|  |  |  |
| --- | --- | --- |
| **Sr No** | **Topics** | **Sign** |
| **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 Datase |  |
| **6** | Implement Batch Gradient Descent with early stopping for Softmax Regression |  |
| **7** | Implement MLP for classification of handwritten digits (MNIST Dataset) |  |
| **8** | Classification of images of clothing using Tensorflow (Fashion MNIST dataset) |  |
| **9** | Implement Regression to predict fuel efficiency using Tensorflow (Auto MPG dataset) |  |

**INDEX**

**Practical No.: 1 Date: / /** **Aim: Linear Regression.**

**Source code:**

# Import necessary libraries

import pandas as pd  # for data manipulation

import numpy as np   # for scientific computing

import seaborn as sns # for visualization

import matplotlib.pyplot as plt # for plotting & visualization

from sklearn import datasets

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

from sklearn.linear\_model import LinearRegression

from sklearn import metrics as mt

# Load the diabetes dataset

diabetes = datasets.load\_diabetes()

# Print dataset description

print(diabetes.DESCR)

# Display feature names and dataset shapes

print(diabetes.feature\_names)

print(diabetes.data.shape)

print(diabetes.target.shape)

# Create a DataFrame from the dataset

db\_df = pd.DataFrame(diabetes.data, columns=diabetes.feature\_names)

print(db\_df.sample(5))

# Add target column 'Progression' to the DataFrame

db\_df['Progression'] = diabetes.target

# Check for missing values

print(db\_df.isna().sum())

# Display summary statistics

print(db\_df.describe())

# Display information about the DataFrame

print(db\_df.info())

# Calculate correlation matrix

corr = db\_df.corr()

# Plot the heatmap of the correlation matrix

plt.subplots(figsize=(8, 8))

sns.heatmap(corr, cmap='RdYlGn', annot=True)

plt.show()

# Define independent (features) and dependent (target) variables

x = db\_df.drop(labels='Progression', axis=1)  # Features

y = db\_df['Progression']  # Target variable

# Split the dataset into train and test sets (75% train, 25% test)

train\_x, test\_x, train\_y, test\_y = train\_test\_split(x, y, test\_size=0.25, random\_state=999)

# Display shapes of train and test sets

print(train\_x.shape)

print(test\_x.shape)

print(train\_y.shape)

print(test\_y.shape)

# Create an instance of the Linear Regression model

lm = LinearRegression()

print(lm)

print(type(lm))

# Fit the model on the training data

lm.fit(train\_x, train\_y)

# Predict on the test set

predicted\_y = lm.predict(test\_x)

# Evaluate the model

print("1) The model explains", np.round(mt.explained\_variance\_score(test\_y, predicted\_y) \* 100, 2), "% variance of the target w.r.t features")

print("2) The Mean Absolute Error of the model is:", np.round(mt.mean\_absolute\_error(test\_y, predicted\_y), 2))

print("3) The R-Square score of the model is", np.round(mt.r2\_score(test\_y, predicted\_y), 2))

