B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

SEMESTER: II

SUBJECT: APPLIED MACHINE AND DEEP LEARNING

NAME: SHWETA SHAM KODI

CLASS: M.SC. COMPUTER SCIENCE PART-1

ROLL NO.: 22

B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

CERTIFICATE

Miss. Shweta Sham Kodi.

Roll No. 22 *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) _______________________

B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

INDEX

Practical No 1

Aim: Implement Linear Regression (Diabetes Dataset).

Background Information:

Linear Regression:

- Linear regression is one of the easiest and most popular Machine Learning algorithms.
- It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
- Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression.
- Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Diabetes Dataset:

- 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.

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)
```

```
In [2]: import matplotlib.pyplot as plt
        import numpy as np
        from sklearn import datasets, linear_model
        from sklearn.metrics import mean_squared_error, r2_score

In [3]: # Load the diabetes dataset
        diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

In [4]: # 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:]

In [5]: # Create linear regression object
        regr = linear_model.LinearRegression()

        # Train the model using the training sets
        regr.fit(diabetes_X_train, diabetes_y_train)

Out[5]: LinearRegression()
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

```
# 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))
```

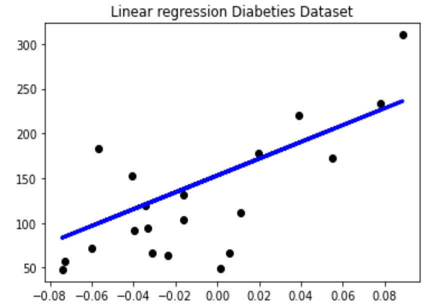
In [6]:
```
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

In [7]:
```
# 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))
```

```
Coefficients: 
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

```
#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()
```

In [9]:
```
#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:

- 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.
- 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.
- 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.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

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)
```

In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
df = pd.read_csv("Iris.csv")
df.head(5)
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa


```
df = df.drop(columns = ['Id'])
df.head(5)
```

In [6]:
```
df = df.drop(columns = ['Id'])
df.head(5)
```

Out[6]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

df.info()

```
In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

df['Species'].value_counts()

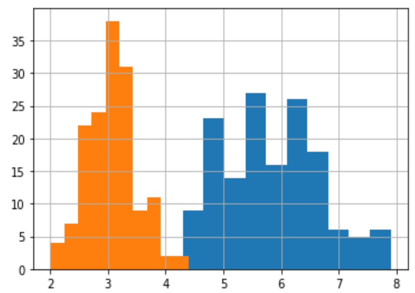
```
In [8]: df['Species'].value_counts()

Out[8]: Iris-setosa        50
        Iris-versicolor    50
        Iris-virginica     50
        Name: Species, dtype: int64
```

df.isnull().sum()

```
In [9]: df.isnull().sum()

Out[9]: SepalLengthCm    0
        SepalWidthCm     0
        PetalLengthCm    0
        PetalWidthCm     0
        Species          0
        dtype: int64
```

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

```
In [10]: df['SepalLengthCm'].hist()
         df['SepalWidthCm'].hist()

Out[10]: <AxesSubplot:>
```

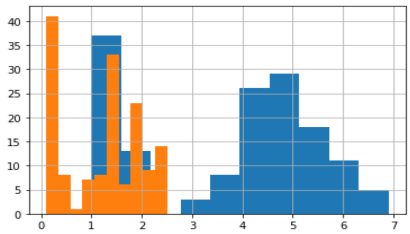


df['PetalLengthCm'] . hist()
df['PetalWidthCm'] . hist()
df . corr()

```
In [14]: df['PetalLengthCm'].hist()
         df['PetalWidthCm'].hist()

         df.corr()
```

Out[14]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954	0.782561
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544	-0.419446
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757	0.949043
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000	0.956464
Species	0.782561	-0.419446	0.949043	0.956464	1.000000



from sklearn.preprocessing **import** LabelEncoder
le **=** LabelEncoder()
df['Species'] **=** le . fit_transform(df['Species'])
df . head(100)

```
In [13]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         df['Species'] = le.fit_transform(df['Species'])
         df.head(100)
```

Out[13]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1
98	5.1	2.5	3.0	1.1	1
99	5.7	2.8	4.1	1.3	1

100 rows × 5 columns

```
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)
```

```
In [16]: 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)

In [17]: from sklearn.linear_model import LogisticRegression
         model = LogisticRegression()

In [19]: model.fit(X_train, Y_train)
         print("Accuracy: ", model.score(X_test, Y_test) * 100)

         Accuracy:  100.0
```

Practical No 3

Aim: Implements Multinomial Logistic Regression (Iris Dataset).

Background Information:

Linear Regression:

- 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.
- 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.
- 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.
- 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'])
```

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.cm as cm
        import random
        import seaborn

In [3]: 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()

In [4]: print(iris_data['DESCR'])

