Lec5:

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.tree import export\_graphviz

import graphviz

import matplotlib.pyplot as plt

import io

data = pd.DataFrame({

    'Credit score': [700, 600, 750, 650, 800],

    'Income': [5000, 3000, 7000, 4000, 6000],

    'Loan amount': [10000, 5000, 15000, 8000, 12000],

    'Loan term': ['short-term', 'long-term', 'short-term', 'long-term', 'short-term'],

    'Employment status': ['employed', 'self-employed', 'unemployed', 'employed', 'other'],

    'Previous delinquencies': ['no', 'yes', 'no', 'no', 'yes'],

    'Loan Approval': ['yes', 'no', 'yes', 'no', 'yes']

})

# Split the dataset into features (X) and target variable (y)

X = data.drop('Loan Approval', axis=1)

y = data['Loan Approval']

# Convert categorical variables to numerical using one-hot encoding

X = pd.get\_dummies(X)

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Build the decision tree classifier

clf = DecisionTreeClassifier()

# Train the decision tree classifier on the training data

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

# Use the trained classifier to predict loan approvals for the test data

y\_pred = clf.predict(X\_test)

# Evaluate the accuracy of the model

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

print("Accuracy: {:.2f}%".format(accuracy \* 100))

# Visualize the decision tree

dot\_data = export\_graphviz(clf, out\_file=None, feature\_names=X.columns, class\_names=['No', 'Yes'], filled=True, rounded=True)

graph = graphviz.Source(dot\_data)

# Save the decision tree plot as a file

image\_data = io.BytesIO()

graph.format = 'png'

graph.render('loan\_decision\_tree', format='png', view=False)

graph.render(filename='loan\_decision\_tree', format='png', view=False)

# Load and display the decision tree plot

image = plt.imread('loan\_decision\_tree.png')

plt.imshow(image)

plt.axis('off')

plt.show()

B:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Generate a random dataset with two features (x1, x2)

DIM = 20      # Number of features (dimension)

CLASSES = 4   # Number of classes

N = 1000      # Number of samples

X, y = make\_blobs(n\_samples=N, centers=CLASSES, n\_features=DIM, cluster\_std=3)

print(np.unique(y))

X[:, 2] = X[:, 2] \* 1000

X[:, 4] = X[:, 4] \* 10000

X[:, 15] = X[:, 15] \* 100

# Divide the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

print(X\_train.shape, X\_test.shape)

# Implement the KNN classification

k = KNeighborsClassifier(n\_neighbors=2)

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

print("KNeighborsClassifier(n\_neighbors=2)")

y\_pred = k.predict(X\_test)

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

print("accuracy =", accuracy)

# Train the KNN classifier on the training set and determine the optimal value of k using

# cross-validation.

param\_grid = {'n\_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]}

cv = GridSearchCV(k, param\_grid, refit=True, verbose=3, n\_jobs=-1)

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

print(cv)

print("print(cv.best\_params\_) =", cv.best\_params\_)

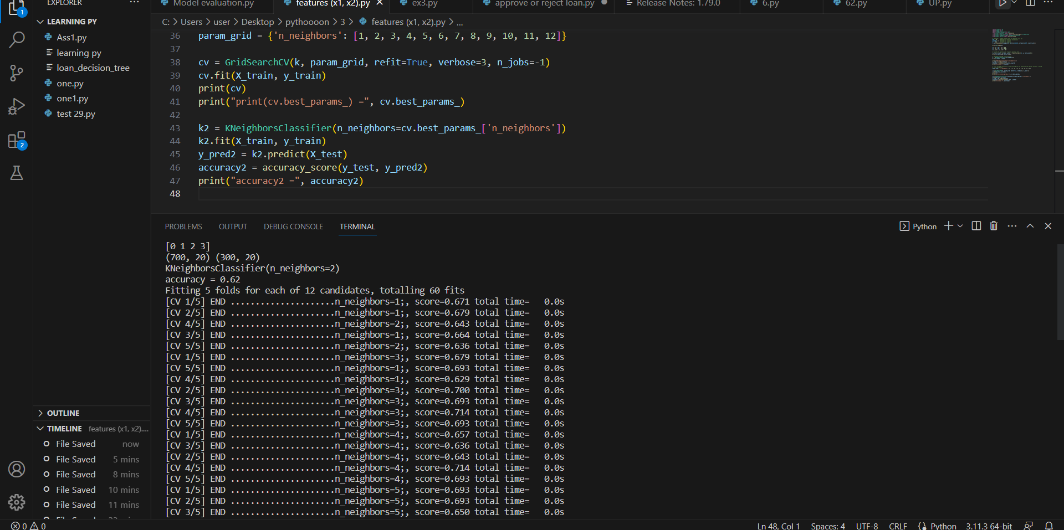
k2 = KNeighborsClassifier(n\_neighbors=cv.best\_params\_['n\_neighbors'])

