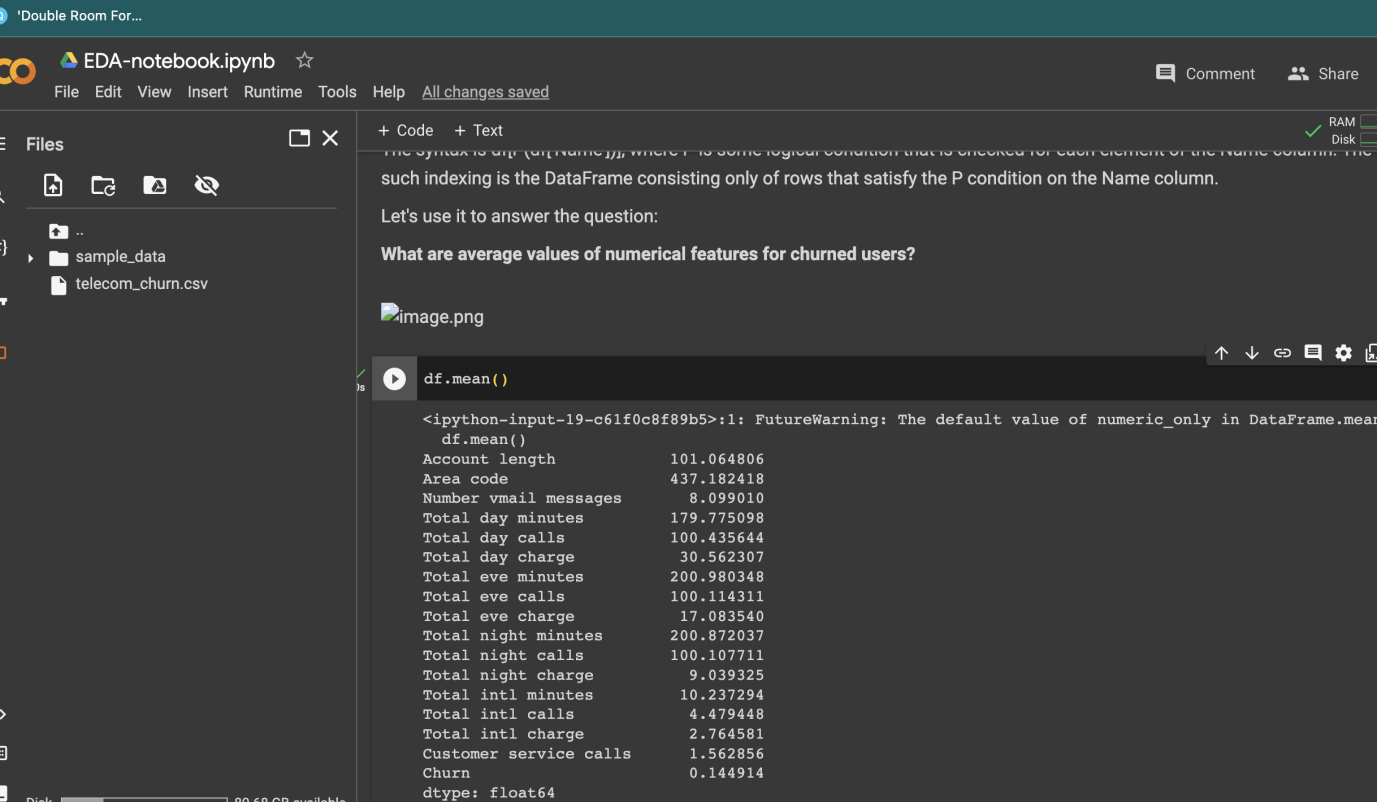
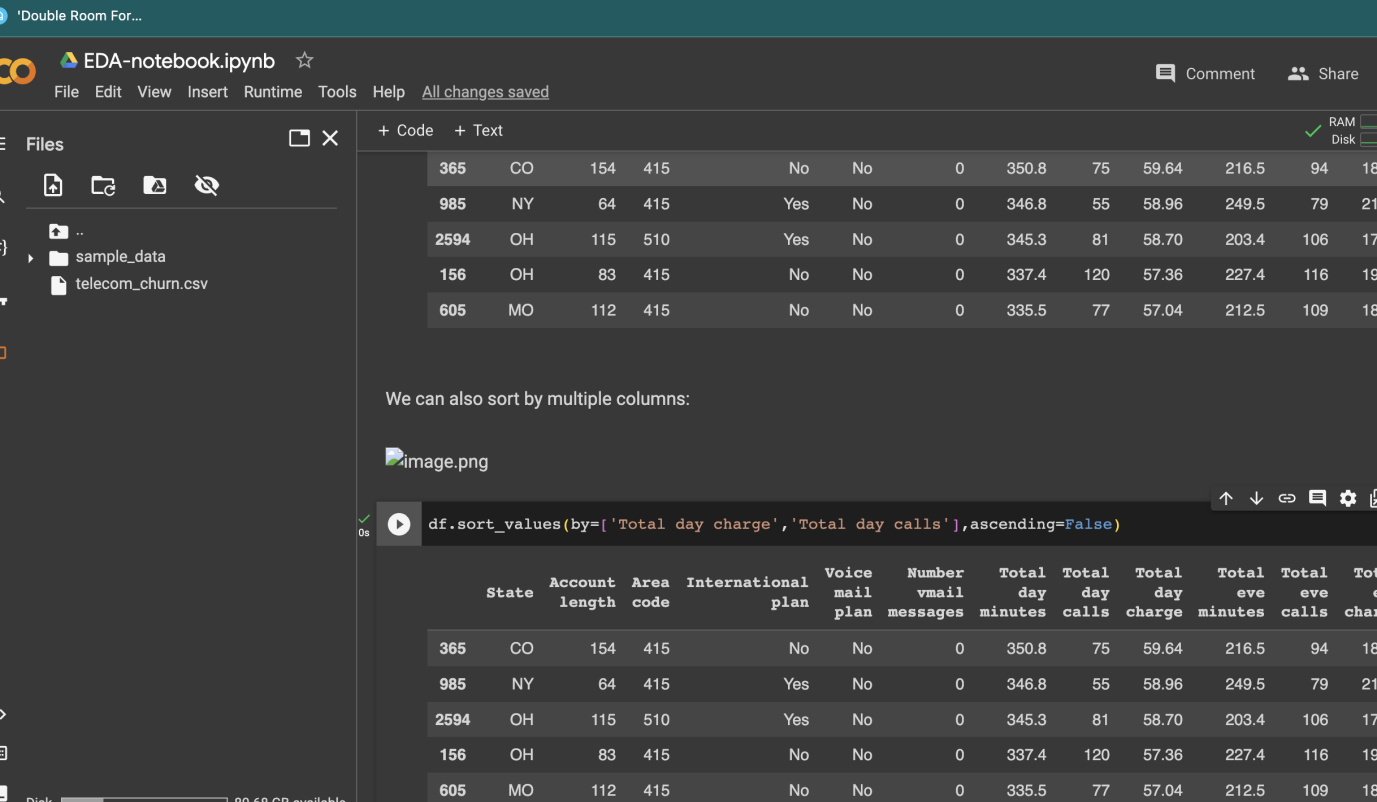
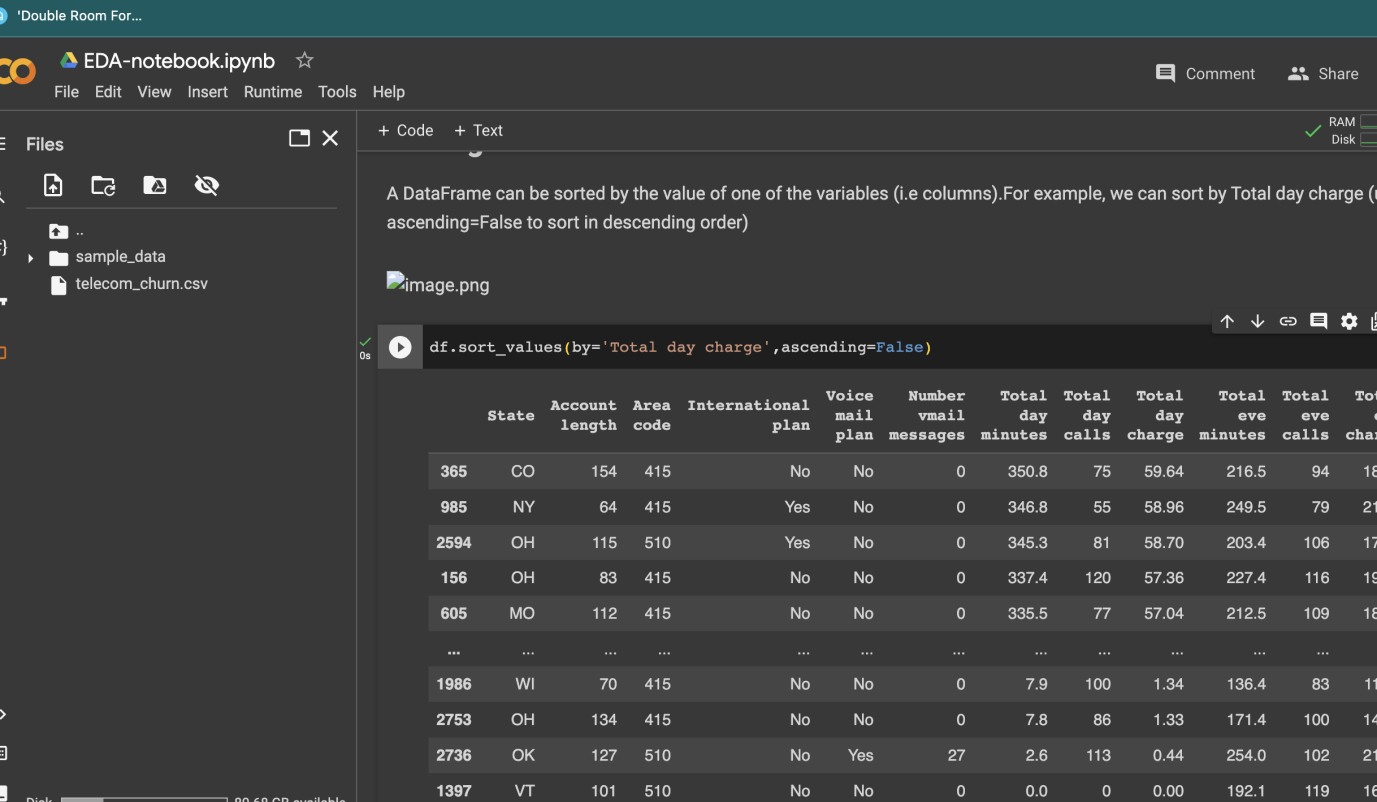
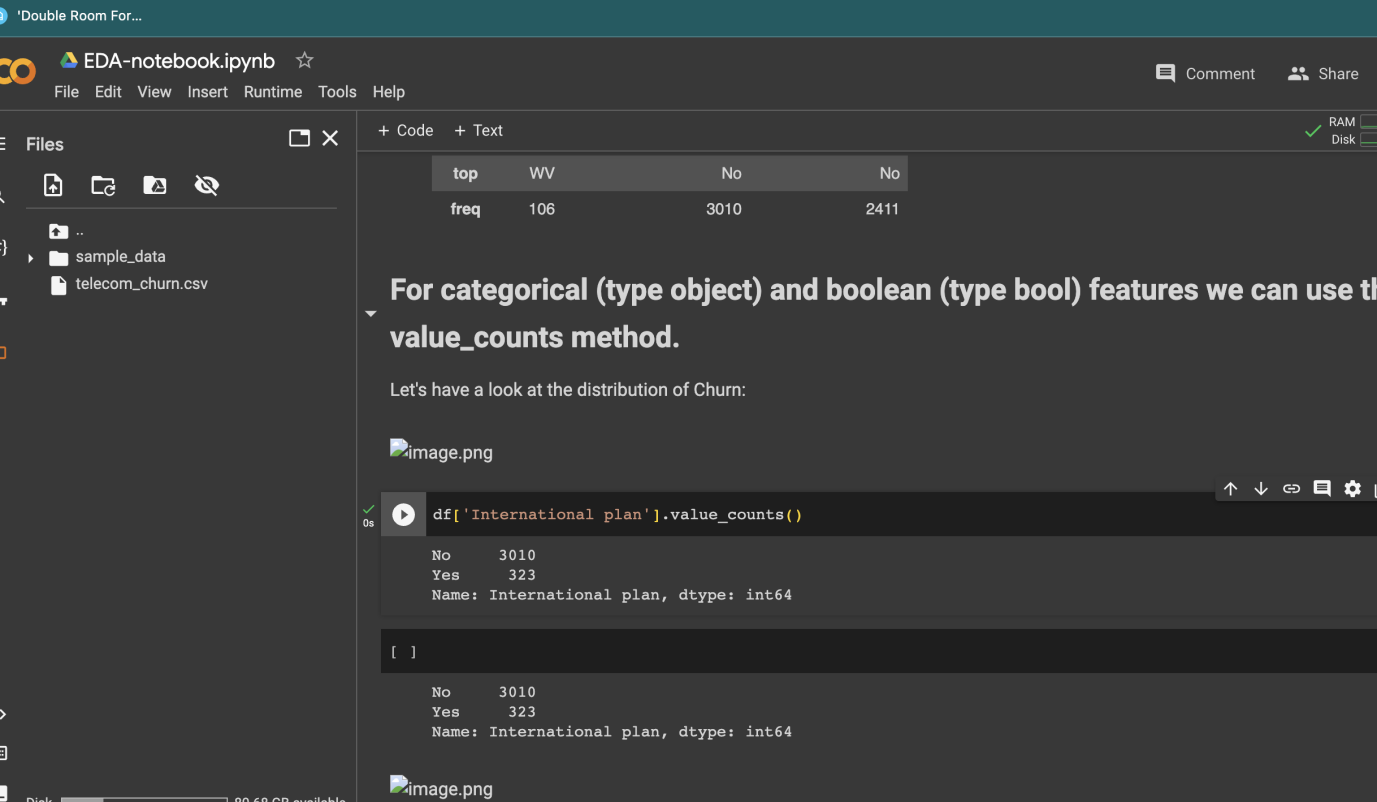
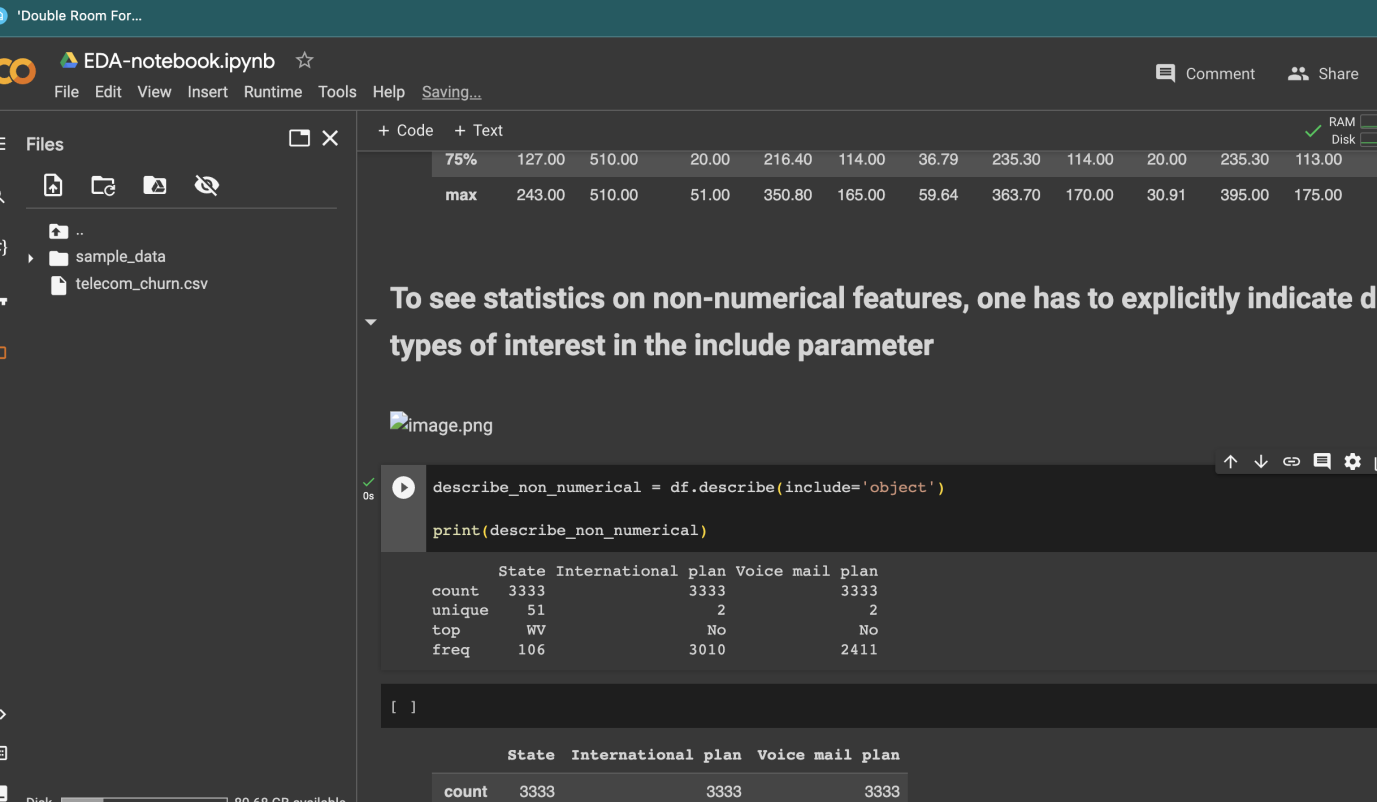
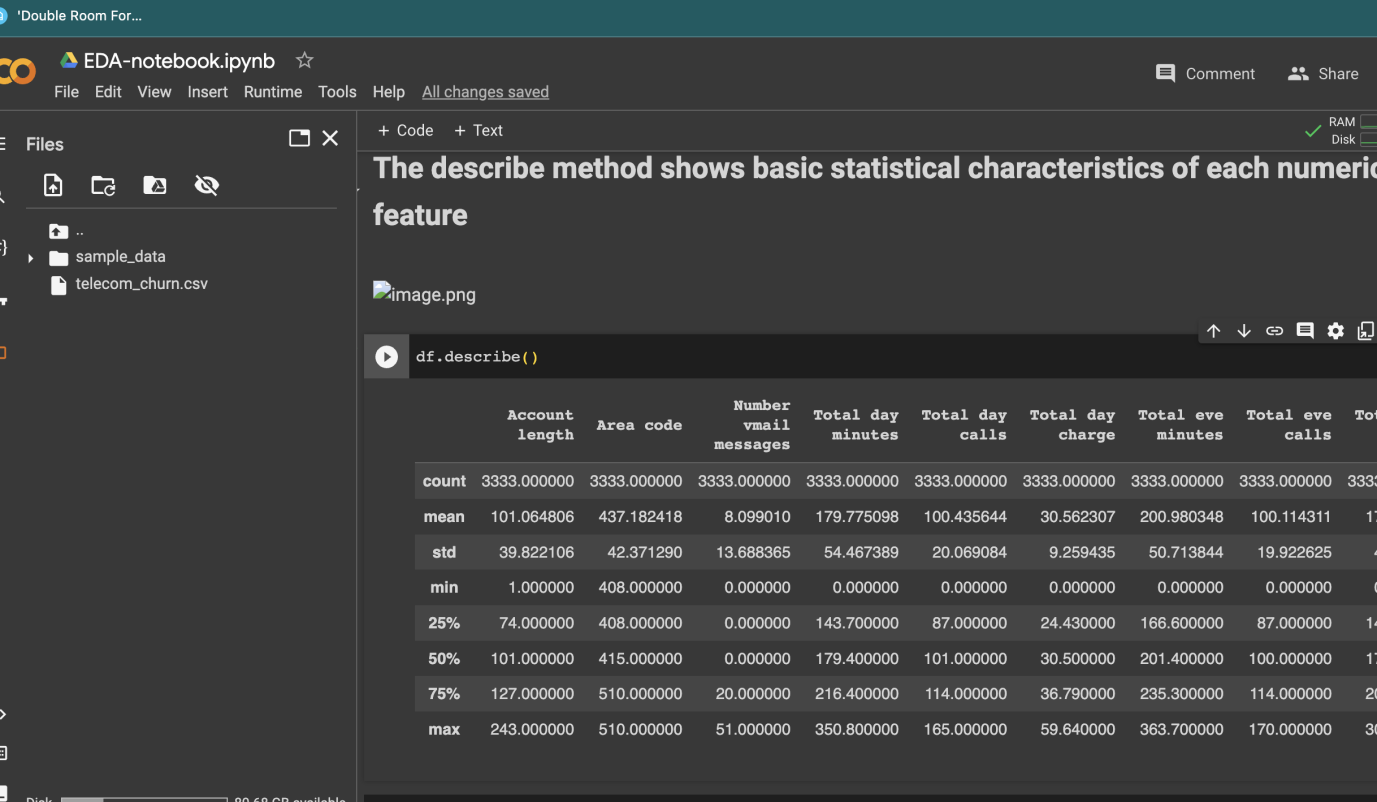
LAB Logbook