.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:
```

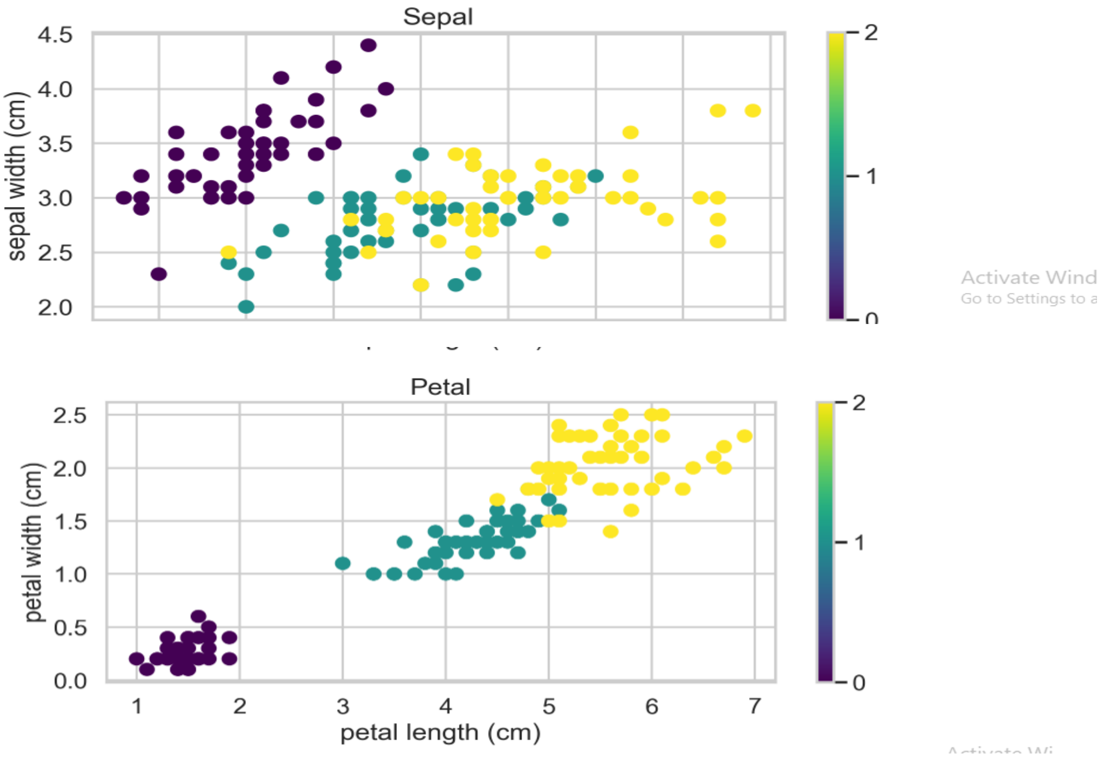
Activate Windows
Go to Settings to activate Windo

```
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')

```
In [5]: 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)

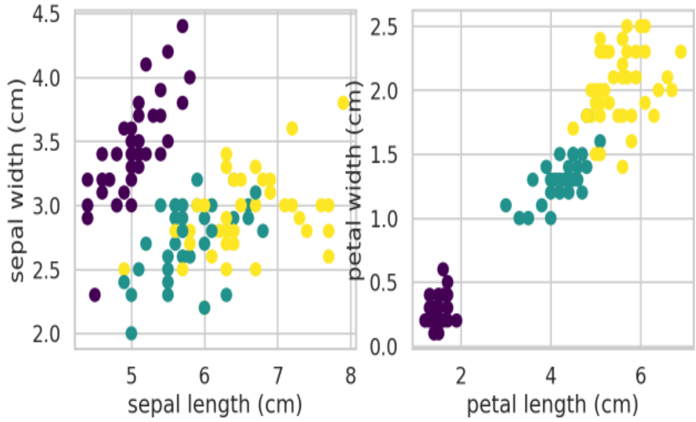
def separate_data():
    """
    A = iris_dataset[0:40]
    tA = iris_dataset[40:50]
    B = iris_dataset[50:90]
    tB = iris_dataset[90:100]
    C = iris_dataset[100:140]

```
    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])
```

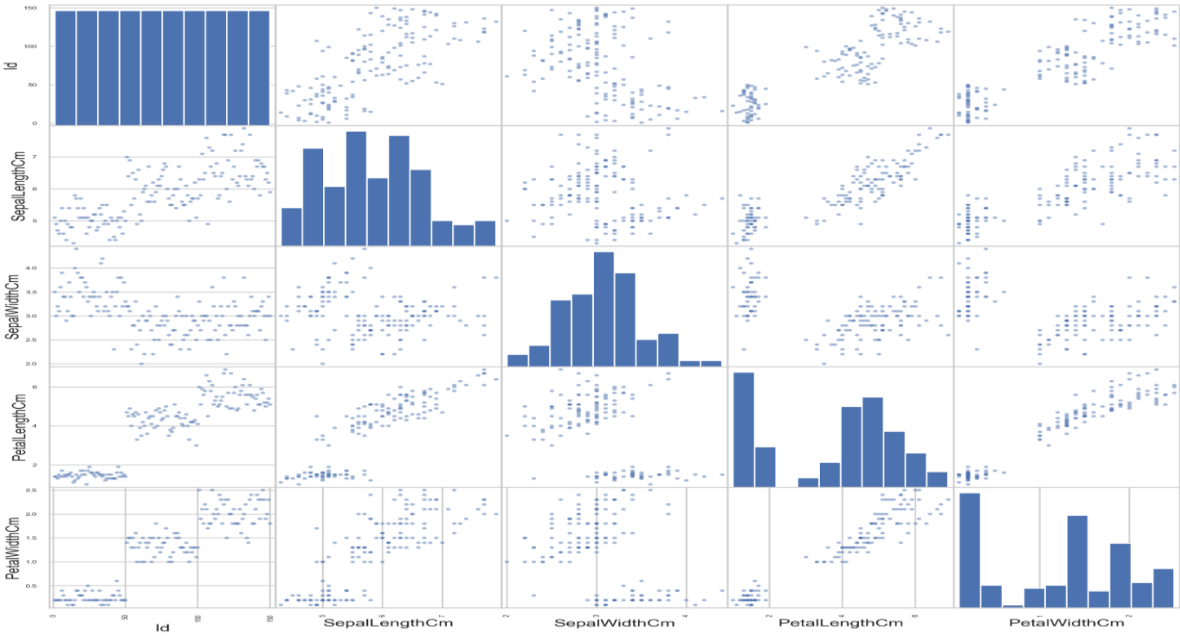
In [7]:
```
random.seed(123)

