Semester: ∏

Subject: Applied MACHINE AND DEEP learning

Name: YUKTA KRISHNA CHAUDHARI

Class: M.Sc.Computer Science PART-1

Roll No.: 10

**CERTIFICATE**

*Miss.* **Yukta Krishna Chaudhari.**

*Roll No.* **10** *Exam Seat No.*  *has satisfactorily completed the Practical in* **Applied Machine And Deep Learning** *as laid down in the regulation of University of Mumbai for the purpose of MSc*

*Computer Science* **Semester-**II **(Practical)** *Examination* **2022-2023.**

*Date:*

*Place:* **Kalyan**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Head*

*Department of Computer Science*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Signature of Examiners*

*Professor In-Charge* 1) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Computer Science*

2) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr. No. | Date | Practical Name | Pg. No. | Remark |
| 1. |  | Implement Linear Regression (Diabetes Dataset) |  |  |
| 2. |  | Implement Logistic Regression (Iris Dataset) |  |  |
| 3. |  | Implements Multinomial Logistic Regression (Iris Dataset) |  |  |
| 4. |  | Implement SVM classifier (Iris Dataset) |  |  |
| 5. |  | Train and fine-tune a Decision Tree for the Moons Dataset |  |  |
| 6. |  | Train an SVM regressor on the California Housing Dataset |  |  |
| 7. |  | Implement Batch Gradient Descent with early stopping for Softmax Regression |  |  |
| 8. |  | Implement MLP for classification of handwritten digits (MNIST Dataset) |  |  |
| 9. |  | Classification of images of clothing using Tensorflow (Fashion MNIST dataset) |  |  |
| 10. |  | Implement Regression to predict fuel efficiency using Tensorflow (Auto MPG dataset) |  |  |

**Practical No 1**

**Aim:-** Implement Linear Regression (Diabetes Dataset)

**Background Information:-**

**Linear Regression:**

1. Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task.
2. Regression models a target prediction value based on independent variables.
3. It is mostly used for finding out the relationship between variables and forecasting.
4. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.
5. There are many names for a regression’s dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand.
6. The independent variables can be called exogenous variables, predictor variables, or regressors.
7. Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable.
8. For example, in finance, linear regression might be used to understand the relationship between a company’s stock price and its earnings, or to predict the future value of a currency based on its past performance.

**Diabetes Dataset:**

1. There are several datasets available online for diabetes prediction. One such dataset is available on Kaggle. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and contains diagnostic measurements of patients to predict whether a patient has diabetes or not.
2. Another dataset is available on CDC which provides access to the latest diabetes data and statistics through the National Diabetes Statistics Report and the Diabetes Report Card.
3. Microsoft Learn also provides a diabetes dataset which has 442 samples with 10 features, making it ideal for getting started with machine learning algorithms.

**Code:-**

**Libraries Required –** matplotlib, numpy, scikit-learn

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the diabetes dataset

diabetes\_X, diabetes\_y **=** datasets**.**load\_diabetes(return\_X\_y**=True**)

# Use only one feature

diabetes\_X **=** diabetes\_X[:, np**.**newaxis, 2]

# Split the data into training/testing sets

diabetes\_X\_train **=** diabetes\_X[:**-**20]

diabetes\_X\_test **=** diabetes\_X[**-**20:]

# Split the targets into training/testing sets

diabetes\_y\_train **=** diabetes\_y[:**-**20]

diabetes\_y\_test **=** diabetes\_y[**-**20:]

# Create linear regression object

regr **=** linear\_model**.**LinearRegression()

# Train the model using the training sets

regr**.**fit(diabetes\_X\_train, diabetes\_y\_train)



# Make predictions using the testing set

diabetes\_y\_pred **=** regr**.**predict(diabetes\_X\_test)

# The coefficients

print('Coefficients: \n', regr**.**coef\_)

# The mean squared error

print('Mean squared error: %.2f'

**%** mean\_squared\_error(diabetes\_y\_test, diabetes\_y\_pred))

# The coefficient of determination: 1 is perfect prediction

print('Coefficient of determination: %.2f'

**%** r2\_score(diabetes\_y\_test, diabetes\_y\_pred))



#Scatter Plot

plt**.**scatter(diabetes\_X\_test, diabetes\_y\_test, color**=**'black')

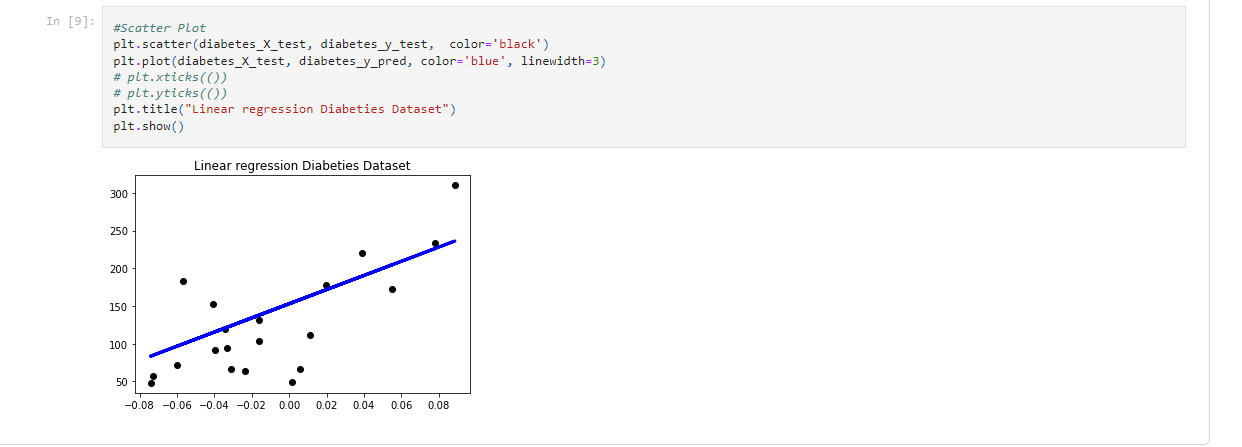
plt**.**plot(diabetes\_X\_test, diabetes\_y\_pred, color**=**'blue', linewidth**=**3)

# plt.xticks(())

# plt.yticks(())

plt**.**title("Linear regression Diabeties Dataset")

plt**.**show()



**Practical No 2**

**Aim:-** Implement Logistic Regression (Iris Dataset)

**Background Information:-**

**Logistic Regression:**

1. Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
2. Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be of a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
3. Logistic Regression is much like Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
4. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
5. The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

**Iris Dataset:**

1. The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
2. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
3. The columns in this dataset are:
   * Id
   * SepalLengthCm
   * SepalWidthCm
   * PetalLengthCm
   * PetalWidthCm
   * Species

**Code:-**

**Libraries Required -** pandas, numpy, os, matplotlib, seaborn

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

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

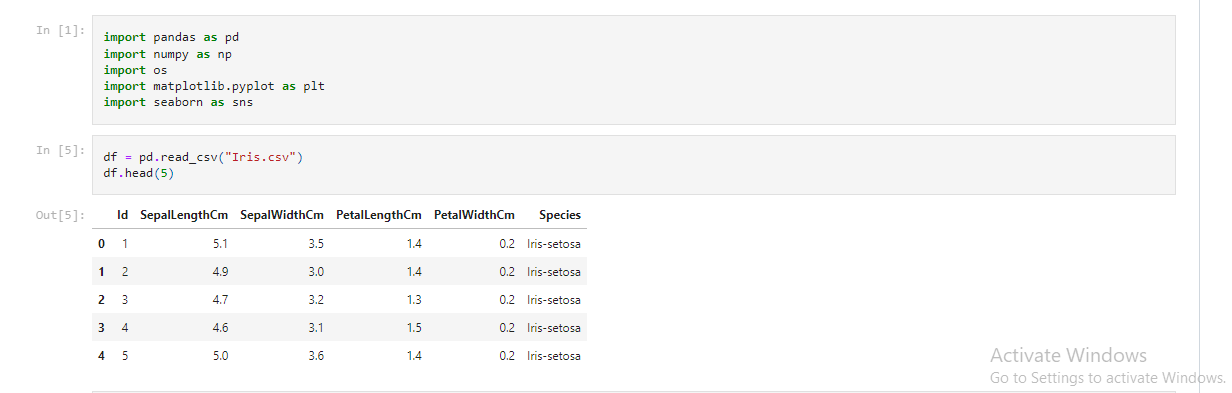
**import** os

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

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

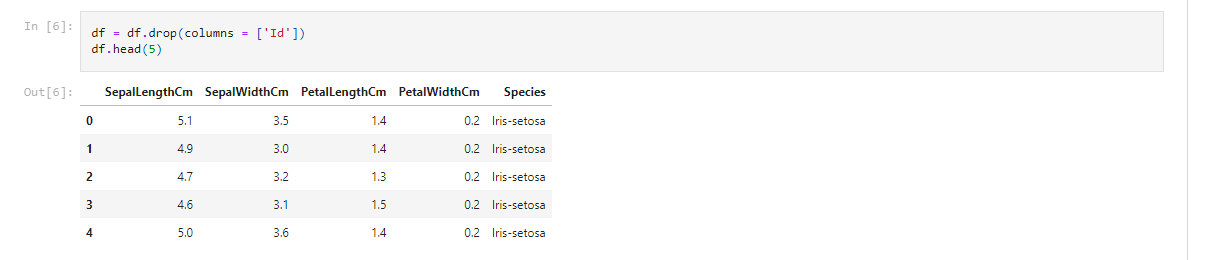
df **=** pd**.**read\_csv("Iris.csv")

df**.**head(5)

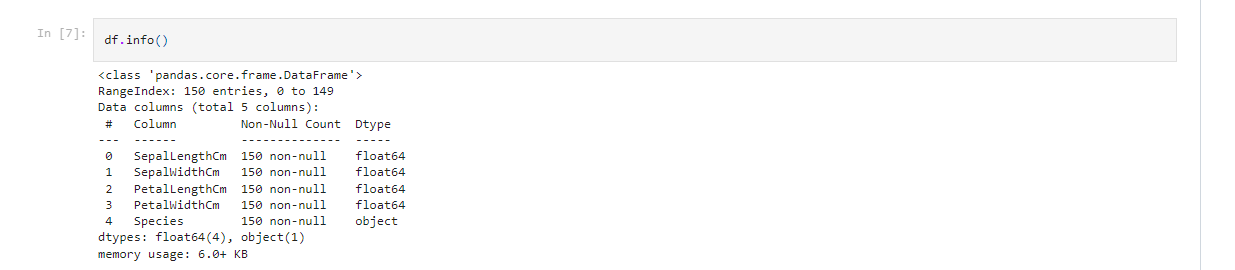


df **=** df**.**drop(columns **=** ['Id'])

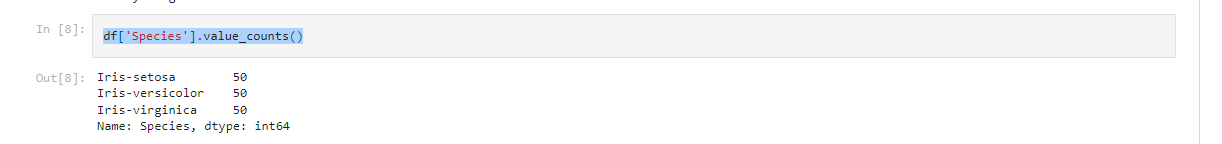
df**.**head(5)



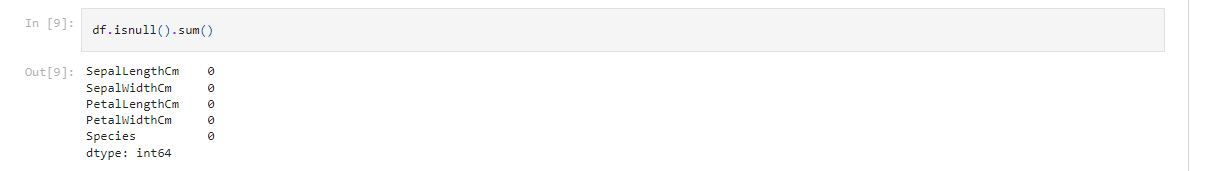
df**.**info()



df['Species']**.**value\_counts()

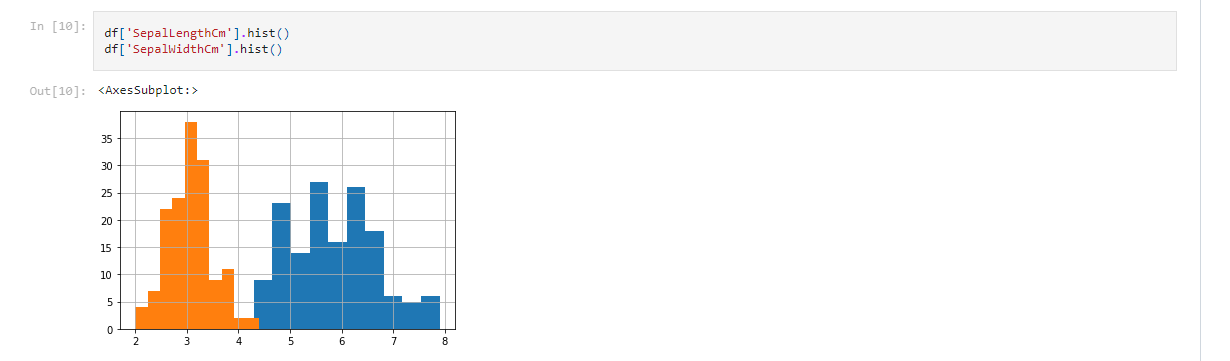


df**.**isnull()**.**sum()



df['SepalLengthCm']**.**hist()