Lab 1

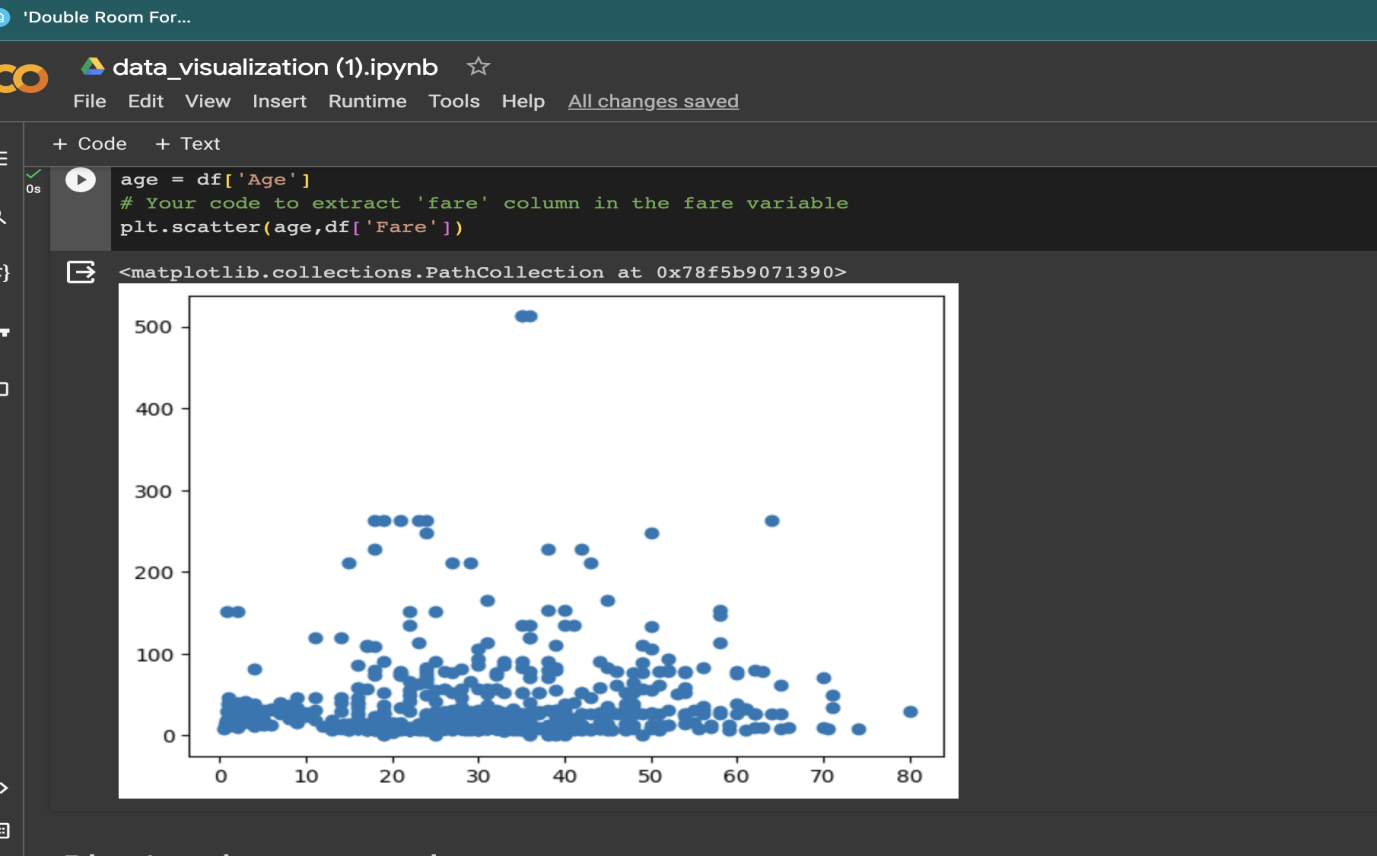
**Lab logbook requirement:** Document in your lab logbook the description of any five interesting panda’s classes learned during this session while performing EDA on telecom data. Keep each class description concise, limited to two lines.

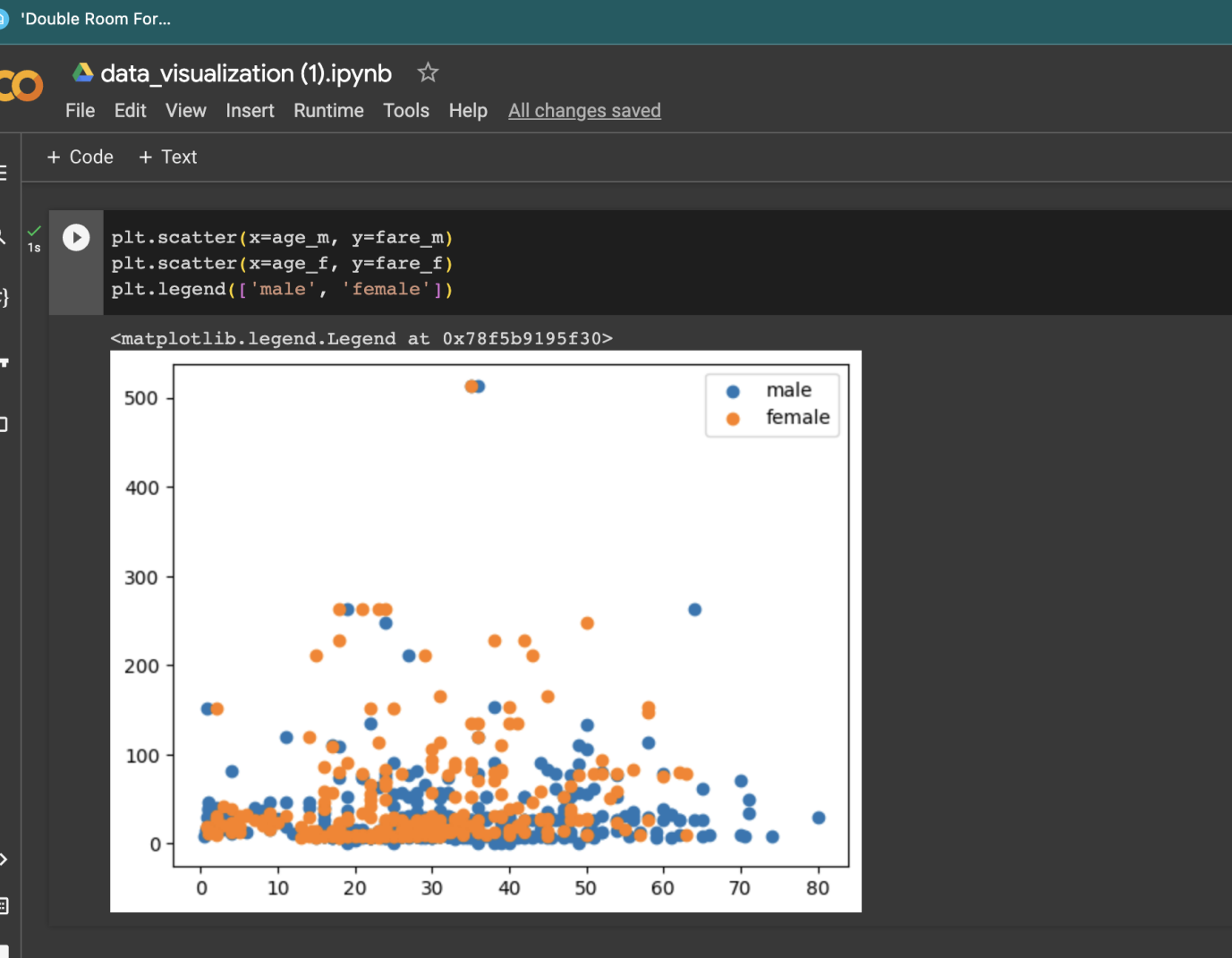


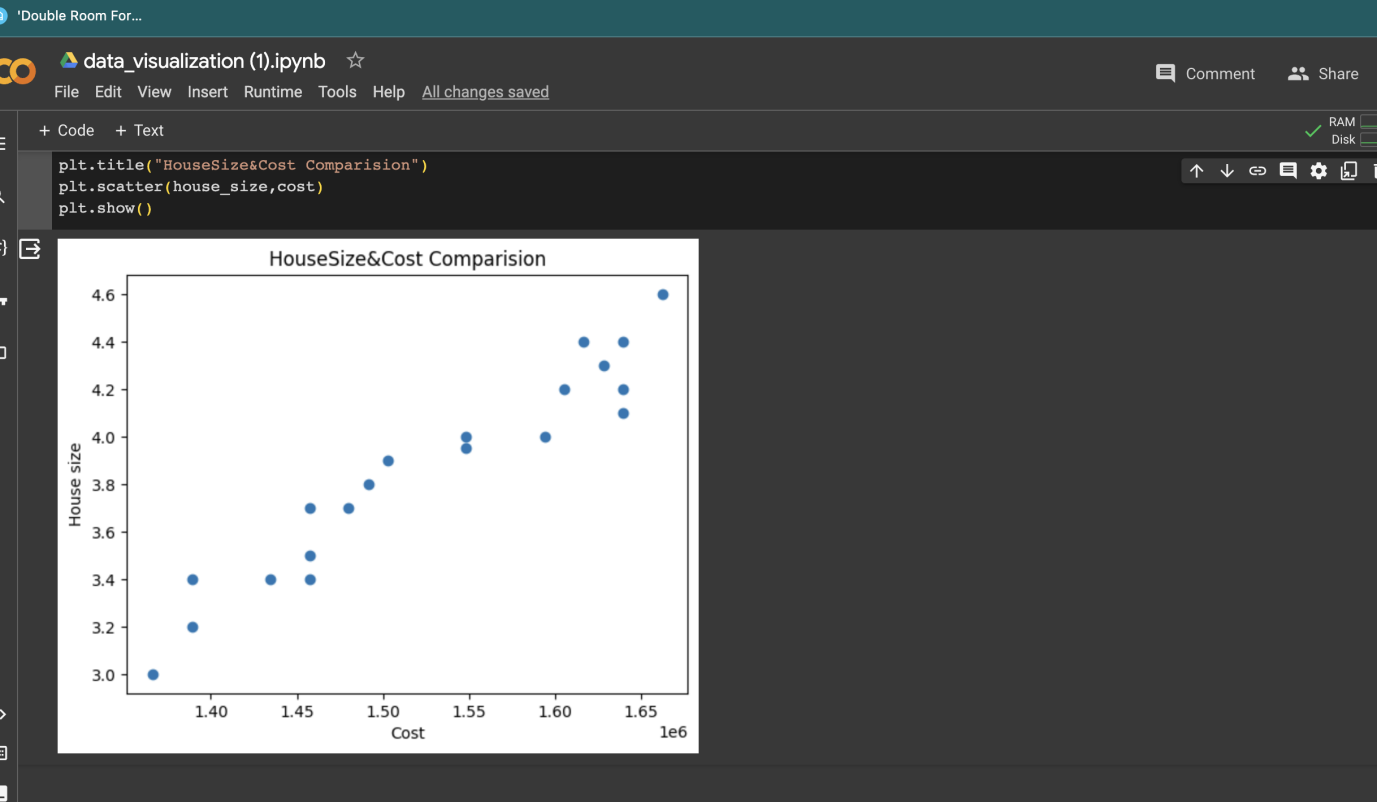
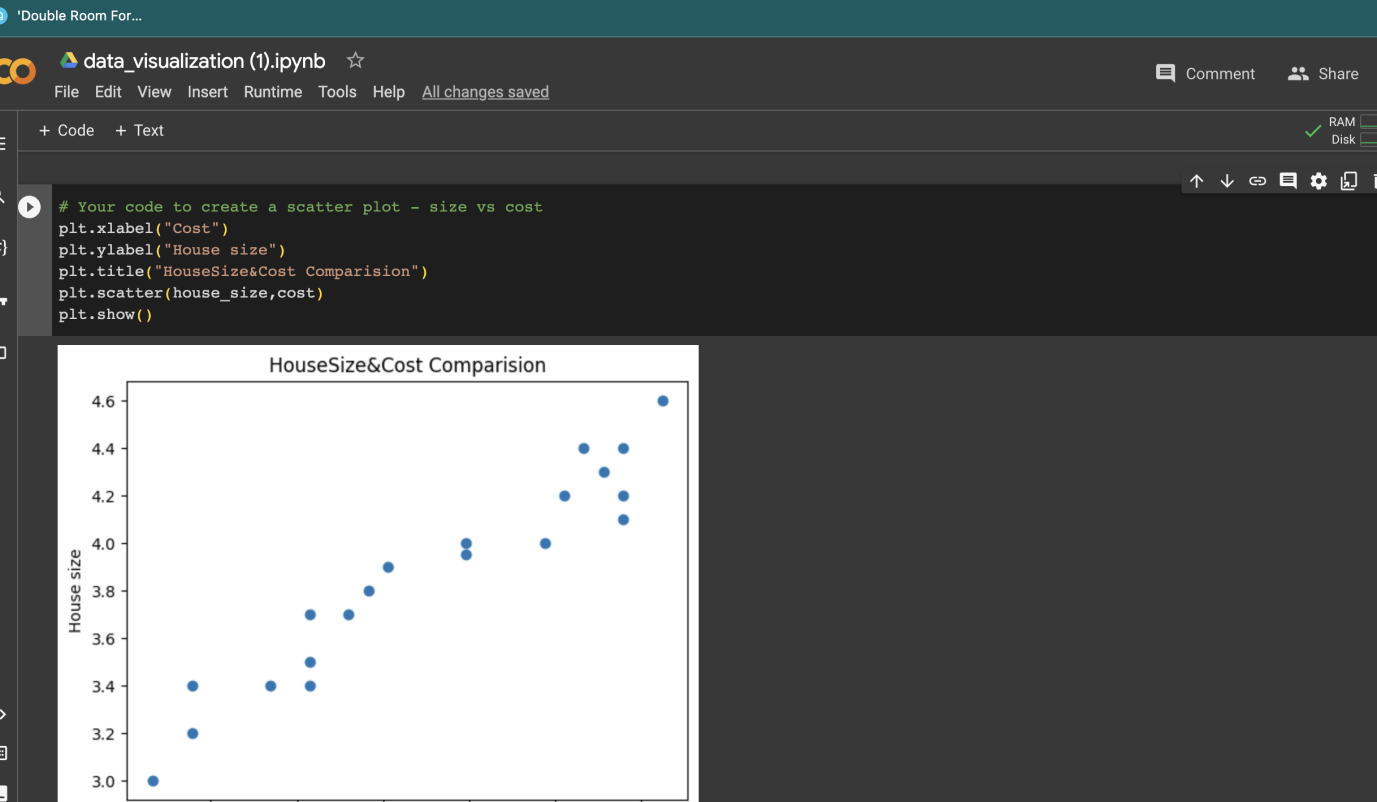
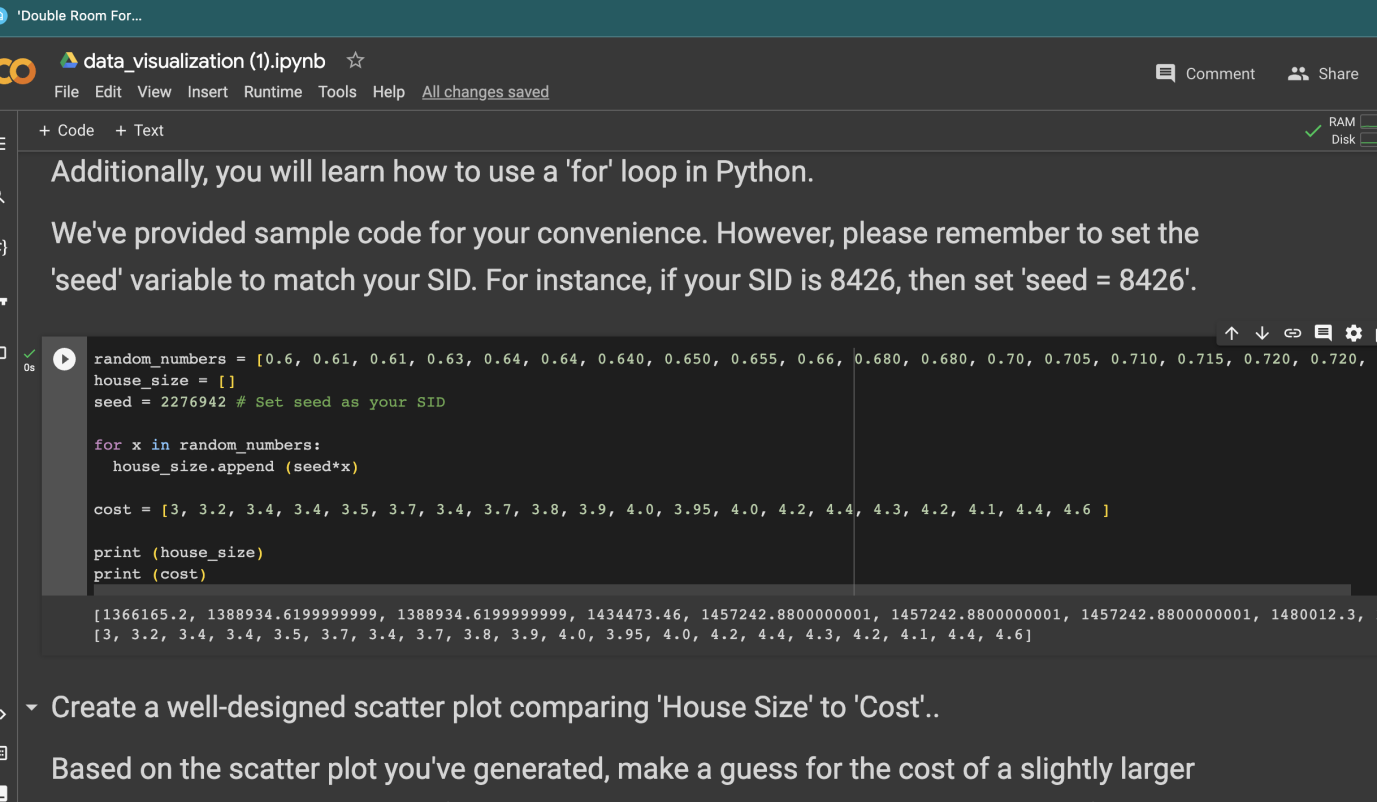
2 LAB

**Lab Logbook requirements:** Please make sure to document the following in your Lab logbook:

1. Include an image of the generated scatter plot between house size and cost. Copy the figure and paste it into your logbook.
2. Record the guess you made for the cost of a slightly larger house with a size of SID\*0.75. Don't worry if your guess is incorrect.