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])
```

In [8]:
```
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])
```

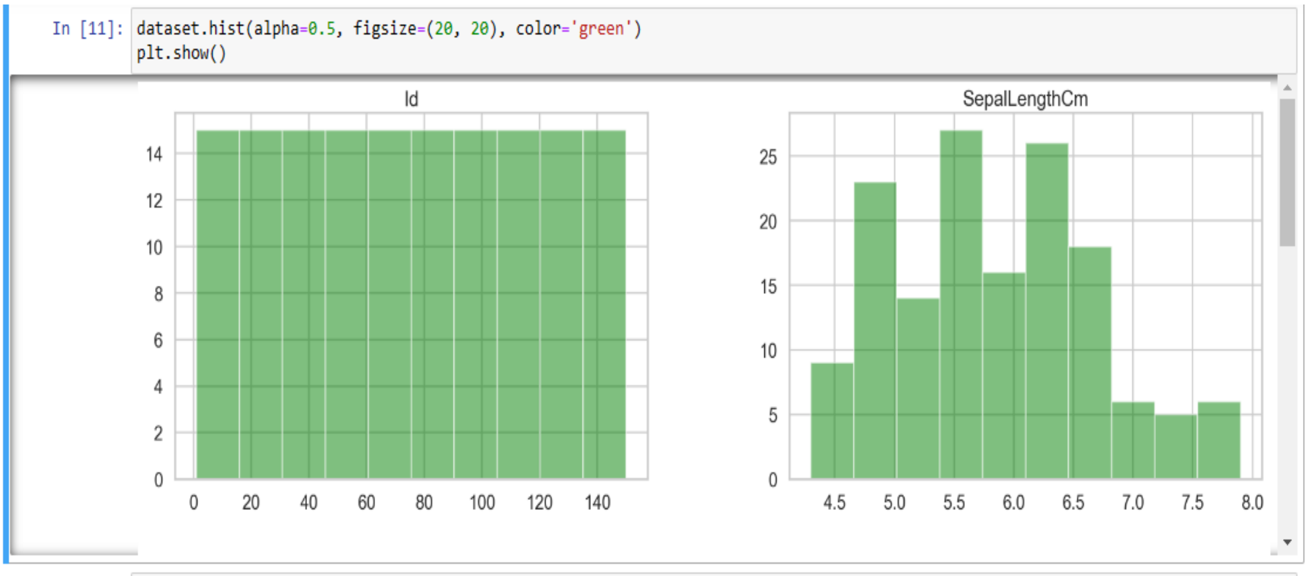
Out[8]: Text(0, 0.5, 'petal width (cm)')




```
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()



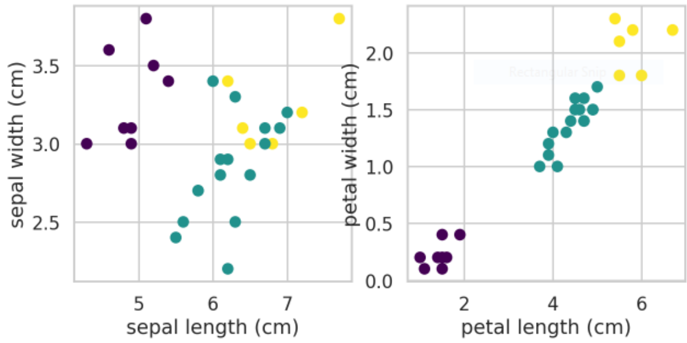
plt.figure(figsize=(10,10));

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])

```
In [11]: 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])

Out[11]: Text(0, 0.5, 'petal width (cm)')
```



B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

Practical No 4

Aim: Implement SVM classifier (Iris Dataset).

Background Information:

SVM Classifier:

- 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.
- 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.
- 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()