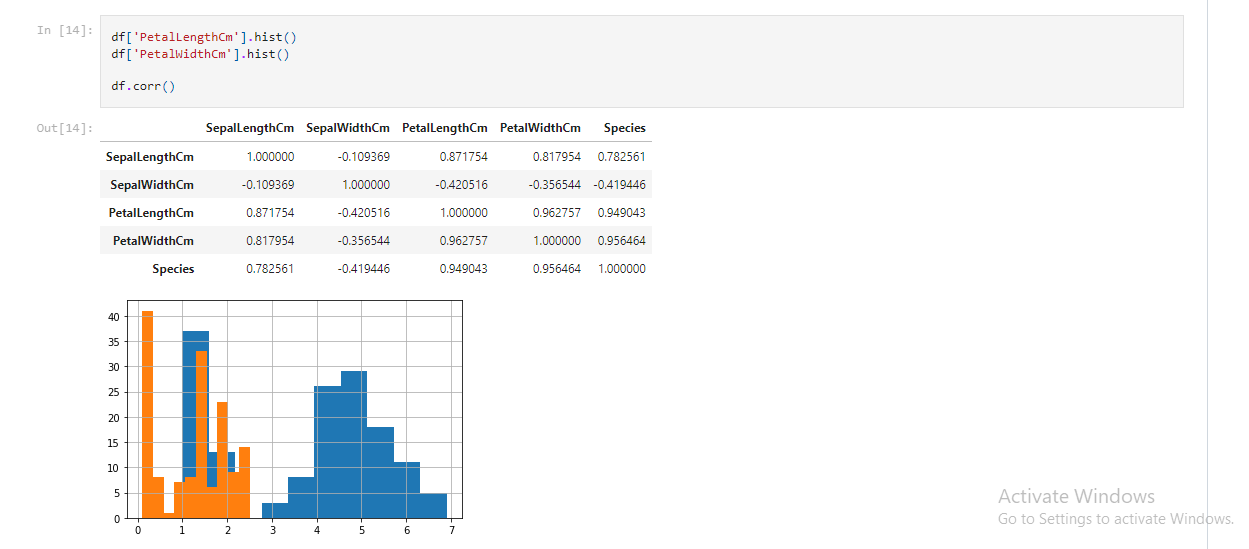
df['SepalWidthCm']**.**hist()



df['PetalLengthCm']**.**hist()

df['PetalWidthCm']**.**hist()

df**.**corr()

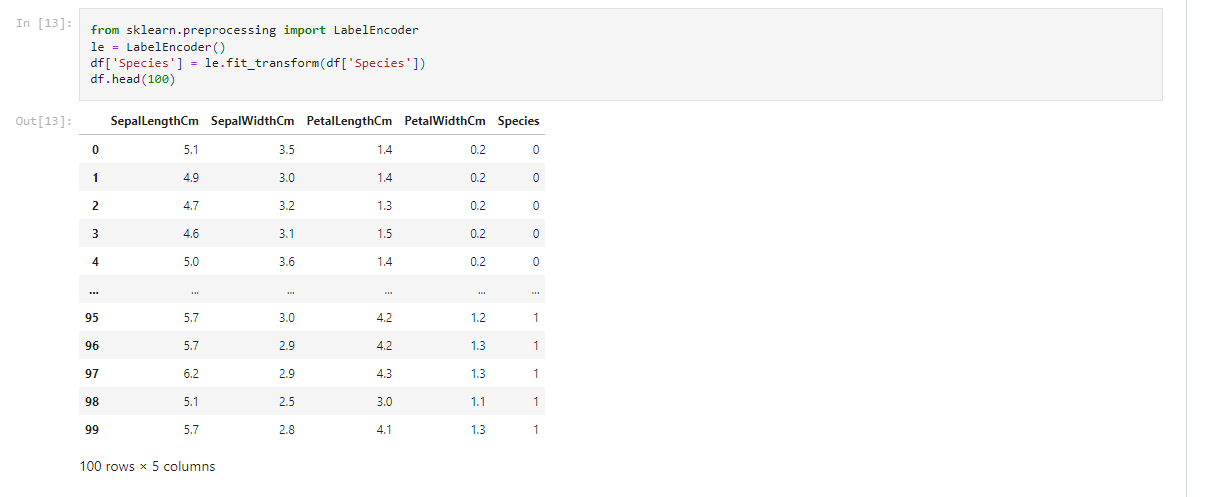


**from** sklearn.preprocessing **import** LabelEncoder

le **=** LabelEncoder()

df['Species'] **=** le**.**fit\_transform(df['Species'])

df**.**head(100)



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

X **=** df**.**drop(columns **=** ['Species'])

Y **=** df['Species']

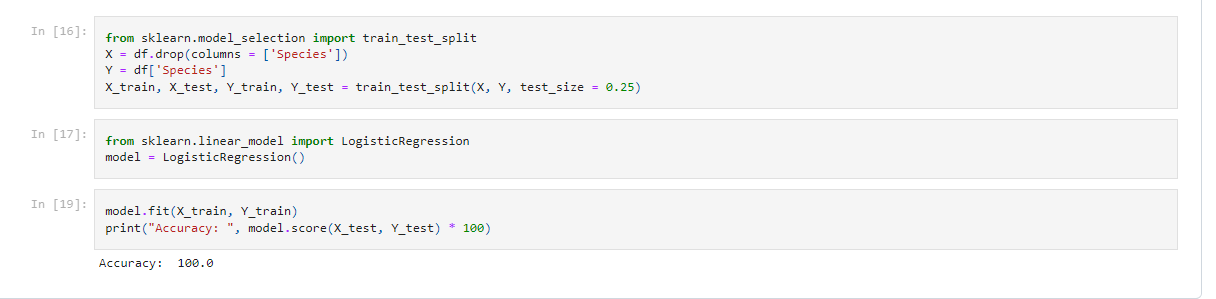
X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(X, Y, test\_size **=** 0.25)

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

model **=** LogisticRegression()

model**.**fit(X\_train, Y\_train)

print("Accuracy: ", model**.**score(X\_test, Y\_test) **\*** 100)



**Practical No 3**

**Aim:-** Implements Multinomial Logistic Regression (Iris Dataset)

**Background Information:-**

**Linear Regression:**

1. Multinomial Logistic Regression is like logistic regression but with a difference, that the target dependent variable can have more than two classes i.e., multiclass or polychotomous.
2. For example, the students can choose a major for graduation among the streams “Science”, “Arts” and “Commerce”, which is a multiclass dependent variable, and the independent variables can be marks, grade in competitive exams, Parents profile, interest etc.
3. Multinomial Logistic Regression is a classification technique that extends the logistic regression algorithm to solve multiclass possible outcome problems, given one or more independent variables.
4. This model is used to predict the probabilities of categorically dependent variable, which has two or more possible outcome classes. Whereas the logistic regression model is used when the dependent categorical variable has two outcome classes for example, students can either “Pass” or “Fail” in an exam or bank manager can either “Grant” or “Reject” the loan for a person.

**Iris Dataset:**

1. The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
2. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
3. The columns in this dataset are:

* Id
* SepalLengthCm
* SepalWidthCm
* PetalLengthCm
  + PetalWidthCm
  + Species

**Code:-**

**Libraries Required -** numpy, random, matplotlib, seaborn

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

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

**import** matplotlib.cm **as** cm

**import** random

**import** seaborn

seaborn.set(style='whitegrid'); seaborn.set\_context('talk')

%matplotlib inline

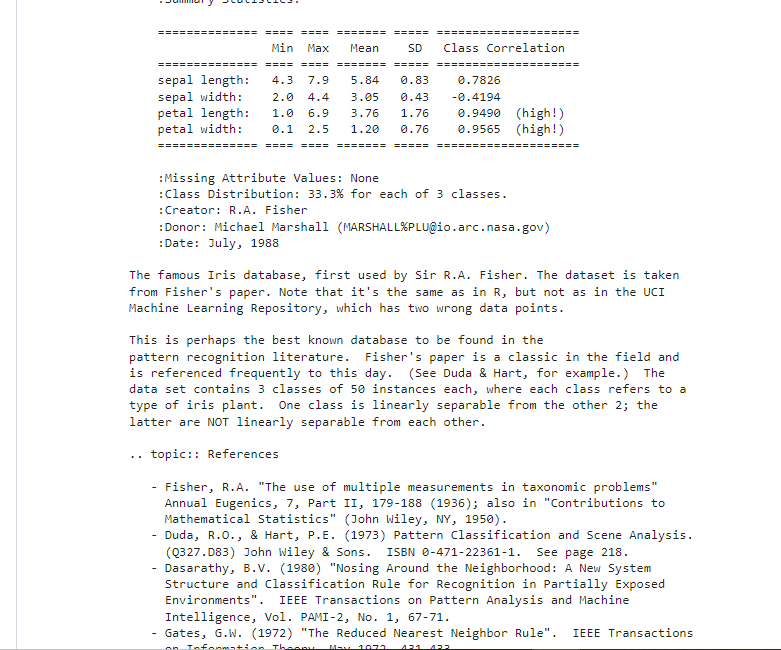
%config InlineBackend.figure\_format = 'retina'

from sklearn.datasets import load\_iris

iris\_data = load\_iris()

print(iris\_data['DESCR'])





n\_samples, n\_features **=** iris\_data**.**data**.**shape

**def** Show\_Diagram(x\_label,y\_label,title):

plt**.**figure(figsize**=**(10,4))

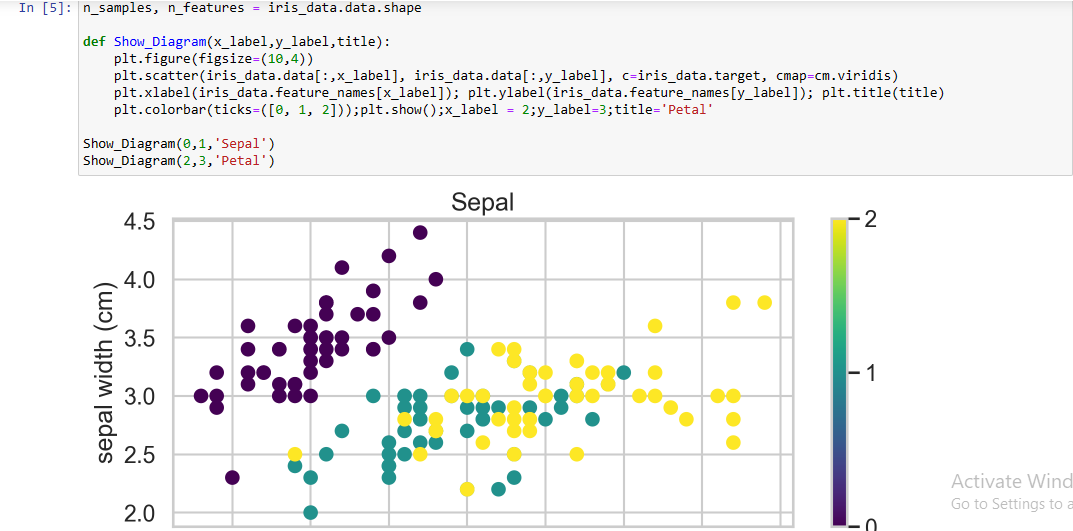
plt**.**scatter(iris\_data**.**data[:,x\_label], iris\_data**.**data[:,y\_label], c**=**iris\_data**.**target, cmap**=**cm**.**viridis)

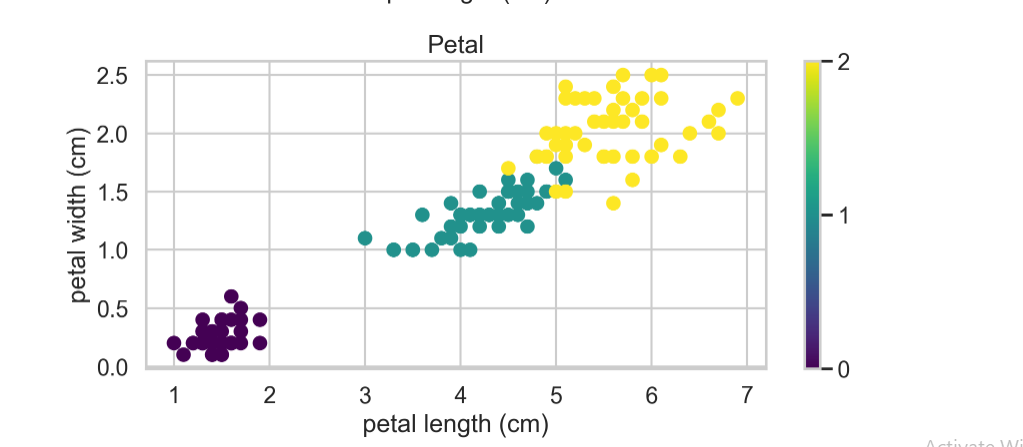
plt**.**xlabel(iris\_data**.**feature\_names[x\_label]); plt**.**ylabel(iris\_data**.**feature\_names[y\_label]); plt**.**title(title)

plt**.**colorbar(ticks**=**([0, 1, 2]));plt**.**show();x\_label **=** 2;y\_label**=**3;title**=**'Petal'

Show\_Diagram(0,1,'Sepal')

Show\_Diagram(2,3,'Petal')





random**.**seed(123)

**def** separate\_data():

""

A **=** iris\_dataset[0:40]

tA **=** iris\_dataset[40:50]

B **=** iris\_dataset[50:90]

tB **=** iris\_dataset[90:100]

C **=** iris\_dataset[100:140]

tC **=** iris\_dataset[140:150]

train **=** np**.**concatenate((A,B,C))

test **=** np**.**concatenate((tA,tB,tC))

**return** train,test

train\_porcent **=** 80 *# Train*

test\_porcent **=** 20 *# Test*

iris\_dataset **=** np**.**column\_stack((iris\_data**.**data,iris\_data**.**target**.**T)) *#Join X and Y*

iris\_dataset **=** list(iris\_dataset)

random**.**shuffle(iris\_dataset)

train\_file , test\_file **=** separate\_data()

train\_X **=** np**.**array([k[:4] **for** k **in** train\_file])

train\_y **=** np**.**array([k[4] **for** k **in** train\_file])

test\_X **=** np**.**array([k[:4] **for** k **in** test\_file])

test\_y **=** np**.**array([k[4] **for** k **in** test\_file])

plt.figure(figsize=(10,10));plt.subplot(2,2,3)