# Display coefficients and intercept of the model

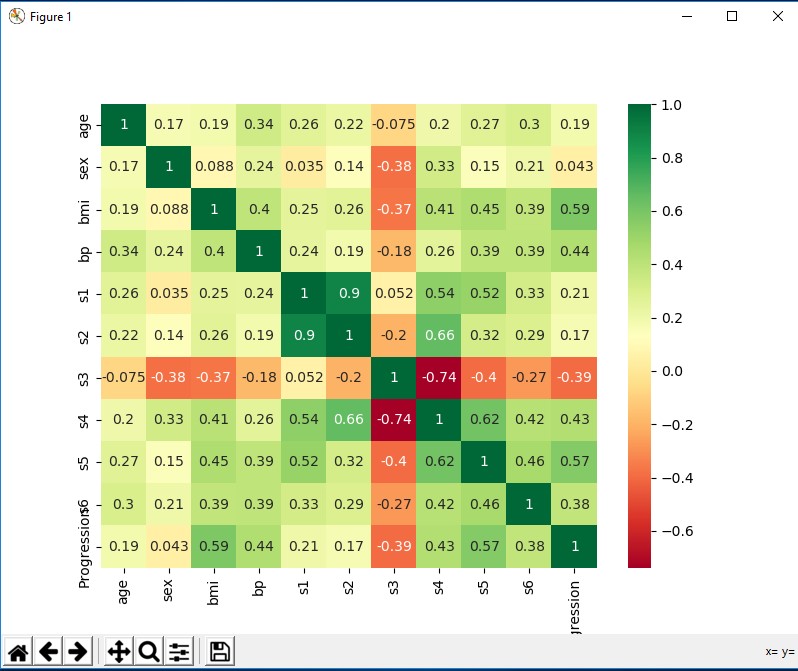
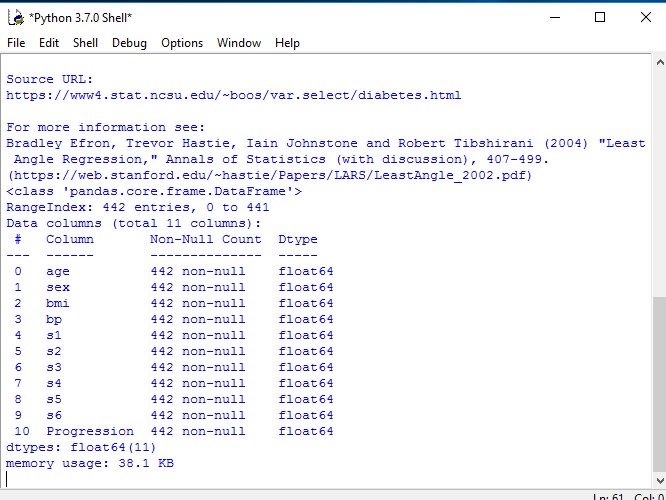
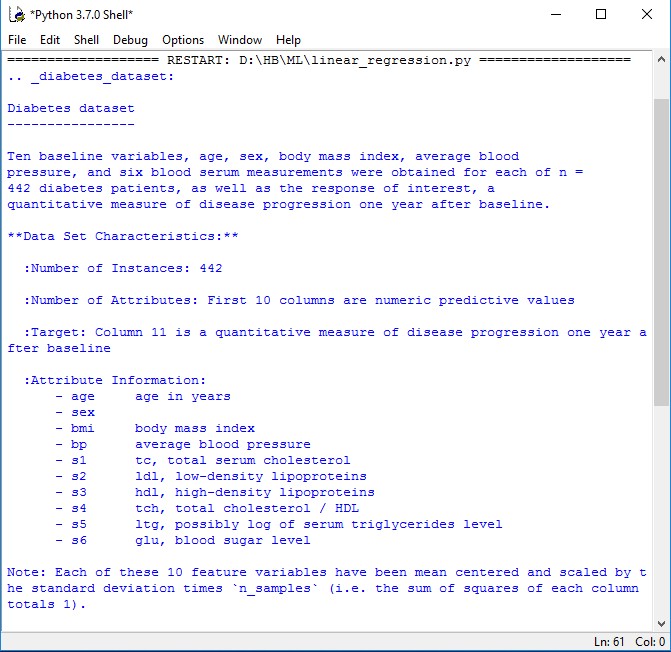
coeff = pd.Series(lm.coef\_, index=train\_x.columns)

intercept = lm.intercept\_

print("Coefficients:\n", coeff)

print("\nIntercept:\n", intercept)

**Output:**



**Practical No.: 2 Date: / /**

**Aim: Logistic Regression.**

**Source Code:**

# Import necessary libraries

import matplotlib.pyplot as plt

import numpy as np

import torch

import torch.nn.functional as F

from sklearn import datasets

from sklearn import preprocessing

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

from collections import Counter

# Load the iris dataset

iris = datasets.load\_iris()

# Select only 2 classes and 2 features (sepal length/width)

X = iris.data[:-50, :2]

y = iris.target[:-50]

# Scale the features

X = preprocessing.scale(X)

# Print data details

print("Class distribution:", Counter(y)) # 50 of each iris flower

print("Flower types:", list(iris.target\_names[:-1])) # Type of flower

print("Feature shape:", X.shape)

# Scatter plot of the two classes

plt.scatter(X[:, 0], X[:, 1], c=y)

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.title('Scatter Plot of Iris Classes')

plt.show()

# Train/Test split

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

# Print sizes of training and test sets

print(f'X train size: {X\_train.shape}')

print(f'X test size: {X\_test.shape}')

print(f'y train size: {y\_train.shape}')