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: #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()
```

Out[2]:

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

#Encoding the categorical column
dataset = dataset.replace({"class": {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})
#Visualize the new dataset
dataset.head()

In [3]:
```
#Encoding the categorical column
dataset = dataset.replace({"class":  {"Iris-setosa":1,"Iris-versicolor":2, "Iris-virginica":3}})
#Visualize the new dataset
dataset.head()
```

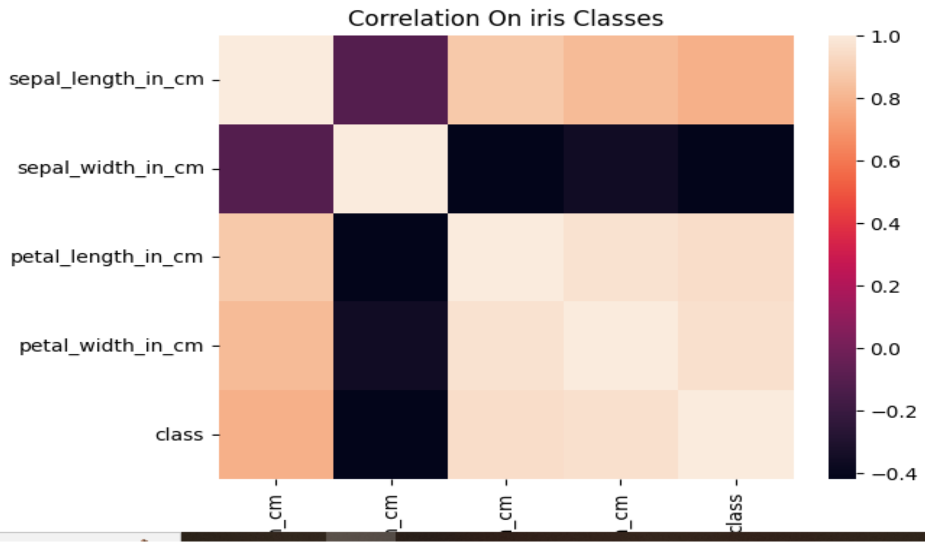
Out[3]:

	sepal_length_in_cm	sepal_width_in_cm	petal_length_in_cm	petal_width_in_cm	class
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation On iris Classes')

In [4]:
```
plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation On iris Classes')
```

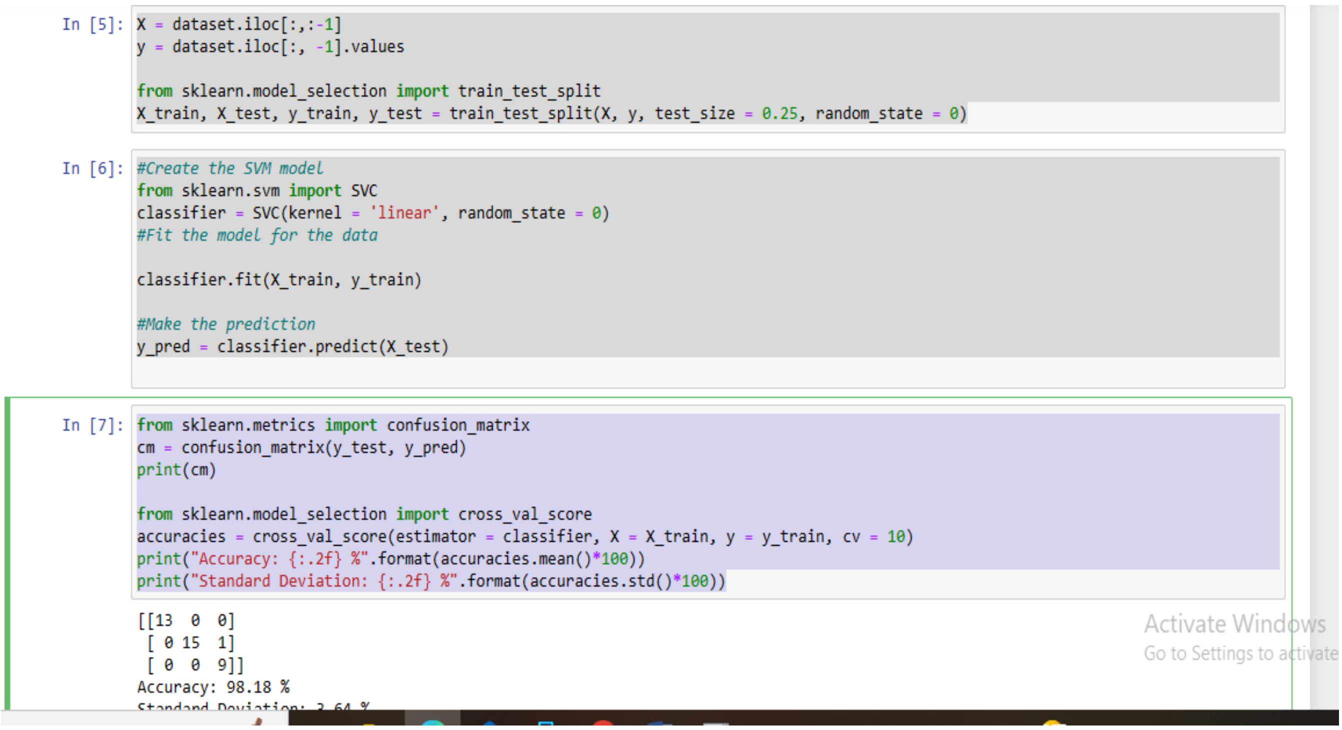
Out[4]: Text(0.5, 1.0, 'Correlation On iris Classes')



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))
```



```
In [5]: 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)

In [6]: #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)

In [7]: 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))

[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy: 98.18 %
Standard Deviation: 3.64 %
```

Practical No 5

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

Background Information:

Decision Tree:

- 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. It is one way to display an algorithm that only contains conditional control statements.
- 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.
- 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.
- The decisions or the test are performed based on features of the given dataset.A decision tree simply asks a question and based on the answer (Yes/No), it further splits the tree into subtrees.
- 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:

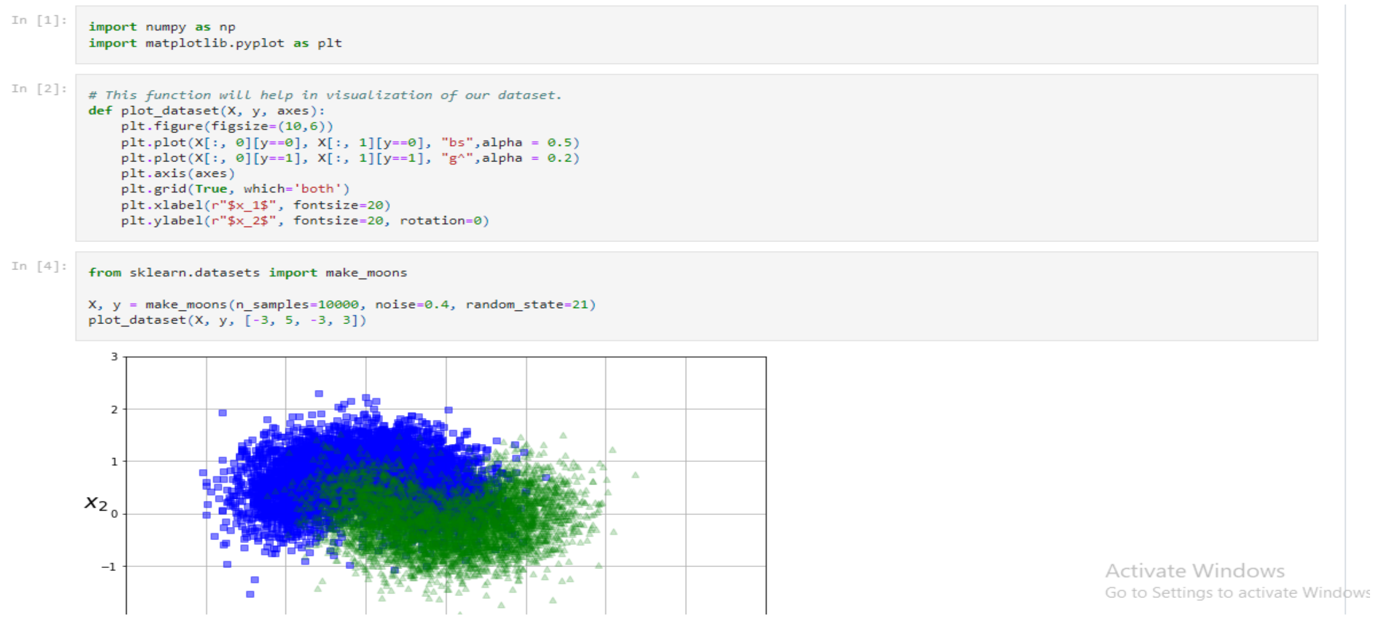
1. Make two interleaving half circles.
2. A simple toy dataset to visualize clustering and classification algorithms.
3. It's taken from Sklearn.

