**DATA SCIENCE & MACHINE LEARNING**

**LAB CYCLE 4 PART -1 KNN**

1. Using the iris data set implement the KNN algorithm. Take different values for Test and training data set .Also use different values for k. Also find the accuracy level.Using the iris data set implement the KNN algorithm. Take different values for Test and training data set .Also use different values for k. Also find the accuracy level.

**PROGRAM**

**import** numpy **as** np

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

**import** pandas **as** pd

dataset **=** pd**.**read\_csv("iris.csv")

X **=** dataset**.**iloc[:, :**-**1]**.**values

y **=** dataset**.**iloc[:, 4]**.**values

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.20)

**from** sklearn.neighbors **import** KNeighborsClassifier

classifier **=** KNeighborsClassifier(n\_neighbors**=**5)

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

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

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

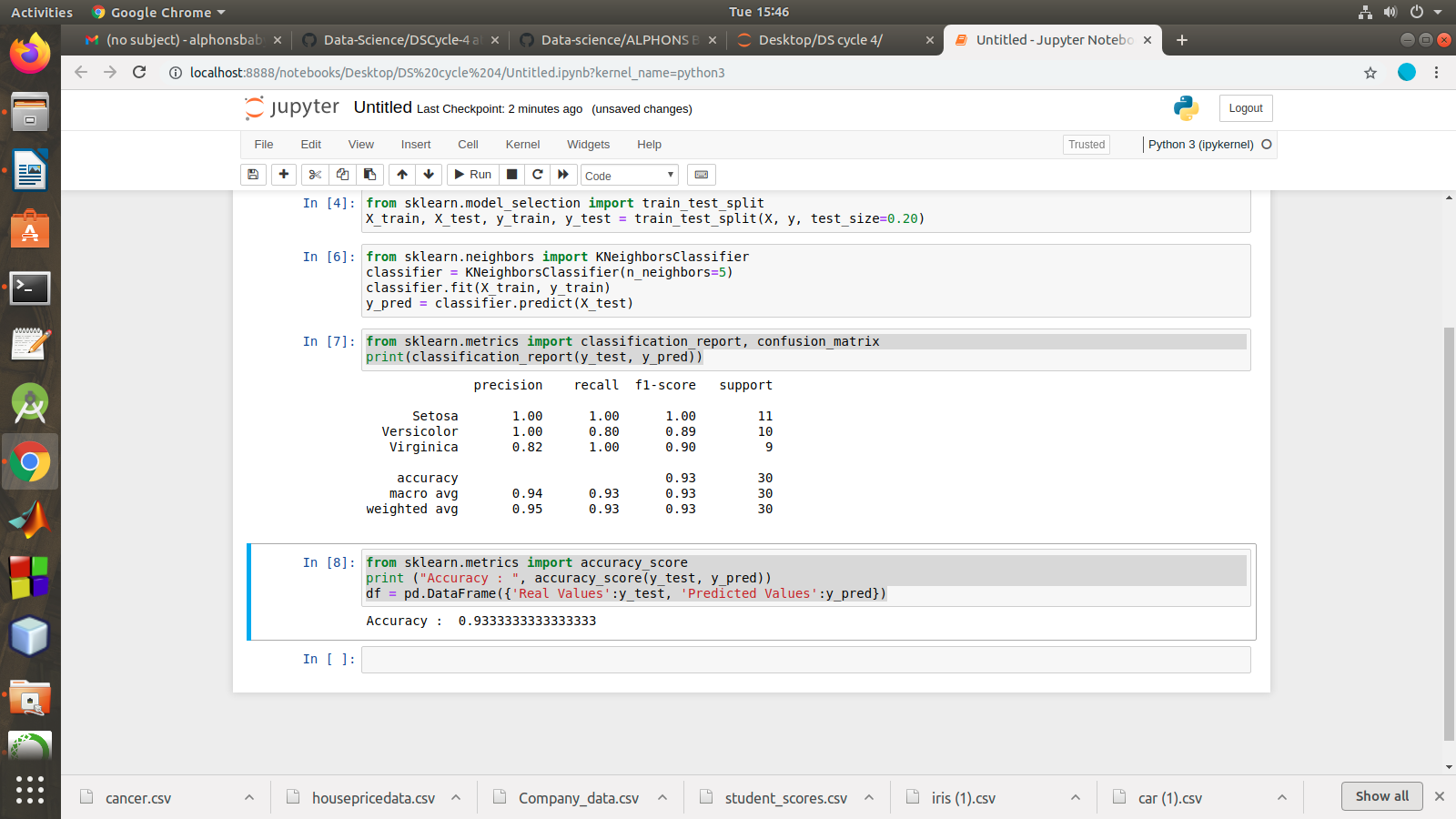
print(classification\_report(y\_test, y\_pred))

**from** sklearn.metrics **import** accuracy\_score

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

df **=** pd**.**DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

**OUTPUT**



2. Download another data set suitable for the KNN and implement the KNN algorithm. Take different

values for Test and training data set .Also use different values for k.

**PROGRAM**

**import** numpy **as** np

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

**import** pandas **as** pd

dataset **=** pd**.**read\_csv("cancer.csv")

dataset**.**head()

dataset**.**info()

X **=** dataset**.**iloc[:, 2:35]**.**values

print(X)

y **=** dataset**.**iloc[:, 1]**.**values

print(y)

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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.20)

**from** sklearn.neighbors **import** KNeighborsClassifier

classifier **=** KNeighborsClassifier(n\_neighbors**=**5)

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

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

**from** sklearn.metrics **import** classification\_report, confusion\_matrix

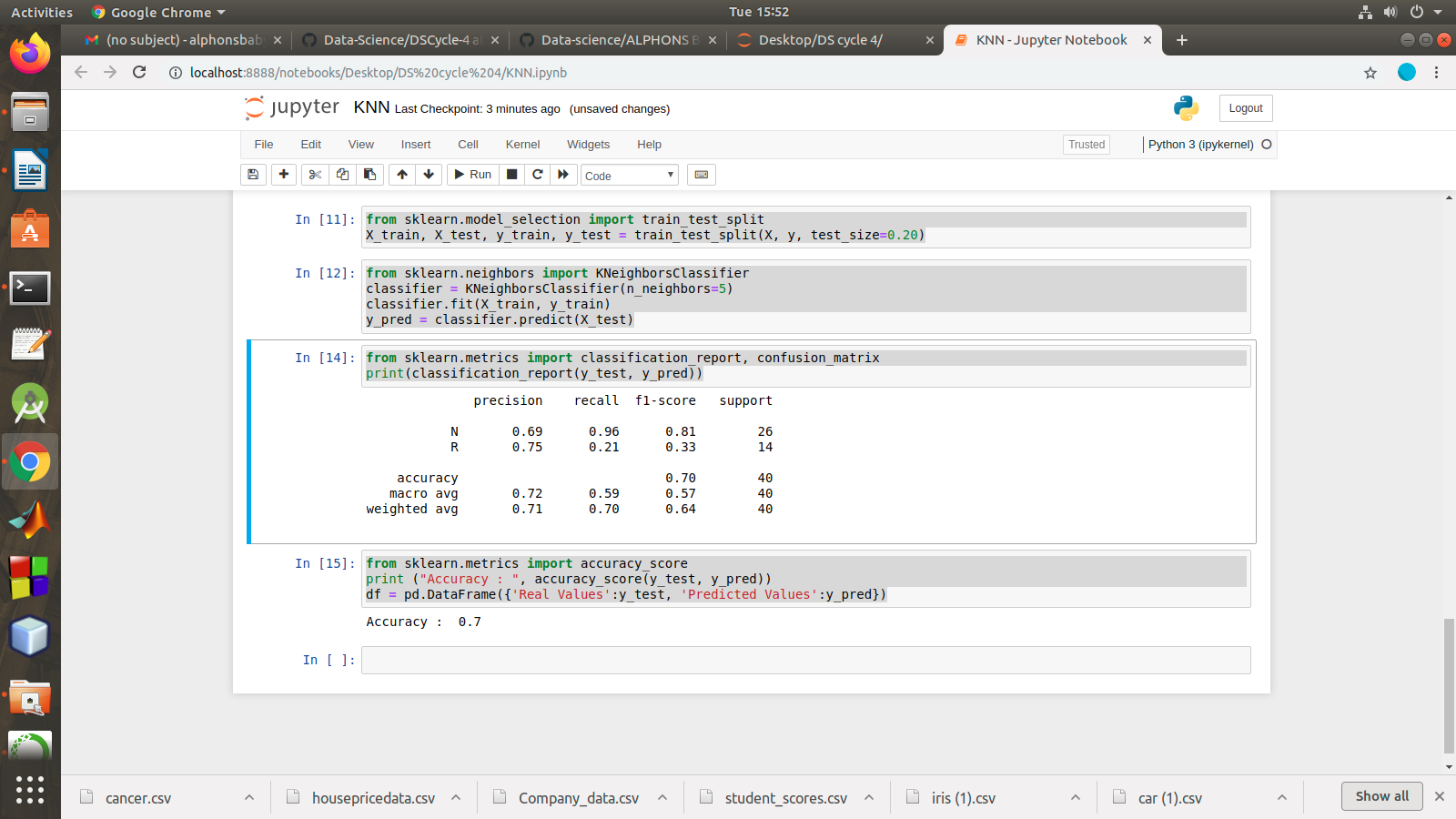
print(classification\_report(y\_test, y\_pred))