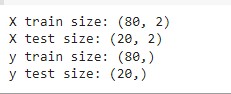
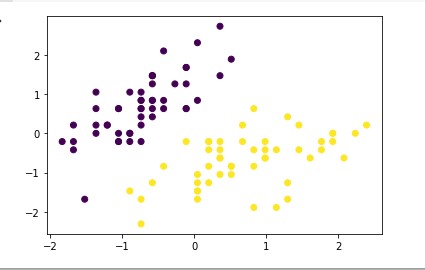
print(f'y test size: {y\_test.shape}')

# Distribution of both classes are roughly equal

print("Training set class distribution:", Counter(y\_train))

# print(f'Validation Accuracy: {acc.item() :.4f}')

**Output:**



**Practical No.: 3 Date: / /**

**Aim: Implement Multinomial Logistic Regression(Iris Dataset).**

**Source Code:**

# Import necessary libraries

from sklearn.linear\_model import LogisticRegression

from sklearn import datasets

from sklearn.preprocessing import StandardScaler

# Load the iris dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Standardize the features

scaler = StandardScaler()

X\_std = scaler.fit\_transform(X)

# Instantiate and fit the Logistic Regression model

clf = LogisticRegression(random\_state=0, multi\_class='multinomial', solver='newton-cg')

model = clf.fit(X\_std, y)

# Define a new observation and predict its class

new\_observation = [[0.5, 0.5, 0.5, 0.5]]

predicted\_class = model.predict(new\_observation)

predicted\_probabilities = model.predict\_proba(new\_observation)

# Output the predictions

print("Predicted class:", predicted\_class)

print("Predicted probabilities:", predicted\_probabilities)

**Output:** array([[0.01982536, 0.74491994, 0.2352547 ]])

**Practical No.: 4 Date:**

**Aim: Implement Support Vector Classifier(Iris Dataset).**

**Source code:**

from sklearn import datasets # To Get iris dataset from sklearn import svm # To fit the svm classifier import numpy as np import matplotlib.pyplot as plt # To visuvalizing the data

# import iris data to model Svm classifier iris\_dataset = datasets.load\_iris() def visuvalize\_sepal\_data(): iris = datasets.load\_iris()

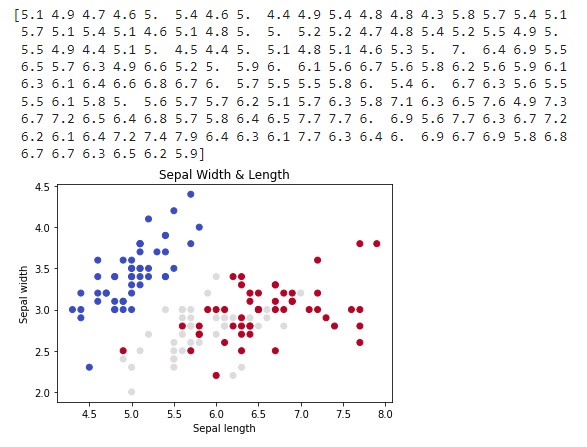
X = iris.data[:, :2] # we only take the first two features.

y = iris.target print(X[:,0])

#cmap- color map,it is a colour representation of 3D values plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm) plt.xlabel('Sepal length') plt.ylabel('Sepal width') plt.title('Sepal Width & Length') plt.show() visuvalize\_sepal\_data()

output

-



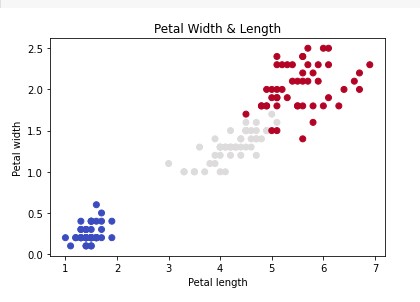
def visuvalize\_petal\_data(): iris = datasets.load\_iris()

X = iris.data[:, 2:] # we only take the last two features. y = iris.target plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm) plt.xlabel('Petal length')

plt.ylabel('Petal width') plt.title('Petal Width & Length') plt.show() visuvalize\_petal\_data()

output

-



# Import necessary libraries

from sklearn import datasets, svm

import numpy as np

import matplotlib.pyplot as plt

# Load the iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2]  # Use only the first two features (Sepal length and width)

y = iris.target

C = 1.0  # SVM regularization parameter

# Create SVM classifiers with different kernels

# SVC with linear kernel

svc = svm.SVC(kernel='linear', C=C).fit(X, y)