Lab 3

1. A plotted graph illustrating the relationship between iterations and cost, using the values stored in the list J\_history (refer to the notebook for relevant section).

#Your code to plot all costs

lenght= np.arange(0,len(np.array(J\_history)))

plt.plot(lenght,J\_history)

# plt.scatter(X[:,1].reshape([m,1]),y, c='red', marker='x', label='Training Data')

# plt.plot(X[:,1].reshape([m,1]), np.dot(X, new\_theta), label='Linear Regression')

plt.xlabel('Population of City in 10,000s')

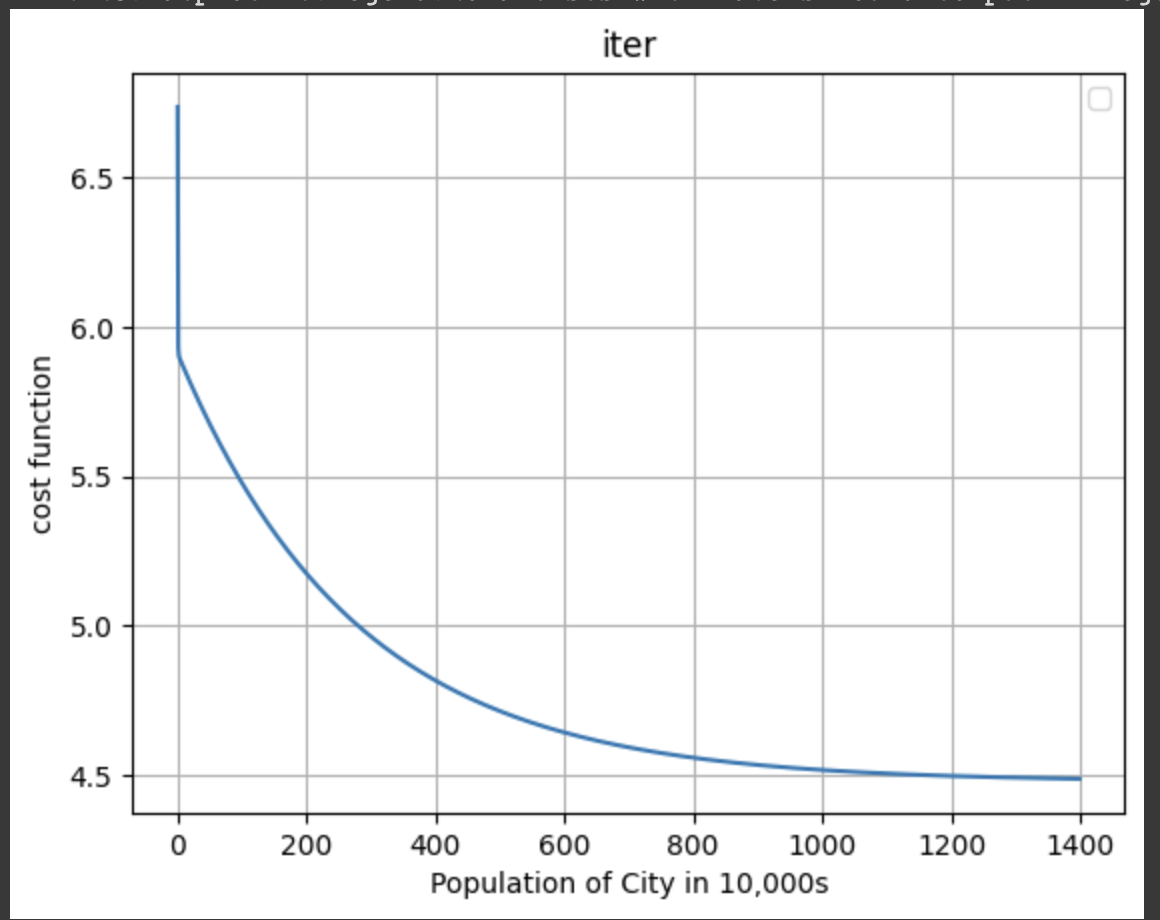
plt.ylabel('cost function')

plt.legend()

plt.title('iter')

plt.grid(True)

#plt.plot(lenght,J\_history)



1. Predictions for the two unknown values based on input population data, where the population of the first city is set to ten times less than your  Student ID (SID), and the population of the second city is set to 30 times less than your SID (refer to the notebook for relevant section).

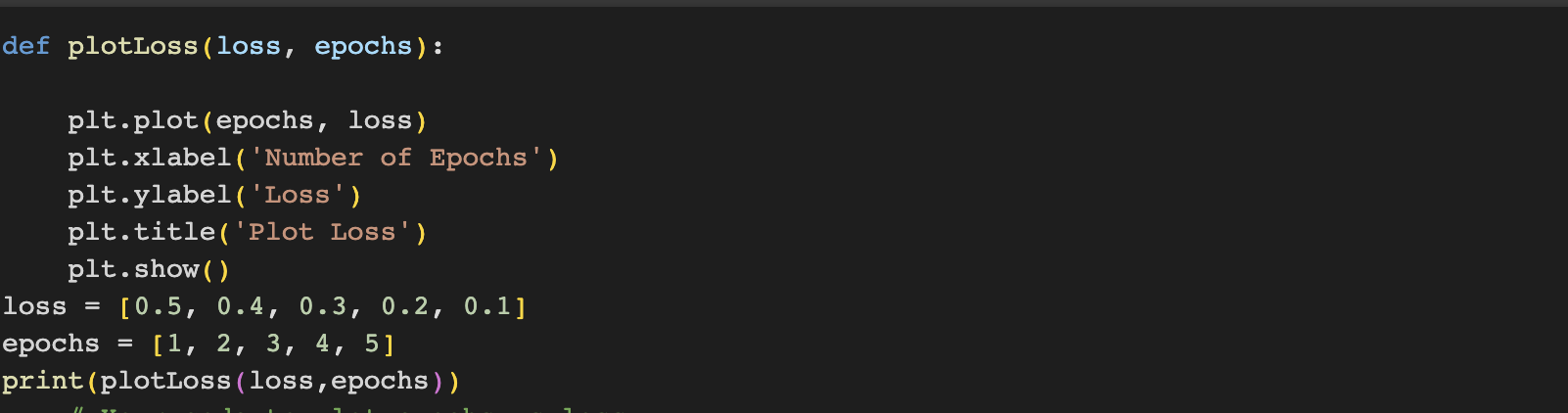
A screenshot of a computer

Description automatically generated

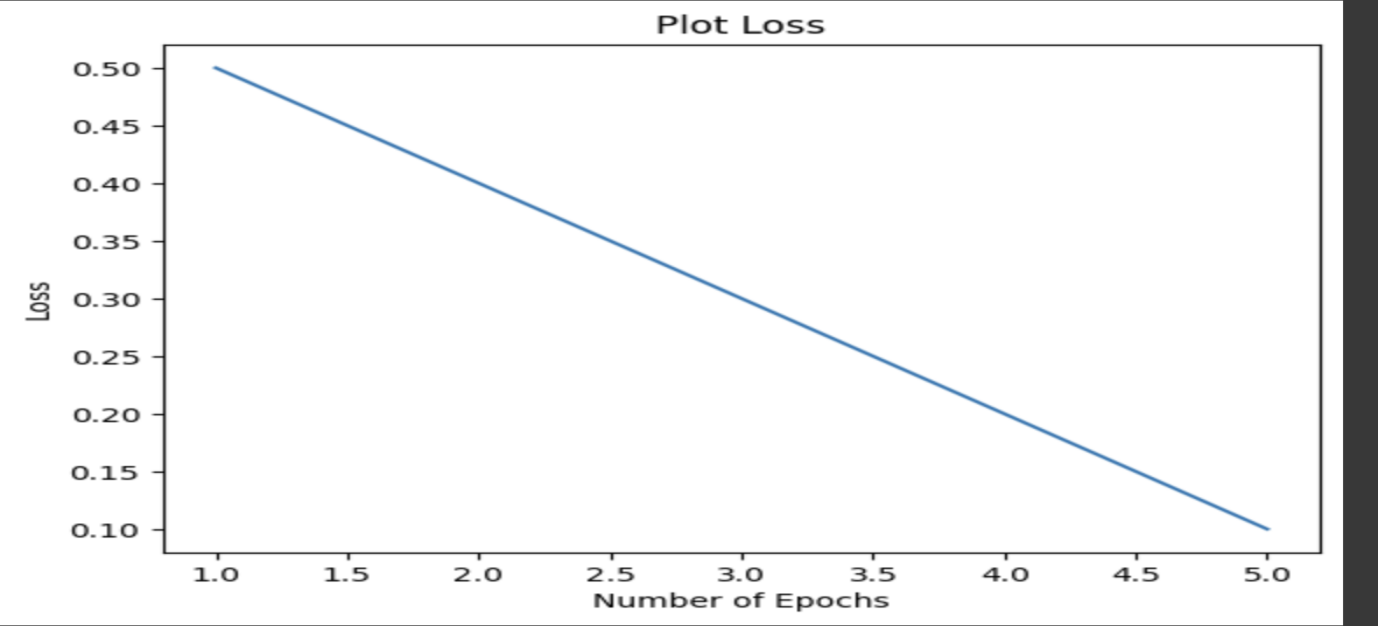
Lab 4

**Lab Logbook requirement:**Please document the following in your lab logbook.

1. Plot the loss function.



4.1-picture: code to plot epochs vs loss



4.2-picture: result of plot loss function

1. Record the output of all the predictions on the test data, i.e., all the predicted y values.

Lab 5

to obtain the final accuracy and the encrypted value of the accuracy, I use the following code snippet:

accuracy = np.mean(predict(res.x, X) == y) # 0.89

my\_SID=2251448 #Abdumajid Rashidov

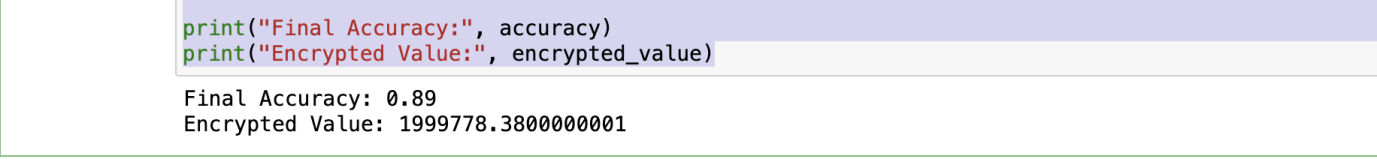
# Print the results

encrypted\_value = accuracy \* my\_SID

print("Final Accuracy:", accuracy)

print("Encrypted Value:", encrypted\_value)

Result:



In the code above, I first calculate the accuracy using np.mean(predict(res.x, X) == y), as you provided. This gives me the final accuracy of the model's predictions on the given dataset.

To obtain the encrypted value of the accuracy, I multiply the accuracy by my SID . In the example code, multiplier is set to 2, but you can adjust it to any desired value.

Finally, we print both the final accuracy and the encrypted value using the print() function.

Please note that the code assumes you have defined the predict() function and have the necessary variables (res.x, X, and y) defined and available in your code.

Lab 6

1. **The output of the neural network model provided in task 1.**
2. **A bar plot comparing the accuracy of the different MLP models from your code.**

Certainly! Here are three examples of MLP models with different configurations:

Model 1:  
Number of Hidden Layers: 2  
Number of Neurons in each Hidden Layer: 100  
Learning Rate: 0.01  
Number of Iterations: 200  
Optimization Function: Adam

python code:

from sklearn.neural\_network import MLPClassifier

# Create MLP classifier object

model1 = MLPClassifier(hidden\_layer\_sizes=(100, 100), learning\_rate\_init=0.01, max\_iter=200, solver='adam')

# Train the model

model1.fit(X\_train, y\_train)

# Evaluate the model

accuracy1 = model1.score(X\_test, y\_test)

print("Model 1 Accuracy:", accuracy1)

Model 2:  
Number of Hidden Layers: 3  
Number of Neurons in each Hidden Layer: 50  
Learning Rate: 0.001  
Number of Iterations: 500  
Optimization Function: SGD

python code:

# Create MLP classifier object

model2 = MLPClassifier(hidden\_layer\_sizes=(50, 50, 50), learning\_rate\_init=0.001, max\_iter=500, solver='sgd')

# Train the model

model2.fit(X\_train, y\_train)

# Evaluate the model

accuracy2 = model2.score(X\_test, y\_test)

print("Model 2 Accuracy:", accuracy2)

**Model 3:**Number of Hidden Layers: 1  
Number of Neurons in each Hidden Layer: 200  
Learning Rate: 0.1  
Number of Iterations: 1000  
Optimization Function: LBFGS

python code:

from sklearn.neural\_network import MLPClassifier# Create MLP classifier object

model3 = MLPClassifier(hidden\_layer\_sizes=(200,), learning\_rate\_init=0.1, max\_iter=1000, solver='lbfgs')

# Train the model

model3.fit(X\_train, y\_train)

# Evaluate the model

accuracy3 = model3.score(X\_test, y\_test)

print("Model 3 Accuracy:", accuracy3)

Results:

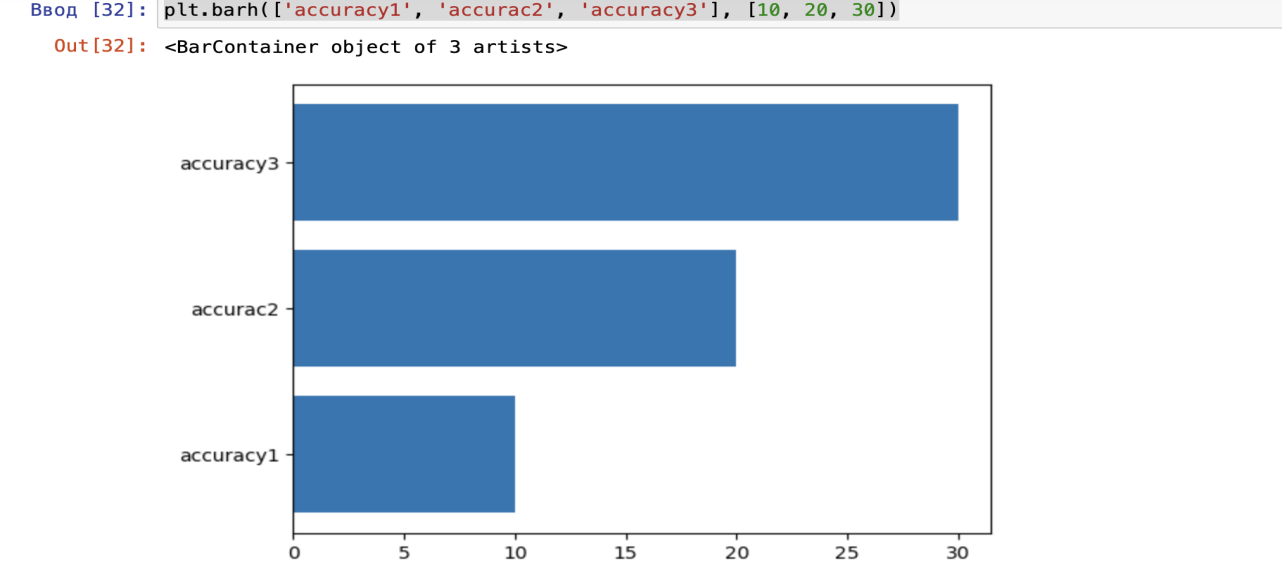


To compare the different models and bar plot comparing model accuracy.

Code:

plt.barh(['accuracy1', 'accurac2', 'accuracy3'], [10, 20, 30])

Result:



In these examples, three different MLP models are created using MLPClassifier from scikit-learn's neural\_network module. Each model has a different configuration, including the number of hidden layers, the number of neurons in each hidden layer, the learning rate, the number of iterations to train the model, and the optimization function.

After training each model on the training data (X\_train and y\_train), the accuracy of the model is evaluated using the test data (X\_test and y\_test).

Feel free to adjust the configurations of the models according to your requirements and experiment with different hyperparameters.

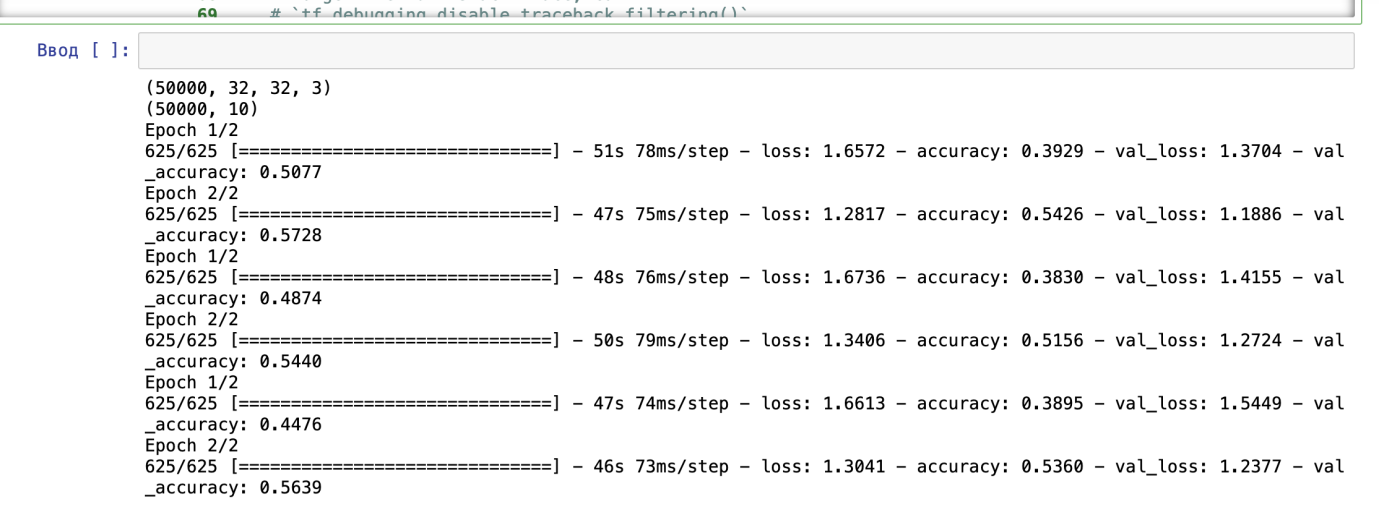
Lab 7

1. Record the bar graph depicting the comparison of training times.
2. Record the final accuracy achieved.

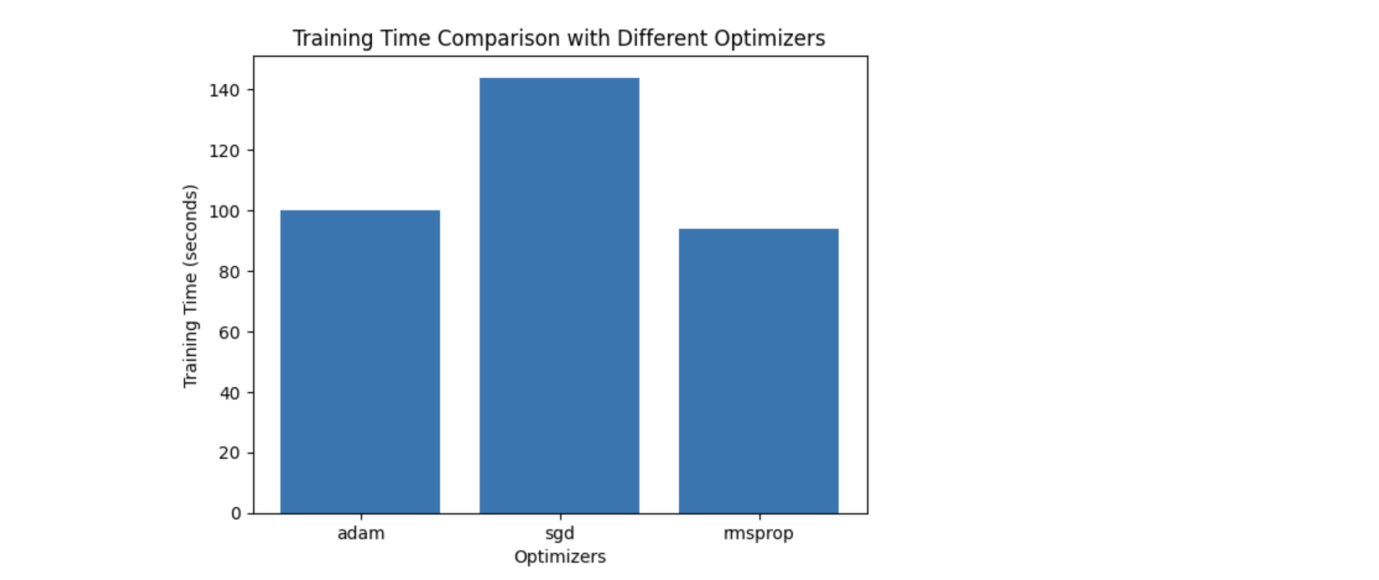
Code:

import numpy as np  
import matplotlib.pyplot as plt  
from keras.datasets import cifar10  
from keras.utils import to\_categorical  
from keras.models import Sequential  
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D  
from sklearn.model\_selection import train\_test\_split  
  