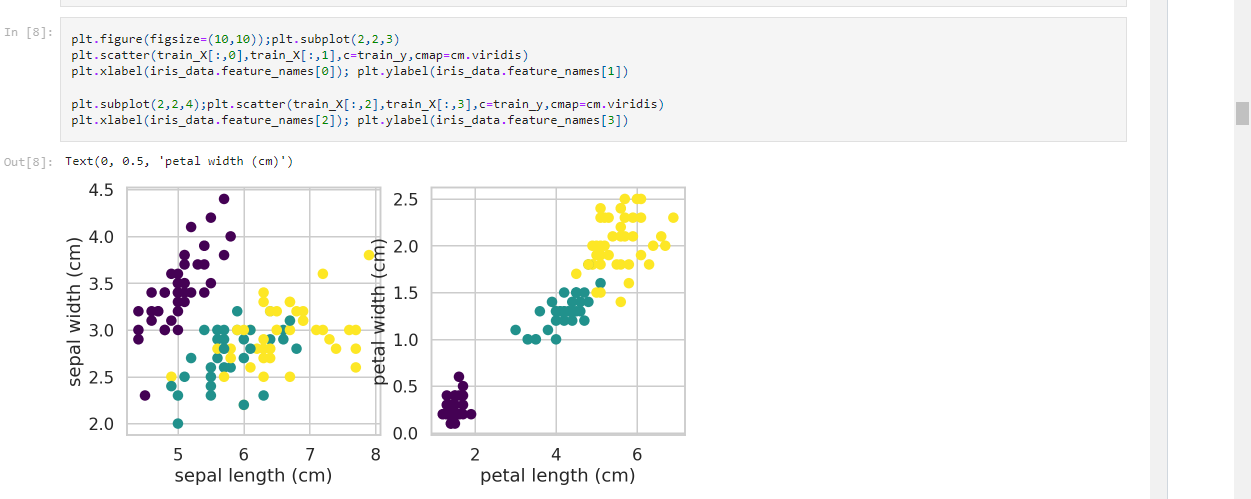
plt.scatter(train\_X[:,0],train\_X[:,1],c=train\_y,cmap=cm.viridis)

plt.xlabel(iris\_data.feature\_names[0]); plt.ylabel(iris\_data.feature\_names[1])

plt.subplot(2,2,4);plt.scatter(train\_X[:,2],train\_X[:,3],c=train\_y,cmap=cm.viridis)

plt.xlabel(iris\_data.feature\_names[2]); plt.ylabel(iris\_data.feature\_names[3])





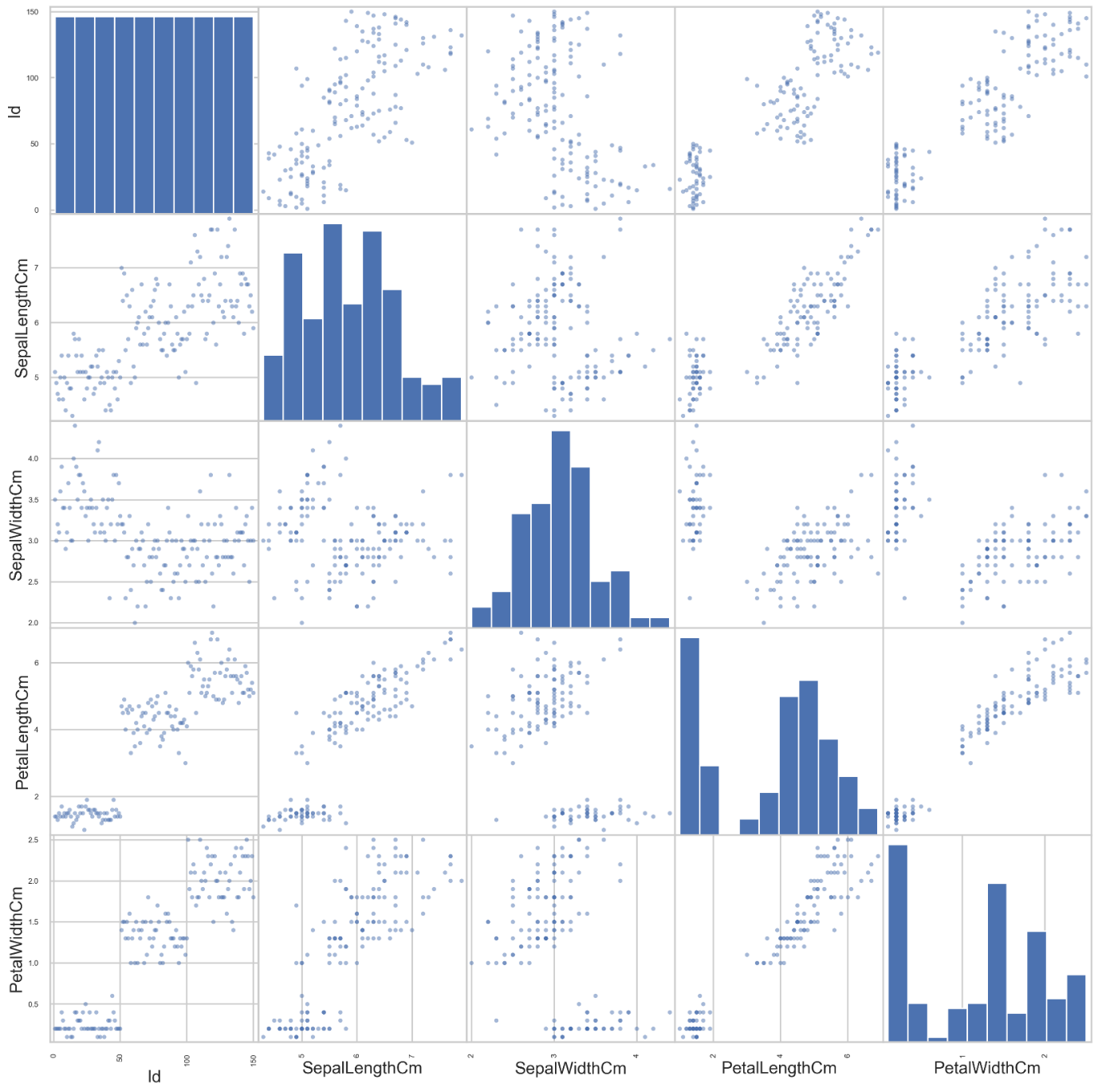
**import** pandas

**from** pandas.plotting **import** scatter\_matrix

dataset **=** pandas**.**read\_csv('Iris.csv')

scatter\_matrix(dataset, alpha**=**0.5, figsize**=**(20, 20))

plt**.**show()



dataset**.**hist(alpha**=**0.5, figsize**=**(20, 20), color**=**'green')

plt**.**show()



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

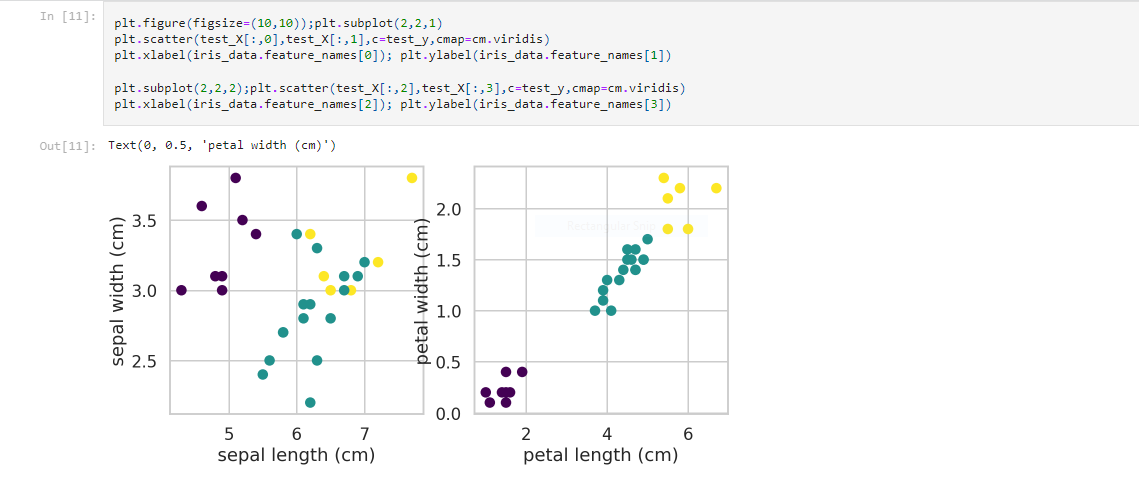
plt.subplot(2,2,1)

plt.scatter(test\_X[:,0],test\_X[:,1],c=test\_y,cmap=cm.viridis)

plt.xlabel(iris\_data.feature\_names[0]); plt.ylabel(iris\_data.feature\_names[1])

plt.subplot(2,2,2);plt.scatter(test\_X[:,2],test\_X[:,3],c=test\_y,cmap=cm.viridis)

plt.xlabel(iris\_data.feature\_names[2]); plt.ylabel(iris\_data.feature\_names[3])



**Practical No 4**

**Aim:-** Implement SVM classifier (Iris Dataset)

**Background Information:-**

**SVM Classifier:**

1. Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
2. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
3. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence algorithm is termed as Support Vector Machine.

**Iris Dataset:**

1. The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.
2. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
3. The columns in this dataset are:

* Id
* SepalLengthCm
* SepalWidthCm
* PetalLengthCm
  + PetalWidthCm
  + Species

**Code:-**

**Libraries Required –** pandas, matplotlib, seaborn

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

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

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

#Define the col names

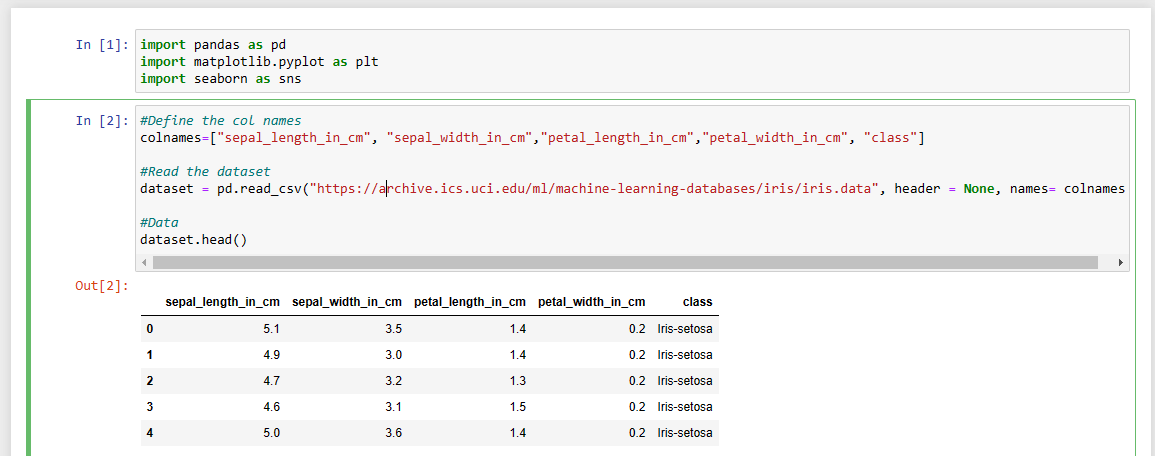
colnames=["sepal\_length\_in\_cm", "sepal\_width\_in\_cm","petal\_length\_in\_cm","petal\_width\_in\_cm", "class"]

#Read the dataset

dataset = pd.read\_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", header = None, names= colnames )

#Data

dataset.head()

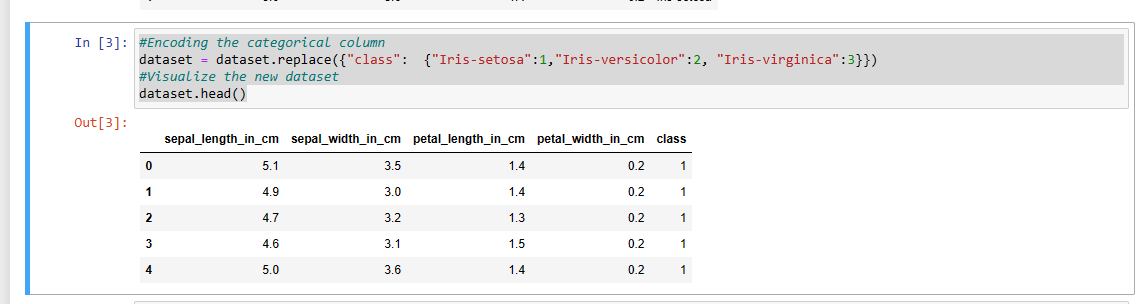


#Encoding the categorical column

dataset = dataset.replace({"class": {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})

#Visualize the new dataset

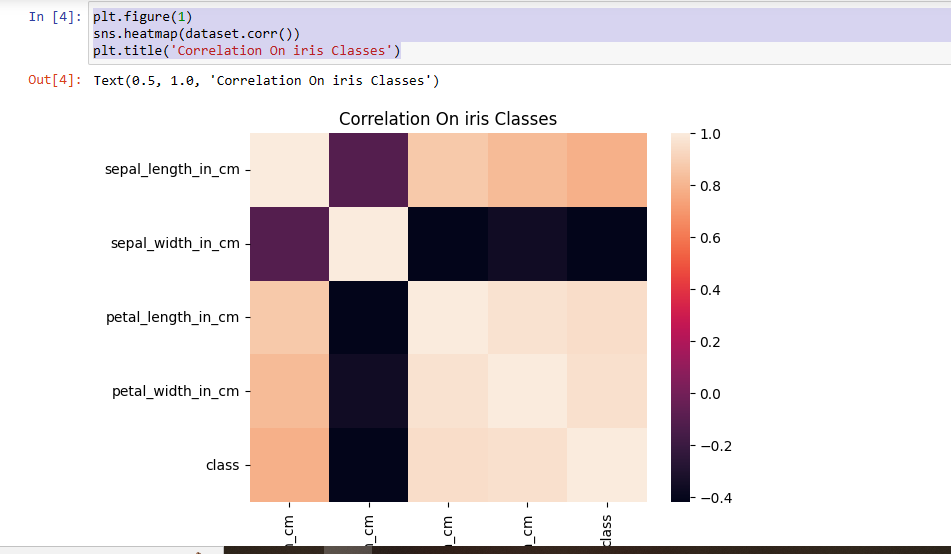
dataset.head()



plt.figure(1)

sns.heatmap(dataset.corr())

plt.title('Correlation On iris Classes')