**from** sklearn.metrics **import** accuracy\_score

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

df **=** pd**.**DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

**OUTPUT**

****

3. Using iris data set, implement naive bayes classification for different naive Bayes classification

algorithms.( (i) gaussian (ii) bernoulli etc)

 Find out the accuracy level w.r.t to each  algorithm

 Display the no:of mislabeled classification from test data set

 List out the class labels of the mismatching records

**import** numpy **as** np

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

**import** pandas **as** pd

dataset **=** pd**.**read\_csv('iris.csv')

X **=** dataset**.**iloc[:,:4]**.**values

y **=** dataset['variety']**.**values

dataset**.**head(5)

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

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

**from** sklearn.naive\_bayes **import** GaussianNB

classifier **=** GaussianNB()

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

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

y\_pred

**from** sklearn.metrics **import** confusion\_matrix

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

**from** sklearn.metrics **import** accuracy\_score

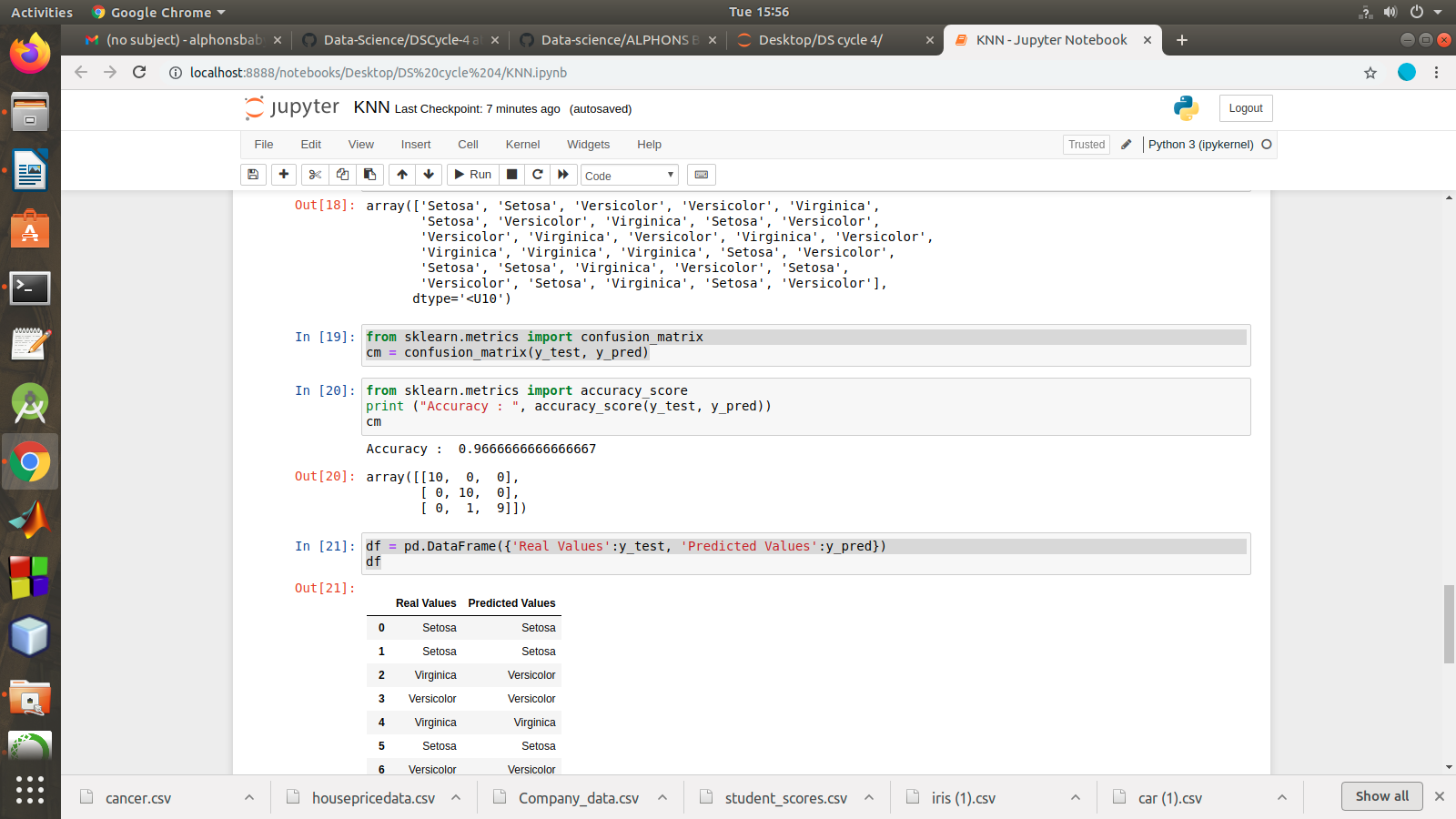
print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

cm

df **=** pd**.**DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

df

**OUTPUT**

****

4. Use car details CSV file and implement decision tree algorithm

 Find out the accuracy level.