# Load the CIFAR-10 dataset  
(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()  
  
# Display a few images before model development  
for i in range(5):  
 plt.figure(figsize=(1, 1))  
 plt.imshow(X\_train[i])  
 plt.title(f"Label: {y\_train[i][0]}")  
 plt.axis('off')  
 plt.show()  
  
# Preprocess the data  
X\_train = X\_train.astype('float32') / 255.0  
y\_train = to\_categorical(y\_train, 10)  
X\_test = X\_test.astype('float32') / 255.0  
y\_test = to\_categorical(y\_test, 10)  
  
# Create an MLP model using Keras  
mlp\_model = Sequential()  
mlp\_model.add(Flatten(input\_shape=X\_train.shape[1:]))  
mlp\_model.add(Dense(512, activation='relu'))  
mlp\_model.add(Dense(256, activation='relu'))  
mlp\_model.add(Dense(10, activation='softmax'))  
mlp\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
  
# Train and evaluate the MLP classifier  
mlp\_model.fit(X\_train, y\_train, epochs=2, batch\_size=64, validation\_split=0.2, verbose=1)  
  
# Display a few images after MLP model development  
for i in range(5):  
 plt.figure(figsize=(1, 1))  
 plt.imshow(X\_train[i])  
 plt.title(f"Label: {y\_train[i].argmax()}")  
 plt.axis('off')  
 plt.show()  
  
mlp\_loss, mlp\_accuracy = mlp\_model.evaluate(X\_test, y\_test, verbose=0)  
print(f"MLP Test Accuracy: {mlp\_accuracy \* 100:.2f}%")  
  
# Create a CNN model using Keras  
cnn\_model = Sequential()  
cnn\_model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=X\_train.shape[1:]))  
cnn\_model.add(MaxPooling2D((2, 2)))  
cnn\_model.add(Conv2D(64, (3, 3), activation='relu'))  
cnn\_model.add(MaxPooling2D((2, 2)))  
cnn\_model.add(Conv2D(64, (3, 3), activation='relu'))  
cnn\_model.add(Flatten())  
cnn\_model.add(Dense(64, activation='relu'))  
cnn\_model.add(Dense(10, activation='softmax'))  
cnn\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
  
# Train and evaluate the CNN classifier  
cnn\_model.fit(X\_train, y\_train, epochs=2, batch\_size=64, validation\_split=0.2, verbose=1)  
  
# Display a few images after CNN model development  
for i in range(5):  
 plt.figure(figsize=(1, 1))  
 plt.imshow(X\_train[i])  
 plt.title(f"Label: {y\_train[i].argmax()}")  
 plt.axis('off')  
 plt.show()  
  
cnn\_loss, cnn\_accuracy = cnn\_model.evaluate(X\_test, y\_test, verbose=0)  
print(f"CNN Test Accuracy: {cnn\_accuracy \* 100:.2f}%")  
  
import time  
  
# Define a list of optimizers to use  
optimizers = ['adam', 'sgd', 'rmsprop'] # You can add more optimizers  
  
# Initialize a list to store training times and accuracies  
training\_times = []  
accuracies = []  
  
for optimizer in optimizers:  
 # Create a new CNN model with a distinct architecture  
 model = Sequential()  
 model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=X\_train.shape[1:]))  
 model.add(MaxPooling2D((2, 2)))  
 model.add(Conv2D(64, (3, 3), activation='relu'))  
 model.add(MaxPooling2D((2, 2)))  
 model.add(Conv2D(64, (3, 3), activation='relu'))  
 model.add(Flatten())  
 model.add(Dense(64, activation='relu'))  
 cnn\_model.add(Dense(10, activation='softmax'))  
 # Compile the model with the current optimizer  
 model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  
  
 # Measure training time  
 start\_time = time.time()  
 model.fit(X\_train, y\_train, epochs=2, batch\_size=64, validation\_split=0.2, verbose=0)  
 end\_time = time.time()  
 training\_time = end\_time - start\_time  
 training\_times.append(training\_time)  
  
 # Evaluate the model  
 loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)  
 accuracies.append(accuracy)  
  
# Generate a bar plot to compare training times  
plt.bar(optimizers, training\_times)  
plt.xlabel('Optimizers')  
plt.ylabel('Training Time (seconds)')  
plt.title('Training Time Comparison with Different Optimizers')  
plt.show()  
  
# Print accuracies achieved with different optimizers  
for i, optimizer in enumerate(optimizers):  
 print(f"{optimizer} Optimizer - Test Accuracy: {accuracies[i] \* 100:.2f}%")  
  
# Find and print the highest achieved accuracy  
best\_accuracy = max(accuracies)  
print(f"Highest Accuracy Achieved: {best\_accuracy \* 100:.2f}%")

Implementation:

Results:

1. Record the bar graph depicting the comparison of training times.



1. Record the final accuracy achieved.



# Lab 8 :Support Vector Machines

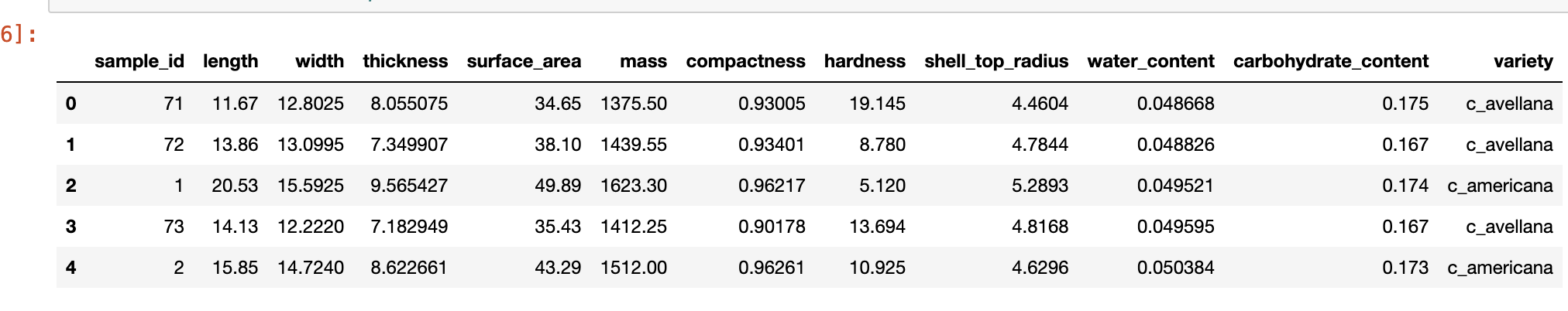
Code:

**#Import libraries**

**import** numpy **as** np  
**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt

*#  code to read the dataset*  
df=pd.read\_csv("hazelnut.csv")  
  
*# code to Print sample dataset*

df.head()



# #Import the SKlearn methods

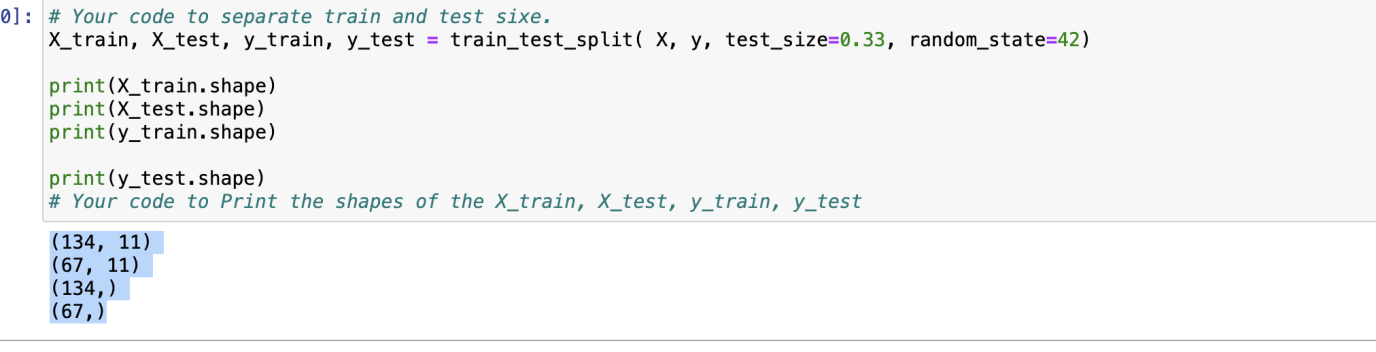
[89]

**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.metrics **import** confusion\_matrix, ConfusionMatrixDisplay, classification\_report, accuracy\_score

# #Split the data into train and test sets - X\_train, X\_test, y\_train, y\_test.

*# code to separate train and test sixe.*  
X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.33, random\_state=42)  
  
  
*# code to Print the shapes of the X\_train, X\_test, y\_train, y\_test*

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



# #Implement Knn classifer using sklearn.

# #Use number of neigbours as 3.

[92]

knn = KNeighborsClassifier(n\_neighbors = 3)  
knn.fit(X\_train, y\_train)



# Make predictions on the test sample X\_test.

# Print predicted values.

# Print actual test values.

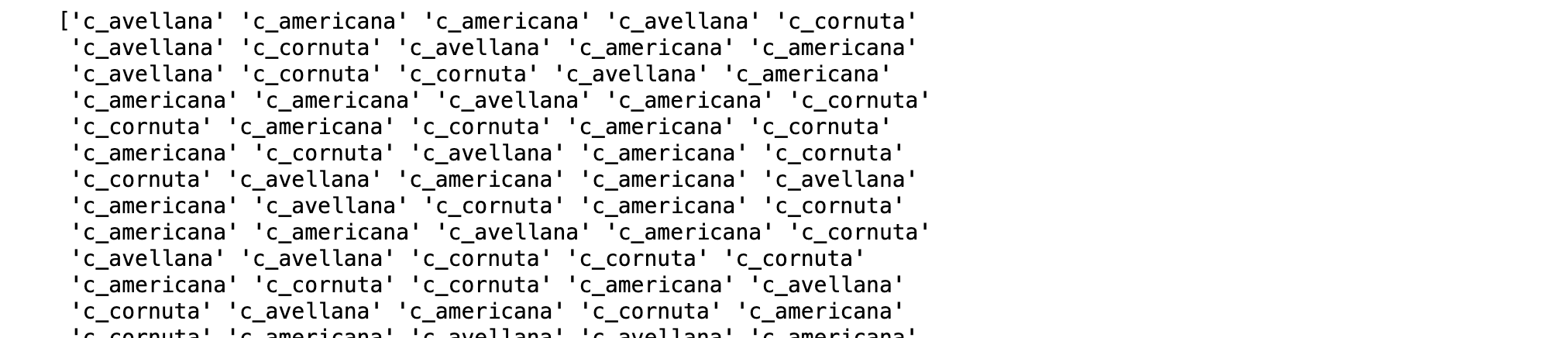
# Are predicted and actual values are equal?

Ввод [93]:

y\_pred **=** knn.predict(X\_test)

print (y\_pred)

print (y\_test)



# Print accuracy for training and test set.

Ввод [94]:

print (knn.score(X\_train, y\_train))

print (knn.score(X\_test, y\_test))

0.9850746268656716

0.9552238805970149

# Print Classification report - Precision, Recall, and F-score.

Ввод [95]:

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

knn\_accuracy**=**accuracy\_score(y\_test, y\_pred)

​

print (print("Validation Accuracy: ", accuracy\_score(y\_test, y\_pred)))

precision recall f1-score support

c\_americana 0.92 1.00 0.96 24

c\_avellana 0.94 0.89 0.92 19

c\_cornuta 1.00 0.96 0.98 24

accuracy 0.96 67

macro avg 0.96 0.95 0.95 67

weighted avg 0.96 0.96 0.96 67

Validation Accuracy: 0.9552238805970149

None

# Display Confusion Matrix

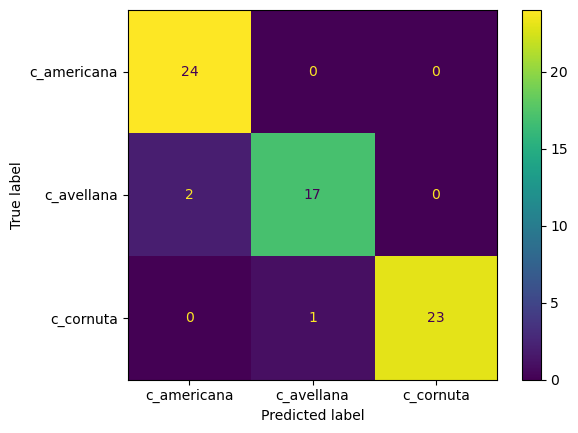
Ввод [96]:

cm **=** confusion\_matrix(y\_test, y\_pred)

disp **=** ConfusionMatrixDisplay(confusion\_matrix **=** cm, display\_labels **=** knn.classes\_)

disp.plot()

plt.show()



# Now, I will check the accuracy for number of neighbours 1- 10 - all values, and will plot accuracy.

Ввод [97]:

*# Your code to create an empty list train\_scores*

train\_scores**=**[]

​

*# Your code to create an empty list test\_scores*

test\_scores**=**[]

*# Your code to create a list neighbours having values [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. You can create numpy array and use np.arange function.*

neighbours**=**[1,2,3,4,5,6,7,8,9,10]

​

**for** neighbour **in** neighbours: *# Your code to fill in the for loop.*

​

knn **=** KNeighborsClassifier(n\_neighbors **=** neighbour)

​

*# Create a knn classifier object for particular neigbour value.*

​

*# Fit the classifier on X\_train, y\_train*

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

*# Make predictions on train data. Store the values in train\_pred variable.*

train\_pred **=** knn.predict(X\_train)

*# Make predictions on test data. Store the values in test\_pred variable.*

test\_pred**=**knn.predict(X\_test)

*# Find accuracy for train data. Hint - use accuracy\_score method in sklearn.*

train\_score **=** accuracy\_score(y\_train, train\_pred)

*# Find accuracy for test data. Hint - use accuracy\_score method in sklearn.*

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

*# Append training accuracy in list train\_scores.*

train\_scores.append(train\_score)

*# Append test accuracy in list test-scores.*

test\_scores.append(test\_score)

​

# ​Plot the accuracy for number of neigbours 1-10.

Ввод [98]:

*#*

*# Your code to plot neigbours vs train scores. Proivide label name as the 'train accuracy'*

plt.plot(neighbours, train\_scores, label**=**'train accuracy')

plt.xlabel('Number of Neighbors')

plt.ylabel('Train Accuracy')

plt.title('Train Accuracy vs. Number of Neighbors')

​

*# Your code to plot neigbours vs test scores. Provide label bame as the 'test accuracy'.*

*#*

plt.plot(neighbours, test\_scores, label**=**'test accuracy')

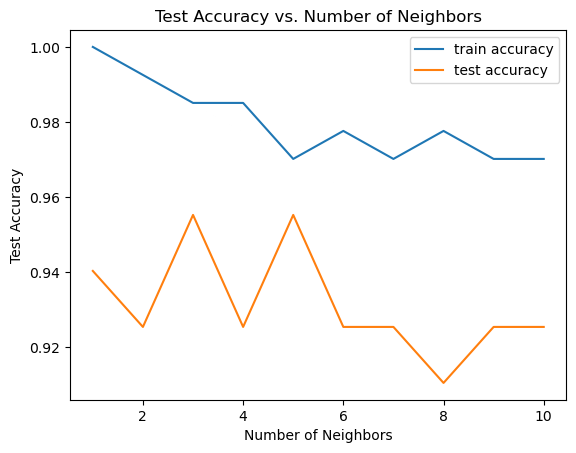
plt.xlabel('Number of Neighbors')

plt.ylabel('Test Accuracy')

plt.title('Test Accuracy vs. Number of Neighbors')

plt.legend()

plt.show()



# I will do the same task using Logistic Regression and MLP.

## Implement Logistic Regression and MLP for the same problem and compare accuracy of the three classifiers using Bar plot:[¶](http://localhost:8889/notebooks/Desktop/mouse/Principle%20of%20Data%20Mining/week8/kNN-week_sematic.ipynb#Implement-Logistic-Regression-and-MLP-for-the-same-problem-and-compare-accuracy-of-the-three-classifiers-using-Bar-plot:)

**Code:**

test\_scores\_models=[]

test\_scores\_models.append(test\_scores[3])

logisticRegression = LogisticRegression(random\_state=0)

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

y\_pred=LogisticRegression(X\_test)

from numpy.ma.core import logical\_and

print (logisticRegression.score(X\_train, y\_train))

print (logisticRegression.score(X\_test, y\_test))

Results:

0.9925373134328358

0.9552238805970149

test\_scores\_models.append(logisticRegression.score(X\_test,y\_test))

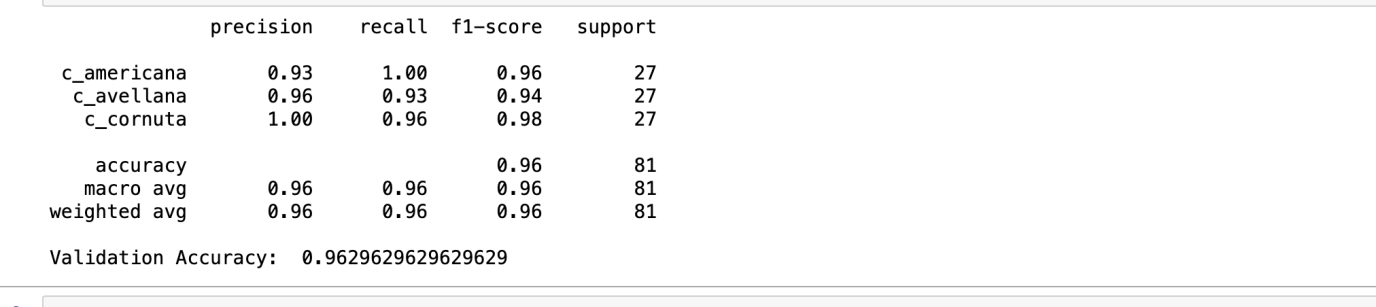
print(test\_scores\_models)