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

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

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

#Create the SVM model

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

#Fit the model for the data

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

#Make the prediction

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

from sklearn.metrics import confusion\_matrix

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

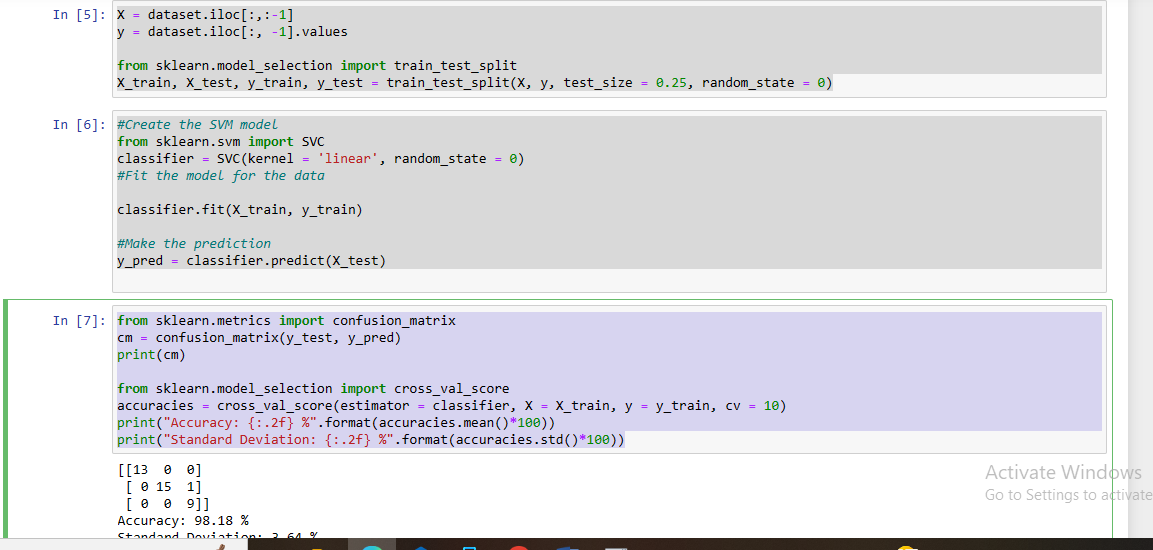
print(cm)

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

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)

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

print("Standard Deviation: {:.2f} %".format(accuracies.std()\*100))



**Practical NO 5**

**Aim:-** Train and fine-tune a Decision Tree for the Moons Dataset

**Background Information:-**

**Decision Tree:**

1. A decision tree is a decision support hierarchical model that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.
2. It is one way to display an algorithm that only contains conditional control statements.
3. Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
4. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.
5. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
6. The decisions or the test are performed based on features of the given dataset.
7. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
8. It is called a decision tree because, like a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
9. To build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
10. A decision tree simply asks a question and based on the answer (Yes/No), it further splits the tree into subtrees.
11. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal but are also a popular tool in machine learning.

**Moons Dataset:**

1. Make two interleaving half circles.
2. A simple toy dataset to visualize clustering and classification algorithms.
3. It's taken from Sklearn.

**Code:-**

**Libraries Required -** numpy, matplotlib

import numpy as np

import matplotlib.pyplot as plt

# This function will help in visualization of our dataset.

def plot\_dataset(X, y, axes):

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

plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs",alpha = 0.5)

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^",alpha = 0.2)

plt.axis(axes)

plt.grid(True, which='both')

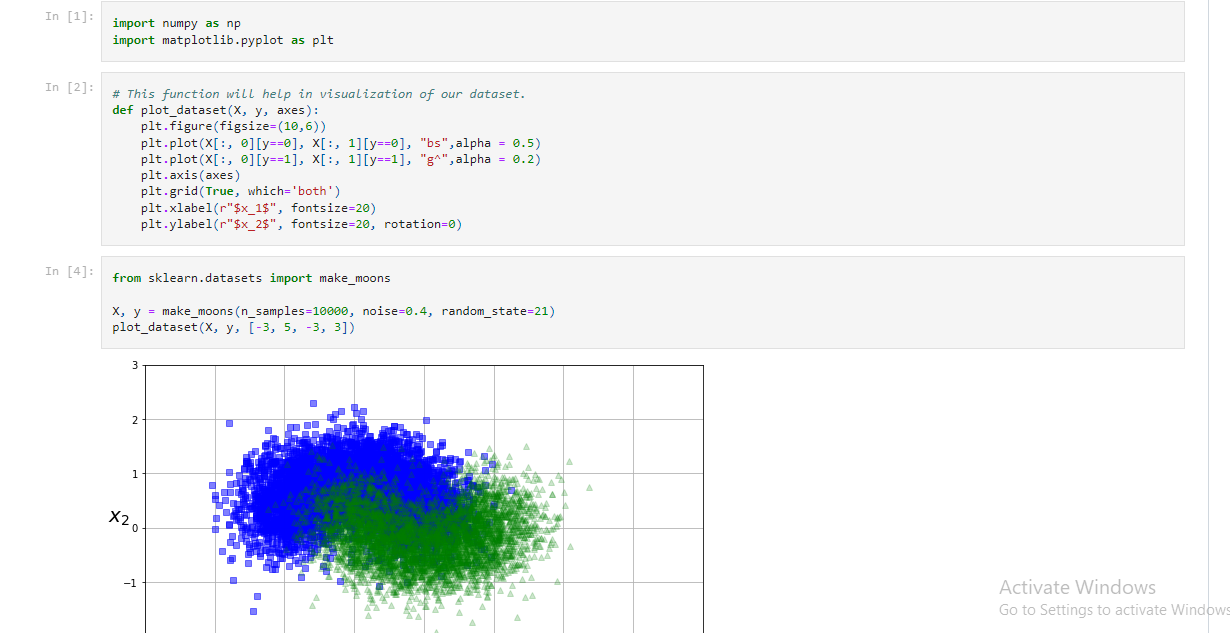
plt.xlabel(r"$x\_1$", fontsize=20)

plt.ylabel(r"$x\_2$", fontsize=20, rotation=0)

from sklearn.datasets import make\_moons

X, y = make\_moons(n\_samples=10000, noise=0.4, random\_state=21)

plot\_dataset(X, y, [-3, 5, -3, 3])



**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.tree import DecisionTreeClassifier

tree\_clf = DecisionTreeClassifier()

**from** sklearn.model\_selection **import** GridSearchCV

parameter **=** {

'criterion' : ["gini", "entropy"],

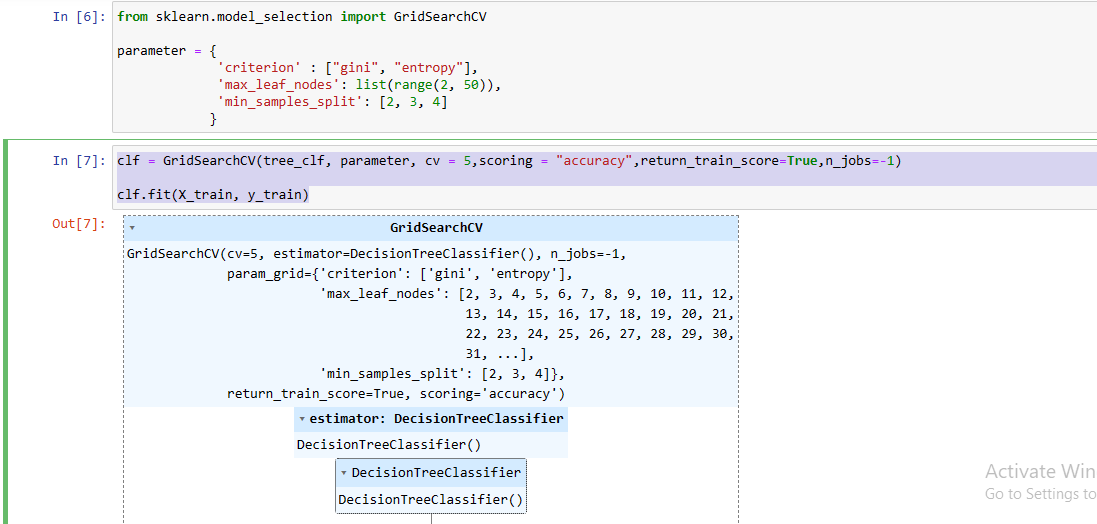
'max\_leaf\_nodes': list(range(2, 50)),

'min\_samples\_split': [2, 3, 4]

}

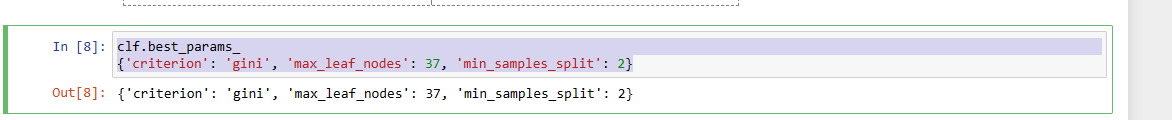
clf = GridSearchCV(tree\_clf, parameter, cv = 5,scoring = "accuracy",return\_train\_score=True,n\_jobs=-1)

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



clf.best\_params\_

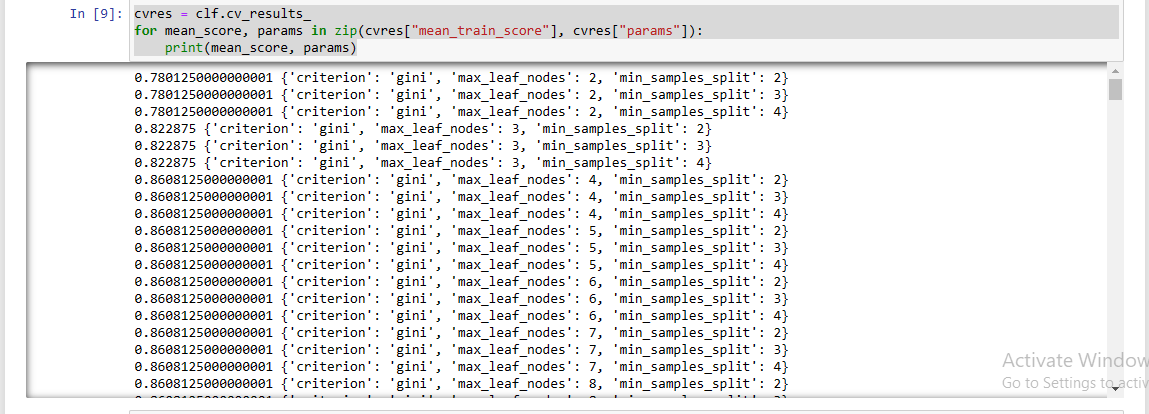
{'criterion': 'gini', 'max\_leaf\_nodes': 37, 'min\_samples\_split': 2}



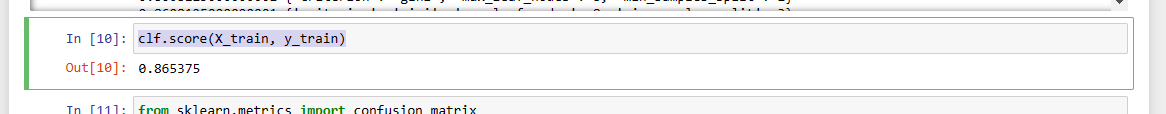
cvres = clf.cv\_results\_

for mean\_score, params in zip(cvres["mean\_train\_score"], cvres["params"]):

print(mean\_score, params)



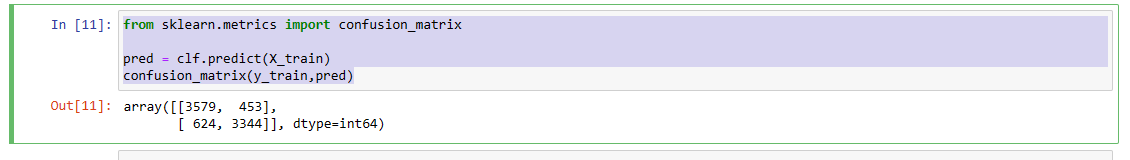
clf.score(X\_train, y\_train)



from sklearn.metrics import confusion\_matrix

pred = clf.predict(X\_train)

confusion\_matrix(y\_train,pred)

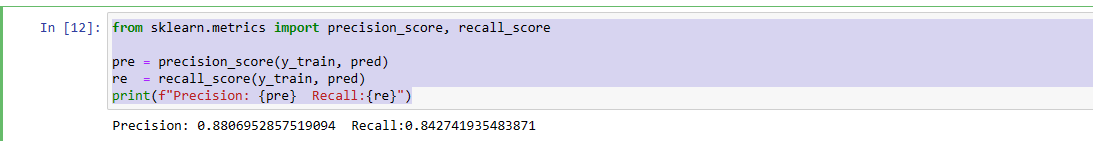


from sklearn.metrics import precision\_score, recall\_score

pre = precision\_score(y\_train, pred)

re = recall\_score(y\_train, pred)

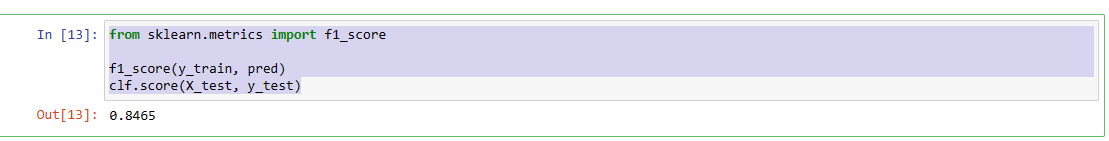
print(f"Precision: {pre} Recall:{re}")



from sklearn.metrics import f1\_score

f1\_score(y\_train, pred)

clf.score(X\_test, y\_test)



**Practical No 6**

**Aim:-** Train an SVM regressor on the California Housing Dataset

**Background Information:-**

**SVM Regressor:**

1. Support Vector Regression as the name suggests is a regression algorithm that supports both linear and non-linear regressions.
2. This method works on the principle of the Support Vector Machine.
3. SVR differs from SVM in the way that SVM is a classifier that is used for predicting discrete categorical labels while SVR is a regressor that is used for predicting continuous ordered variables.
4. In simple regression, the idea is to minimize the error rate while in SVR the idea is to fit the error inside a certain threshold which means, work of SVR is to approximate the best value within a given margin called ε- tube.

