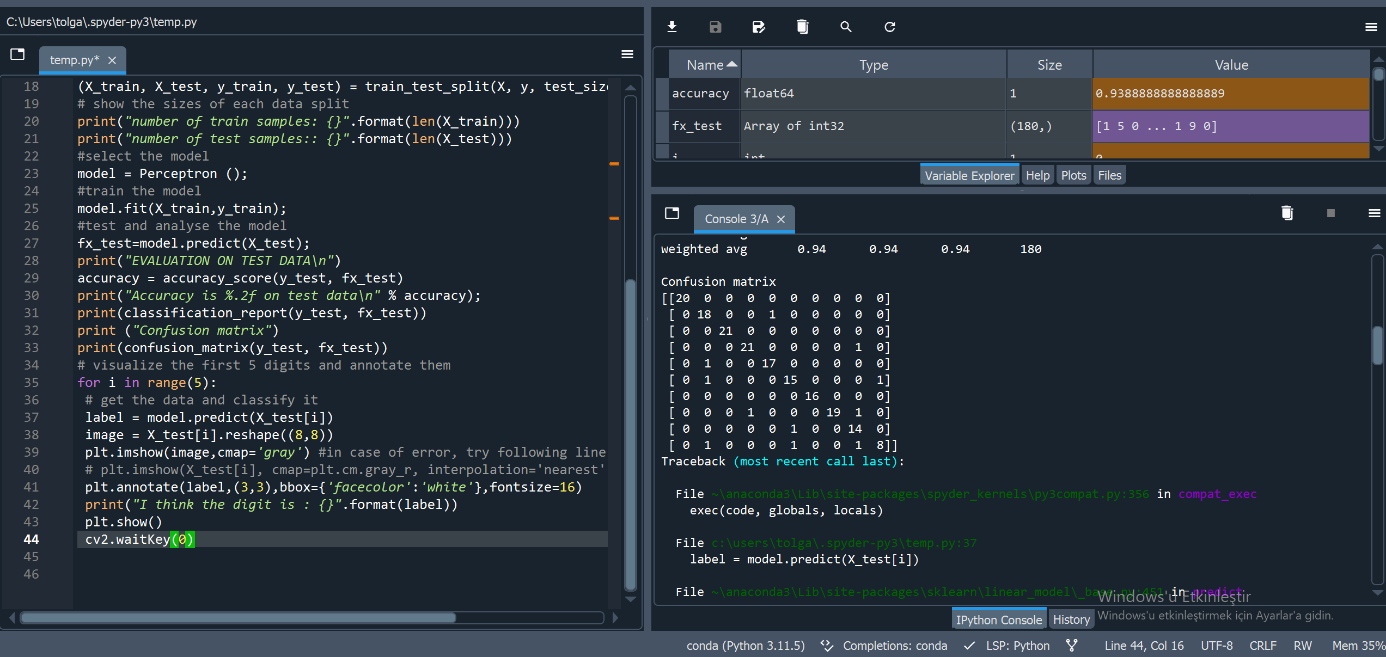
i) Running the example code results in the following confusion matrix



The second line is for ‘1’ class

Precision = TP / (TP+FP)

TP (True Positives) = 18

FP (False Positives) = 3

Precision= 18/(18+3) = 0.857

ii)

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

import matplotlib.pyplot as plt

from sklearn import datasets

import numpy as np

import cv2

# load the MNIST digits dataset

mnist = datasets.load\_digits()

X = mnist.data

y = mnist.target

# Standard normalization

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# split dataset into training and test set

(X\_train, X\_test, y\_train, y\_test) = train\_test\_split(X, y, test\_size=0.10, random\_state=1)

# show the sizes of each data split

print("Number of train samples: {}".format(len(X\_train)))

print("Number of test samples: {}".format(len(X\_test)))

# Parameter tuning

best\_accuracy = 0

best\_max\_iter = 0

best\_tol = 0

for max\_iter in [100, 500, 1000, 1500]:

for tol in [0.0001, 0.001, 0.01, 0.1]:

model = Perceptron(max\_iter=max\_iter, tol=tol)

model.fit(X\_train, y\_train)

fx\_test = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, fx\_test)

print(f"Max\_iter: {max\_iter}, Tol: {tol}, Accuracy: {accuracy}")