 Display the no:of mislabeled classification from test data set

 List out the class labels of the mismatching records

**import** os

**import** numpy **as** np

**import** pandas **as** pd

**import** numpy **as** np**,** pandas **as** pd

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

**from** sklearn **import** tree, metrics, model\_selection

data **=** pd**.**read\_csv('car.csv',names**=**['buying','maint','doors','persons','lug\_boot','safety','class'])

data**.**head()

data**.**info()

data['class'],class\_names **=** pd**.**factorize(data['class'])

print(class\_names)

print(data['class']**.**unique())

data['buying'],\_ **=** pd**.**factorize(data['buying'])

data['maint'],\_ **=** pd**.**factorize(data['maint'])

data['doors'],\_ **=** pd**.**factorize(data['doors'])

data['persons'],\_ **=** pd**.**factorize(data['persons'])

data['lug\_boot'],\_ **=** pd**.**factorize(data['lug\_boot'])

data['safety'],\_ **=** pd**.**factorize(data['safety'])

data**.**head()

data**.**info()

X **=** data**.**iloc[:,:**-**1]

y **=** data**.**iloc[:,**-**1]

*# split data randomly into 70% training and 30% test*

X\_train, X\_test, y\_train, y\_test **=** model\_selection**.**train\_test\_split(X, y, test\_size**=**0.3, random\_state**=**0)

*# train the decision tree*

dtree **=** tree**.**DecisionTreeClassifier(criterion**=**'entropy', max\_depth**=**3, random\_state**=**0)

dtree**.**fit(X\_train, y\_train)

*# use the model to make predictions with the test data*

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

*# how did our model perform?*

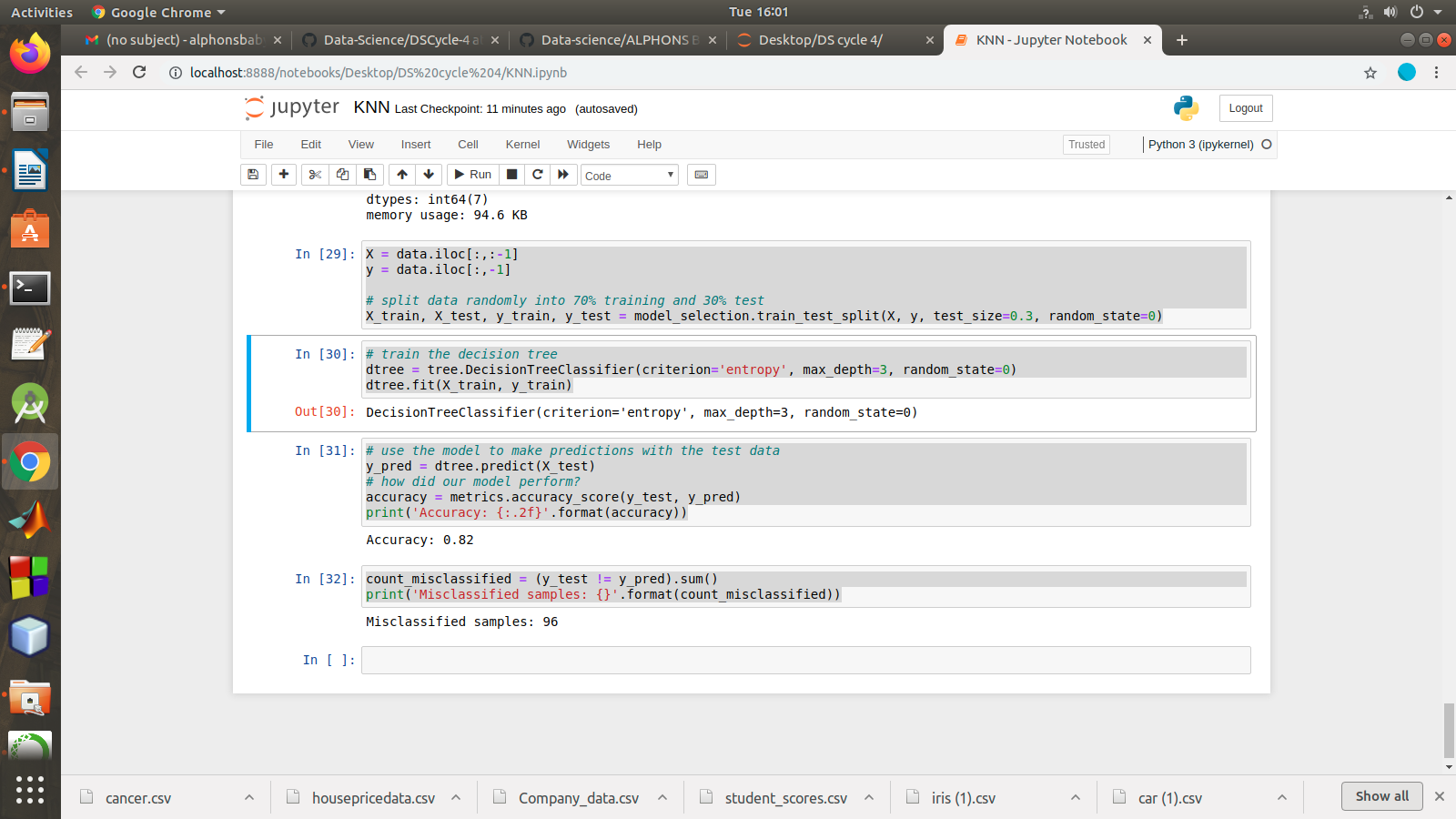
accuracy **=** metrics**.**accuracy\_score(y\_test, y\_pred)

print('Accuracy: {:.2f}'**.**format(accuracy))

count\_misclassified **=** (y\_test **!=** y\_pred)**.**sum()

print('Misclassified samples: {}'**.**format(count\_misclassified))

**OUTPUT**



**LAB CYCLE 4 PART -2 REGRESSION**

Implement Simple and multiple linear regression for the data sets ‘student\_score.csv’ and ‘company\_data .csv’ respectively

*Implement Simple and multiple linear regression for the data sets ‘student\_score.csv’*

*# and ‘company\_data .csv’ respectively.*

**import** numpy **as** np

**import** pandas **as** pd

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

*#data set contains details of no.of hours spend by students for studt and their marks*

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

student**.**head()

student**.**describe()

student**.**info()

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

Xax**=**student**.**iloc[:,0]

Yax**=**student**.**iloc[:,1]

plt**.**scatter(Xax,Yax)

plt**.**xlabel("No.of hours")

plt**.**ylabel("Score")

plt**.**title("Student scores")

plt**.**show()

*#Perform the simple linear regression model*

*#Equation: Y=w0+w1.x*

*#Here Y(marks)=w0+w1.x*

*#Create x as hours and Y as marks*

X **=** student**.**iloc[:, :**-**1]

y **=** student**.**iloc[:, 1]

print(X)

print(y)

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

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

print(X\_train)

**from** sklearn.linear\_model **import** LinearRegression

regressor **=** LinearRegression()

regressor**.**fit(X\_train, y\_train)

print(regressor**.**intercept\_)

print(regressor**.**coef\_)

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

**for**(i,j) **in** zip(y\_test,y\_pred):

**if** i**!=**j:

print("Actual value :",i,"Predicted value :",j)

print("Number of mislabeled points from test data set :", (y\_test **!=** y\_pred)**.**sum())

**from** sklearn **import** metrics

print("Mean Absolute error :", metrics**.**mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared error :", metrics**.**mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared error :", np**.**sqrt(metrics**.**mean\_squared\_error(y\_test,y\_pred)))

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

c**=**X\_test['Hours']**.**count()

xax**=**np**.**arange(c)

print(xax)

X\_axis **=** np**.**arange(len(xax))

plt**.**bar(X\_axis**-**0.2, y\_test, 0.6, label**=**'Actual')

plt**.**bar(X\_axis**+**0.2, y\_pred, 0.6, label**=**'Predicted')

plt**.**xlabel("Test Records")