# LinearSVC (linear kernel)

lin\_svc = svm.LinearSVC(C=C, max\_iter=10000).fit(X, y)

# SVC with RBF kernel

rbf\_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)

# SVC with polynomial (degree 3) kernel

poly\_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)

# Define a step size in the mesh

h = .02

# Create a mesh to plot in

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),

                     np.arange(y\_min, y\_max, h))

# Titles for the plots

titles = [

    'SVC with linear kernel',

    'LinearSVC (linear kernel)',

    'SVC with RBF kernel',

    'SVC with polynomial (degree 3) kernel'

]

# Plot decision boundaries for each classifier

for i, clf in enumerate((svc, lin\_svc, rbf\_svc, poly\_svc)):

    plt.subplot(2, 2, i + 1)

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

    # Predict for each point in the mesh

    Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)

    # Put the result into a color plot

    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')

    plt.xlabel('Sepal length')

    plt.ylabel('Sepal width')

    plt.xlim(xx.min(), xx.max())

    plt.ylim(yy.min(), yy.max())

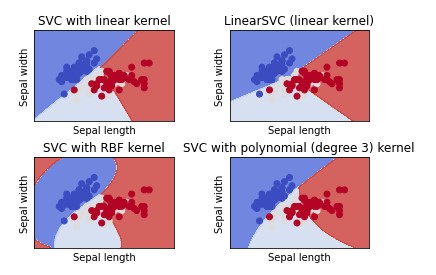
    plt.xticks(())

    plt.yticks(())

    plt.title(titles[i])

# Show the plots

plt.show()

output-

**Practical No.:5 Date: / /**

**Aim: Train and Find Tune Decision Tree for the moon dataset.**

**Source code:**

from sklearn.datasets import make\_moons

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Generate the Moons dataset

X, y = make\_moons(n\_samples=1000, noise=0.2, random\_state=42)

# Split the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define the model

model = DecisionTreeClassifier(random\_state=42)

# Define the parameter grid

params = {

    'max\_depth': [None, 5, 10, 20],

    'min\_samples\_split': [2, 5, 10],

    'min\_samples\_leaf': [1, 2, 4]

}

# Setup GridSearchCV

grid\_search = GridSearchCV(model, params, cv=5, n\_jobs=-1)

# Fit GridSearchCV to the training data

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and the best model

best\_params = grid\_search.best\_params\_

best\_model = grid\_search.best\_estimator\_

# Print the best parameters

print(f"Best parameters: {best\_params}")

# Predict and evaluate the best model

y\_pred = best\_model.predict(X\_test)

print(f"Accuracy of the best model: {accuracy\_score(y\_test, y\_pred):.4f}")

**Output:**



**Practical No.:6 Date: / /**

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

**Source 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]

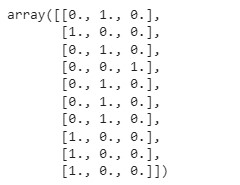
one\_hot(y\_valid[:10]) y\_train\_prob = one\_hot(y\_train) y\_valid\_prob = one\_hot(y\_valid) y\_test\_prob = one\_hot(y\_test)

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)

**Output:**





# Practical No.:7 Date: / / Aim: Implement MLP for classification of handwritten digits (MNIST Dataset)

## Code:-

import numpy as np import matplotlib.pyplot as plt import random

from keras.datasets import mnist from keras.models import Sequential from keras.layers.core import Dense, Dropout, Activation from keras.utils import np\_utils

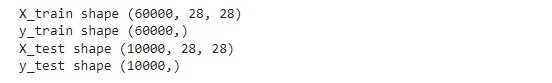
(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