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_max\_iter = max\_iter

best\_tol = tol

# Report the best parameters

print(f"Best parameters - Max\_iter: {best\_max\_iter}, Tol: {best\_tol}, Accuracy: {best\_accuracy}")

# train the model with the best parameters

model = Perceptron(max\_iter=best\_max\_iter, tol=best\_tol)

model.fit(X\_train, y\_train)

# test and analyze the model

fx\_test = model.predict(X\_test)

print("EVALUATION ON TEST DATA\n")

accuracy = accuracy\_score(y\_test, fx\_test)

print("Accuracy is %.2f on test data\n" % accuracy)

print(classification\_report(y\_test, fx\_test))

print("Confusion matrix")

print(confusion\_matrix(y\_test, fx\_test))

# visualize the first 5 digits and annotate them

for i in range(5):

# get the data and classify it

label = model.predict(X\_test[i:i+1])

image = X\_test[i].reshape((8, 8))

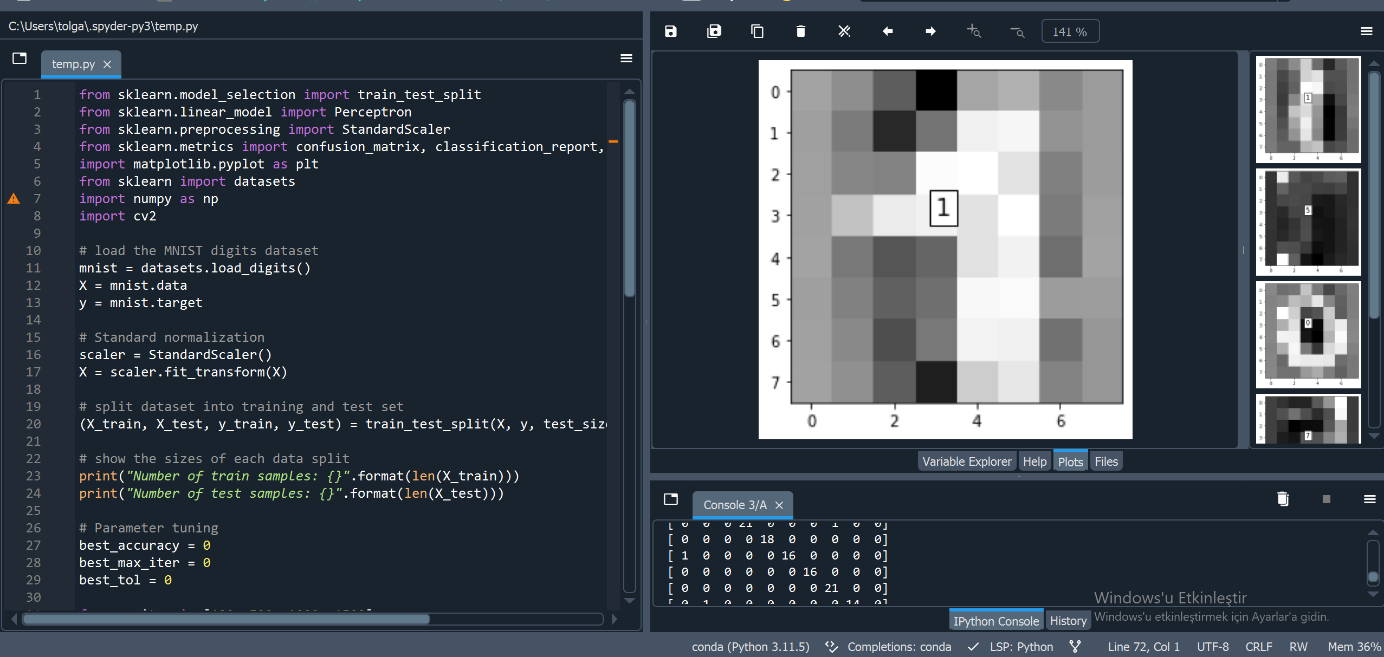
plt.imshow(image, cmap='gray')