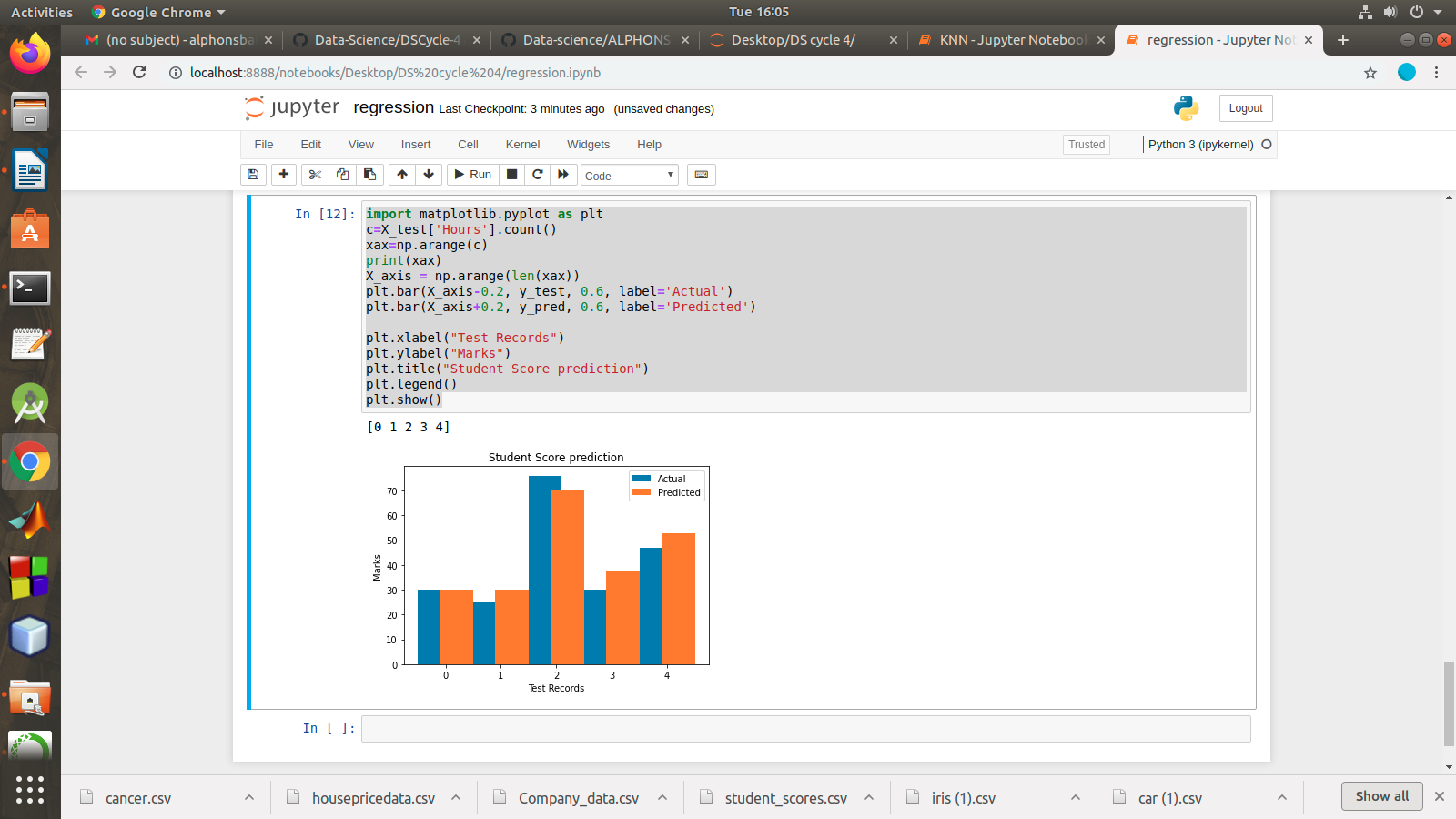
plt**.**ylabel("Marks")

plt**.**title("Student Score prediction")

plt**.**legend()

plt**.**show()

**OUTPUT**

****

multiple linear regression for the data sets ‘student\_score.csv’ and ‘company\_data .csv’ respectively

**import** pandas **as** pd

**import** numpy **as** np

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

advertising **=** pd**.**read\_csv('Company\_data.csv')

advertising**.**head()

advertising**.**describe()

Out[2]:

advertising**.**info()

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

**import** seaborn **as** sns

sns**.**pairplot(advertising, x\_vars**=**['TV', 'Radio', 'Newspaper'],

y\_vars**=**'Sales', height**=**5, aspect**=**1, kind**=**'scatter')

plt**.**show()

*#perform the multiple linear regression model*

*#Equation : Y=w0+w1.x1 + w2.x2 + w3.x3*

*#Here Y(sales)=w0+w1.x1(TV)+w2.x2(Radio)+w3.x3(Newspaper)*

*#create x and Y as sales*

X **=** advertising**.**iloc[:, :**-**1]

print(X)

y **=** advertising**.**iloc[:, **-**1]

print(y)

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

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

print(X\_train)

**from** sklearn.linear\_model **import** LinearRegression

regressor **=** LinearRegression()

regressor**.**fit(X\_train, y\_train)

print(regressor**.**intercept\_)

print(regressor**.**coef\_)

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

**for**(i,j) **in** zip(y\_test,y\_pred):

**if** i**!=**j:

print("Actual value :",i,"Predicted value :",j)

print("Number of mislabeled points from test data set :", (y\_test **!=** y\_pred)**.**sum())

**from** sklearn **import** metrics

print("Mean Absolute error :", metrics**.**mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared error :", metrics**.**mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared error :", np**.**sqrt(metrics**.**mean\_squared\_error(y\_test,y\_pred)))

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

c**=**X\_test['TV']**.**count()

xax**=**np**.**arange(c)

print(xax)

X\_axis **=** np**.**arange(len(xax))

plt**.**bar(X\_axis**-**0.2, y\_test, 0.6, label**=**'Actual')

plt**.**bar(X\_axis**+**0.2, y\_pred, 0.6, label**=**'Predicted')

plt**.**xlabel("Sales")