**California Housing Dataset:**

1. The data contains information from the 1990 California census. So, although it may not help you with predicting current housing prices like the Zillow Zestimate dataset, it does provide an accessible introductory dataset for teaching people about the basics of machine learning.
2. The data pertains to the houses found in each California district and some summary stats about them based on the 1990 census data. Be warned the data isn't cleaned so there are some preprocessing steps required!
3. The columns are as follows; their names are self-explanatory:

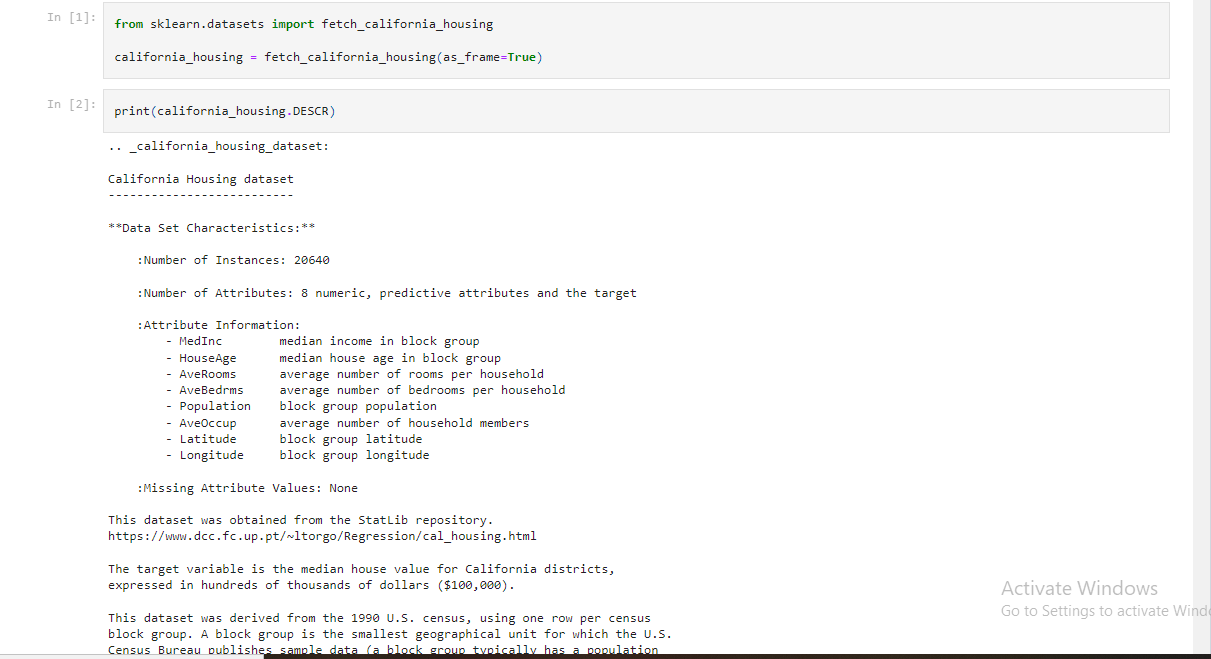
* longitude
* latitude
* housing\_median\_age
* total\_rooms
* total\_bedrooms
* population
* households
* median\_income
* median\_house\_value
* ocean\_proximity

**Code:-**

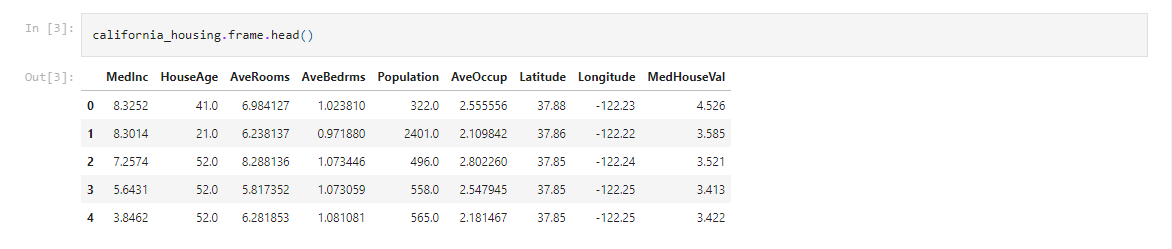
**from** sklearn.datasets **import** fetch\_california\_housing

california\_housing **=** fetch\_california\_housing(as\_frame**=True**)

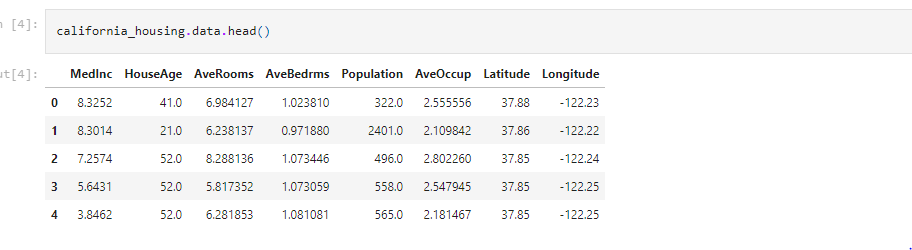
print(california\_housing**.**DESCR)



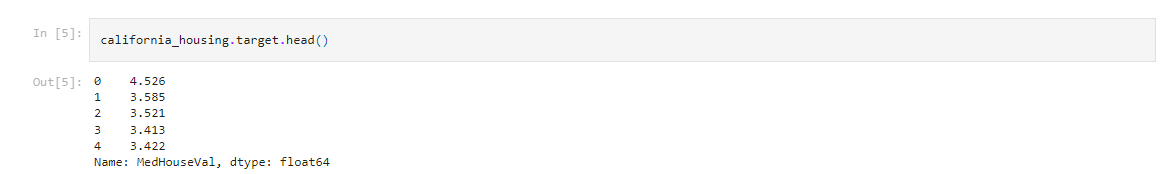
california\_housing**.**frame**.**head()



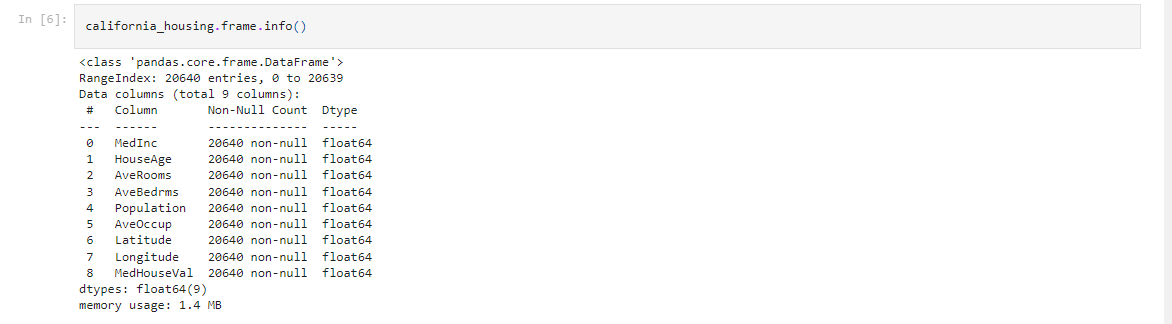
california\_housing**.**data**.**head()



california\_housing**.**target**.**head()



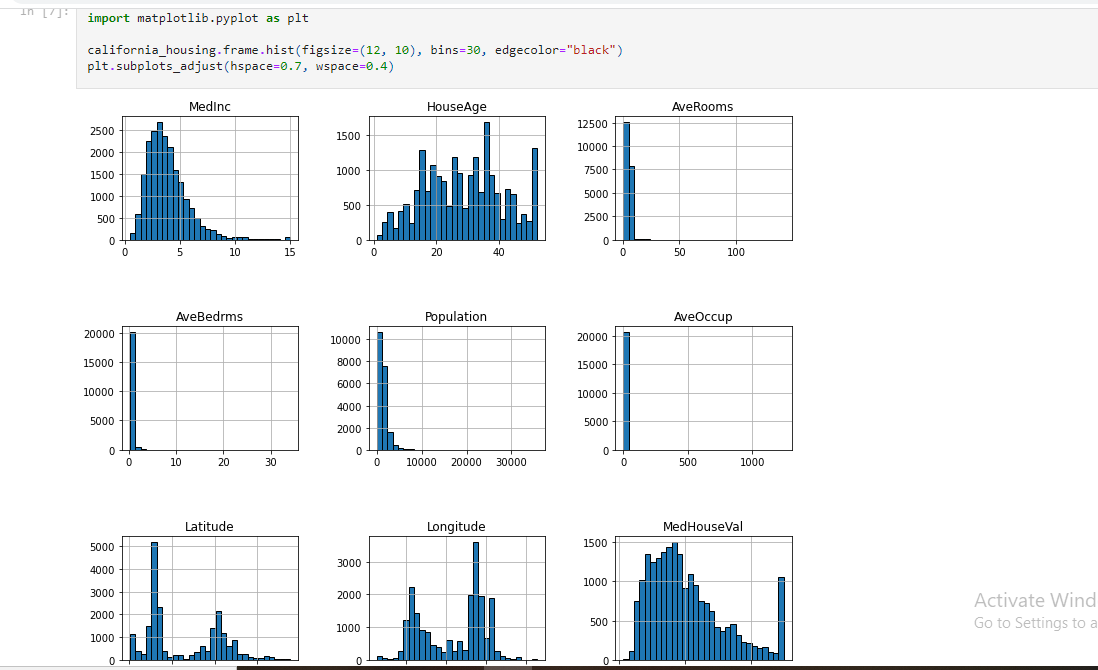
california\_housing**.**frame**.**info()



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

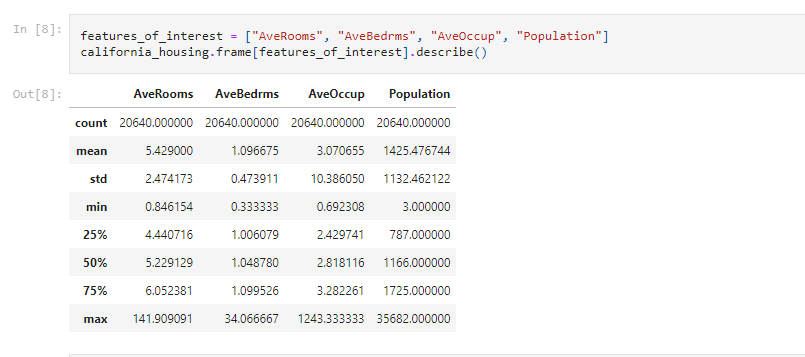
california\_housing**.**frame**.**hist(figsize**=**(12, 10), bins**=**30, edgecolor**=**"black")

plt**.**subplots\_adjust(hspace**=**0.7, wspace**=**0.4)



features\_of\_interest **=** ["AveRooms", "AveBedrms", "AveOccup", "Population"]

california\_housing**.**frame[features\_of\_interest]**.**describe()



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

sns**.**scatterplot(data**=**california\_housing**.**frame, x**=**"Longitude", y**=**"Latitude",

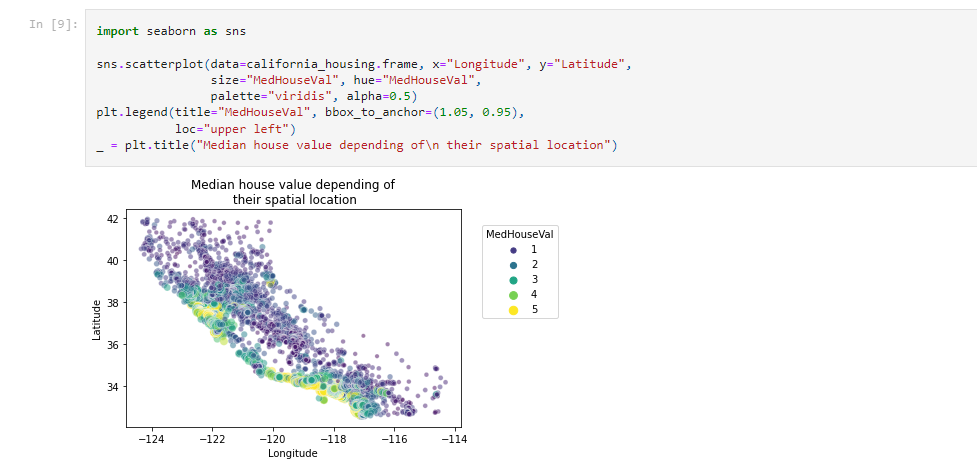
size**=**"MedHouseVal", hue**=**"MedHouseVal",

palette**=**"viridis", alpha**=**0.5)

plt**.**legend(title**=**"MedHouseVal", bbox\_to\_anchor**=**(1.05, 0.95),

loc**=**"upper left")

\_ **=** plt**.**title("Median house value depending of\n their spatial location")