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)
```

```
In [6]: from sklearn.model_selection import GridSearchCV

        parameter = {
                    'criterion' : ["gini", "entropy"],
                    'max_leaf_nodes': list(range(2, 50)),
                    'min_samples_split': [2, 3, 4]
                   }

In [7]: clf = GridSearchCV(tree_clf, parameter, cv = 5,scoring = "accuracy",return_train_score=True,n_jobs=-1)

        clf.fit(X_train, y_train)

Out[7]: GridSearchCV
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                            13, 14, 15, 16, 17, 18, 19, 20, 21,
                                            22, 23, 24, 25, 26, 27, 28, 29, 30,
                                            31, ...],
                         'min_samples_split': [2, 3, 4]},
             return_train_score=True, scoring='accuracy')
estimator: DecisionTreeClassifier
DecisionTreeClassifier()
DecisionTreeClassifier
DecisionTreeClassifier()
```



```
clf.best_params_
{'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 2}
```

```
In [8]: clf.best_params_
        {'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 2}

Out[8]: {'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)
```

```
In [9]: cvres = clf.cv_results_
        for mean_score, params in zip(cvres["mean_train_score"], cvres["params"]):
            print(mean_score, params)

0.7801250000000001 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 2}
0.7801250000000001 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 3}
0.7801250000000001 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 4}
0.822875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 2}
0.822875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 3}
0.822875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 4}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 2}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 3}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 4}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 2}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 3}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 4}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 2}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 3}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 4}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 2}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 3}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 4}
0.8608125000000001 {'criterion': 'gini', 'max_leaf_nodes': 8, 'min_samples_split': 2}
```


clf.score(X_train, y_train)

```
In [10]: clf.score(X_train, y_train)
Out[10]: 0.865375

In [11]: from sklearn.metrics import confusion_matrix
```

from sklearn.metrics import confusion_matrix

pred = clf.predict(X_train)
confusion_matrix(y_train,pred)

```
In [11]: from sklearn.metrics import confusion_matrix

         pred = clf.predict(X_train)
         confusion_matrix(y_train,pred)
Out[11]: array([[3579,  453],
                [ 624, 3344]], dtype=int64)
```

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}")

```
In [12]: 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}")

         Precision: 0.8806952857519094  Recall:0.842741935483871
```

from sklearn.metrics import f1_score

f1_score(y_train, pred)
clf.score(X_test, y_test)

```
In [13]: from sklearn.metrics import f1_score

         f1_score(y_train, pred)
         clf.score(X_test, y_test)

Out[13]: 0.8465
```

Practical No 6

Aim: Train an SVM regressor on the California Housing Dataset.

Background Information:

SVM Regressor:

- Support Vector Regression as the name suggests is a regression algorithm that supports both linear and non-linear regressions.
- This method works on the principle of the Support Vector Machine.
- 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.
- 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)

```
In [1]: from sklearn.datasets import fetch_california_housing

        california_housing = fetch_california_housing(as_frame=True)

In [2]: print(california_housing.DESCR)

.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
```


california_housing . frame . head()

```
In [3]: california_housing.frame.head()
Out[3]:
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

california_housing . data . head()

```
In [4]: california_housing.data.head()
```

Out[4]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

california_housing . target . head()

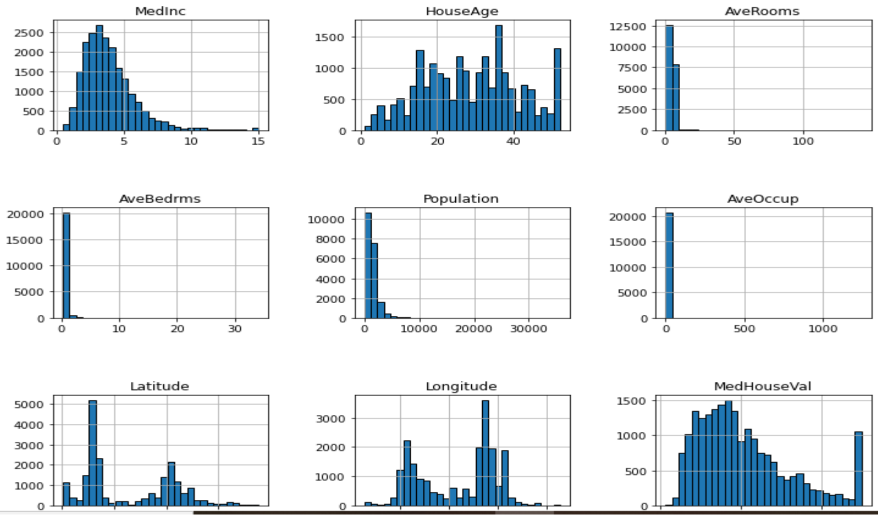
```
In [5]: california_housing.target.head()

Out[5]: 0    4.526
        1    3.585
        2    3.521
        3    3.413
        4    3.422
        Name: MedHouseVal, dtype: float64
```

california_housing . frame . info()

```
In [6]: california_housing.frame.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
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)
```