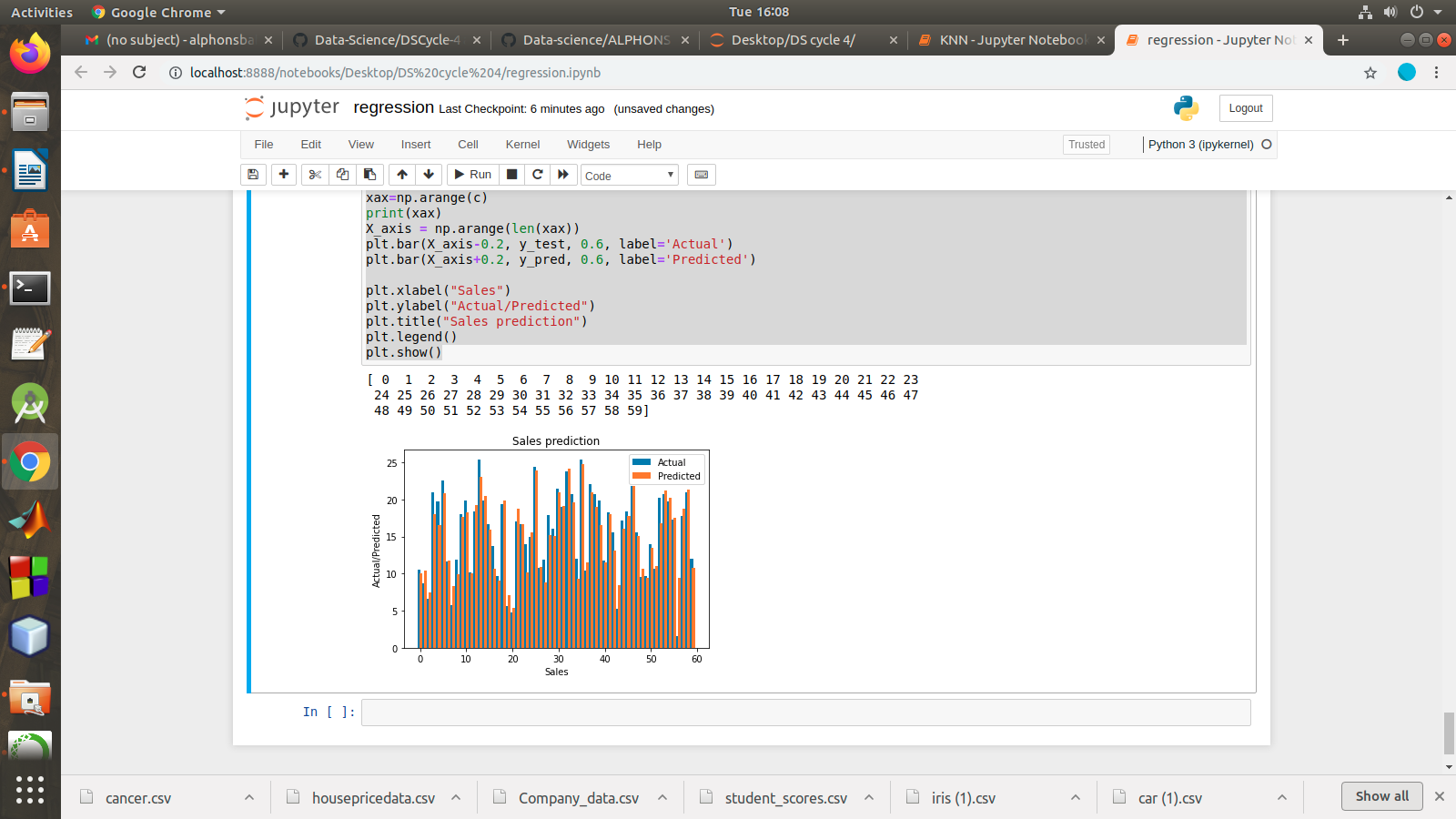
plt**.**ylabel("Actual/Predicted")

plt**.**title("Sales prediction")

plt**.**legend()

plt**.**show()

**OUTPUT**



**LAB CYCLE 4 PART -3 NEURAL NETWOR**

**Experiment:**

1. Create a neural network for the given ‘houseprice.csv’ to predict the whether price of the house is above or below median value or not

Keras allows you to train your model using stochastic, batch, or minibatch gradient descent.

* **Batch Gradient Descent**. Batch size is set to the total number of examples in the training dataset.
* **Stochastic Gradient Descent**. Batch size is set to one.
* **Minibatch Gradient Descent**. Batch size is set to more than one and less than the total number of examples in the training dataset.

**Cost Functions**

The three most common loss(cost) functions are:

1. ‘*binary\_crossentropy*‘ for binary classification.
2. ‘*sparse\_categorical\_crossentropy*‘ for multi-class classification.
3. ‘*mse*‘ (mean squared error) for regression.

**Different Optimizers**

* SGD.
* RMSprop.
* Adam.
* Adadelta.
* Adagrad.
* Adamax.
* Nadam.
* Ftrl.

**Fit the Model**

Fitting the model requires that you first select the training configuration, such as the number of epochs (loops through the training dataset) and the batch size (number of samples in an epoch used to estimate model error).

A subset of the training dataset is used to estimate the error and update the weights.

The number of examples from the training dataset used in the estimate of the error gradient is called the [batch size](https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/)

A batch size of 32 means that 32 samples from the training dataset will be used to estimate the error gradient before the model weights are updated. One [training epoch](https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/) means that the learning algorithm has made one pass through the training dataset, where examples were separated into randomly selected “batch size” groups.

**Activation Functions**

**ReLU** function is suitable for using in hidden layers

**Sigmoid** function is commonly used in output layer for binary classification

**Softmax** activation function should be used in the output layer in case of multiclass classification.

**import** tensorflow **as** tf

**import** keras

**import** pandas

**import** sklearn

**import** matplotlib

**import** pandas **as** pd

df **=** pd**.**read\_csv('housepricedata.csv')

df

dataset **=** df**.**values

dataset

X **=** dataset[:,0:10]

Y **=** dataset[:,10]

**from** sklearn **import** preprocessing

min\_max\_scaler **=** preprocessing**.**MinMaxScaler()

X\_scale **=** min\_max\_scaler**.**fit\_transform(X)

X\_scale

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

X\_train, X\_val\_and\_test, Y\_train, Y\_val\_and\_test **=** train\_test\_split(X\_scale, Y, test\_size**=**0.3)

X\_val, X\_test, Y\_val, Y\_test **=** train\_test\_split(X\_val\_and\_test, Y\_val\_and\_test, test\_size**=**0.5)

print(X\_train**.**shape, X\_val**.**shape, X\_test**.**shape, Y\_train**.**shape, Y\_val**.**shape, Y\_test**.**shape)

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense

model **=** Sequential([

Dense(32, activation**=**'relu', input\_shape**=**(10,)),

Dense(32, activation**=**'relu'),

Dense(1, activation**=**'sigmoid'),

])

model**.**compile(optimizer**=**'sgd',

loss**=**'binary\_crossentropy',

metrics**=**['accuracy'])

hist **=** model**.**fit(X\_train, Y\_train,

batch\_size**=**32, epochs**=**100,

validation\_data**=**(X\_val, Y\_val))

model**.**evaluate(X\_test, Y\_test)[1]

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

plt**.**plot(hist**.**history['loss'])

plt**.**plot(hist**.**history['val\_loss'])

plt**.**title('Model loss')

plt**.**ylabel('Loss')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'upper right')

plt**.**show()

plt**.**plot(hist**.**history['accuracy'])

plt**.**plot(hist**.**history['val\_accuracy'])

plt**.**title('Model accuracy')

plt**.**ylabel('Accuracy')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'lower right')

plt**.**show()

plt**.**plot(hist**.**history['accuracy'])

plt**.**plot(hist**.**history['val\_accuracy'])

plt**.**title('Model accuracy')

plt**.**ylabel('Accuracy')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'lower right')

plt**.**show()

plt**.**plot(hist**.**history['loss'])

plt**.**plot(hist**.**history['val\_loss'])

plt**.**title('Model loss')

plt**.**ylabel('Loss')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'upper right')

plt**.**show()

plt**.**plot(hist\_2**.**history['accuracy'])

plt**.**plot(hist\_2**.**history['val\_accuracy'])

plt**.**title('Model accuracy')

plt**.**ylabel('Accuracy')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'lower right')

plt**.**show()

**from** keras.layers **import** Dropout

**from** keras **import** regularizers

model\_3 **=** Sequential([

Dense(1000, activation**=**'relu', kernel\_regularizer**=**regularizers**.**l2(0.01), input\_shape**=**(10,)),

Dropout(0.3),

Dense(1000, activation**=**'relu', kernel\_regularizer**=**regularizers**.**l2(0.01)),

Dropout(0.3),

Dense(1000, activation**=**'relu', kernel\_regularizer**=**regularizers**.**l2(0.01)),

Dropout(0.3),

Dense(1000, activation**=**'relu', kernel\_regularizer**=**regularizers**.**l2(0.01)),

Dropout(0.3),

Dense(1, activation**=**'sigmoid', kernel\_regularizer**=**regularizers**.**l2(0.01)),