**Output:-**



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

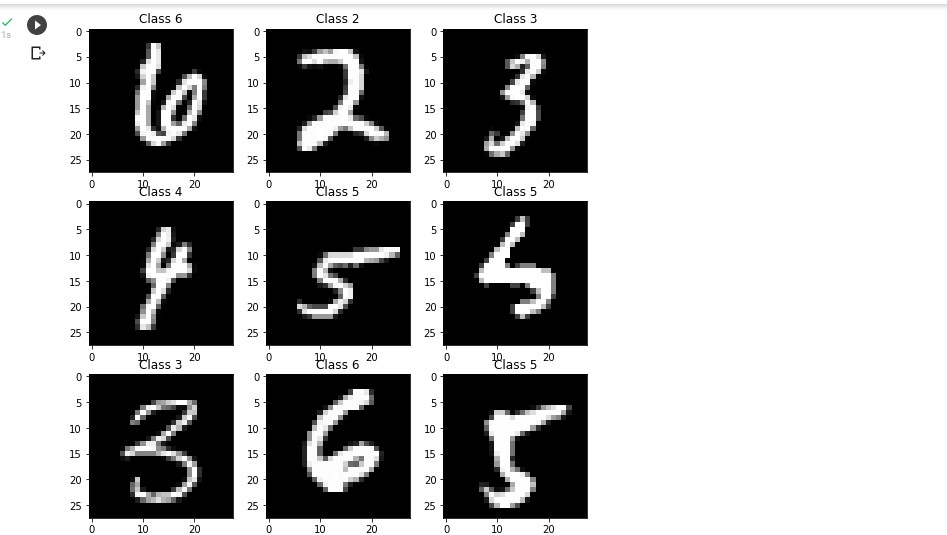
## Output:-



plt.rcParams['figure.figsize'] = (9,9) # Make the figures a bit bigger for i in range(9):

plt.subplot(3,3,i+1) num = random.randint(0, len(X\_train)) plt.imshow(X\_train[num], cmap='gray', interpolation='none') plt.title("Class {}".format(y\_train[num]))

## Output:-



plt.tight\_layout()

## Output:-



def matprint(mat, fmt="g"):

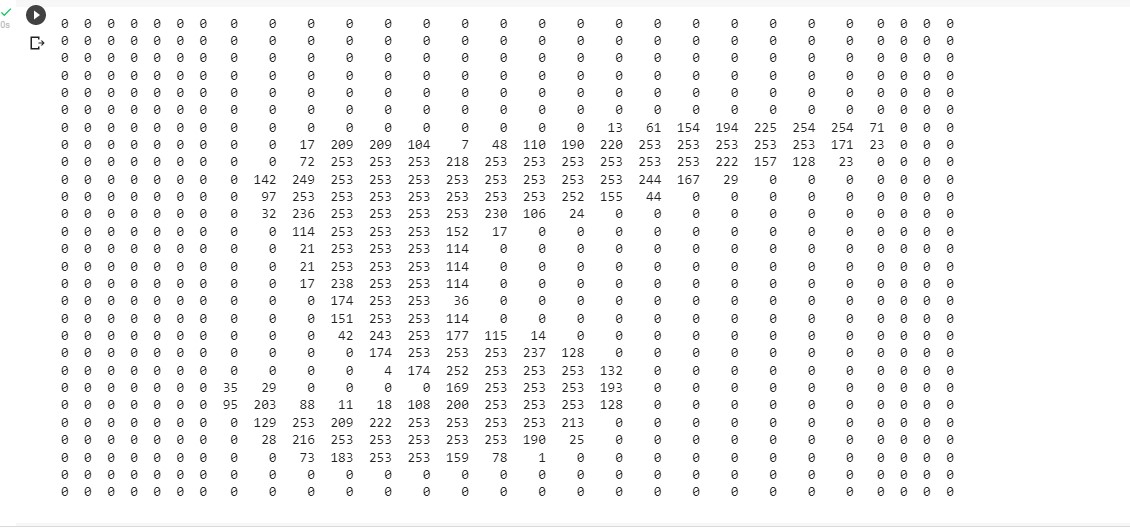
col\_maxes = [max([len(("{:"+fmt+"}").format(x)) for x in col]) for col in mat.T] for x in mat: for i, y in enumerate(x):

print(("{:"+str(col\_maxes[i])+fmt+"}").format(y), end=" ")

print("")

matprint(X\_train[num])

## Output:-



X\_train = X\_train.reshape(60000, 784)

X\_test = X\_test.reshape(10000, 784)

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255

X\_test /= 255 print("Training matrix shape", X\_train.shape) print("Testing matrix shape", X\_test.shape)