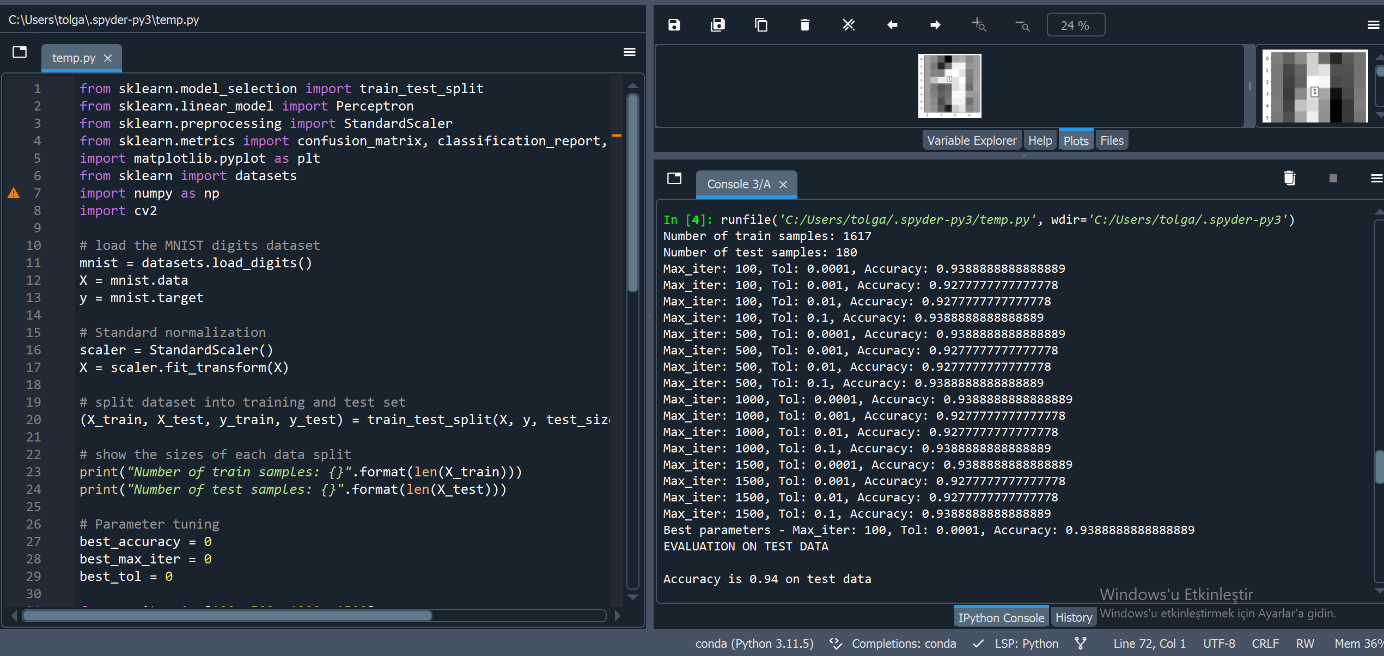
plt.annotate(label[0], (3, 3), bbox={'facecolor': 'white'}, fontsize=16)

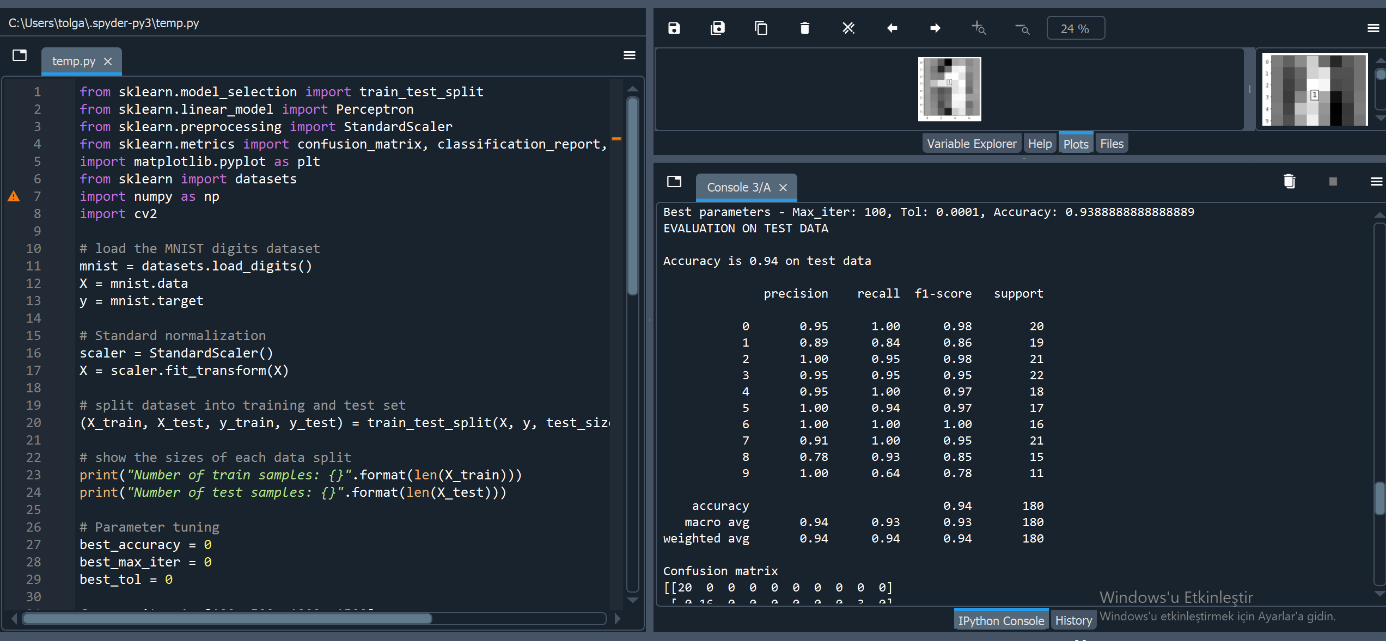
print("I think the digit is : {}".format(label[0]))