])

model\_3**.**compile(optimizer**=**'adam',

loss**=**'binary\_crossentropy',

metrics**=**['accuracy'])

hist\_3 **=** model\_3**.**fit(X\_train, Y\_train,

batch\_size**=**32, epochs**=**100,

validation\_data**=**(X\_val, Y\_val))

plt**.**plot(hist\_3**.**history['loss'])

plt**.**plot(hist\_3**.**history['val\_loss'])

plt**.**title('Model loss')

plt**.**ylabel('Loss')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'upper right')

plt**.**ylim(top**=**1.2, bottom**=**0)

plt**.**show()

plt**.**plot(hist\_3**.**history['accuracy'])

plt**.**plot(hist\_3**.**history['val\_accuracy'])

plt**.**title('Model accuracy')

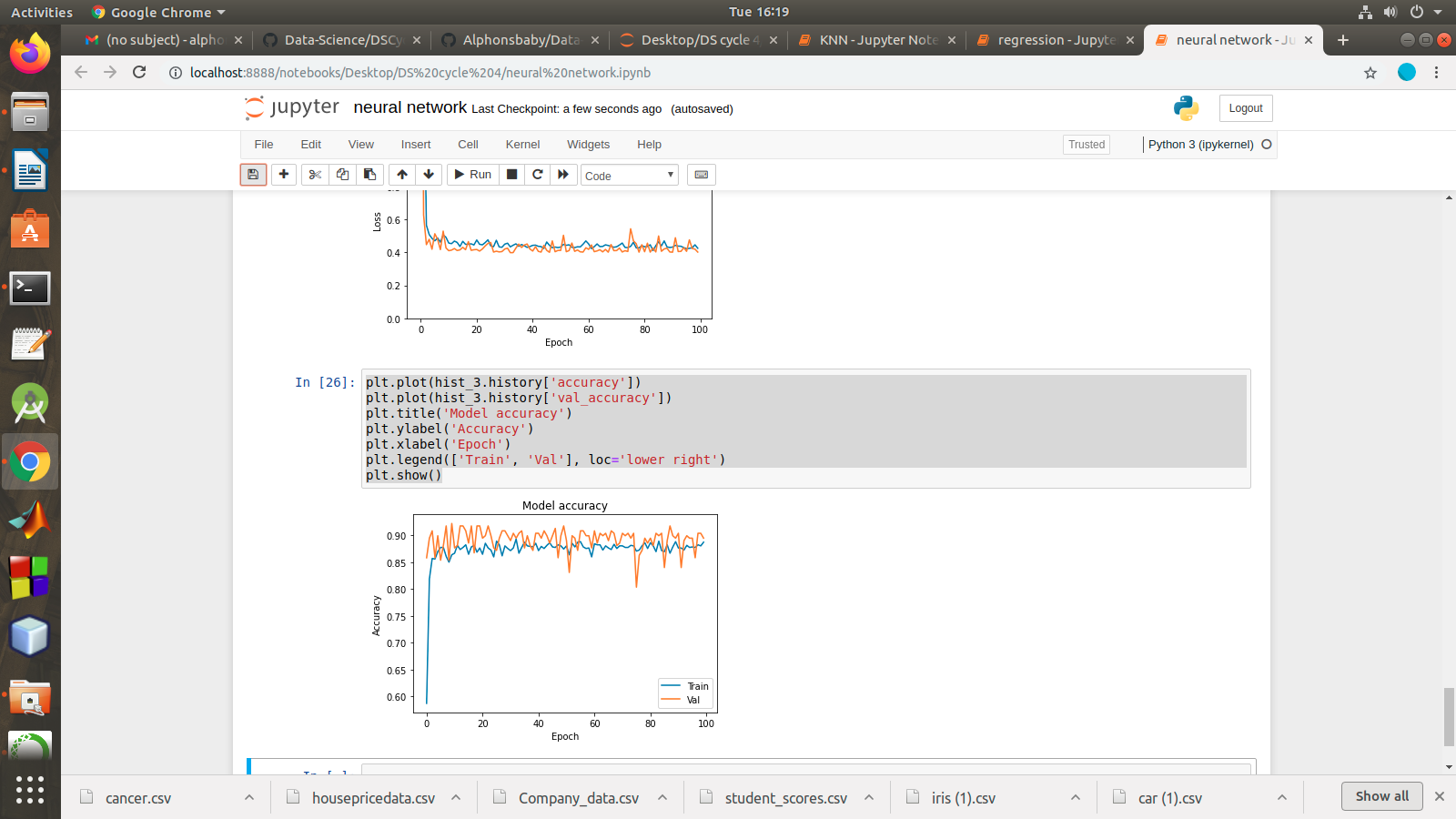
plt**.**ylabel('Accuracy')

plt**.**xlabel('Epoch')

plt**.**legend(['Train', 'Val'], loc**=**'lower right')

plt**.**show()

**OUTPUT**



**LAB CYCLE 5 PART -1 SVM**

1. Linear SVC The objective of a Linear SVC (Support Vector Classifier) is **to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes**, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

**CountVectorizer:**

Creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

Given a data set of support tickets. Each ticket also has an associated "urgency score" of between 0 and 3, and where 0 is "very urgent" and 3 is "not urgent". It would be useful if we could have a machine guess how urgent a ticket is, based on the description, so the urgent tickets can be resolved first.

For the given data set, perform text classification using SVM and find out the accuracy of the model.

PROGRAM

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

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

from sklearn.svm import LinearSVC

with open("tickets.txt") as f:

tickets = f.read().strip().split("\n")

with open("labels\_4.txt") as f:

labels = f.read().strip().split("\n")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tickets, labels, test\_size=0.1, random\_state=1337)

vectorizer = CountVectorizer()

svm = LinearSVC()

X\_train = vectorizer.fit\_transform(X\_train)

X\_test = vectorizer.transform(X\_test)

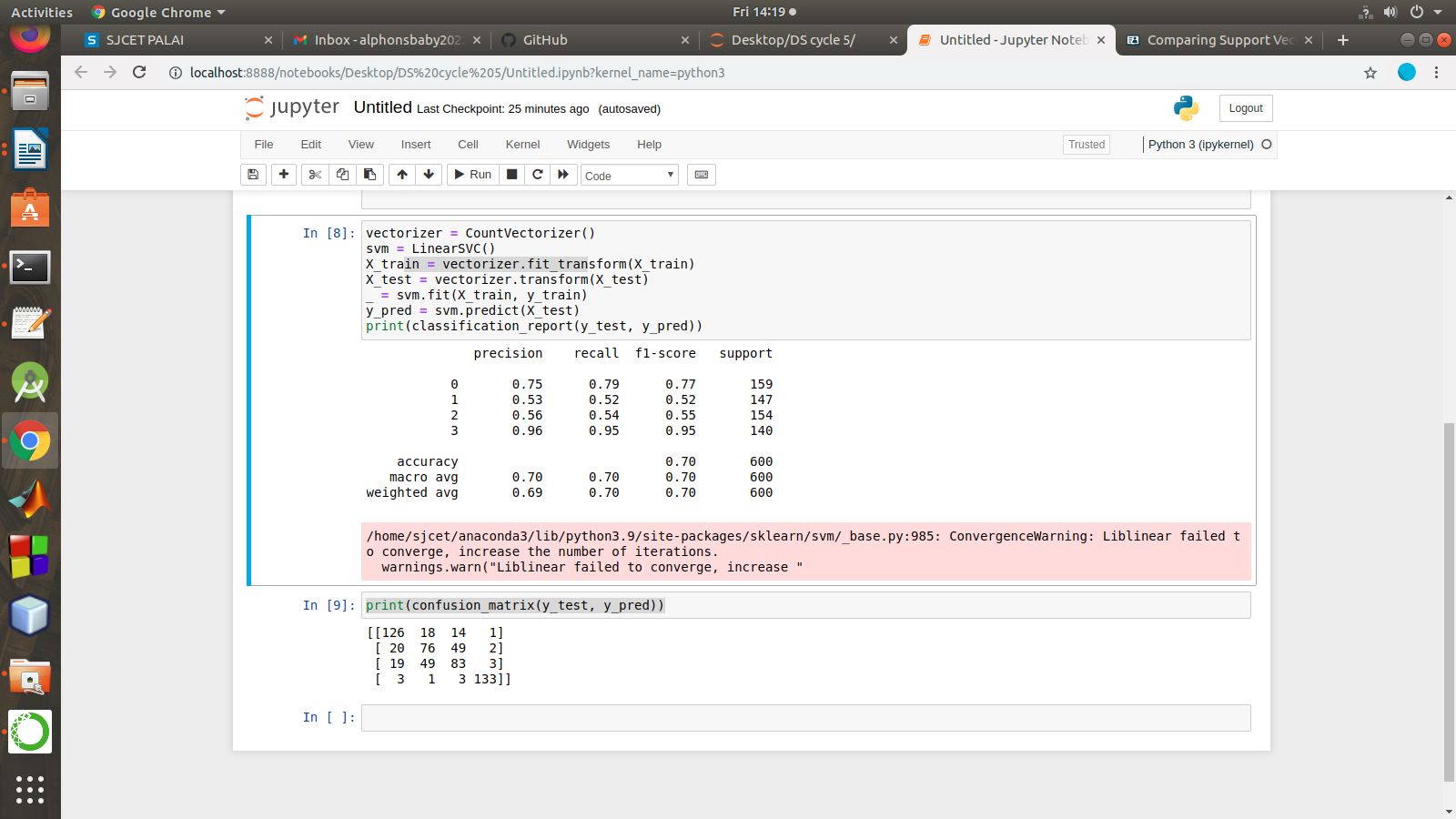
\_ = svm.fit(X\_train, y\_train)