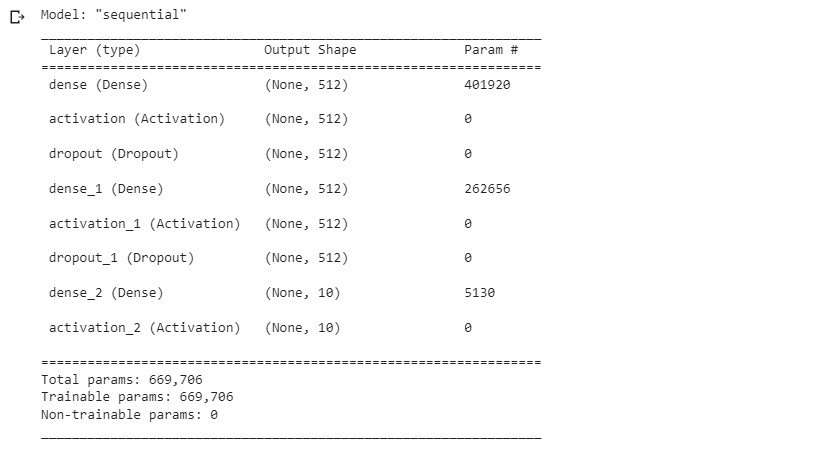
## Output:-



nb\_classes = 10

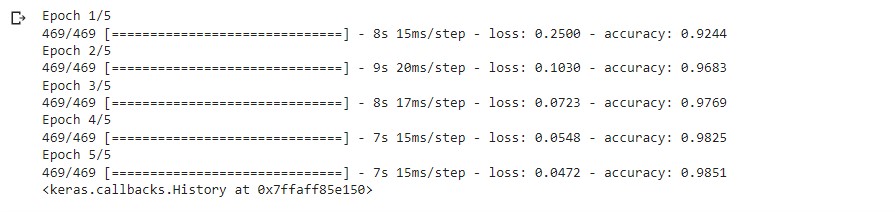
Y\_train = np\_utils.to\_categorical(y\_train, nb\_classes) Y\_test = np\_utils.to\_categorical(y\_test, nb\_classes) model = Sequential() model.add(Dense(512, input\_shape=(784,))) model.add(Activation('relu')) model.add(Dropout(0.2)) model.add(Dense(512)) model.add(Activation('relu')) model.add(Dropout(0.2)) model.add(Dense(10)) model.add(Activation('softmax')) model.summary()

## Output:-



model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy']) model.fit(X\_train, Y\_train, batch\_size=128, epochs=5, verbose=1)

## Output:-



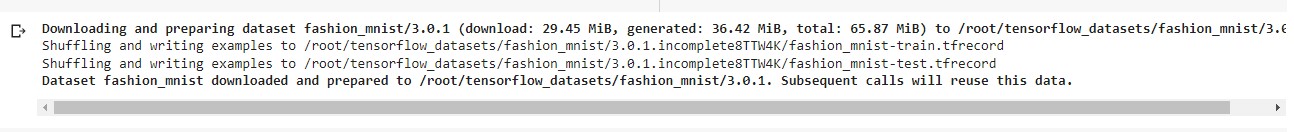
score = model.evaluate(X\_test, Y\_test) print('Test score:', score[0]) print('Test accuracy:', score[1])

## Output:-



# Practical No.:8 Date: / / Aim: Classification Of images Of Clothing Tensorflow (Fashion MNIST Dataset

**Code:-** import tensorflow as tf import tensorflow\_datasets as tfds tfds.disable\_progress\_bar() import math import numpy as np import matplotlib.pyplot as plt import logging logger = tf.get\_logger() logger.setLevel(logging.ERROR) dataset, metadata = tfds.load('fashion\_mnist', as\_supervised=True, with\_info=True) train\_dataset, test\_dataset = dataset['train'], dataset['test'] **Output:-**



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

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

num\_train\_examples = metadata.splits['train'].num\_examples num\_test\_examples = metadata.splits['test'].num\_examples print("Number of training examples: {}".format(num\_train\_examples)) print("Number of test examples: {}".format(num\_test\_examples)) **Output:-**



def normalize(images, labels):

images = tf.cast(images, tf.float32)

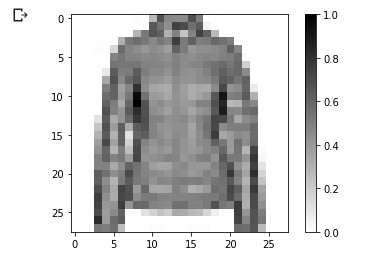
images /= 255 return images, labels train\_dataset = train\_dataset.map(normalize) test\_dataset = test\_dataset.map(normalize) train\_dataset = train\_dataset.cache() test\_dataset = test\_dataset.cache() for image, label in test\_dataset.take(1): break

image = image.numpy().reshape((28,28))

plt.figure()

plt.imshow(image, cmap=plt.cm.binary) plt.colorbar() plt.grid(False) plt.show()