plt.show()

cv2.waitKey(0)



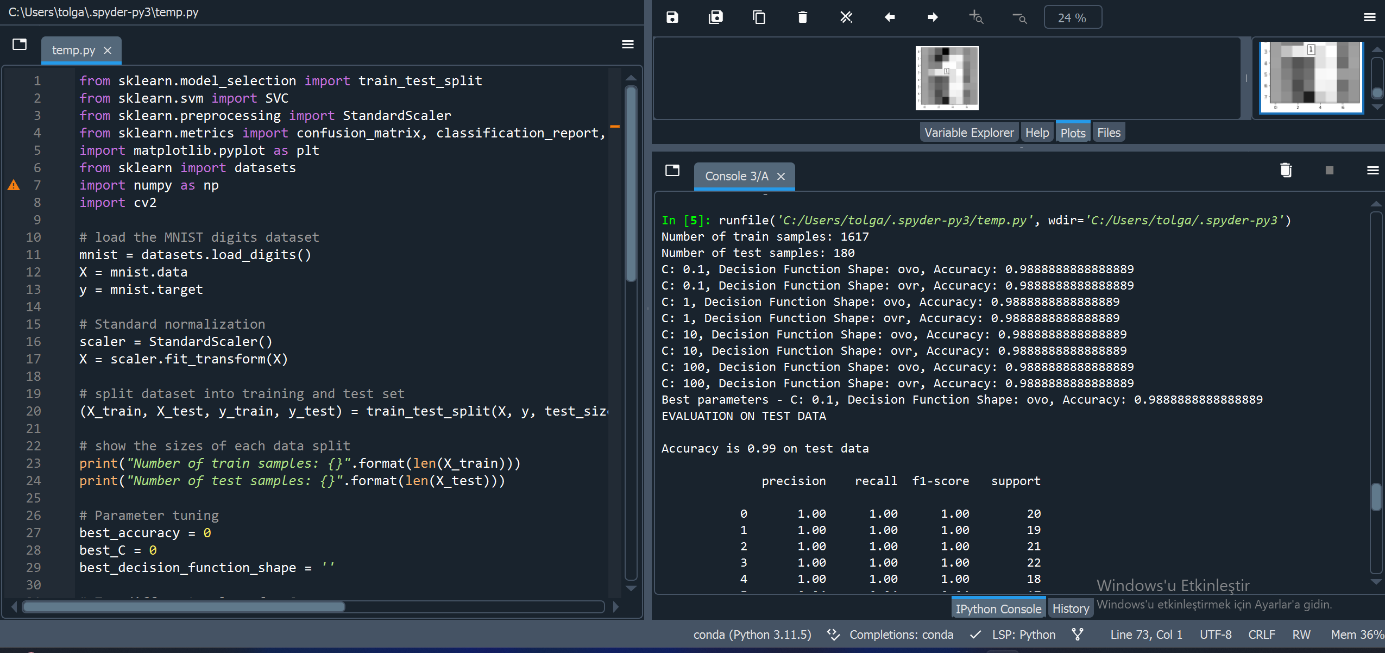


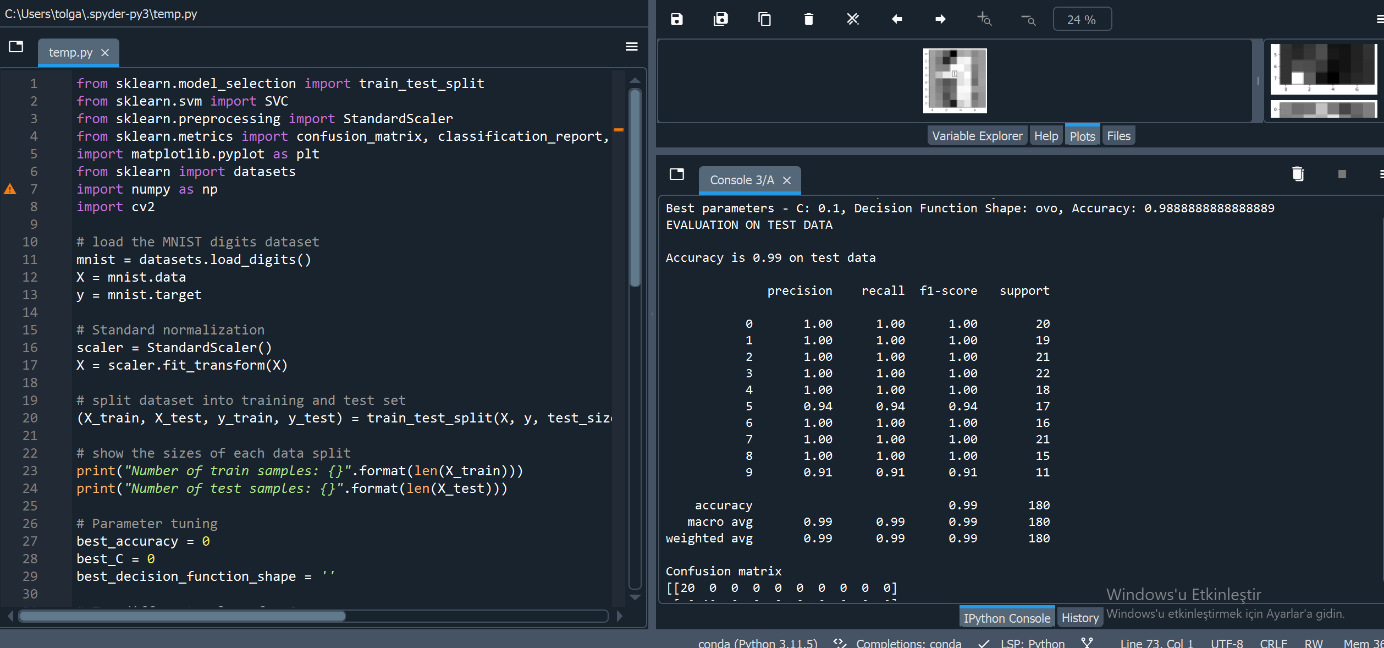


"This code includes standard normalization and performs a grid search on the max\_iter and tol parameters for the Perceptron model. The goal is to determine the combination of these parameters that provides the best performance. The best selected parameters are then used to train the final model and the effectiveness of the model is evaluated on test data "The effect of normalization may vary depending on the characteristics of the algorithm and the dataset. Therefore, it is always important to conduct experiments to observe and analyze the results."