In [7]:
```
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()

In [8]:
```
features_of_interest = ["AveRooms", "AveBedrms", "AveOccup", "Population"]
california_housing.frame[features_of_interest].describe()
```

Out[8]:

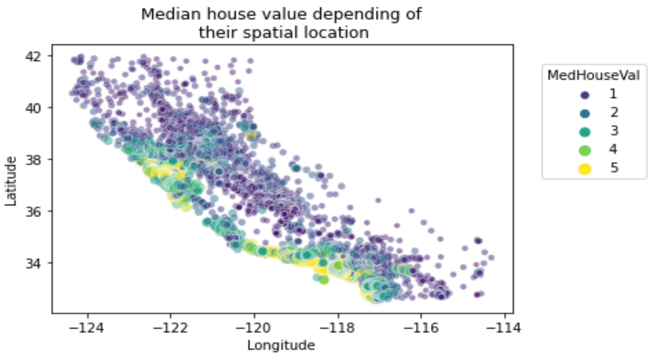
	AveRooms	AveBedrms	AveOccup	Population
count	20640.000000	20640.000000	20640.000000	20640.000000
mean	5.429000	1.096675	3.070655	1425.476744
std	2.474173	0.473911	10.386050	1132.462122
min	0.846154	0.333333	0.692308	3.000000
25%	4.440716	1.006079	2.429741	787.000000
50%	5.229129	1.048780	2.818116	1166.000000
75%	6.052381	1.099526	3.282261	1725.000000
max	141.909091	34.066667	1243.333333	35682.000000

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")

```
In [9]: 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")
```

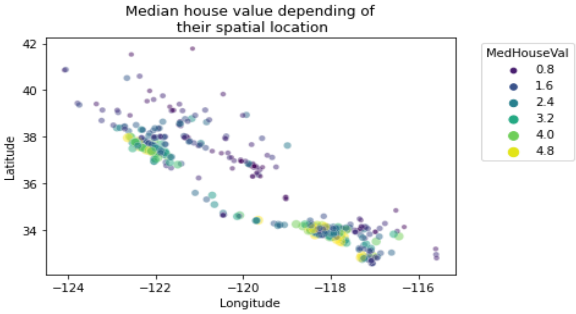


import numpy **as** np

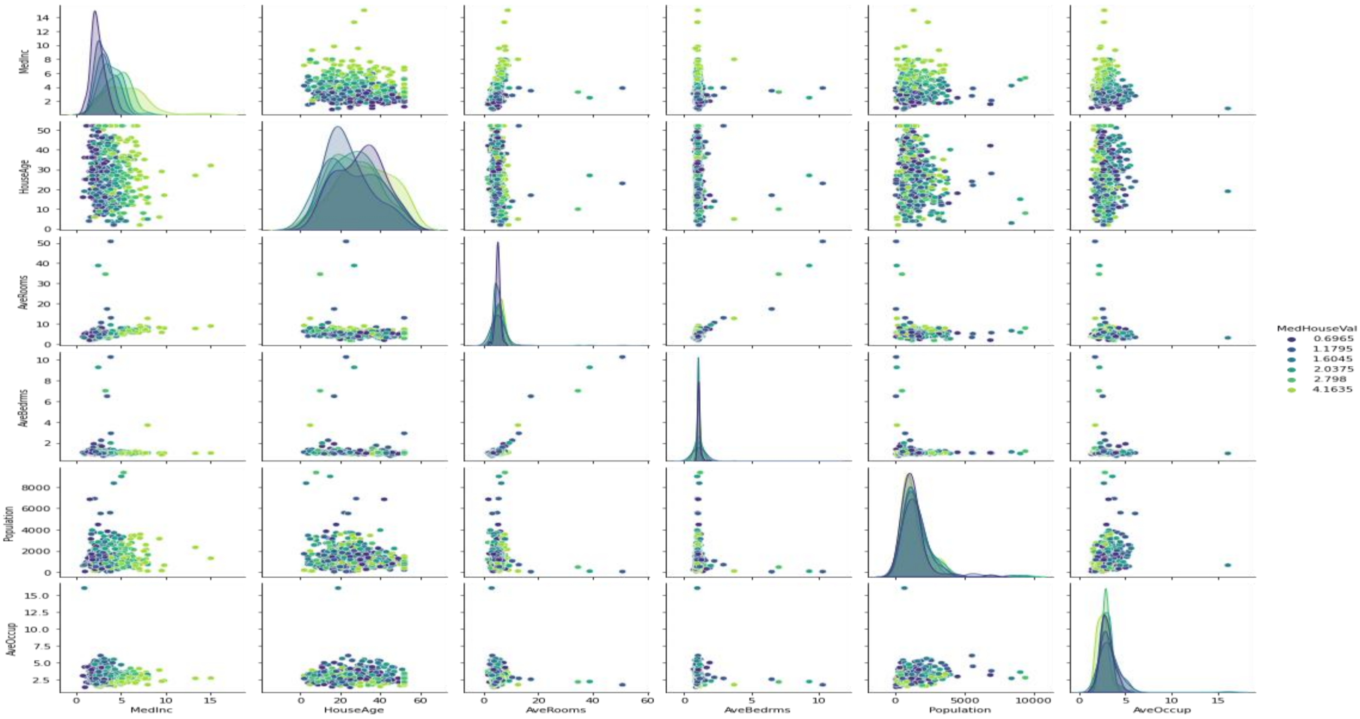
rng = np . random . RandomState(0)
indices = rng . choice(np . arange(california_housing . frame . shape[0]), size=500,
                  replace=**False**)

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")

In [10]:
```
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}")

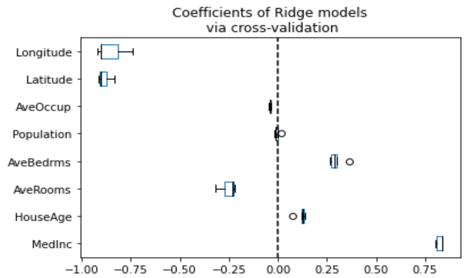
```
In [15]:
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import RidgeCV
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_validate

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}")

R2 score: 0.553 ± 0.062
```