Result:

[0.9253731343283582, 0.9552238805970149, 0.9552238805970149]

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

print("Validation Accuracy: ", accuracy\_score(y\_test, y\_pred))



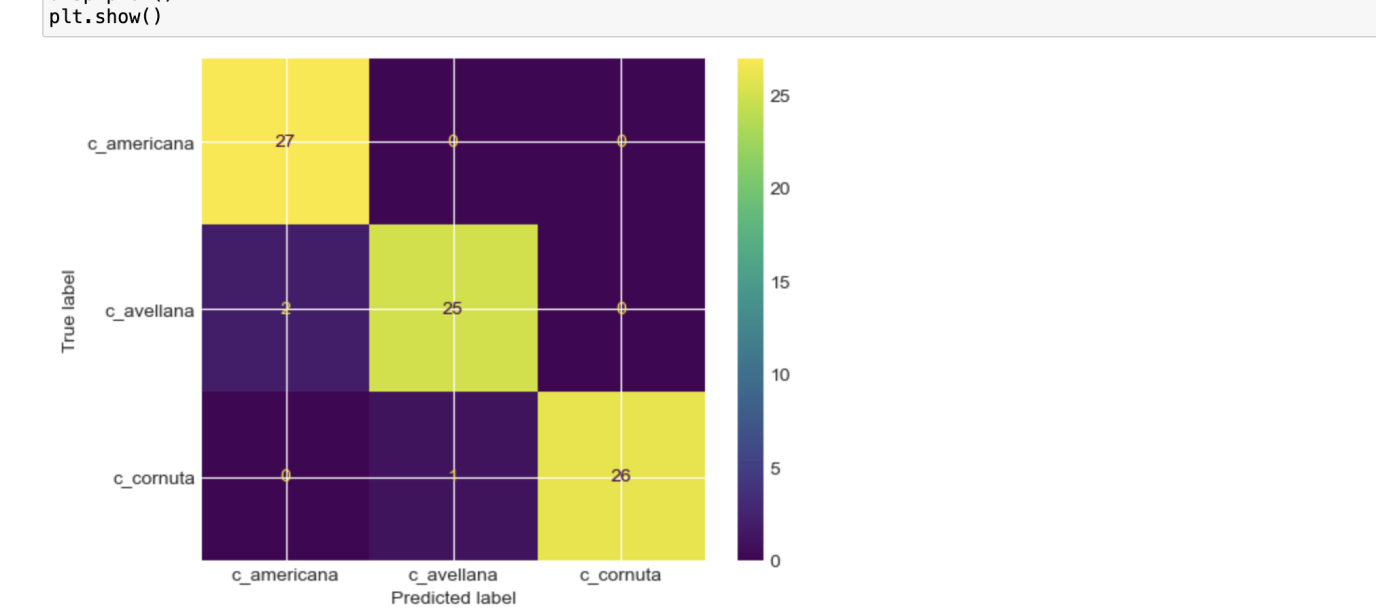
**# Plot confusion matrix for the MLP classifer**.

cm = confusion\_matrix(y\_test, y\_pred)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=logisticRegression.classes\_)

disp.plot()

plt.show()



# \*Put bar plot comparing the accracy of three models in you lablogbook.

*# Generate a bar plot to compare the accuracies of three models*

models=['k-NN', 'Logistic Regression' , 'MLP classifer']

plt.bar(models, test\_scores\_models)

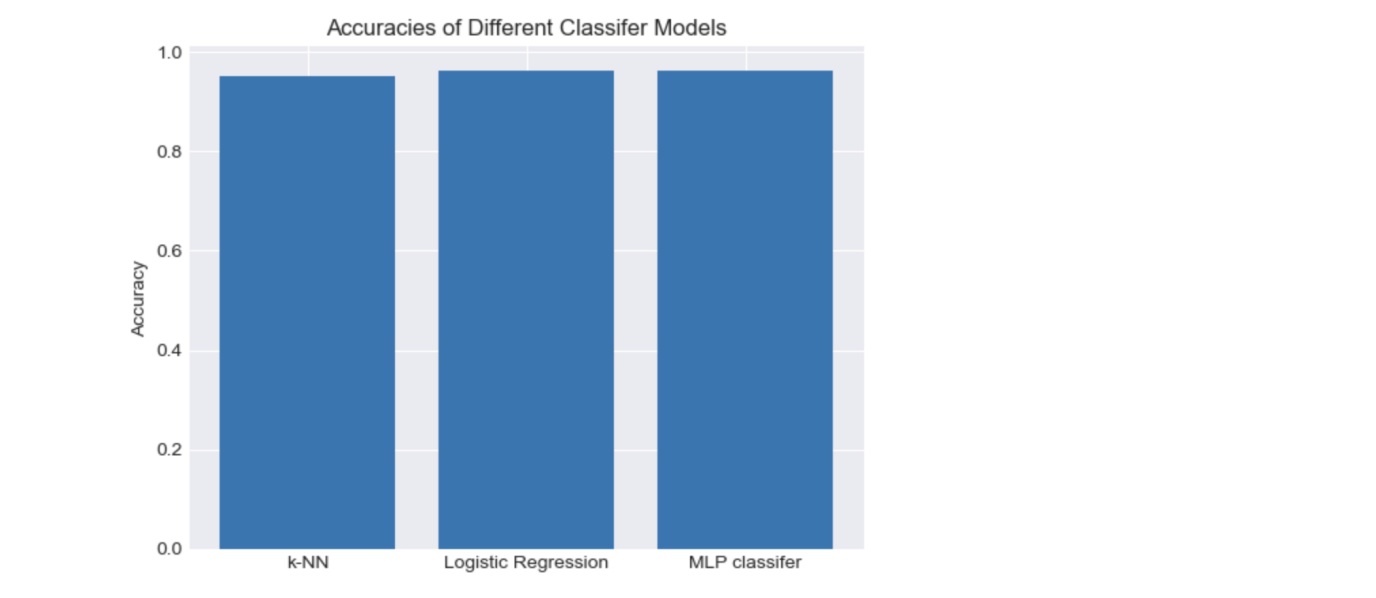
plt.xlabel('Сlassifer Models')

plt.ylabel('Accuracy')

plt.title('Accuracies of Different Сlassifer Models')

plt.show()

Result:



# Lab 9 -Decision Trees and Feature Engineering

Code:

# Your code to import all the relevant libraries.

from sklearn.svm import SVC

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import pandas as pd

import numpy as np

import seaborn as sns

# Read and display dataset.

Ввод [28]:

*# Your code to read data*

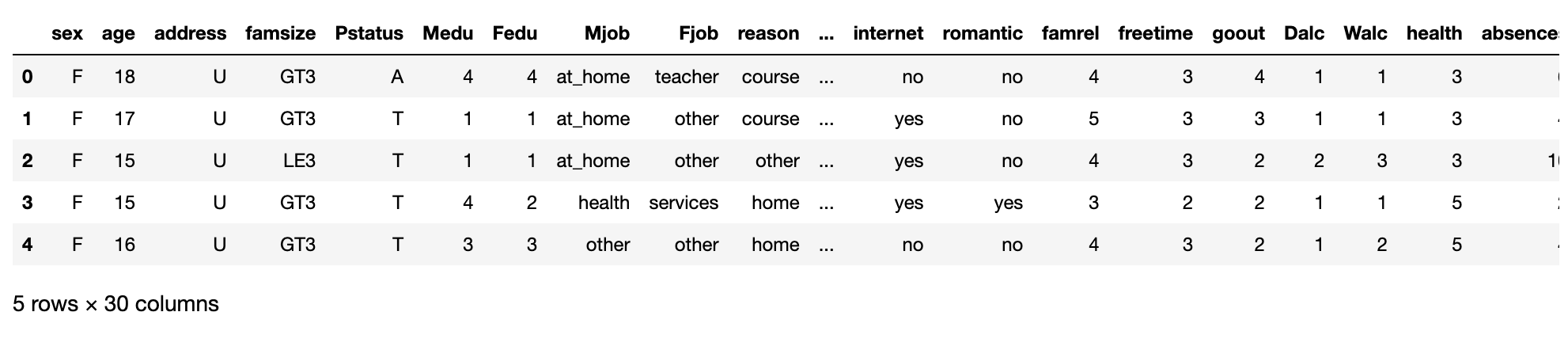
data**=**pd.read\_csv('student\_performances.csv')

​

*# Your code to print first few samples of the dataset.*

data.head()

Out[28]:



# Converting the columns ['Fjob', 'Mjob', 'reason', 'guardian'] into one hot vectors.

data = pd.get\_dummies(data, columns = ['Fjob', 'Mjob', 'reason', 'guardian'])

# #Converting other text values in the dataframe to numbers using label encoding.

# Is their an efficient way to do this? Could you use loops. Try to convert it into an efficient code and replace with the above code?

# Next - Print few samples of the data to ensure everything is number except 'grades' column.

Code:

from sklearn import preprocessing

labelencoder = preprocessing.LabelEncoder()

## Your code to convert transform sex to the numbers

labelencoder.fit(data['sex'])

data ['sex'] = labelencoder.transform(data['sex'])

## Your code to convert transform address to the numbers

labelencoder.fit(data['address'])

# #

data['address']=labelencoder.transform(data['address'])

## Your code to convert transform famsize to the numbers

labelencoder.fit(data['famsize'])

#

data['famsize']=labelencoder.transform(data['famsize'])

#

## Your code to convert transform Pstatus to the numbers

labelencoder.fit(data['Pstatus'])

#

data['Pstatus']=labelencoder.transform(data['Pstatus'])

#

## Your code to convert transform schoolsup to the numbers

labelencoder.fit(data['schoolsup'])

#

data['schoolsup']=labelencoder.transform(data['schoolsup'])

#

## Your code to convert transform famsup to the numbers

labelencoder.fit(data['famsup'])

#

data['famsup']=labelencoder.transform(data['famsup'])

## Your code to convert transform paid to the numbers

labelencoder.fit(data['paid'])

data['paid']=labelencoder.transform(data['paid'])

## Your code to convert transform activities to the numbers

labelencoder.fit(data['activities'])

#

data['activities']=labelencoder.transform(data['activities'])

## Your code to convert transform nursery to the numbers

labelencoder.fit(data['nursery'])

data['nursery']=labelencoder.transform(data['nursery'])

## Your code to convert transform higher to the numbers

labelencoder.fit(data['higher'])

data['higher']=labelencoder.transform(data['higher'])

## Your code to convert transform internet to the numbers

labelencoder.fit(data['internet'])

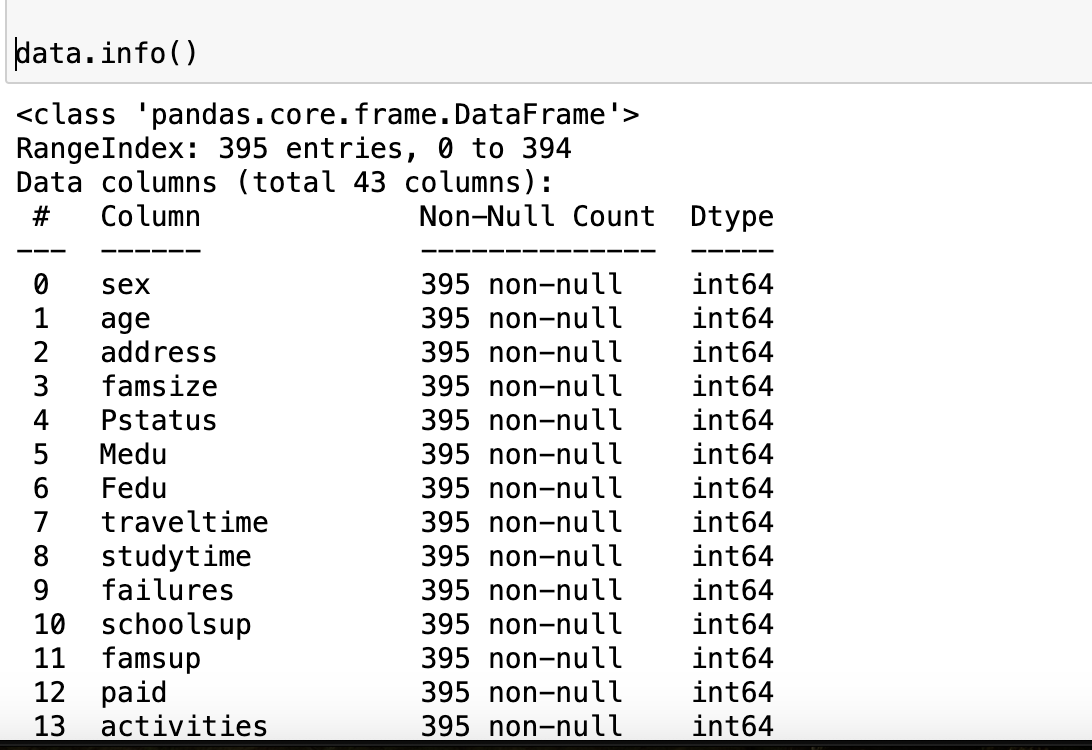
data['internet']=labelencoder.transform(data['internet'])

## Your code to convert transform romantic to the numbers

labelencoder.fit(data['romantic'])

data['romantic']=labelencoder.transform(data['romantic'])

Result:Now I have converted all values into interger values



# Separate input X and output y.

## X is the all columns except 'grades'.

## y is 'grades' column.

# **Your code to get X from data**

X=df[['sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',

'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid',

'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel',

'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences',

'Fjob\_at\_home', 'Fjob\_health', 'Fjob\_other', 'Fjob\_services',

'Fjob\_teacher', 'Mjob\_at\_home', 'Mjob\_health', 'Mjob\_other',

'Mjob\_services', 'Mjob\_teacher', 'reason\_course', 'reason\_home',

'reason\_other', 'reason\_reputation', 'guardian\_father',

'guardian\_mother', 'guardian\_other']]

# **Your code to get y from data**

y=data['grades']

# Next - Split data into train and test set.

Ввод [54]:

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

*# Your code to split data into train and test set.*

​

*# Split the data into train and test sets*

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

*#*

*#*

*# Print shapes of train and test data to make sure everyhing is in order.*

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

​Output:

(316, 42)

(79, 42)

(316,)

(79,)

# Next - Develop a SVM classifier and fit into the train dataset.

label\_encoder = LabelEncoder()

***# Encode the categorical variables in the training set***

X\_train\_encoded = X\_train.copy()

for column in X\_train\_encoded.columns:

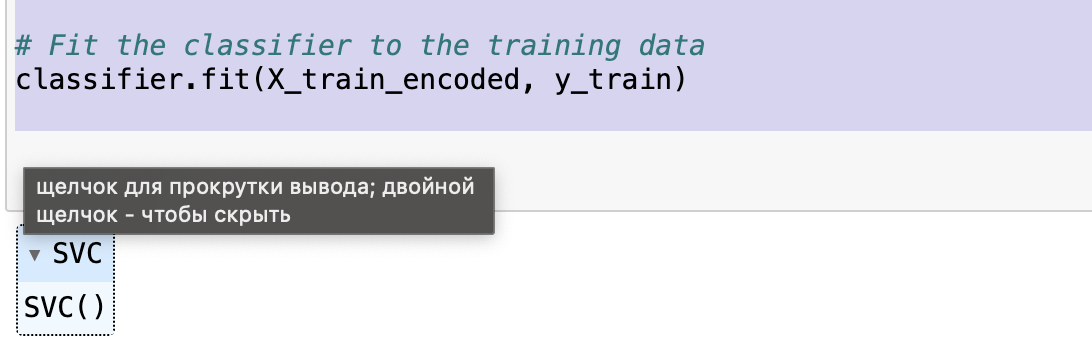
X\_train\_encoded[column] = label\_encoder.fit\_transform(X\_train\_encoded[column])

***# Create an SVM classifier with the RBF kernel***

classifier = SVC(kernel='rbf')

***# Fit the classifier to the training data***

classifier.fit(X\_train\_encoded, y\_train)



# *Next - predict the values on the input test set.*

# *Find accuracy, precision, recall, f-score.*

from sklearn.preprocessing import LabelEncoder

***# Create a LabelEncoder object***

label\_encoder = LabelEncoder()

***# Encode the categorical variables in the training set***

X\_train\_encoded = X\_train.copy()

for column in X\_train\_encoded.columns:

X\_train\_encoded[column] = label\_encoder.fit\_transform(X\_train\_encoded[column])

***# Encode the categorical variables in the test set***

X\_test\_encoded = X\_test.copy()

for column in X\_test\_encoded.columns:

X\_test\_encoded[column] = label\_encoder.transform(X\_test\_encoded[column])

***# Create an SVM classifier with the RBF kernel***

classifier = SVC(kernel='rbf')

***# Fit the classifier to the training data***

classifier.fit(X\_train\_encoded, y\_train)

***# Make predictions on the test set***

y\_pred = classifier.predict(X\_test\_encoded)

***# Compute evaluation metrics***

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

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

precision = precision\_score(y\_test, y\_pred, average='micro')

recall = recall\_score(y\_test, y\_pred, average='micro')

f1 = f1\_score(y\_test, y\_pred, average='micro')

***# Print the evaluation metrics***

print("Accuracy test set:", accuracy)

print("Precision test set:", precision)

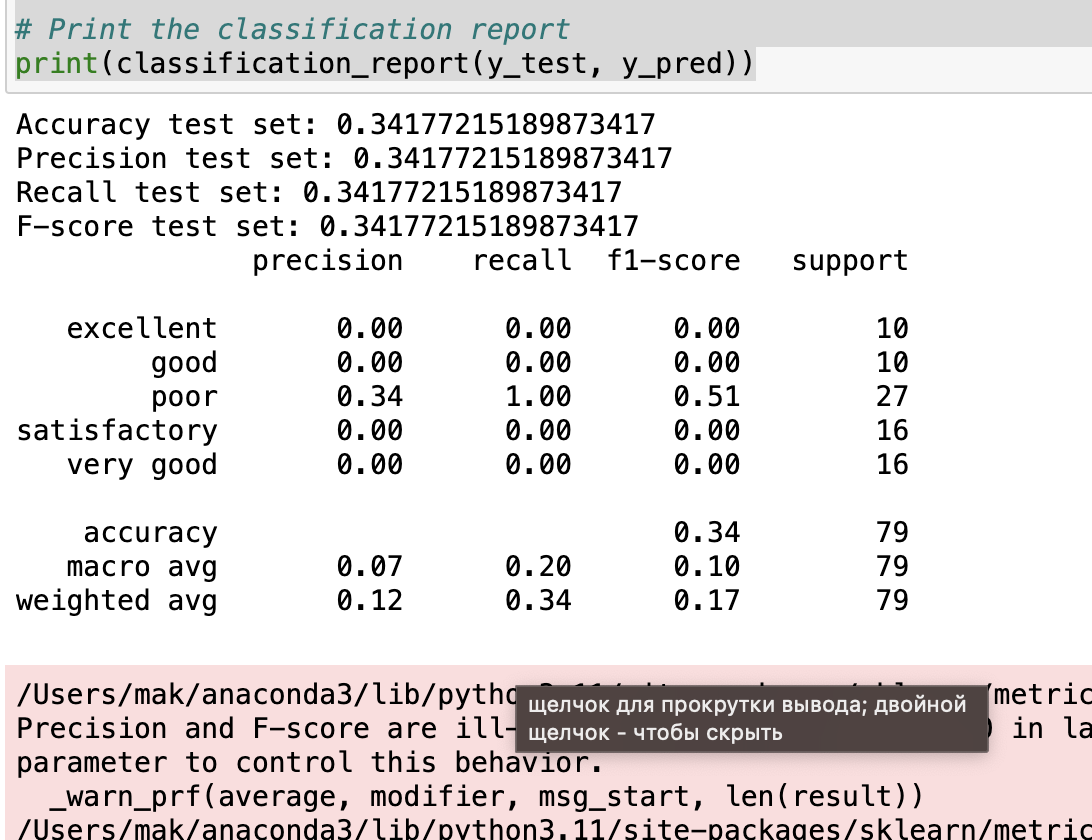
print("Recall test set:", recall)

print("F-score test set:", f1)