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

rng **=** np**.**random**.**RandomState(0)

indices **=** rng**.**choice(np**.**arange(california\_housing**.**frame**.**shape[0]), size**=**500,

replace**=False**)

sns**.**scatterplot(data**=**california\_housing**.**frame**.**iloc[indices],

x**=**"Longitude", y**=**"Latitude",

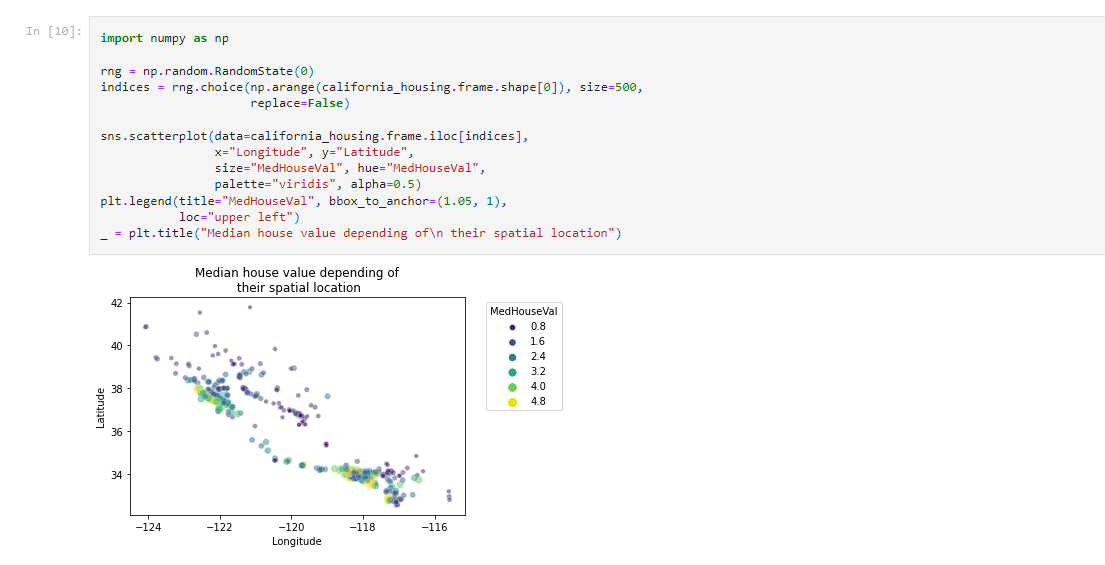
size**=**"MedHouseVal", hue**=**"MedHouseVal",

palette**=**"viridis", alpha**=**0.5)

plt**.**legend(title**=**"MedHouseVal", bbox\_to\_anchor**=**(1.05, 1),

loc**=**"upper left")

\_ **=** plt**.**title("Median house value depending of\n their spatial location")



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

*# Drop the unwanted columns*

columns\_drop **=** ["Longitude", "Latitude"]

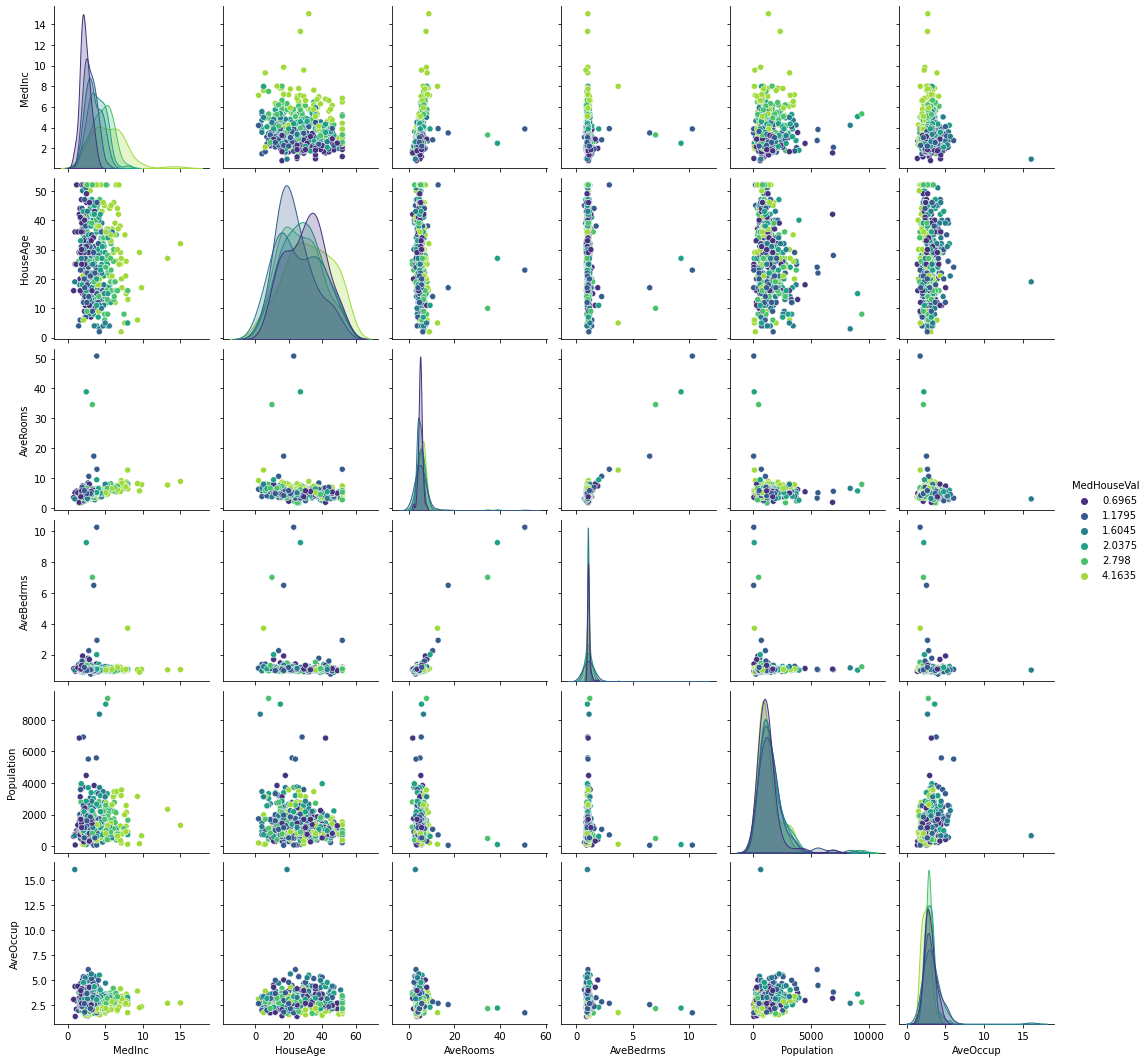
subset **=** california\_housing**.**frame**.**iloc[indices]**.**drop(columns**=**columns\_drop)

*# Quantize the target and keep the midpoint for each interval*

subset["MedHouseVal"] **=** pd**.**qcut(subset["MedHouseVal"], 6, retbins**=False**)

subset["MedHouseVal"] **=** subset["MedHouseVal"]**.**apply(**lambda** x: x**.**mid)

\_ **=** sns**.**pairplot(data**=**subset, hue**=**"MedHouseVal", palette**=**"viridis")



**from** sklearn.preprocessing **import** StandardScaler

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

**from** sklearn.pipeline **import** make\_pipeline

**from** sklearn.model\_selection **import** cross\_validate

alphas **=** np**.**logspace(**-**3, 1, num**=**30)

model **=** make\_pipeline(StandardScaler(), RidgeCV(alphas**=**alphas))

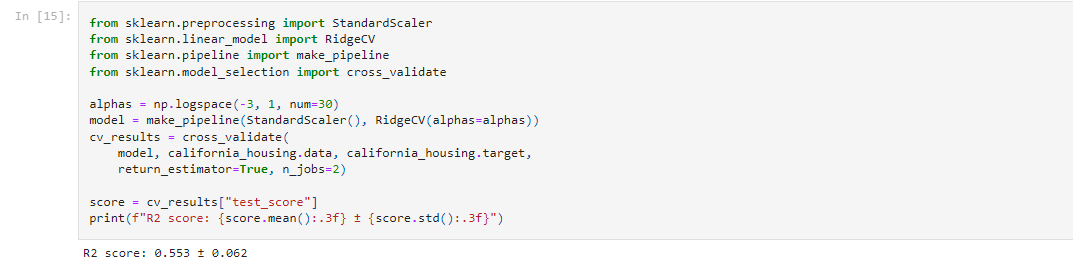
cv\_results **=** cross\_validate(

model, california\_housing**.**data, california\_housing**.**target,

return\_estimator**=True**, n\_jobs**=**2)

score **=** cv\_results["test\_score"]

print(f"R2 score: {score**.**mean():.3f} ± {score**.**std():.3f}")



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

coefs **=** pd**.**DataFrame(

[est[**-**1]**.**coef\_ **for** est **in** cv\_results["estimator"]],

columns**=**california\_housing**.**feature\_names

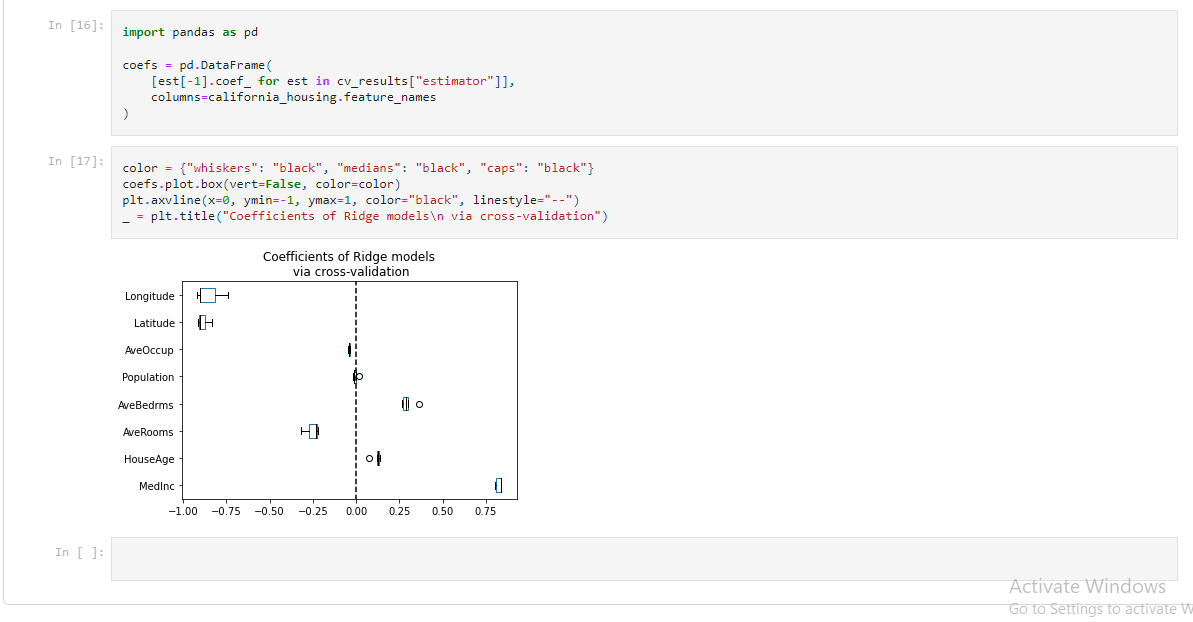
)

color **=** {"whiskers": "black", "medians": "black", "caps": "black"}

coefs**.**plot**.**box(vert**=False**, color**=**color)

plt**.**axvline(x**=**0, ymin**=-**1, ymax**=**1, color**=**"black", linestyle**=**"--")

\_ **=** plt**.**title("Coefficients of Ridge models\n via cross-validation")



**Practical No 7**

**Aim:-** Implement Batch Gradient Descent with early stopping for Softmax Regression

**Background Information:-**

**Batch Gradient Descent:**

1. Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples.
2. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.
3. Computes gradient using the whole Training sample.
4. Slow and computationally expensive algorithm.
5. Not suggested for huge training samples.
6. Deterministic in nature.
7. Gives optimal solution given sufficient time to converge.
8. No random shuffling of points is required.
9. Can’t escape shallow local minima easily.
10. Convergence is slow.

**SoftMax Regression:**

1. SoftMax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes in the target column.
2. In binary logistic regression, the labels were binary, that is for ith observation,
3. Softmax Regression (synonyms: Multinomial Logistic, Maximum Entropy Classifier, or just Multi-class Logistic Regression) is a generalization of logistic regression that we can use for multi-class classification (under the assumption that the classes are mutually exclusive).
4. In contrast, we use the (standard) Logistic Regression model in binary classification tasks.