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")
```

In [16]:
```
import pandas as pd

coefs = pd.DataFrame(
    [est[-1].coef_ for est in cv_results["estimator"]],
    columns=california_housing.feature_names
)
```

In [17]:
```
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")
```



In [ ]:

B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

Practical No 7

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

Background Information:

Batch Gradient Descent:

- 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.
- 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.
- Computes gradient using the whole Training sample.
- Slow and computationally expensive algorithm.
- Not suggested for huge training samples.
- Deterministic in nature.
- Gives optimal solution given sufficient time to converge.
- No random shuffling of points is required.
- Can't escape shallow local minima easily.
- Convergence is slow.

SoftMax Regression:

- 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.
- 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).

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]
```

```
In [2]:
iris=load_iris()
X=iris['data']
y=iris['target']
```

```
In [3]:
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:]]
```

```
In [4]:
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
```

```
In [5]:
y_valid[:10]

Out[5]: array([1, 0, 1, 2, 1, 1, 1, 0, 0, 0])
```


one_hot (y_valid [: 10])

```
In [6]:
one_hot(y_valid[:10])

Out[6]: array([[0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

```
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)
```

In [7]:
```
y_train_prob = one_hot(y_train)
y_valid_prob = one_hot(y_valid)
y_test_prob = one_hot(y_test)
```

In [8]:
```
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)
```

```
5 3
```

Practical No 8

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

Background Information:

MLPClassifier:

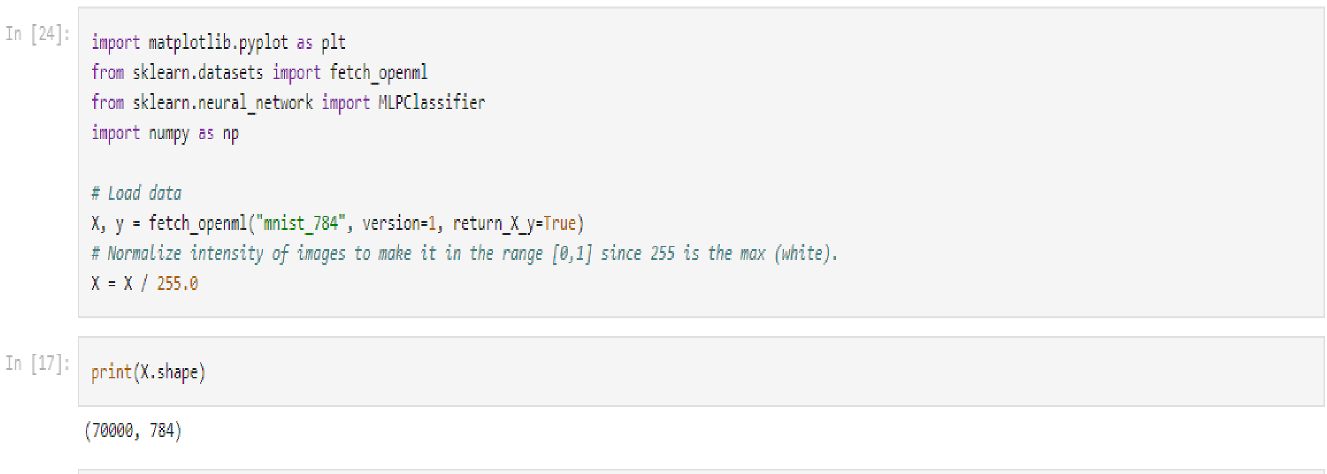
- MLPClassifier stands for Multi-layer Perceptron classifier which in the name itself connects to a Neural Network.
- Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLPClassifier relies on an underlying Neural Network to perform the task of classification.
- 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:

- 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.
- The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets.
- 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.
- 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.
- The MNIST database contains 60,000 training images and 10,000 testing images.
- 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.

<u>Code</u>:

```
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)
```



```
In [24]: 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

In [17]: print(X.shape)

         (70000, 784)
```

```
# 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)
```

```
In [18]: # 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)

         Iteration 1, loss = 0.42635367
         Iteration 2, loss = 0.15133481
         Iteration 3, loss = 0.11926082
         Iteration 4, loss = 0.10128421
         Iteration 5, loss = 0.08698448
         Iteration 6, loss = 0.08018627
         Iteration 7, loss = 0.07544472
         Iteration 8, loss = 0.06650726
         Iteration 9, loss = 0.06502276
         Iteration 10, loss = 0.05670472
         Iteration 11, loss = 0.05228727
         Iteration 12, loss = 0.05194876
         Iteration 13, loss = 0.04580530
         Iteration 14, loss = 0.04507070
         Iteration 15, loss = 0.04141424
         Iteration 16, loss = 0.03988480
         Iteration 17, loss = 0.03980626
         Iteration 18, loss = 0.03593785
         Iteration 19, loss = 0.03619045
         Iteration 20, loss = 0.03170852
         Iteration 21, loss = 0.03625169
         Iteration 22, loss = 0.03089518
         Iteration 23, loss = 0.02846908
```

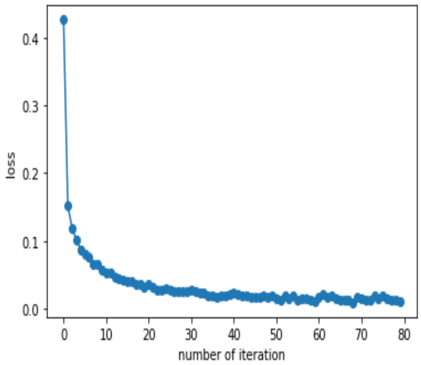


```
print("Training set score: %f" % classifier.score(X_train, y_train))
print("Test set score: %f" % classifier.score(X_test, y_test))
```

```
         Iteration 75, loss = 0.01495040
         Iteration 76, loss = 0.01856423
         Iteration 77, loss = 0.01455814
         Iteration 78, loss = 0.01311380
         Iteration 79, loss = 0.01177643
         Iteration 80, loss = 0.01160100
         Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

Out[18]: MLPClassifier(hidden_layer_sizes=(50, 20, 10), learning_rate_init=0.1,
                       max_iter=100, random_state=1, solver='sgd', verbose=10)

In [19]: print("Training set score: %f" % classifier.score(X_train, y_train))
         print("Test set score: %f" % classifier.score(X_test, y_test))