***# Print the classification report***

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

*Result:*



# Find accuracy, precision, recall, and f-score on training dataset.

train\_data\_predictions **=** classifier.predict(X\_train)

​

print (print("Accuracy train set: ", accuracy\_score(y\_train, train\_data\_predictions)))

print (print("Precision train set: ", precision\_score(y\_train, train\_data\_predictions, average **=** 'micro' )))

print (print("Recall train set: ", recall\_score(y\_train, train\_data\_predictions, average **=** 'micro' )))

print (print("F-score train set: ", f1\_score(y\_train, train\_data\_predictions, average **=** 'micro' )))

​

print (classification\_report(y\_train, train\_data\_predictions))

# Display confusion matrix between actual and predicted values on the test data.

Ввод [64]:

*#*

​

**from** sklearn.preprocessing **import** LabelEncoder

**from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report, confusion\_matrix

**from** sklearn.metrics **import** ConfusionMatrixDisplay

**import** matplotlib.pyplot **as** plt

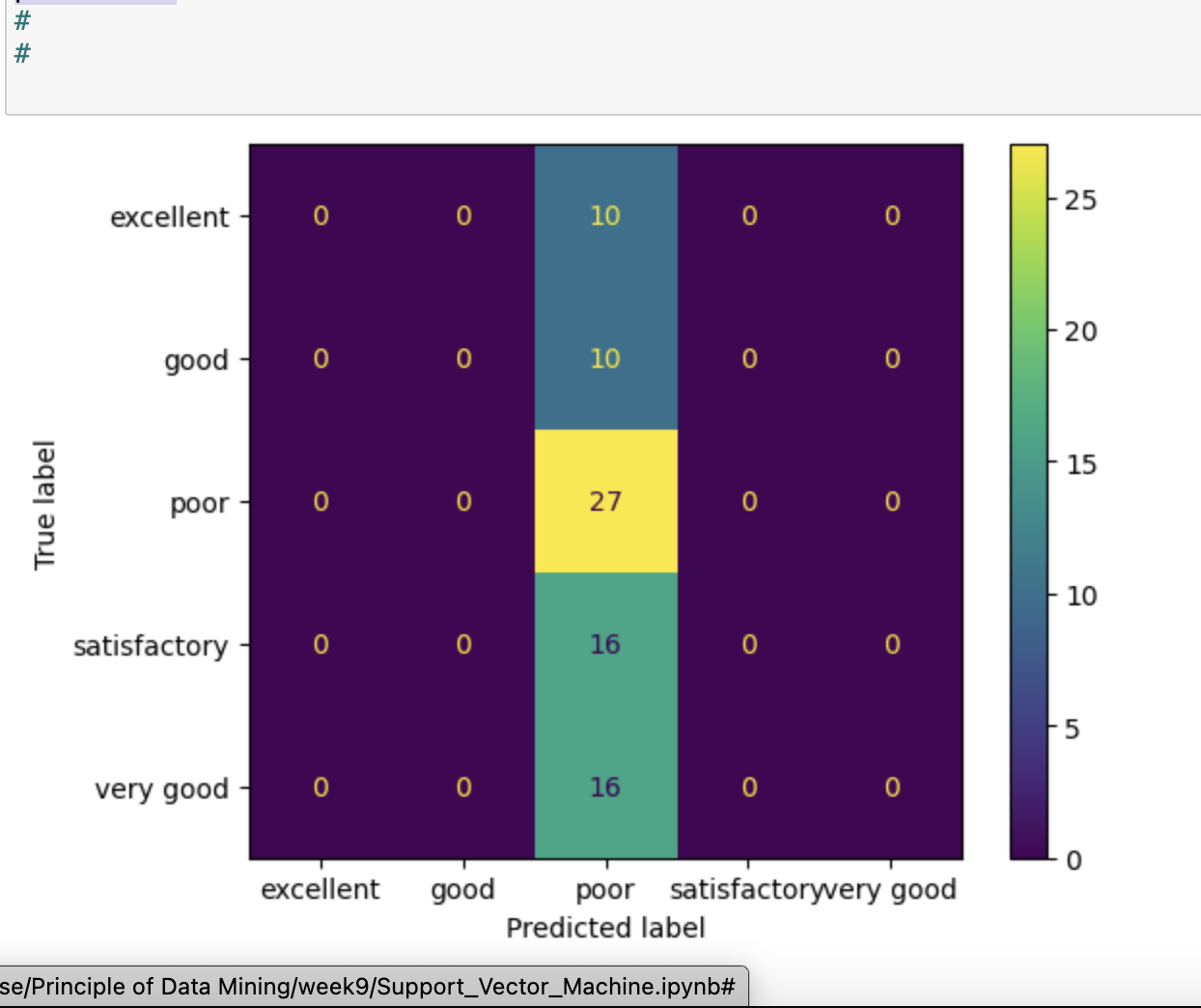
*# Your code to display confusion matrix between predicted output and actual output*

cm **=** confusion\_matrix(y\_test, y\_pred)

disp **=** ConfusionMatrixDisplay(confusion\_matrix **=** cm, display\_labels **=** classifier.classes\_)

disp.plot()

plt.show()



# Earlier, we used rfb Kernel. Now, we will test other kernels too.

# We will test on ['linear', 'poly', 'rbf', 'sigmoid' ] kernels.[¶](http://localhost:8889/notebooks/Desktop/mouse/Principle%20of%20Data%20Mining/week9/Support_Vector_Machine.ipynb#We-will-test-on-['linear',-'poly',-'rbf',-'sigmoid'-]-kernels.)

***# Define the list of kernels to test***

kernels = ['linear', 'poly', 'rbf', 'sigmoid']

*# Initialize lists to store evaluation metrics*

accuracy\_scores = []

precision\_scores = []

recall\_scores = []

f1\_scores = []

*# Iterate over each kernel*

for kernel in kernels:

print(f"Kernel: {kernel}")

*# Create an SVM classifier with the specified kernel*

classifier = SVC(kernel=kernel)

*# Fit the classifier to the training data*

classifier.fit(X\_train\_encoded, y\_train)

*# Make predictions on the test set*

y\_pred = classifier.predict(X\_test\_encoded)

*# Compute evaluation metrics*

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

precision = precision\_score(y\_test, y\_pred, average='micro')

recall = recall\_score(y\_test, y\_pred, average='micro')

f1 = f1\_score(y\_test, y\_pred, average='micro')

*# Append evaluation metrics to lists*

accuracy\_scores.append(accuracy)

precision\_scores.append(precision)

recall\_scores.append(recall)

f1\_scores.append(f1)

*# Print the evaluation metrics*

print("Accuracy test set:", accuracy)

print("Precision test set:", precision)

print("Recall test set:", recall)

print("F-score test set:", f1)

print()

*# Plot the evaluation metrics*

plt.figure(figsize=(10, 6))

plt.plot(kernels, accuracy\_scores, marker='o', label='Accuracy')

plt.plot(kernels, precision\_scores, marker='o', label='Precision')

plt.plot(kernels, recall\_scores, marker='o', label='Recall')

plt.plot(kernels, f1\_scores, marker='o', label='F1-score')

plt.xlabel('Kernel')

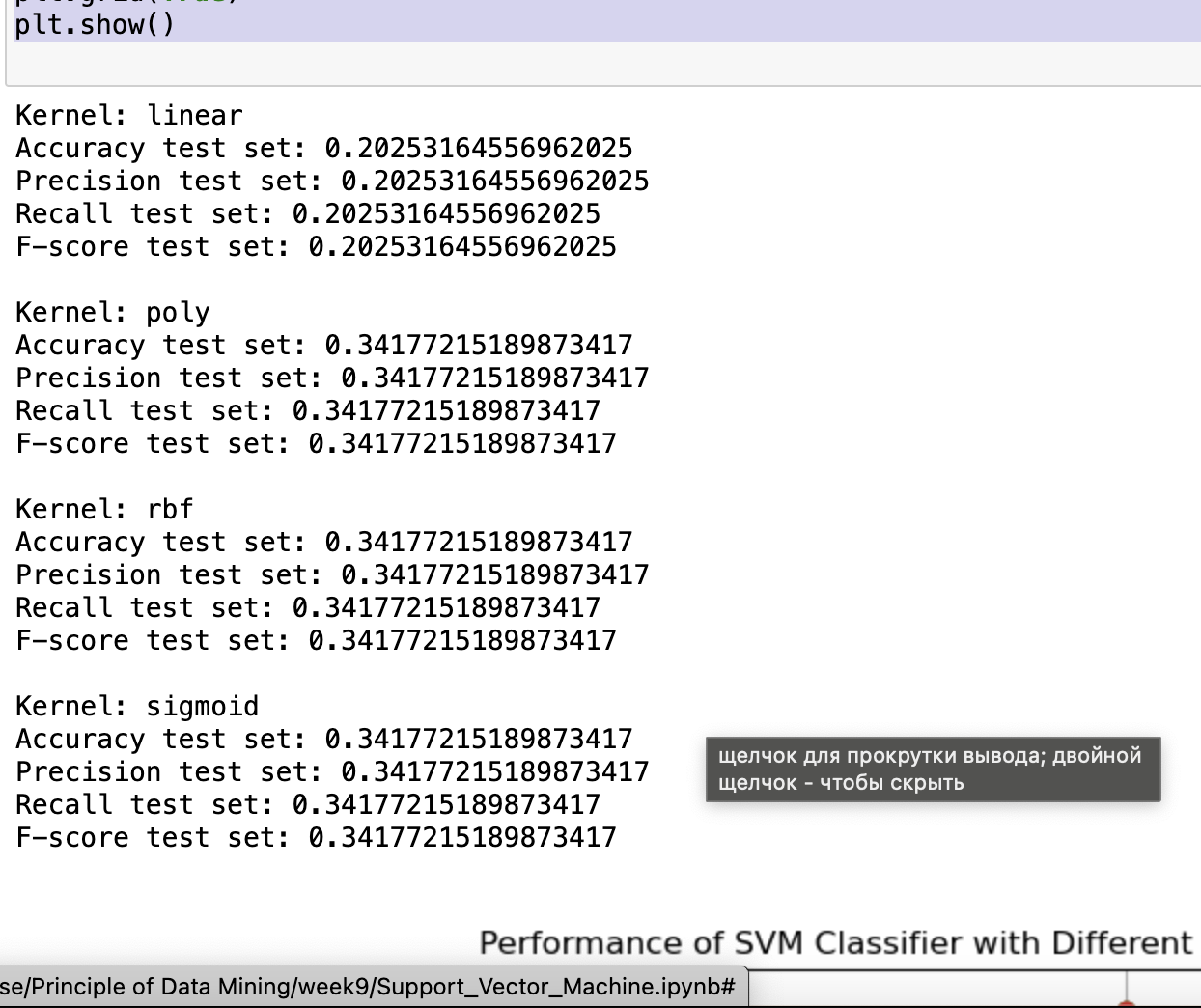
plt.ylabel('Score')

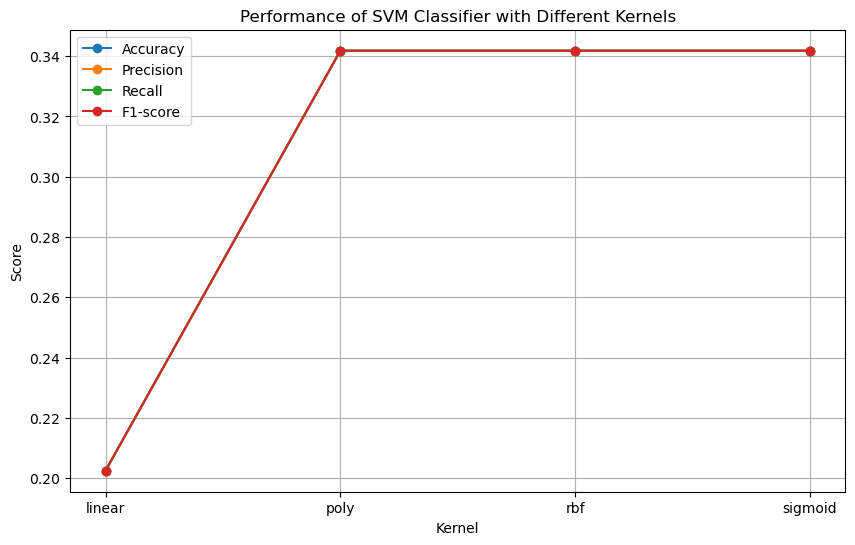
plt.title('Performance of SVM Classifier with Different Kernels')

plt.legend()

plt.grid(True)

plt.show()





# Lab 10 - Clustering

Import Libraries

import numpy as np

import pandas as pd

from pandas import DataFrame, Series

from IPython.display import Image

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

try:

from StringIO import StringIO

except ImportError:

from io import StringIO

import pydotplus

from sklearn import preprocessing

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

from sklearn.tree import DecisionTreeClassifier

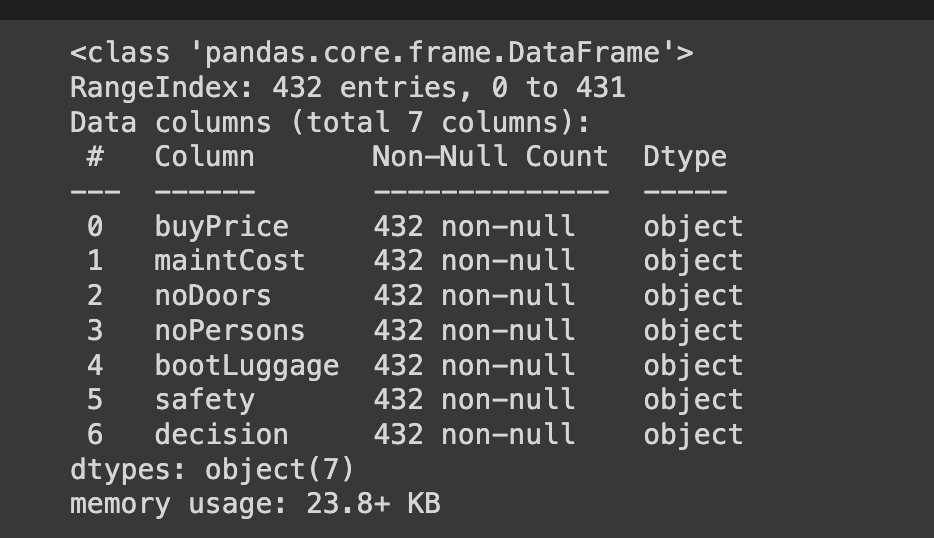
from sklearn import tree

%matplotlib inline

We have 4 sample car evaluation csv files.Now we read all of them.

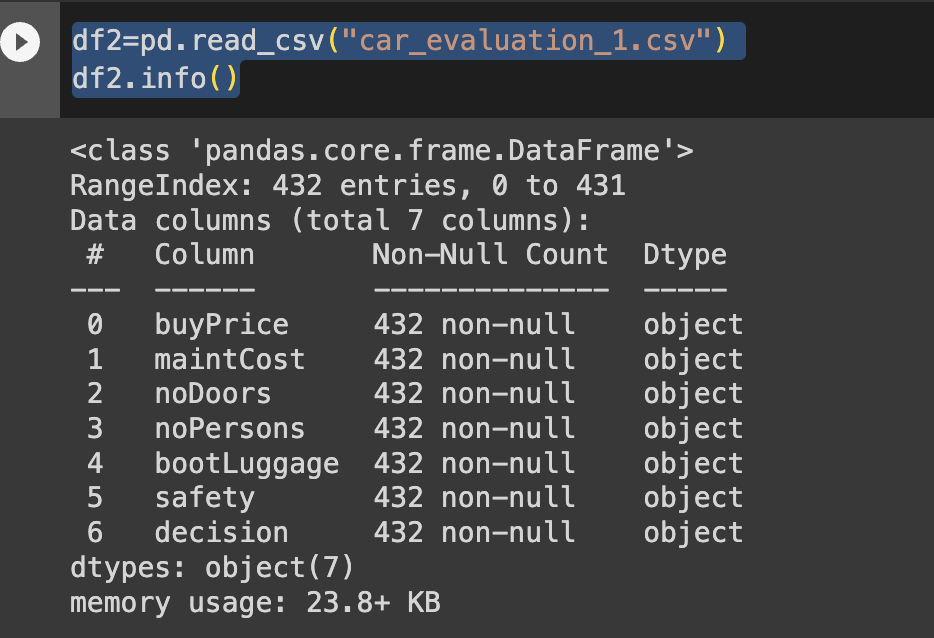
df1=pd.read\_csv("car\_evaluation\_0.csv")

df1.info()



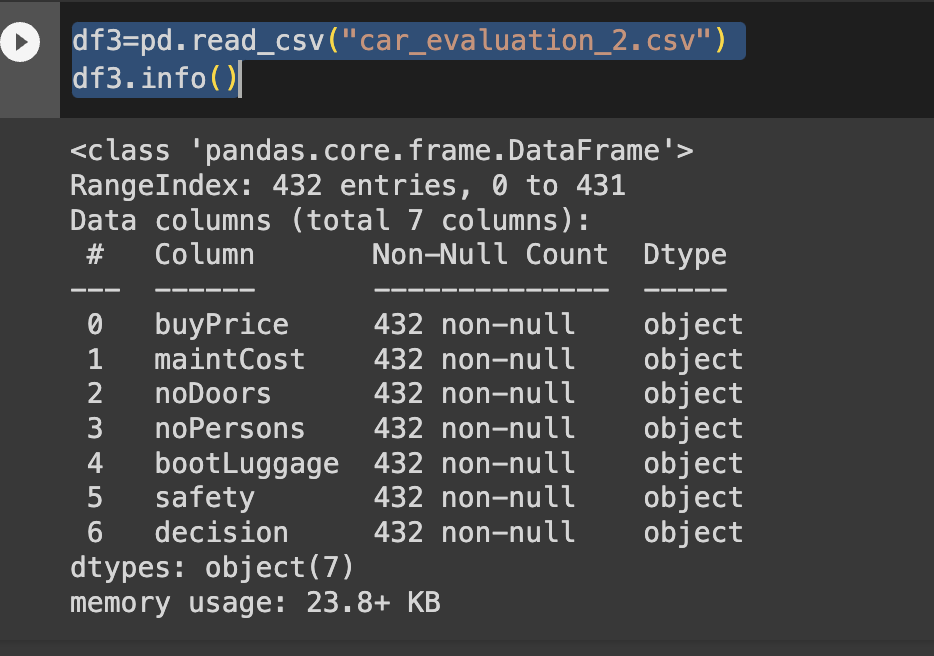
df2=pd.read\_csv("car\_evaluation\_1.csv")

df2.info()