**Code:-**

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

**import** scipy **as** sp

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

**from** sklearn.datasets **import** load\_iris

iris**=**load\_iris()

X**=**iris['data']

y**=**iris['target']

X\_with\_bias **=** np**.**c\_[np**.**ones([len(X), 1]), X]

np**.**random**.**seed(1234)

test\_ratio **=** 0.2

validation\_ratio **=** 0.2

total\_size **=** len(X\_with\_bias)

test\_size **=** int(total\_size **\*** test\_ratio)

validation\_size **=** int(total\_size **\*** validation\_ratio)

train\_size **=** total\_size **-** test\_size **-** validation\_size

rnd\_indices **=** np**.**random**.**permutation(total\_size)

X\_train **=** X\_with\_bias[rnd\_indices[:train\_size]]

y\_train **=** y[rnd\_indices[:train\_size]]

X\_valid **=** X\_with\_bias[rnd\_indices[train\_size:**-**test\_size]]

y\_valid **=** y[rnd\_indices[train\_size:**-**test\_size]]

X\_test **=** X\_with\_bias[rnd\_indices[**-**test\_size:]]

y\_test **=** y[rnd\_indices[**-**test\_size:]]

**def** one\_hot(Y):

nclasses**=**Y**.**max()**+**1

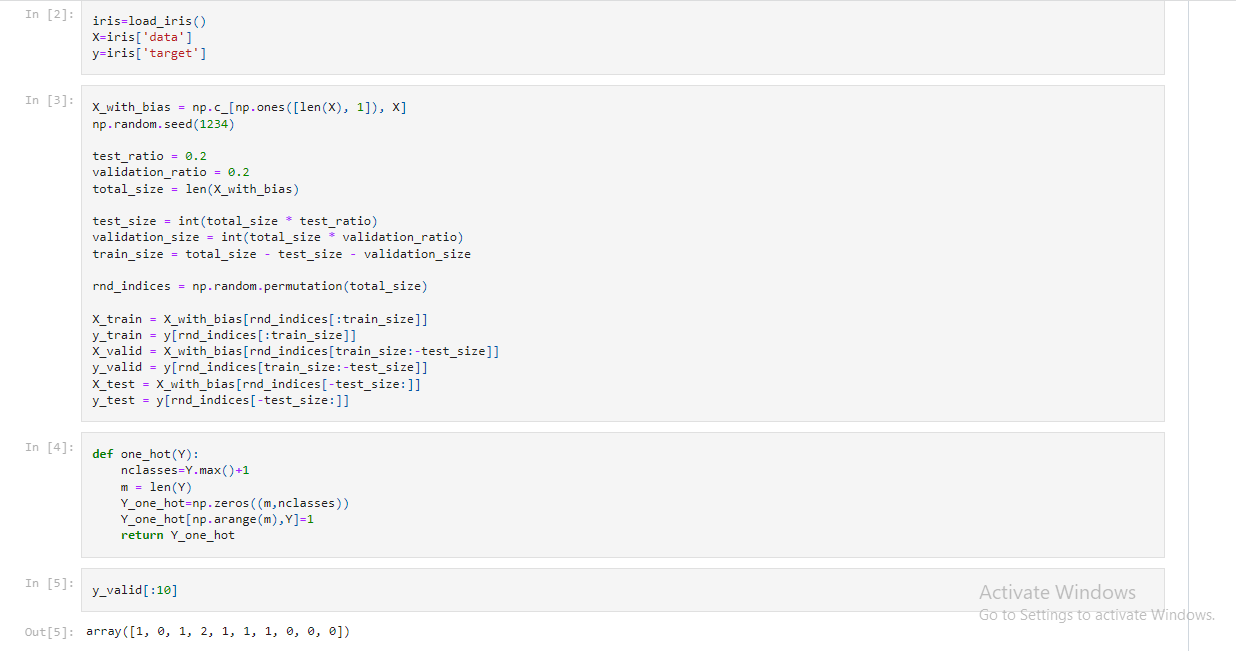
m **=** len(Y)

Y\_one\_hot**=**np**.**zeros((m,nclasses))

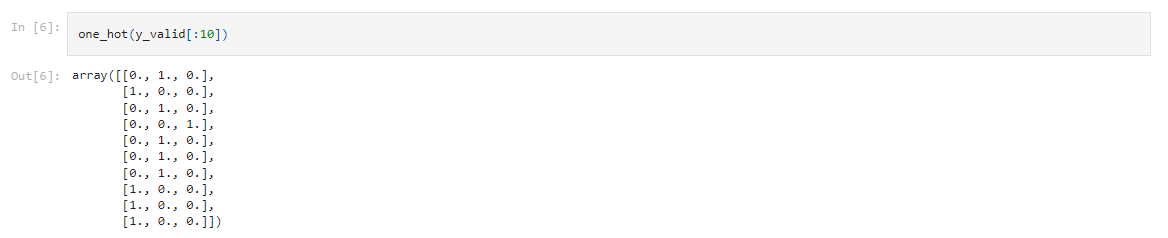
Y\_one\_hot[np**.**arange(m),Y]**=**1

**return** Y\_one\_hot

y\_valid[:10]



one\_hot(y\_valid[:10])



y\_train\_prob **=** one\_hot(y\_train)

y\_valid\_prob **=** one\_hot(y\_valid)

y\_test\_prob **=** one\_hot(y\_test)

**def** softmax(sk\_X):

top **=** np**.**exp(sk\_X)

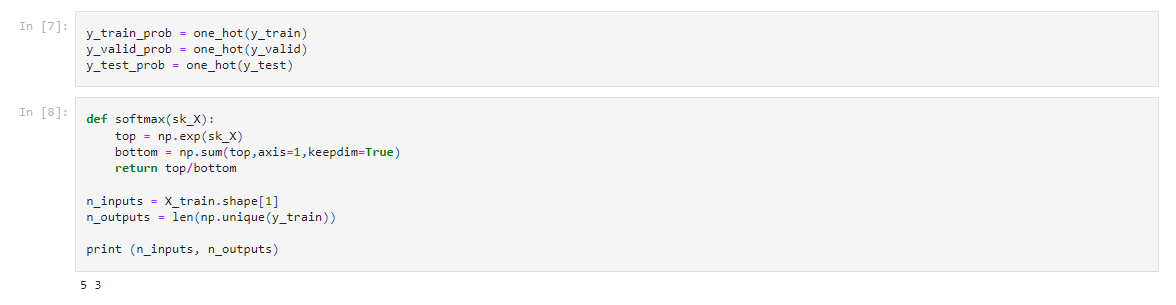
bottom **=** np**.**sum(top,axis**=**1,keepdim**=True**)

**return** top**/**bottom

n\_inputs **=** X\_train**.**shape[1]

n\_outputs **=** len(np**.**unique(y\_train))

print (n\_inputs, n\_outputs)



**Practical No 8**

**Aim:-** Implement MLP for classification of handwritten digits (MNIST Dataset)

**Background Information:-**

**MLPClassifier:**

1. MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network.
2. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.
3. One similarity though, with Scikit-Learn’s other classification algorithms is that implementing MLPClassifier takes no more effort than implementing Support Vectors or Naive Bayes or any other classifiers from Scikit-Learn.

**MNIST Dataset:**

1. The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.
2. The database is also widely used for training and testing in the field of machine learning.
3. It was created by "re-mixing" the samples from NIST's original datasets.
4. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments.
5. Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.
6. The MNIST database contains 60,000 training images and 10,000 testing images.
7. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.
8. The original creators of the database keep a list of some of the methods tested on it.

**Code:-**

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.neural\_network import MLPClassifier

import numpy as np

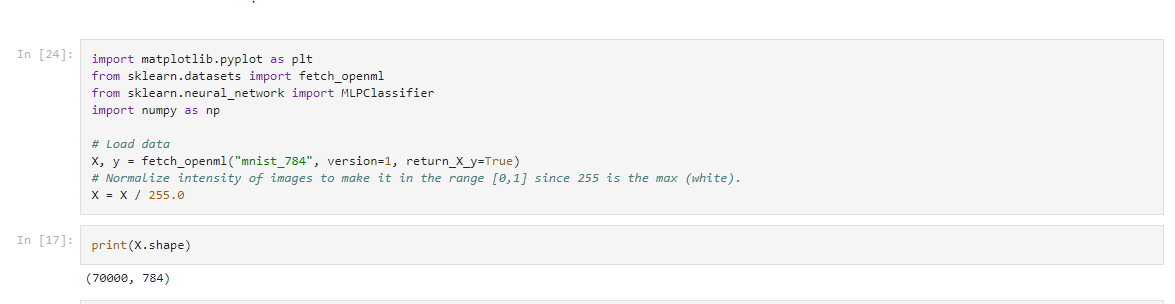
# Load data

X, y = fetch\_openml("mnist\_784", version=1, return\_X\_y=True)

# Normalize intensity of images to make it in the range [0,1] since 255 is the max (white).

X = X / 255.0

print(X.shape)



*# Split the data into train/test sets*

X\_train, X\_test = X[:60000], X[60000:]

y\_train, y\_test = y[:60000], y[60000:]

classifier = MLPClassifier(

hidden\_layer\_sizes=(50,20,10),

max\_iter=100,

alpha=1e-4,

solver="sgd",

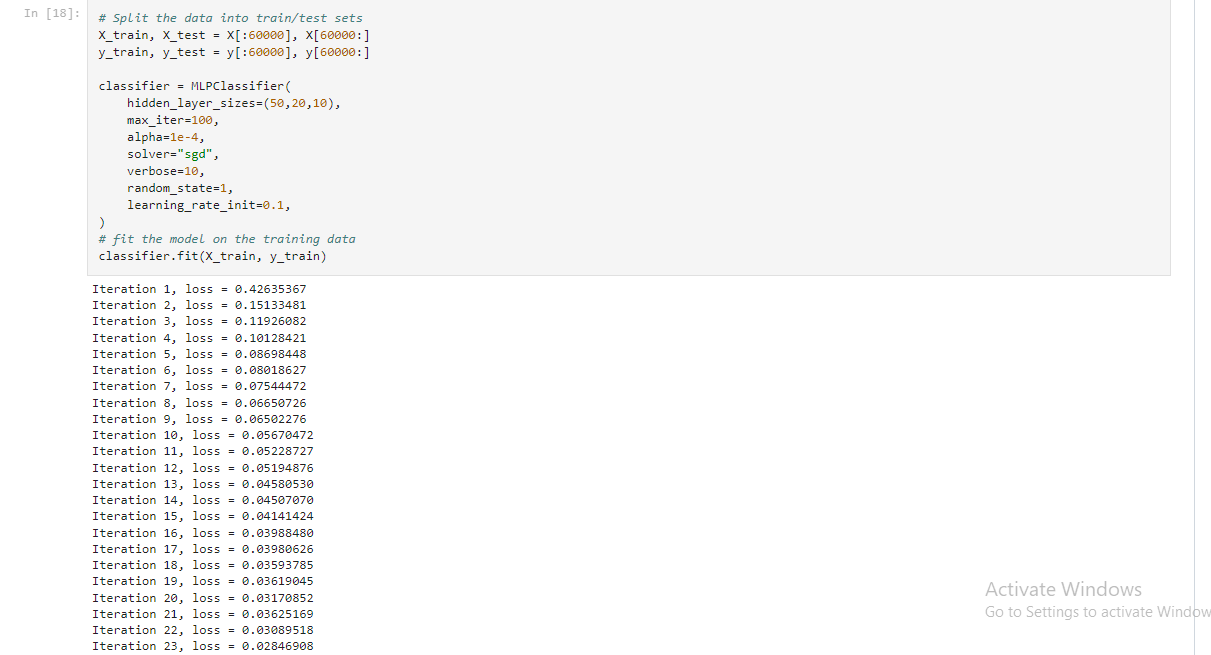
verbose=10,

random\_state=1,

learning\_rate\_init=0.1,)

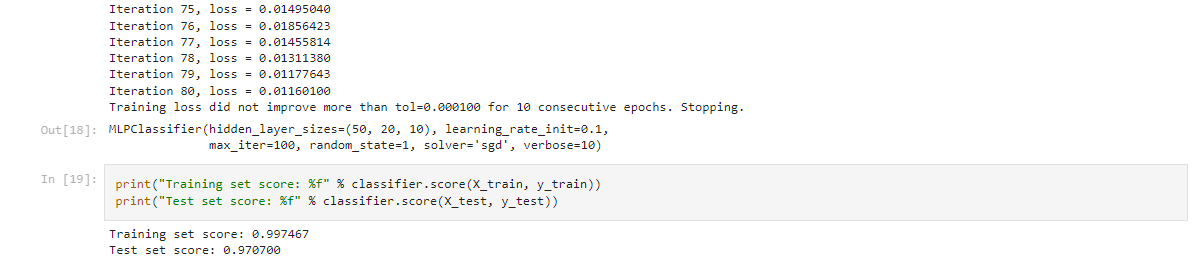
*# fit the model on the training data*

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



print("Training set score: %f" % classifier.score(X\_train, y\_train))

print("Test set score: %f" % classifier.score(X\_test, y\_test))



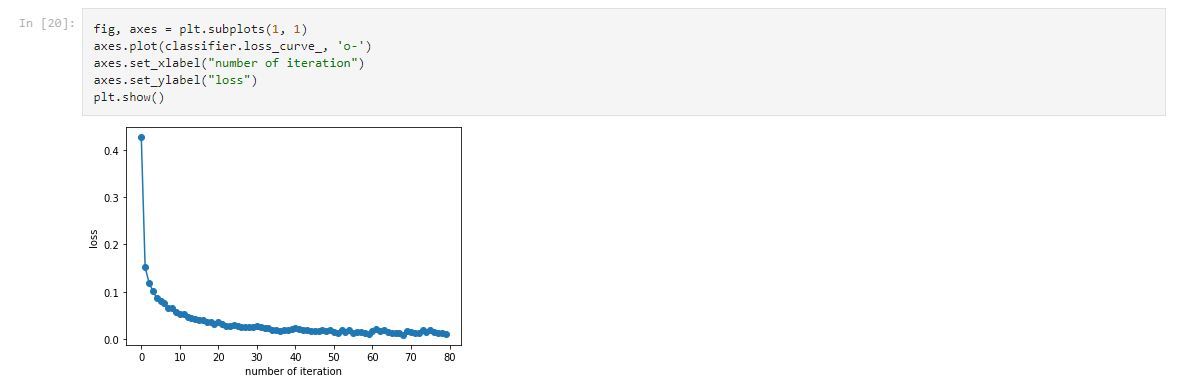
fig, axes = plt.subplots(1, 1)

axes.plot(classifier.loss\_curve\_, 'o-')