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

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

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

OUTPUT



**LAB CYCLE 5 PART -2 SVM**

Given dataset contains 200 records and five columns, two of which describe the customer’s annual income and spending score. The latter is a value from 0 to 100. The higher the number, the more this customer has spent with the company in the past:

Functions to familiarize:

The purpose of Kmeans.fit() is to train the model with data.

The purpose of Kmeans.predict() is to apply a trained model to data

==================================================

Q. Using k means clustering create 6 clusters of customers based on their spending pattern.

1. Visualize the same in a scatter plot with each cluster in a different color scheme.
2. Display the cluster labels of each point.(print cluster indexes)
3. Display the cluster centers.
4. Use different values of K and visualize the same using scatter plot

PROGRAM

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

customers = pd.read\_csv('customer\_data.csv')

customers.head()

1)

kmeans = KMeans(n\_clusters=6, random\_state=0)

kmeans.fit(points)

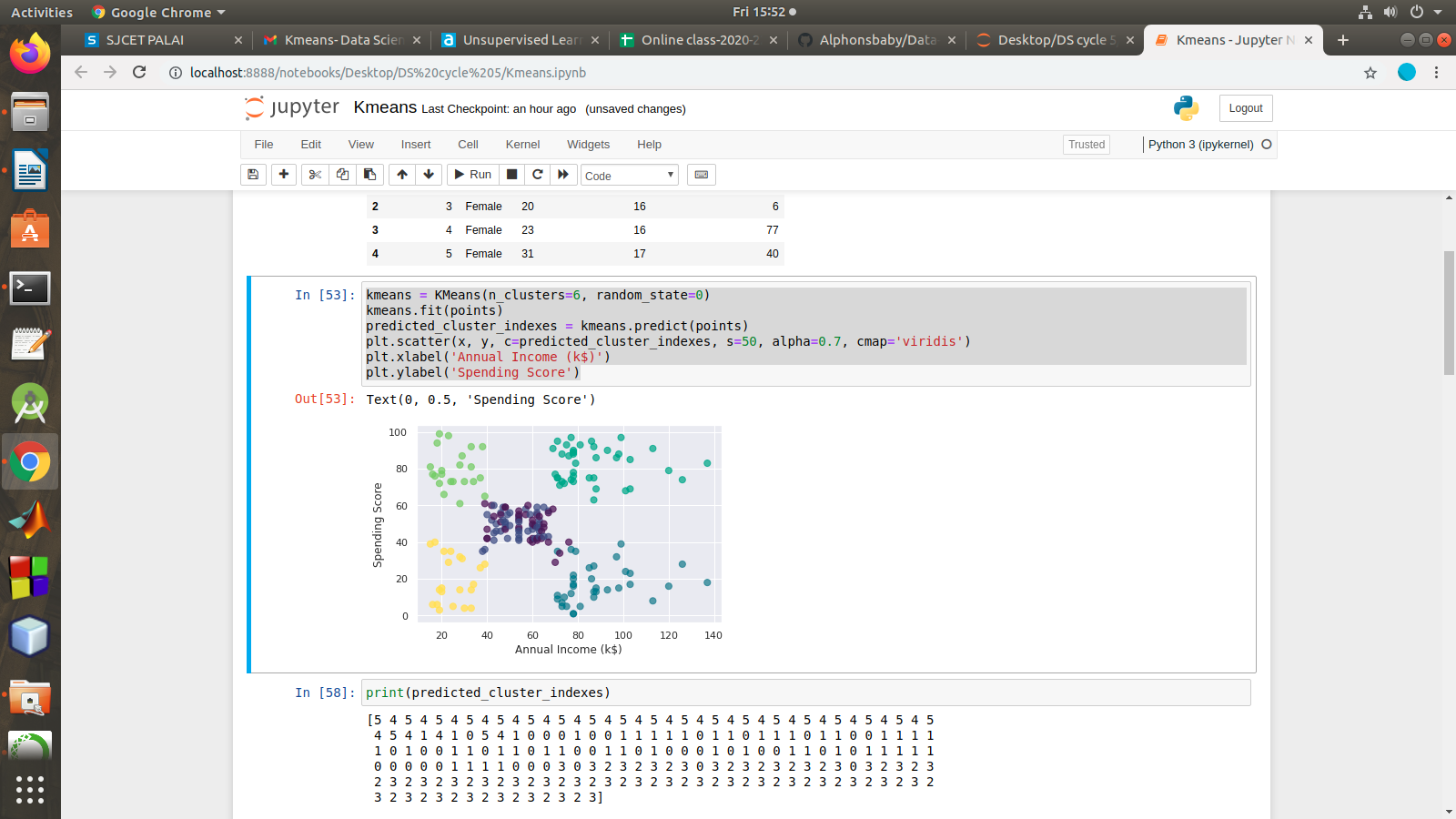
predicted\_cluster\_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted\_cluster\_indexes, s=50, alpha=0.7, cmap='viridis')

plt.xlabel('Annual Income (k$)')