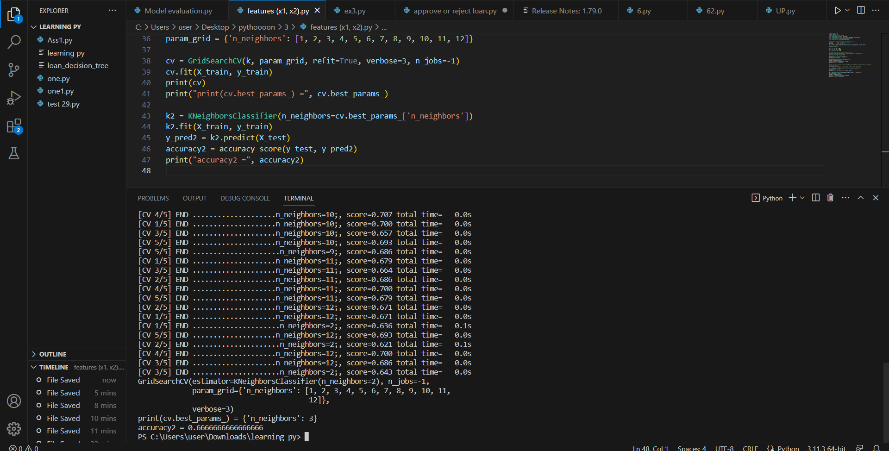
k2.fit(X\_train, y\_train)

y\_pred2 = k2.predict(X\_test)

accuracy2 = accuracy\_score(y\_test, y\_pred2)