         Training set score: 0.997467
         Test set score: 0.970700
```

```
fig, axes = plt.subplots(1, 1)
axes.plot(classifier.loss_curve_, 'o-')
axes.set_xlabel("number of iteration")
axes.set_ylabel("loss")
plt.show()
```

```
In [20]: fig, axes = plt.subplots(1, 1)
         axes.plot(classifier.loss_curve_, 'o-')
         axes.set_xlabel("number of iteration")
         axes.set_ylabel("loss")
         plt.show()
```

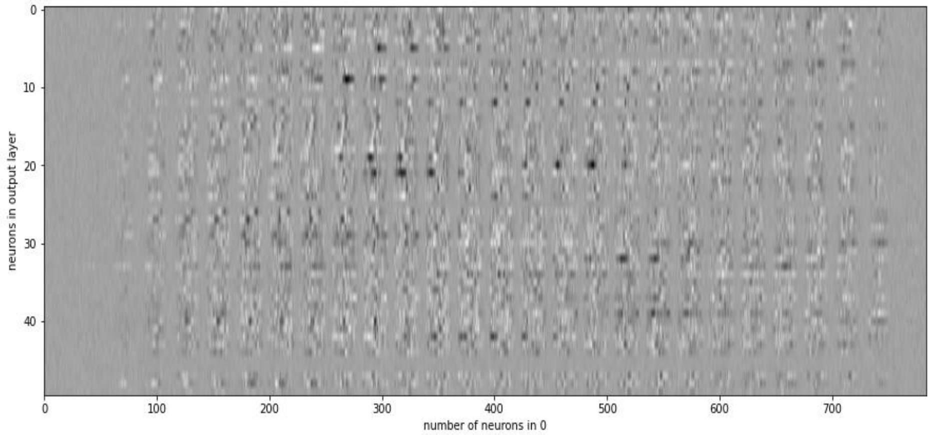



```
len(classifier.intercepts_) == len(classifier.coefs_) == 4
```

```
In [21]: len(classifier.intercepts_) == len(classifier.coefs_) == 4

Out[21]: True
```

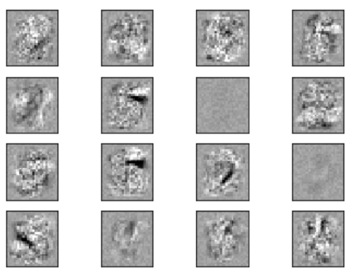
```
target_layer = 0 #0 is input, 1 is 1st hidden etc
fig, axes = plt.subplots(1, 1, figsize=(15,6))
axes.imshow(np.transpose(classifier.coefs_[target_layer]), cmap=plt.get_cmap("gray"),
aspect="auto")
axes.set_xlabel(f"number of neurons in {target_layer}")
axes.set_ylabel("neurons in output layer")
plt.show()
```

In [25]:
```
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()
```

In [26]:
```
# 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:

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.
- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
- 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:

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

```
In [6]: train_images.shape

Out[6]: (60000, 28, 28)
```

```
len(train_labels)
```

```
In [7]: len(train_labels)

Out[7]: 60000
```

train_labels

```
In [8]: train_labels
```

```
Out[8]: array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

There are 10,000 images in the test set. Again, each image is represented as 28 x 28 pixels:

test_images.shape

```
In [9]: test_images.shape
```

```
Out[9]: (10000, 28, 28)
```

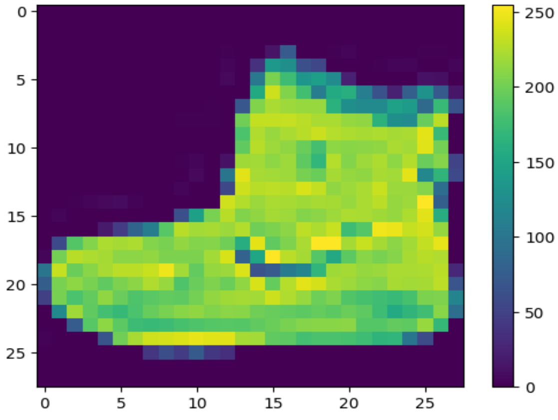
len(test_labels)

```
In [10]: len(test_labels)
```

```
Out[10]: 10000
```

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(**False**)
plt.show()

```
In [11]: 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:

- Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables.
- 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.
- It predicts continuous/real values such as temperature, age, salary, price, etc.

TensorFlow:

- TensorFlow is a free and open-source software library for machine learning and artificial intelligence.
- It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- 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:
   a. mpg: continuous
   b. cylinders: multi-valued discrete
   c. displacement: continuous
   d. horsepower: continuous
   e. weight: continuous

f. acceleration: continuous
g. model year: multi-valued discrete
h. origin: multi-valued discrete
i. car name: string (unique for each instance)

5. 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__)
```

```
In [1]: # Use seaborn for pairplot.
        !pip install -q seaborn

In [2]: 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)

In [4]: import tensorflow as tf

        from tensorflow import keras
        from tensorflow.keras import layers

        print(tf.__version__)

        2.12.0
```

```
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()
```

```
In [5]: 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)
```

```
In [6]: dataset = raw_dataset.copy()
        dataset.tail()
```

Out[6]:

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4	135.0	84.0	2295.0	11.6	82	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1


```
dataset.isna().sum()
```

```
In [7]: dataset.isna().sum()

Out[7]: MPG             0
        Cylinders       0
        Displacement    0
        Horsepower      6
        Weight          0
        Acceleration    0
        Model Year      0
        Origin          0
        dtype: int64
```

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()

```
In [8]: dataset = dataset.dropna()

In [9]: dataset['Origin'] = dataset['Origin'].map({1: 'USA', 2: 'Europe', 3: 'Japan'})

In [10]: dataset = pd.get_dummies(dataset, columns=['Origin'], prefix='', prefix_sep='')
         dataset.tail()

Out[10]:
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Europe	Japan	USA
393	27.0	4	140.0	86.0	2790.0	15.6	82	0	0	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	1	0	0
395	32.0	4	135.0	84.0	2295.0	11.6	82	0	0	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	0	0	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	0	0	1

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')