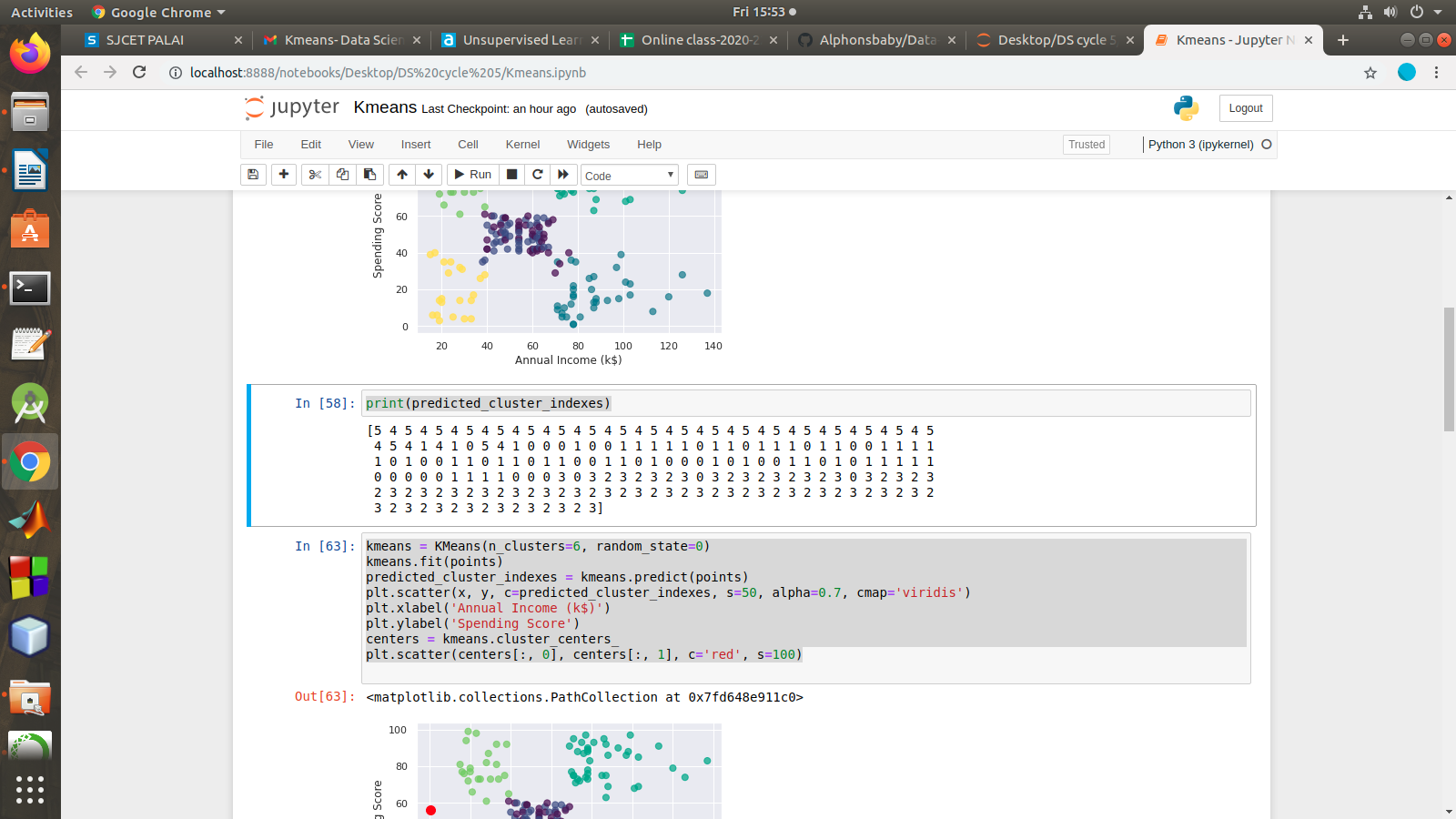
plt.ylabel('Spending Score')

OUTPUT



2)print(predicted\_cluster\_indexes)

OUTPUT



3)kmeans = KMeans(n\_clusters=6, random\_state=0)

kmeans.fit(points)

predicted\_cluster\_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted\_cluster\_indexes, s=50, alpha=0.7, cmap='viridis')

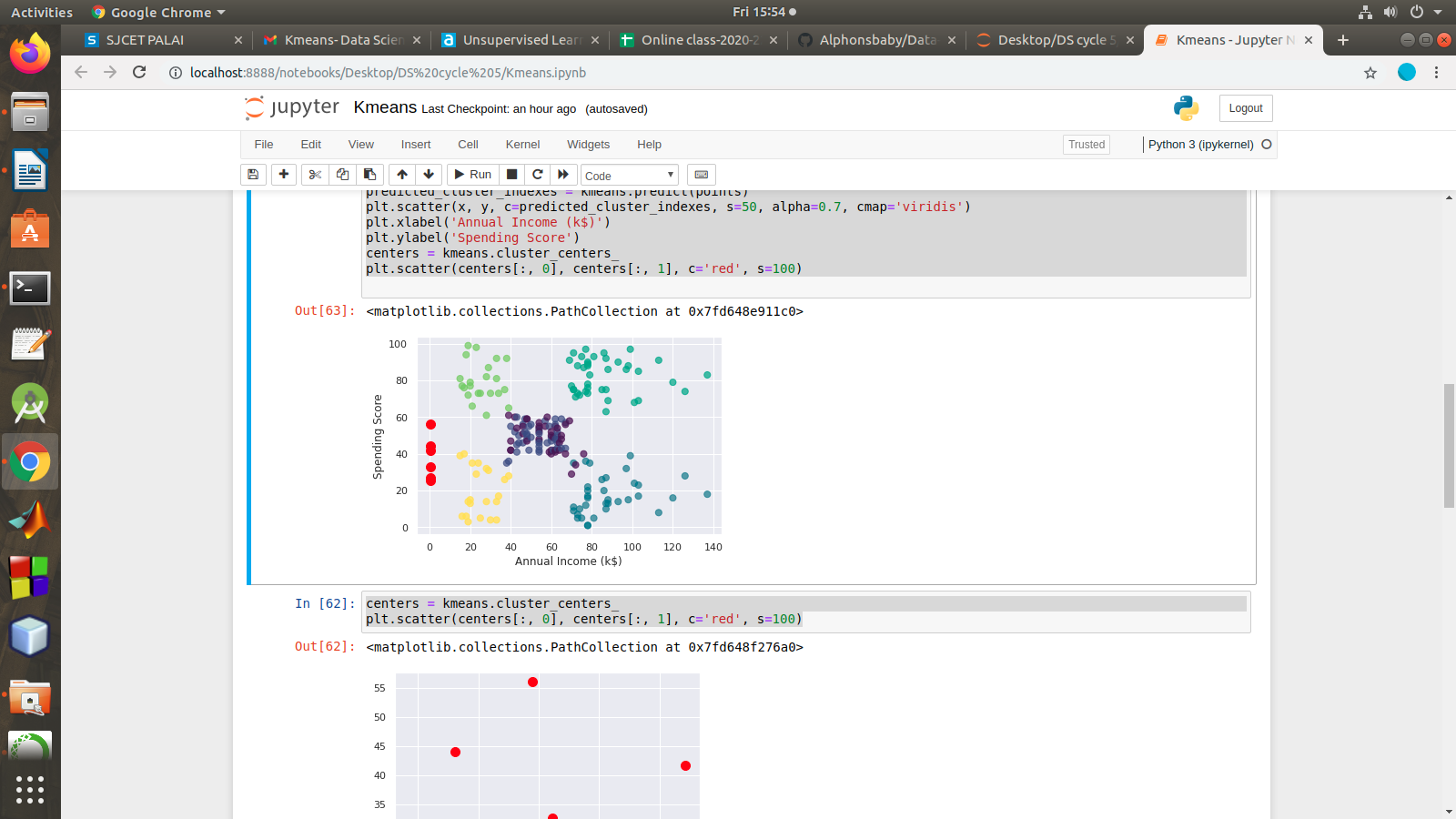
plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)

OUTPUT



4)

kmeans = KMeans(n\_clusters=4, random\_state=0)

kmeans.fit(points)

predicted\_cluster\_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted\_cluster\_indexes, s=50, alpha=0.7, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)

OUTPUT