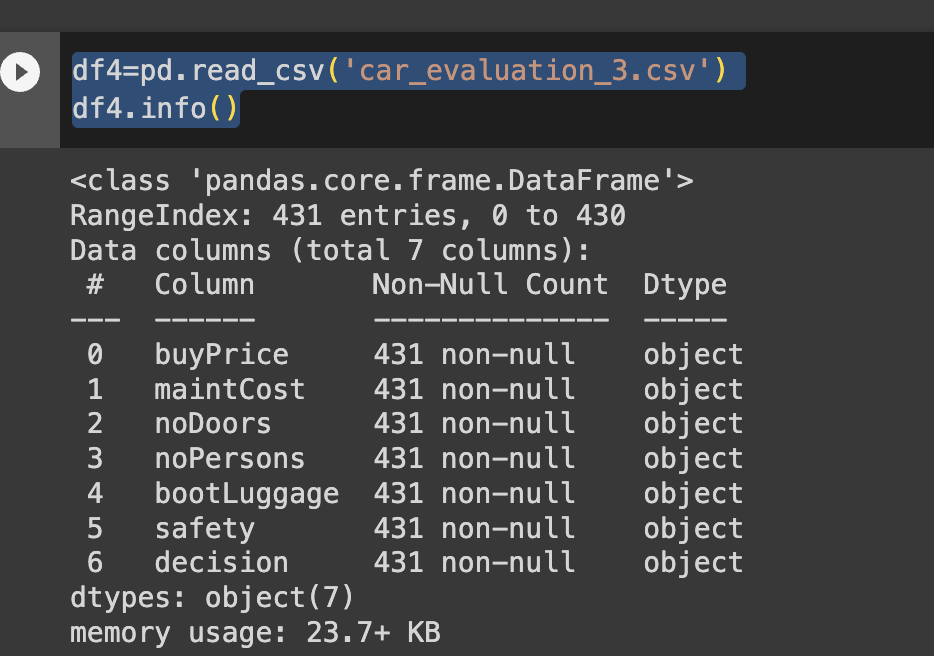
df3=pd.read\_csv("car\_evaluation\_2.csv")

df3.info()



df4=pd.read\_csv('car\_evaluation\_3.csv')

df4.info()

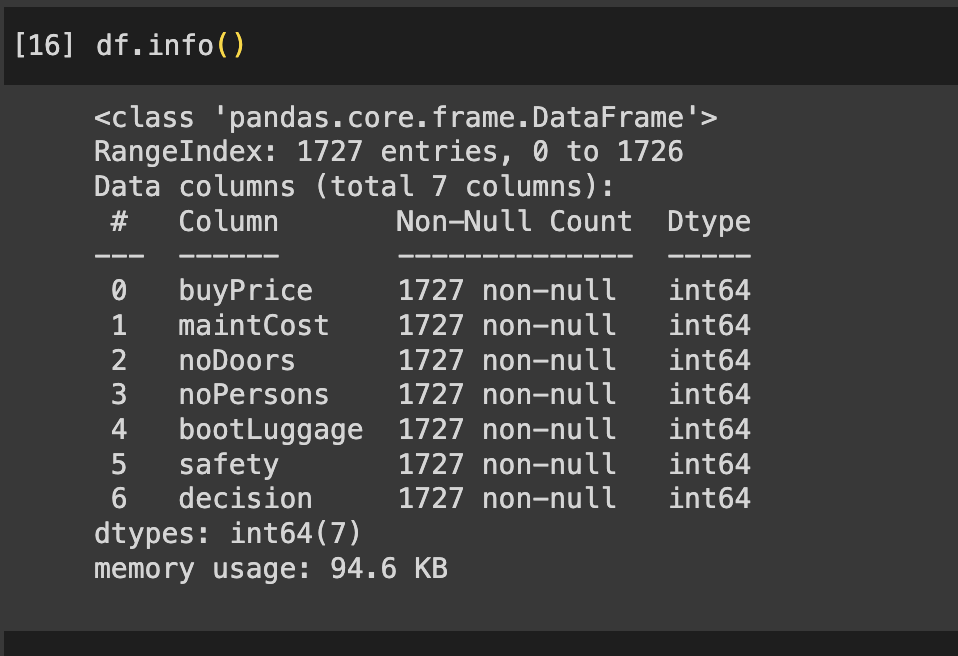


Then I concatenate these files

df=pd.concat([df1,df2,df3,df4], ignore\_index=True)

Result:

df.info()

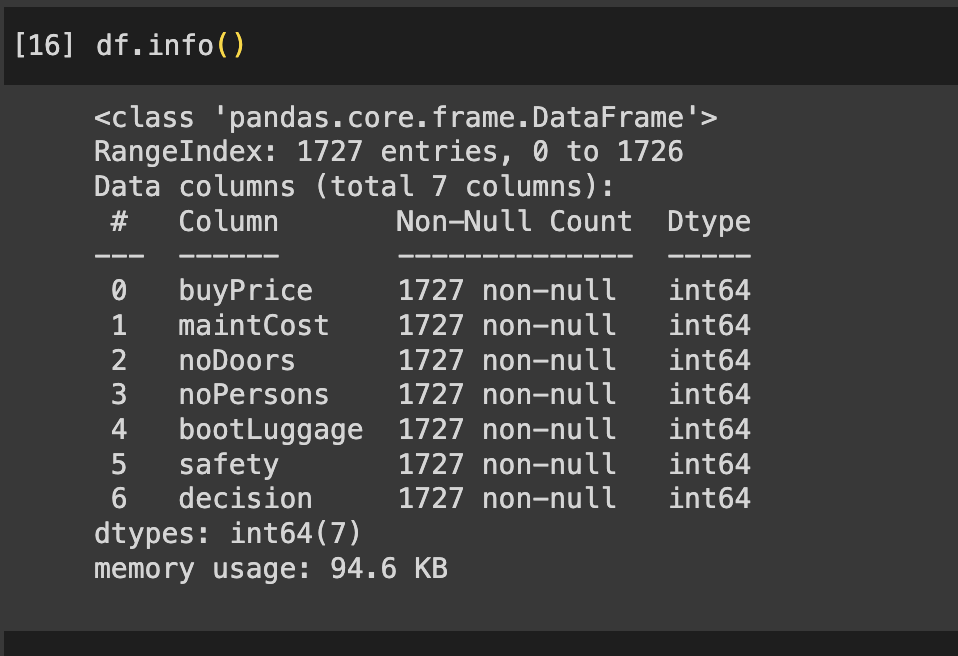


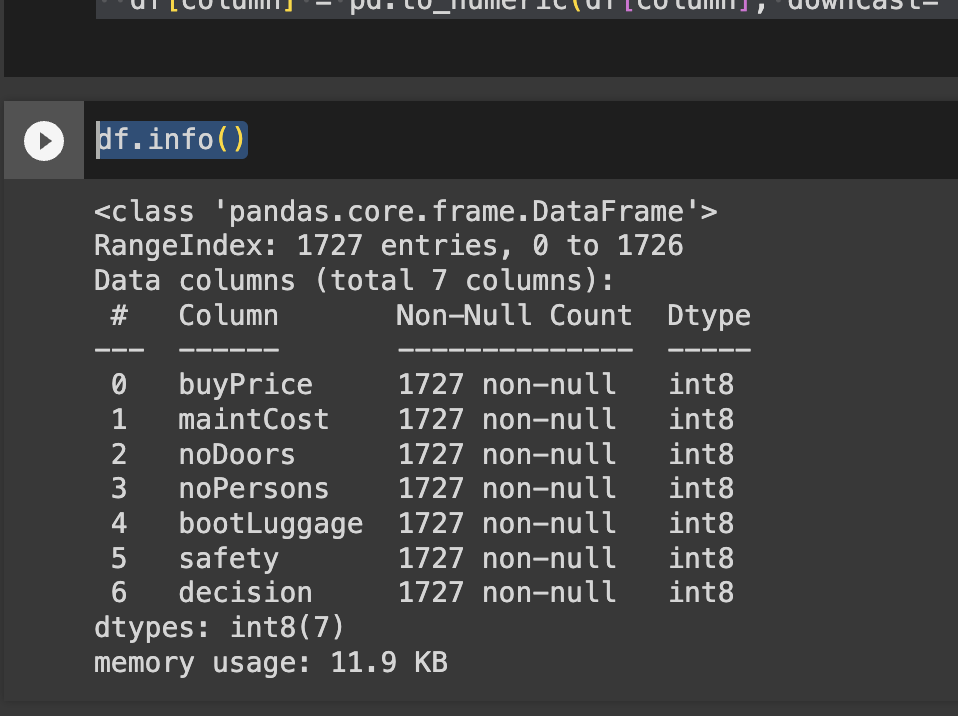
and I convert columns’values into interger to save memory usage.

columns=['buyPrice', 'maintCost', 'noDoors', 'noPersons', 'bootLuggage', 'safety', 'decision']

for column in columns:

df[column] = pd.to\_numeric(df[column], downcast='integer')

df.info()



Now We can reduce the memory usage 9 times

Let’s use train\_test\_spit to split a dataset into two subsets: a training set and a testing

X\_train,X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.1, random\_state=42)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(1554, 6)

(173, 6)

(1554,)

(173,)

clf = DecisionTreeClassifier(criterion='entropy')

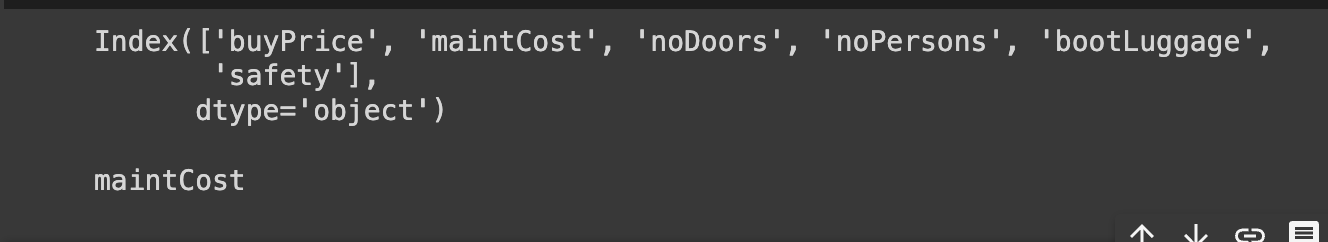
clf = clf.fit(X\_train, y\_train)

feature\_name = df.columns[:-1]

target\_name = df.columns[1]

print(feature\_name,'\n')

print(target\_name)  
Output:



The code snippet you provided is used to visualize a decision tree classifier using the graphviz library

dot\_data = StringIO()

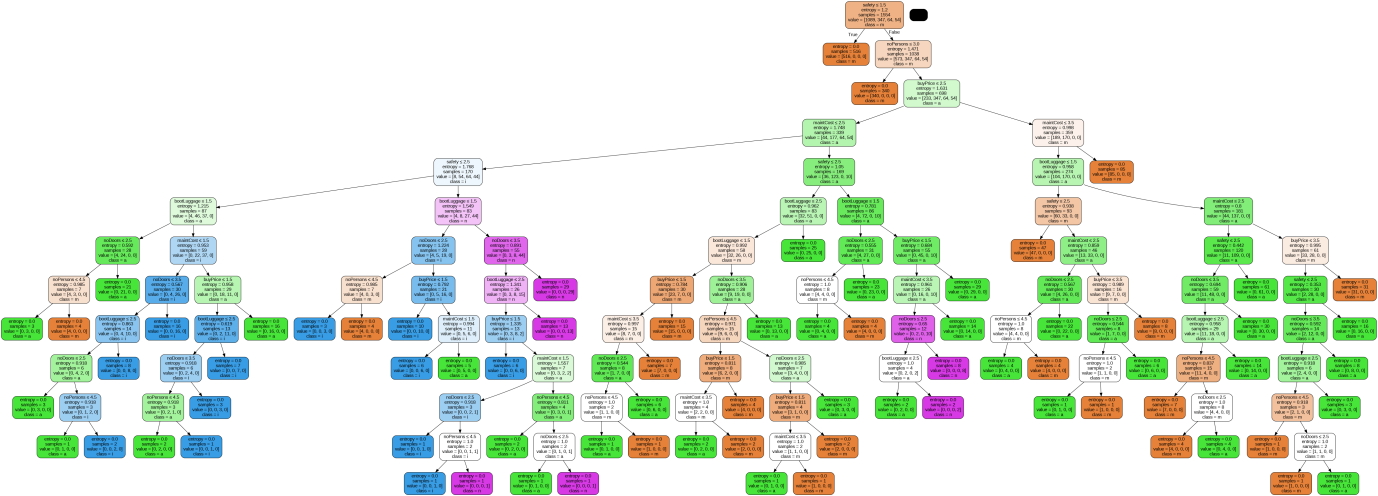
tree.export\_graphviz(clf, out\_file=dot\_data,feature\_names=feature\_name,

class\_names=target\_name, filled=True, rounded=True, special\_characters=True)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

Print the graph:

Image(graph.create\_png())



1. What is the decision criterion at each node of the decision tree?

Determine the best splits at each node of the decision tree.

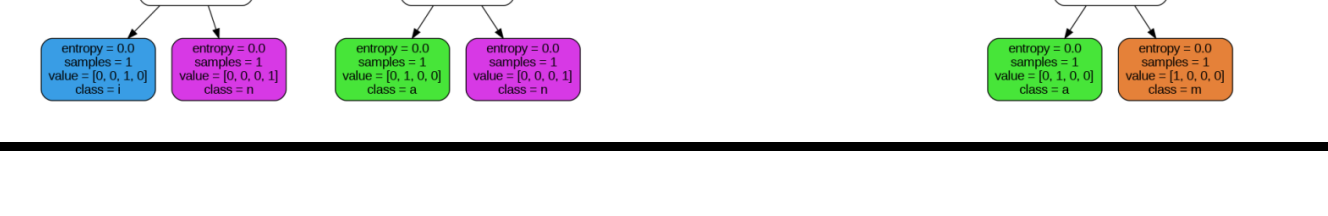
2.How is entropy/gini index/loss change at each node?

it is going down.

3. Is entropy decreasing or increasing in as we move down in tree?

it is decreasing.

4. How many samples are left at each node? Can you see any pattern?

there is one sample left.

5.What information is contained at leaf node?

 the leaf nodes may contain the predicted average house prices based on the training instances that reach them.

# Lab 11 -Hierarchical Clustering

Code:

# My code to import libraries:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

# Code to read file

df=pd.read\_csv("Live.csv")

df.head()

\*Note this csv file is located in same file with code file



# Code to print the shape of the data

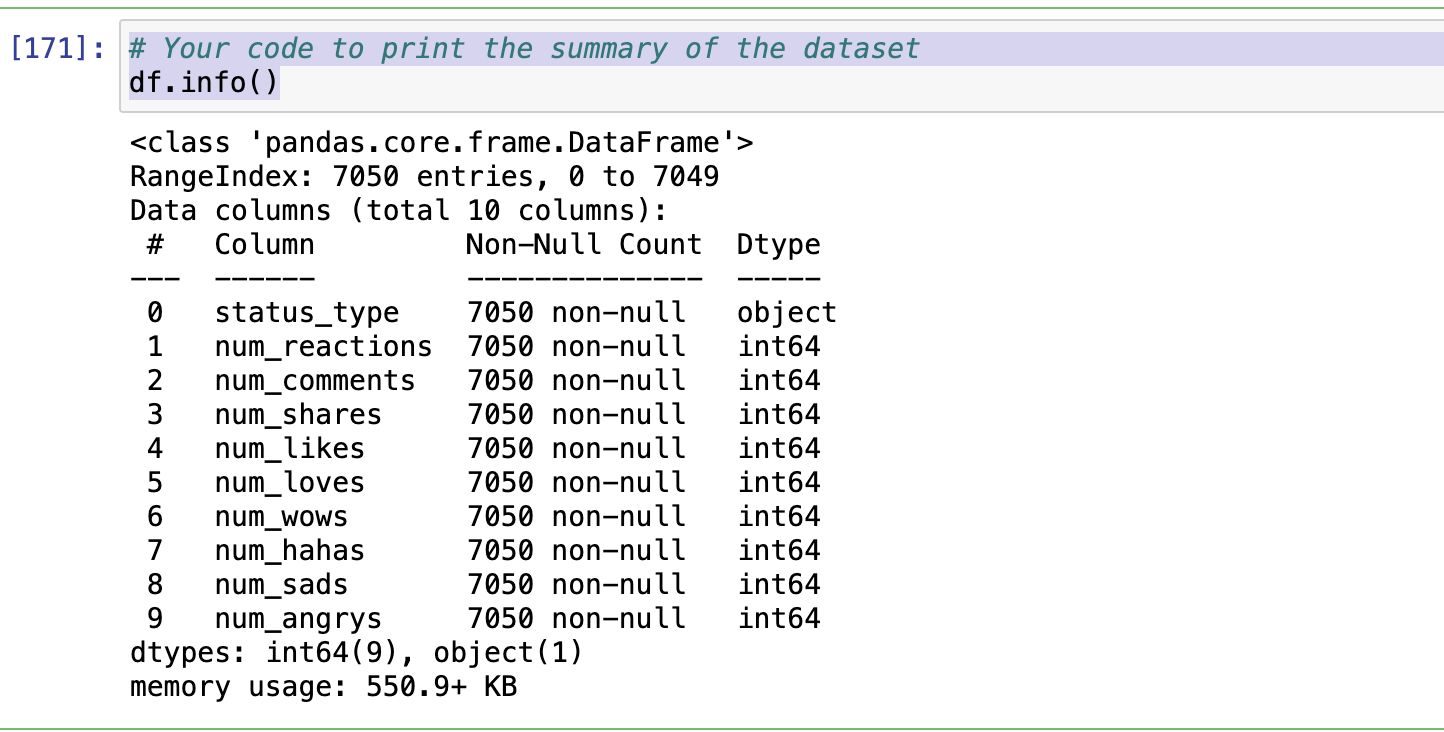
df.shape

**Result:**

(7050, 16)

# Your code to print the summary of the dataset

df.info()



Here is summary of the dataset:

it shows that there is one object , the rests are interger

We need transform object in interger:

# First,I need only X for training the model. The y is normally unknown. However, we will use y for the testing the accuracy of the trained model.