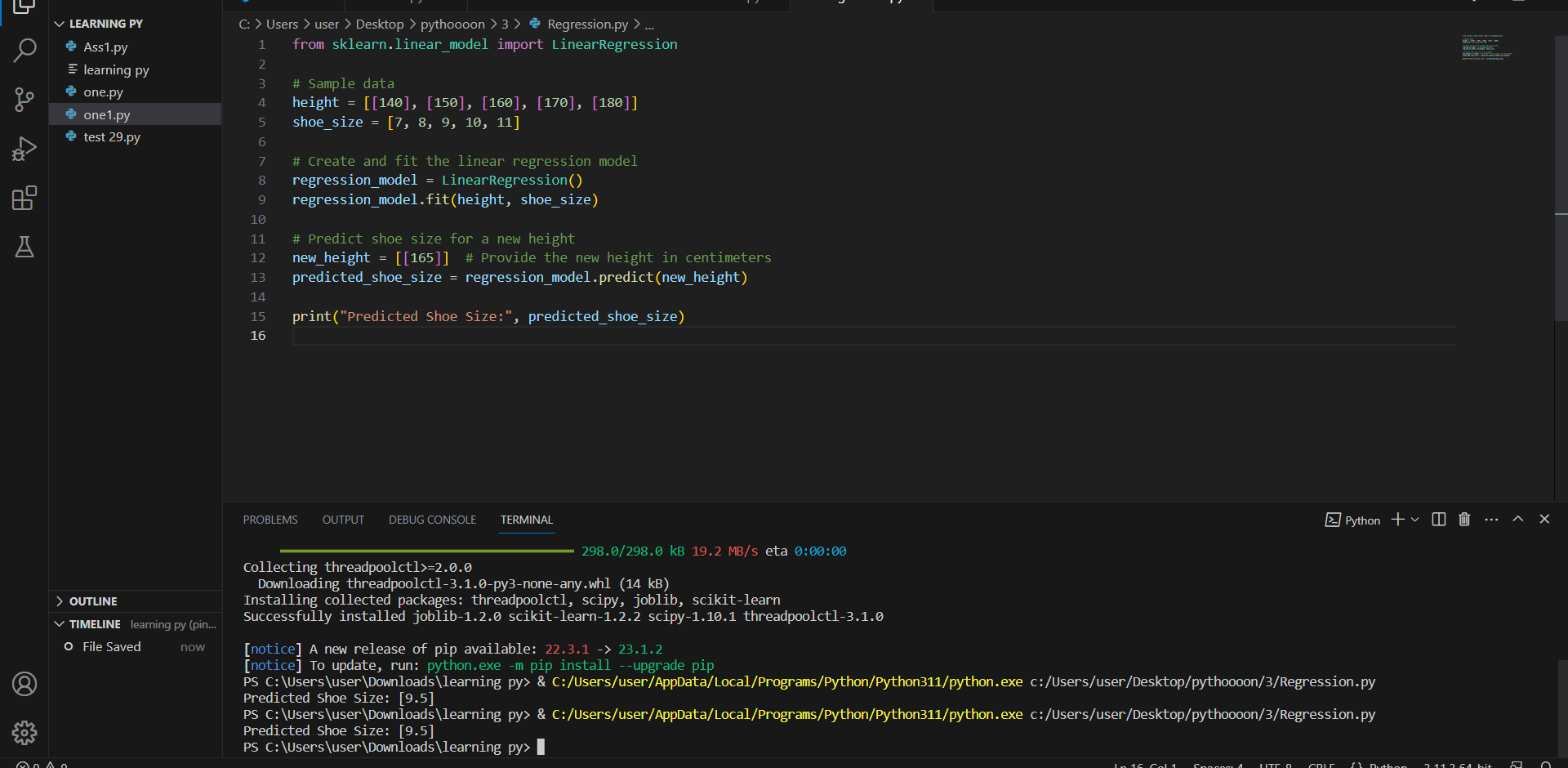
print("accuracy2 =", accuracy2)





C:

Regression:



Model evaluation B :

Accuracy = TN+TP/ TN+ FP + TP + FN

=(20+30)/(20+10+30+30)=0.5

0.5 accuracy means that the model predict 50% of the actual values are correct which means that this result not good because it's not help us to make a good decision>

Precision = TP/ TP + FP

=(20)/(30+20)=0.4

0.4 of the precision means that the model is not very good.

The model predict the positive class correctly for 40%, where the model predict the positive class negative for 60%.

Recall =TP/ TP + FN

=(20)/(20+10)=0.6

Recall of 0.6 means that out of all the actual positive class, the model correctly identified 60% of them.

F1 Score = 2 \* Precision \* Recall / Precision + Recall

=(2\*0.4\*0.6)/(0.4+0.6)=0.48

Ex3:

*#imported library*

**from** random **import** randint

*#create a list of integers*

rand\_int**=**[]

**for** i **in** range(1,11):

num**=**randint(1,10)

rand\_int**.**append(num)

rand\_int

*#calculate mean*

mean **=** sum(rand\_int) **/** len(rand\_int)

mean

*#calculate standard deviation*

differences **=** [(value **-** mean)**\*\***2 **for** value **in** rand\_int]*#x-mean\*

sum\_diff**=**sum(differences) *#sum of x-mean*

stand\_d**=**(sum\_diff **/** (len(rand\_int) **-** 1)) **\*\*** 0.5

stand\_d

*#calculate z-score*

z\_scores **=** [(value **-** mean) **/** stand\_d **for** value **in** rand\_int]

z\_scores

min\_x**=**min(rand\_int)

max\_x**=**max(rand\_int)

*#calculate min-max*

min\_max**=**[(value**-**min\_x)**/**(max\_x**-**min\_x) **for** value **in** rand\_int]

min\_max