iii) Modified Code Fragment

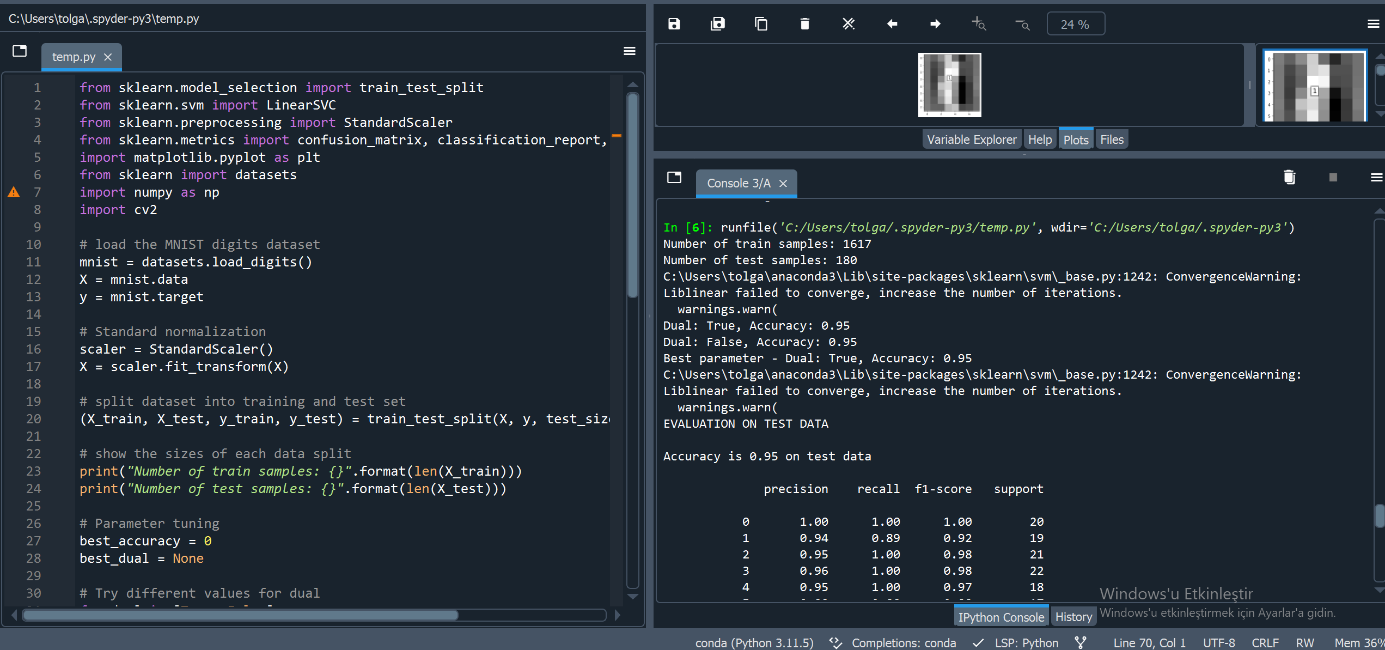
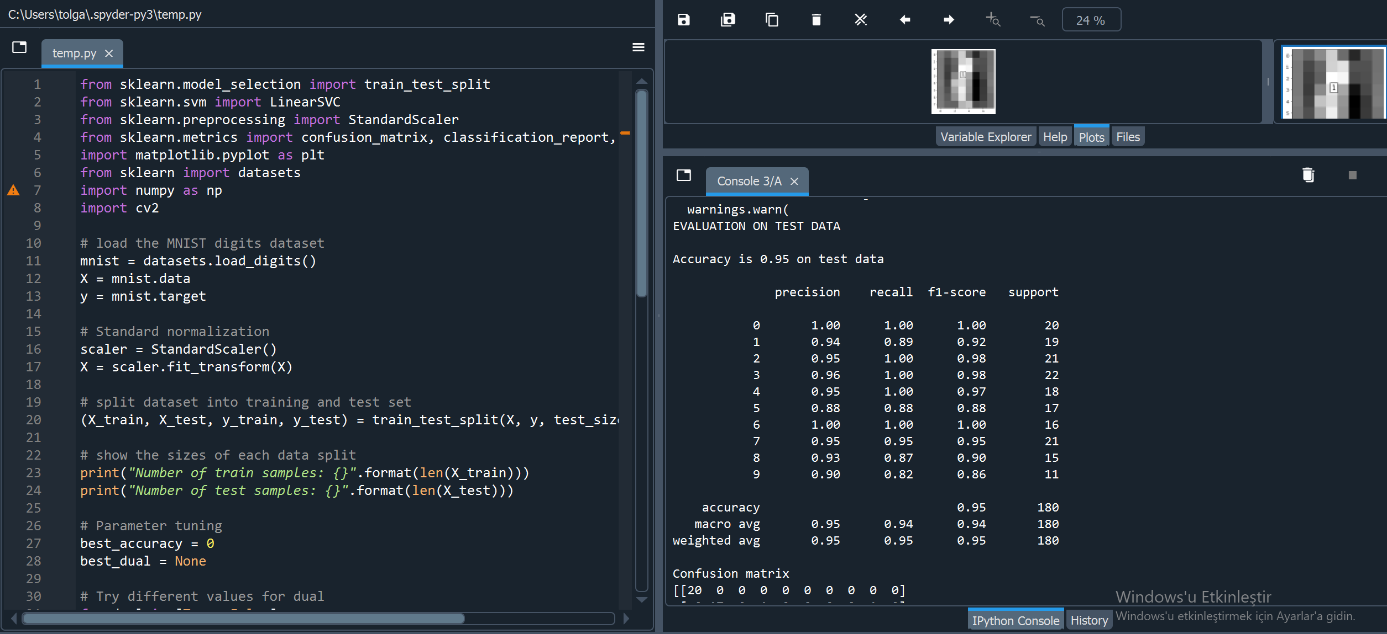
model = svm.SVC(kernel = "linear")