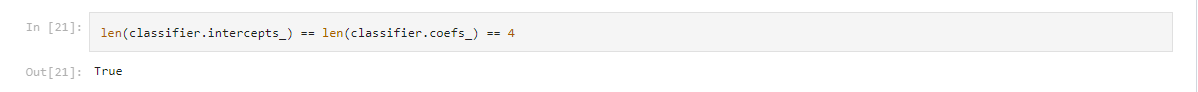
axes.set\_xlabel("number of iteration")

axes.set\_ylabel("loss")

plt.show()



len(classifier.intercepts\_) == len(classifier.coefs\_) == 4



target\_layer = 0 *#0 is input, 1 is 1st hidden etc*

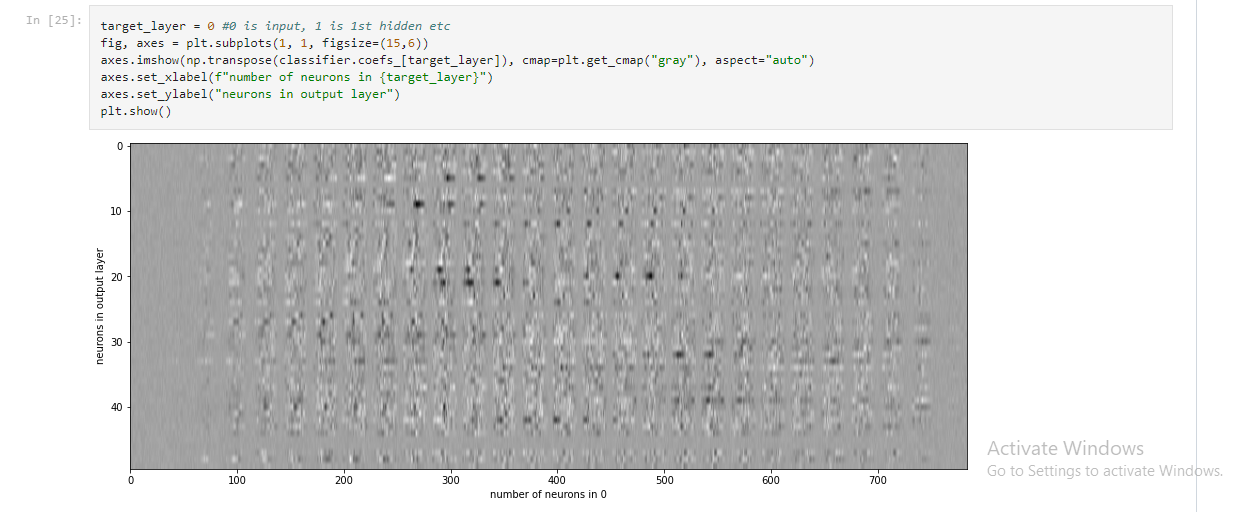
fig, axes = plt.subplots(1, 1, figsize=(15,6))

axes.imshow(np.transpose(classifier.coefs\_[target\_layer]), cmap=plt.get\_cmap("gray"), aspect="auto")

axes.set\_xlabel(f"number of neurons in {target\_layer}")

axes.set\_ylabel("neurons in output layer")

plt.show()



*# choose layer to plot*

target\_layer = 0 *#0 is input, 1 is 1st hidden etc*

fig, axes = plt.subplots(4, 4)

vmin, vmax = classifier.coefs\_[0].min(), classifier.coefs\_[target\_layer].max()

for coef, ax in zip(classifier.coefs\_[0].T, axes.ravel()):

ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray, vmin=0.5 \* vmin, vmax=0.5 \* vmax)

ax.set\_xticks(())

ax.set\_yticks(())

plt.show()



**Practical No 9**

**Aim:-** Classification of images of clothing using Tensorflow (Fashion MNIST dataset)

**Background Information:-**

**Classification:**

1. The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.
2. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
3. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.

**TensorFlow:**

1. TensorFlow is a free and open-source software library for machine learning and artificial intelligence.
2. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
3. TensorFlow can be used in a wide variety of programming languages, including Python, JavaScript, C++, and Java.
4. This flexibility lends itself to a range of applications in many different sectors.

**Fashion MNIST Dataset:**

1. Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples.
2. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms.
3. Each training and test example is assigned to one of the following labels:
   * 0 T-shirt/top
   * 1 Trouser
   * 2 Pullover
   * 3 Dress
   * 4 Coat
   * 5 Sandal
   * 6 Shirt
   * 7 Sneaker
   * 8 Bag
   * 9 Ankle boot

**Code:-**

*# TensorFlow and tf.keras*

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

*# Helper libraries*

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

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

print(tf**.**\_\_version\_\_)

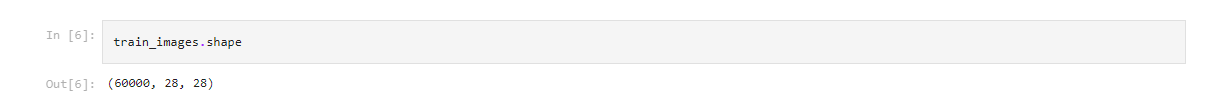
fashion\_mnist **=** tf**.**keras**.**datasets**.**fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) **=** fashion\_mnist**.**load\_data()

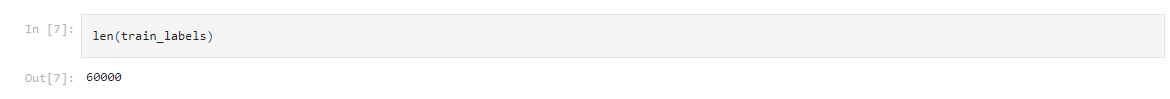
class\_names **=** ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

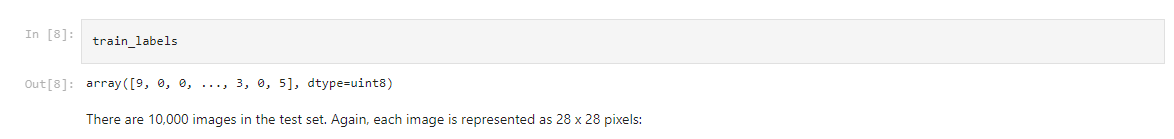
train\_images**.**shape



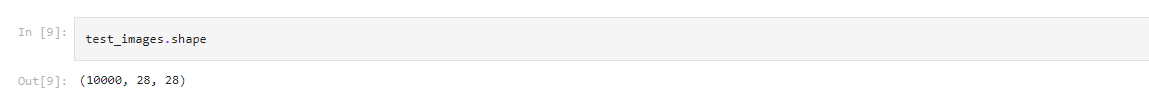
len(train\_labels)



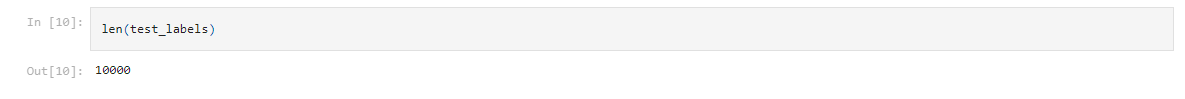
train\_labels



test\_images**.**shape



len(test\_labels)



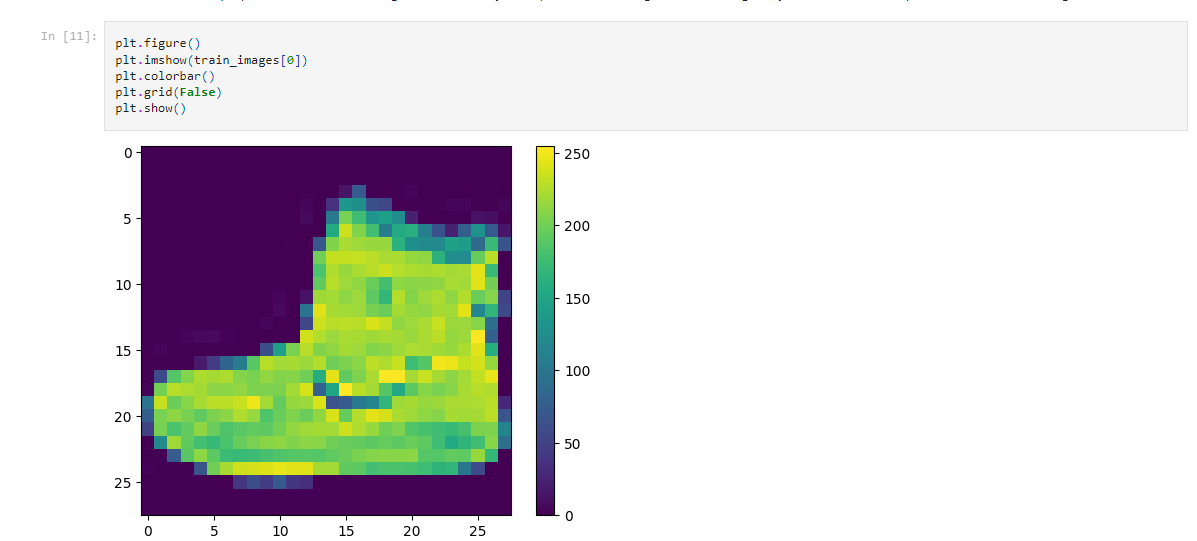
plt**.**figure()

plt**.**imshow(train\_images[0])

plt**.**colorbar()

plt**.**grid(**False**)

plt**.**show()



train\_images **=** train\_images **/** 255.0

test\_images **=** test\_images **/** 255.0

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

**for** i **in** range(25):

plt**.**subplot(5,5,i**+**1)

plt**.**xticks([])

plt**.**yticks([])

plt**.**grid(**False**)

plt**.**imshow(train\_images[i], cmap**=**plt**.**cm**.**binary)

plt**.**xlabel(class\_names[train\_labels[i]])

plt**.**show()



**Practical No 10**

**Aim:-** Implement Regression to predict fuel efficiency using Tensorflow (Auto MPG dataset)

**Background Information:-**

**Regression:**

1. Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.
2. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed.
3. It predicts continuous/real values such as temperature, age, salary, price, etc.

**TensorFlow:**

1. TensorFlow is a free and open-source software library for machine learning and artificial intelligence.
2. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
3. TensorFlow can be used in a wide variety of programming languages, including Python, JavaScript, C++, and Java.

**Auto MPG Dataset:**

1. The data is technical spec of cars. The dataset is downloaded from UCI Machine Learning Repository.
2. Number of Instances: 398
3. Number of Attributes: 9 including the class attribute
4. Attribute Information:
5. mpg: continuous
6. cylinders: multi-valued discrete
7. displacement: continuous
8. horsepower: continuous
9. weight: continuous
10. acceleration: continuous
11. model year: multi-valued discrete
12. origin: multi-valued discrete
13. car name: string (unique for each instance)
14. Missing Attribute Values: horsepower has 6 missing values

**Code:-**

# Use seaborn for pairplot.

**!**pip install -q seaborn

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

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

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

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

# Make NumPy printouts easier to read*.*

np**.**set\_printoptions(precision**=**3, suppress**=True**)

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

**from** tensorflow **import** keras

**from** tensorflow.keras **import** layers

print(tf**.**\_\_version\_\_)



url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data'

column\_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',

'Acceleration', 'Model Year', 'Origin']

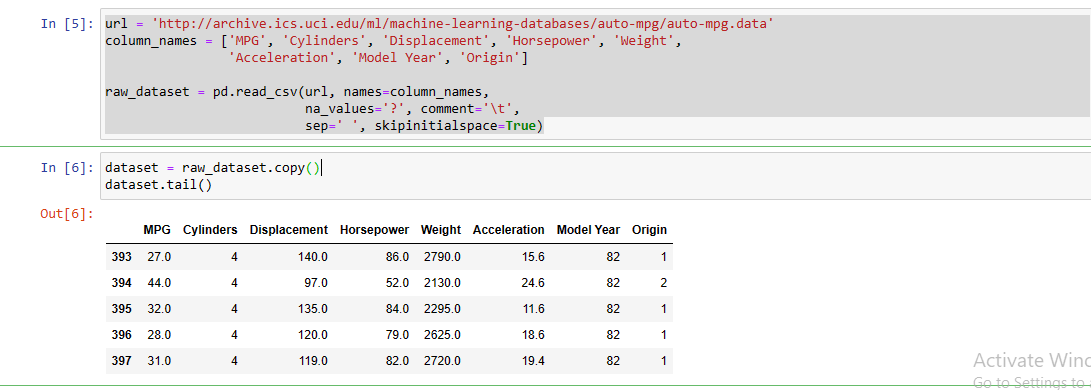
raw\_dataset = pd.read\_csv(url, names=column\_names,

na\_values='?', comment='\t',

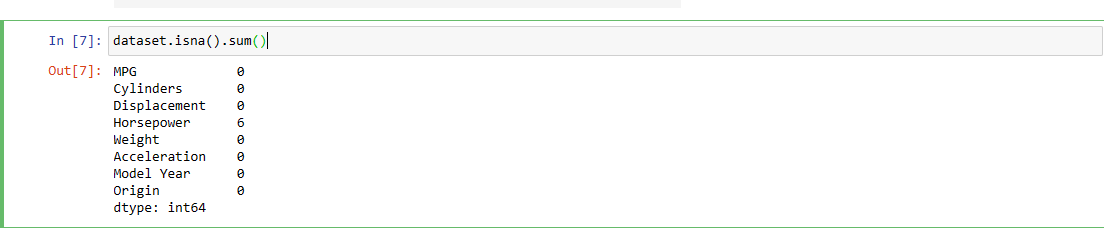
sep=' ', skipinitialspace=True)

dataset = raw\_dataset.copy()

dataset.tail()



dataset.isna().sum()

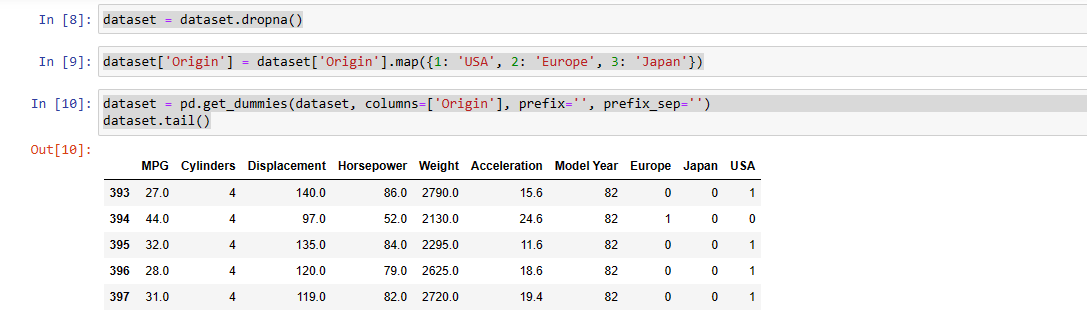


dataset = dataset.dropna()

dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})

dataset = pd.get\_dummies(dataset, columns=['Origin'], prefix='', prefix\_sep='')

dataset.tail()



train\_dataset = dataset.sample(frac=0.8, random\_state=0)

test\_dataset = dataset.drop(train\_dataset.index)

sns.pairplot(train\_dataset[['MPG', 'Cylinders', 'Displacement', 'Weight']], diag\_kind='kde')