X = df

y = df['status\_type']

# Secondly,perform label encoding on the categorical data.

*#*

*# Your code to perform the label encoding the X.*

**from** sklearn.preprocessing **import** LabelEncoder

le**=**LabelEncoder()

X['status\_type']**=**le.fit\_transform(X['status\_type'])

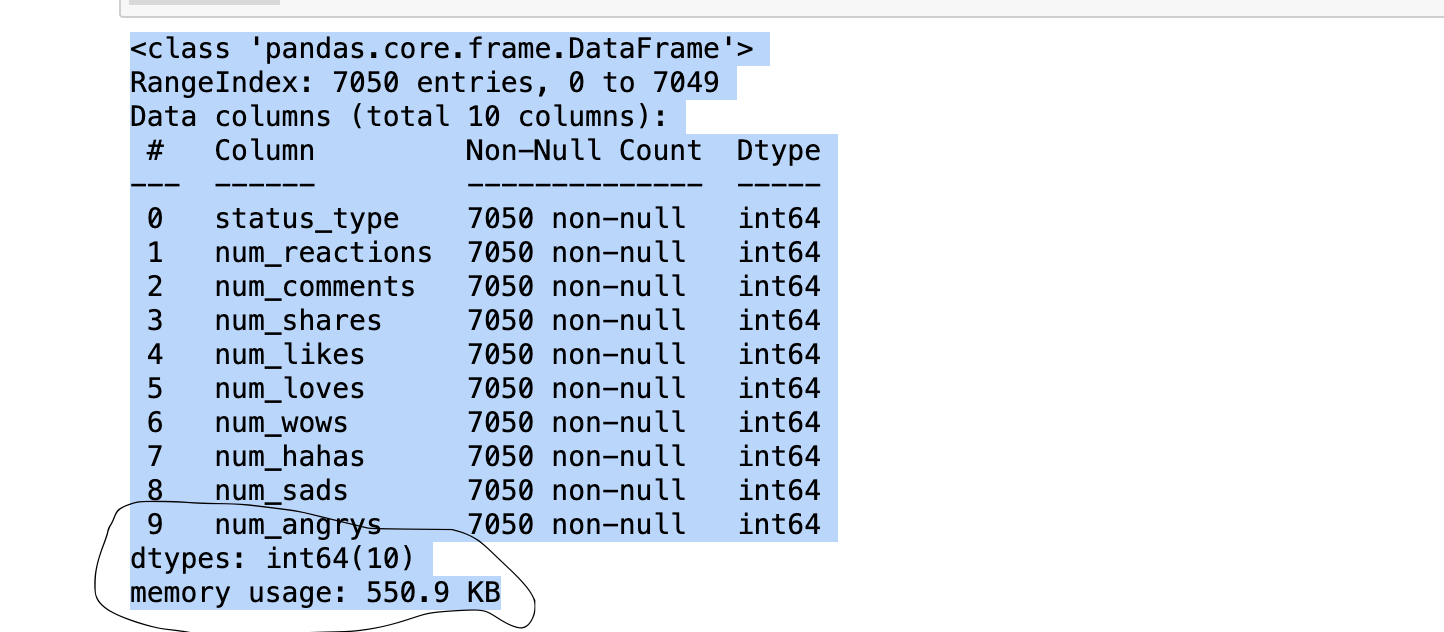
y**=**le.transform(y)

*#*

**Now we can see all value are interger:**

# Code to preview the dataset

df.info()



# Perfrom the feature scaling on X. Use MinMaxScaler:

*#*

*# Your code to perform the feature scaling on X*

**from** sklearn.preprocessing **import** MinMaxScaler

​

*# Assuming X is your dataset*

​

*# Initialize the scaler*

scaler **=** MinMaxScaler()

*# Perform feature scaling on the dataset*

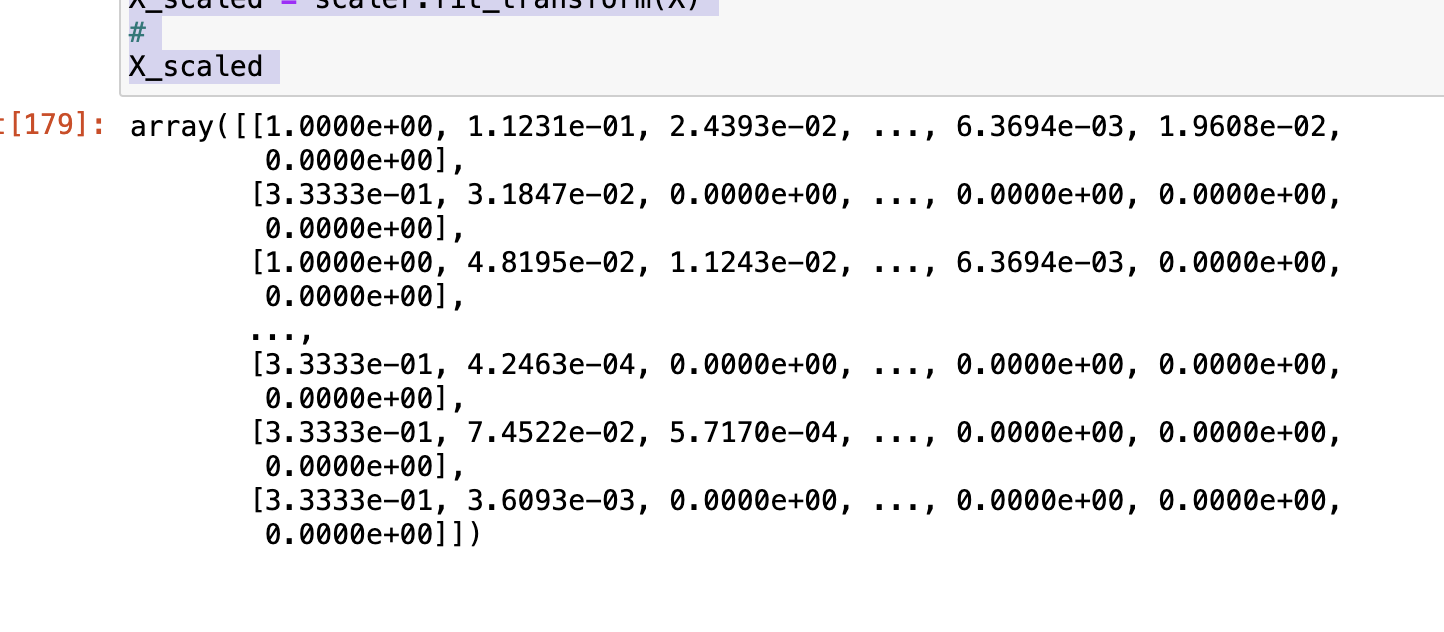
X\_scaled **=** scaler.fit\_transform(X)

*#*

X\_scaled

Out[179]:

Result:



Note: Every value is between 0 and 2:

Next step:

Develop KMeans clustering using 3 cluster:

**Code:**

import numpy as np

from sklearn.cluster import KMeans

# Assuming X is your dataset

accuracy=[]

K\_Means\_Models=[]

intertia=[]

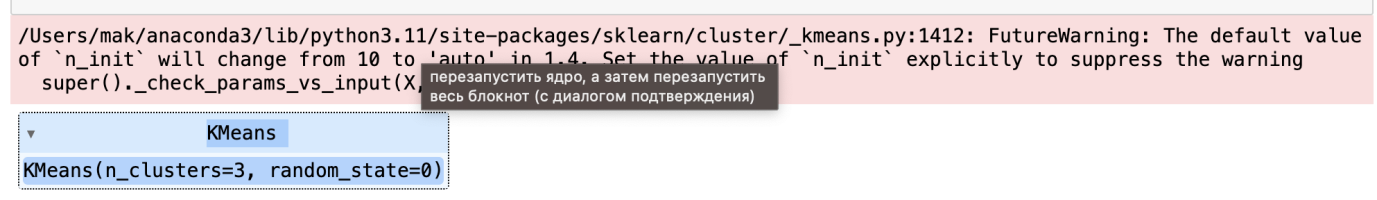
# Initialize the K-means model with 3 clusters

kmeans = KMeans(n\_clusters=2,random\_state=0)

# Fit the model to the dataset

kmeans.fit(X)

**Result:**

****

# Print the cluster centers.

Ввод [143]:

K\_Means\_Models.append(2)

K\_Means\_Models

Output:

[2]

# Print the intertia.

Ввод [182]:

np.set\_printoptions(precision**=**4)

kmeans.cluster\_centers\_

Out[182]:

array([[1.6788e+00, 2.2414e+02, 1.2211e+02, 3.0806e+01, 2.1123e+02,

1.0908e+01, 1.1530e+00, 5.5681e-01, 1.9742e-01, 9.6396e-02],

[2.9858e+00, 5.2289e+02, 5.2344e+03, 4.9162e+02, 4.0201e+02,

1.0192e+02, 7.9716e+00, 7.5390e+00, 2.5106e+00, 9.3617e-01]])

# Code to print the inertia

kmeans.inertia\_

Out[183]:

4986397875.914606

# Check the number of samples correctly classified.

Ввод [184]:

labels **=** kmeans.labels\_

​

*# check how many of the samples were correctly labeled*

correct\_labels **=** sum(y **==** labels)

​

print("Result: %d out of %d samples were correctly labeled." **%** (correct\_labels, y.size))

**Output:**

Result: 64 out of 7050 samples were correctly labeled.

# Check the accuracy of the KMean.

Ввод [185]:

print('Accuracy score: {0:0.2f}'. format(correct\_labels**/**float(y.size)))

​

Accuracy score: 0.03

# Then,Run KMeans for different cluster center numers 1-10 and plot inertia.

#import KMeans

# from sklearn.cluster import KMeans

# # Your code to run K-Mean for k = 1-10 and plot number of clusters vs inertia results.

# cs=[]

# for i in range(1,11):

# kmeans=KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

# kmeans.fit(X)

# cs.append(kmeans.inertia\_)

# plt.plot(range(1,11),cs)

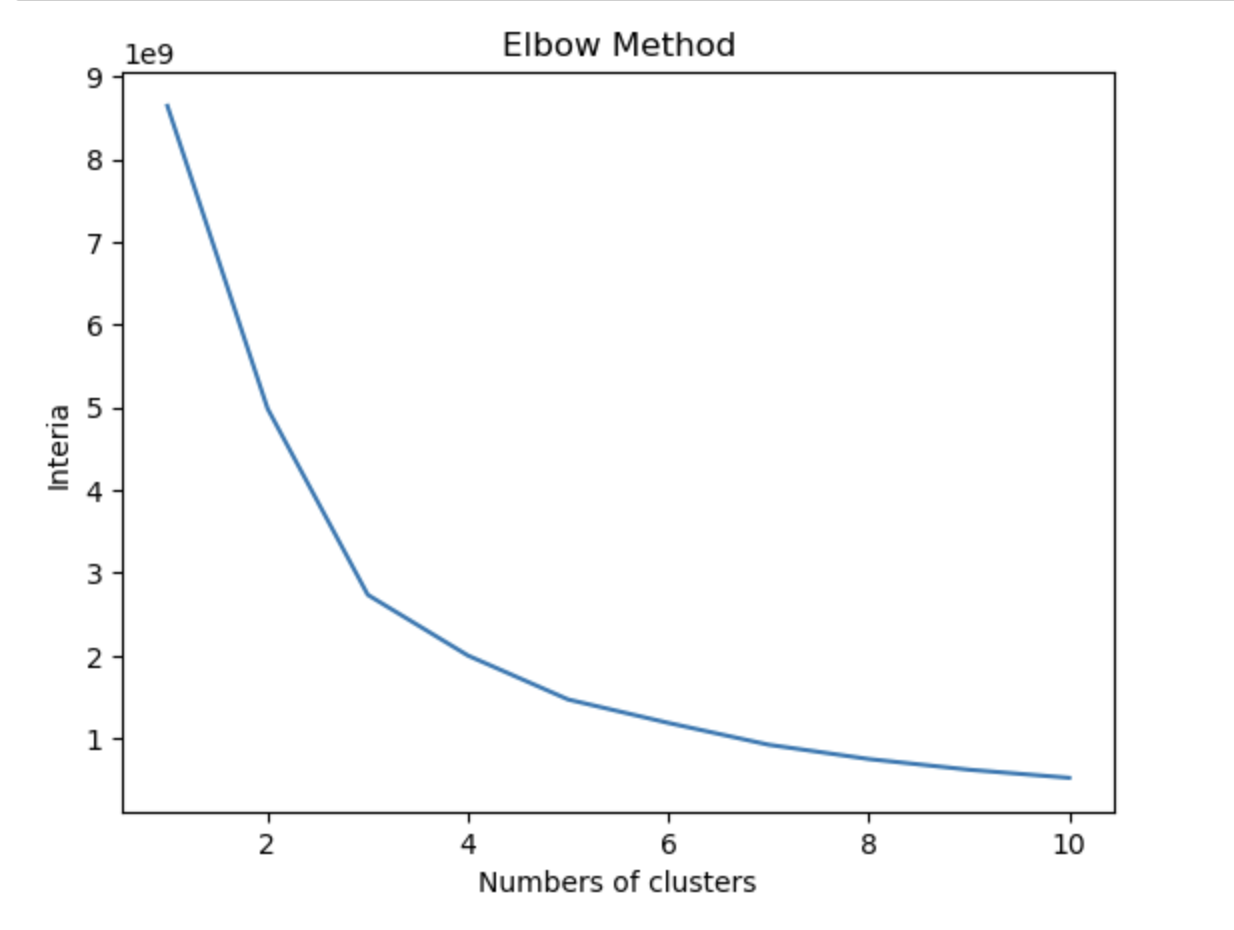
# plt.title("Elbow Method")

# plt.xlabel("Numbers of clusters")

# plt.ylabel("Interia")

# plt.show()

Output:



**Code**:

# Implement the KMeans for K=3 and check accuracy

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans. fit (X)

# check how many of the samples were correctly labeled

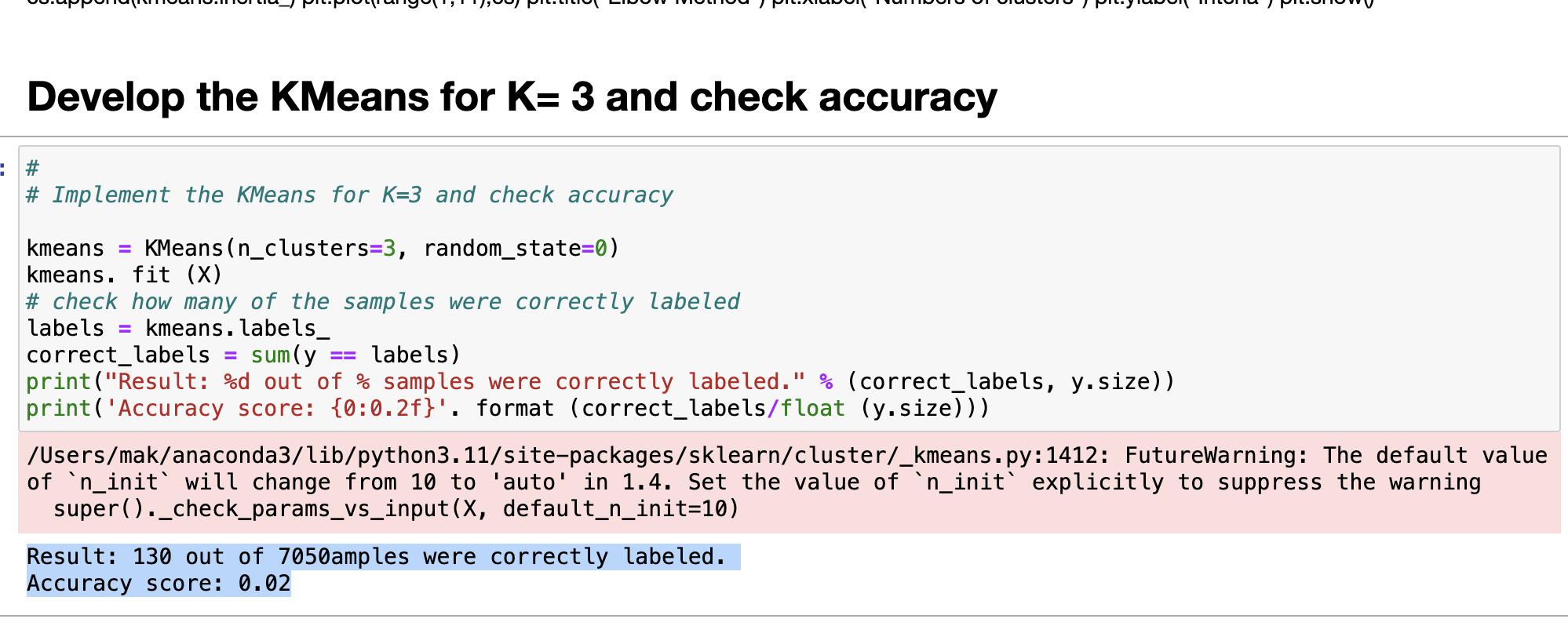
labels = kmeans.labels\_

correct\_labels = sum(y == labels)

print("Result: %d out of % samples were correctly labeled." % (correct\_labels, y.size))

print('Accuracy score: {0:0.2f}'. format (correct\_labels/float (y.size)))

Result:



1. the optimal number of clusters using K-means algorithm for this dataset.

Now I am using K=means algorithm for this dataset , K=4.

Code:

# Implement the KMeans for K= 4 and check accuracy

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans. fit (X)

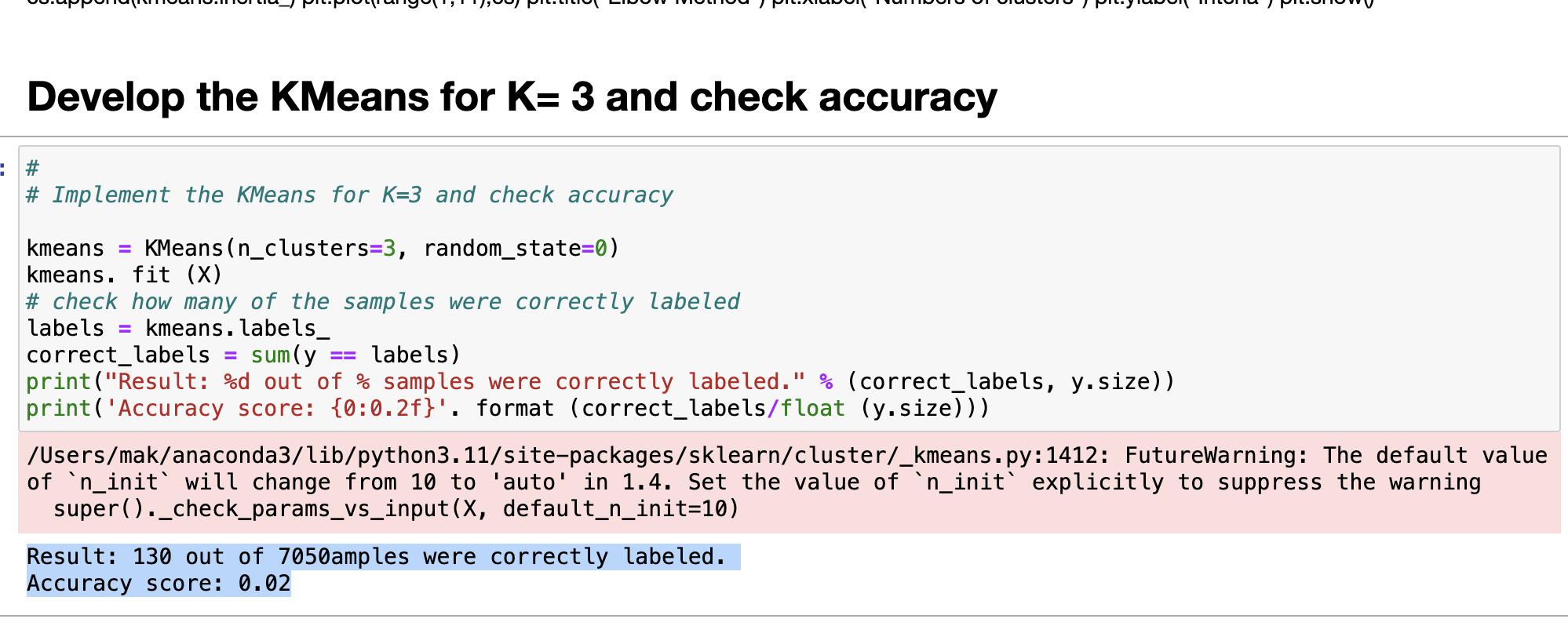
# check how many of the samples were correctly labeled

labels = kmeans.labels\_

correct\_labels = sum(y == labels)

print("Result: %d out of % samples were correctly labeled." % (correct\_labels, y.size))

print('Accuracy score: {0:0.2f}'. format (correct\_labels/float (y.size)))



Lab 12