This code loops over different C and decision\_function\_shape values ​​and reports the best parameters based on accuracy. The best parameters are then used to train the final SVC model and its performance is evaluated on test data. Adjustments to C and decision function shape values ​​can have varying effects on the performance of the model, so it is important to observe and analyze the results.

iv)

model = LinearSVC ( C=1.0, dual=False) # yes to lagrange dual solution

The LinearSVC class in the scikit-learn library has a parameter called dual, and setting this parameter to False uses the primal optimization problem, while setting it to True uses the dual optimization problem. Generally, the choice between dual=False and dual=True depends on the relationship between the number of samples (n\_samples) and the number of features (n\_features). If n\_samples is much larger than n\_features, it will usually be more efficient to set dual=False.

This code tests LinearSVC with different dual values ​​and reports the best parameter based on accuracy. In practice, the performance comparison between dual=False and dual=True depends on the characteristics and characteristics of a particular dataset. Generally, for datasets with a large number of samples and a small number of features, setting dual=False can lead to better performance.

v)

Code using Logistic Regression using different solvers ('newton-cg', 'lbfgs', 'liblinear'). The best dissolution values ​​will be determined accordingly.

This code tests Logistic Regression with different solvers and reports accuracy, basic metric and good solver. Since Logistic Regression's default solver is 'lbfgs', the default values ​​will be determined based on the best solver.  
  
from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

import matplotlib.pyplot as plt

from sklearn import datasets

import numpy as np

import cv2

# load the MNIST digits dataset

mnist = datasets.load\_digits()

X = mnist.data

y = mnist.target

# Standard normalization

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# split dataset into training and test set

(X\_train, X\_test, y\_train, y\_test) = train\_test\_split(X, y, test\_size=0.10, random\_state=1)

# show the sizes of each data split

print("Number of train samples: {}".format(len(X\_train)))

print("Number of test samples: {}".format(len(X\_test)))

# Parameter tuning

best\_accuracy = 0

best\_solver = None

# Try different solvers

for solver in ['newton-cg', 'lbfgs', 'liblinear']:

model = LogisticRegression(solver=solver)

model.fit(X\_train, y\_train)

fx\_test = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, fx\_test)

print(f"Solver: {solver}, Accuracy: {accuracy}")

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_solver = solver

# Report the best parameter

print(f"Best solver - Solver: {best\_solver}, Accuracy: {best\_accuracy}")

# train the model with the best parameter

model = LogisticRegression(solver=best\_solver)

model.fit(X\_train, y\_train)

# test and analyze the model

fx\_test = model.predict(X\_test)

print("EVALUATION ON TEST DATA\n")

accuracy = accuracy\_score(y\_test, fx\_test)

print("Accuracy is %.2f on test data\n" % accuracy)

print(classification\_report(y\_test, fx\_test))

print("Confusion matrix")

print(confusion\_matrix(y\_test, fx\_test))

# visualize the first 5 digits and annotate them

for i in range(5):

# get the data and classify it

label = model.predict(X\_test[i:i+1])

image = X\_test[i].reshape((8, 8))

plt.imshow(image, cmap='gray')

plt.annotate(label[0], (3, 3), bbox={'facecolor': 'white'}, fontsize=16)

print("I think the digit is : {}".format(label[0]))

plt.show()

cv2.waitKey(0)  
  
  