## Output:-



plt.figure(figsize=(10,10)) i = 0

## Output:-



for (image, label) in test\_dataset.take(25):

image = image.numpy().reshape((28,28))

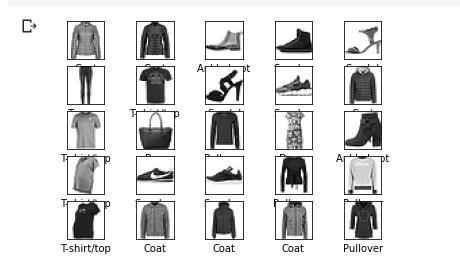
plt.subplot(5,5,i+1) plt.xticks([]) plt.yticks([]) plt.grid(False)

plt.imshow(image, cmap=plt.cm.binary) plt.xlabel(class\_names[label])

i += 1

plt.show()

## Output:-



model = tf.keras.Sequential([ tf.keras.layers.Flatten(input\_shape=(28, 28, 1)), ]) model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])

BATCH\_SIZE = 32 train\_dataset = train\_dataset.cache().shuffle(num\_train\_examples).batch(BATCH\_SIZE) test\_dataset = test\_dataset.cache().batch(BATCH\_SIZE) test\_loss, test\_accuracy = model.evaluate(test\_dataset, steps=math.ceil(num\_test\_examples/32)) print('Accuracy on test dataset:', test\_accuracy)

## Output:-



for test\_images, test\_labels in test\_dataset.take(1):

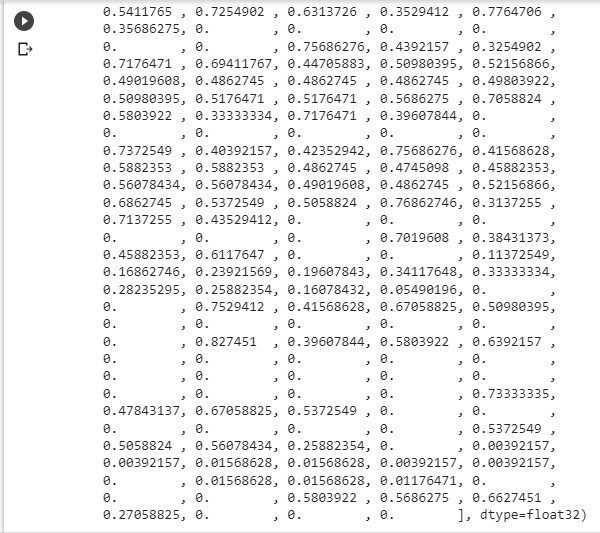
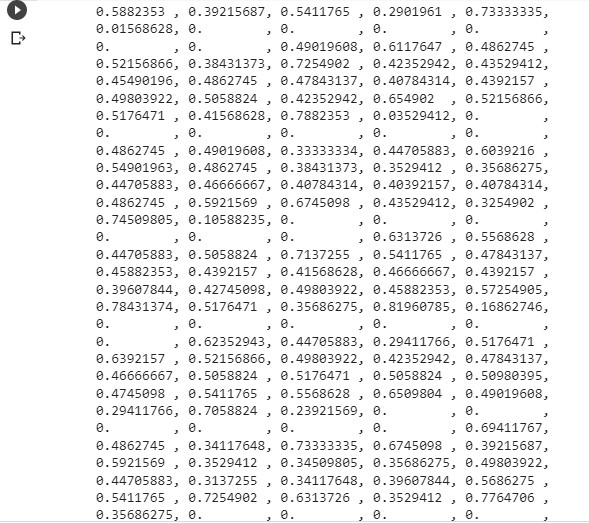
test\_images = test\_images.numpy() test\_labels = test\_labels.numpy() predictions = model.predict(test\_images) predictions.shape

## Output:-



predictions[0]

## Output:-



## Practical No.: 9 Date: / /

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

**Source code:**

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

url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/autompg/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: 'Japa n'}) dataset = pd.get\_dummies(dataset, columns=['Origin'], prefix='', prefix\_se p='') dataset.tail()

## Output-



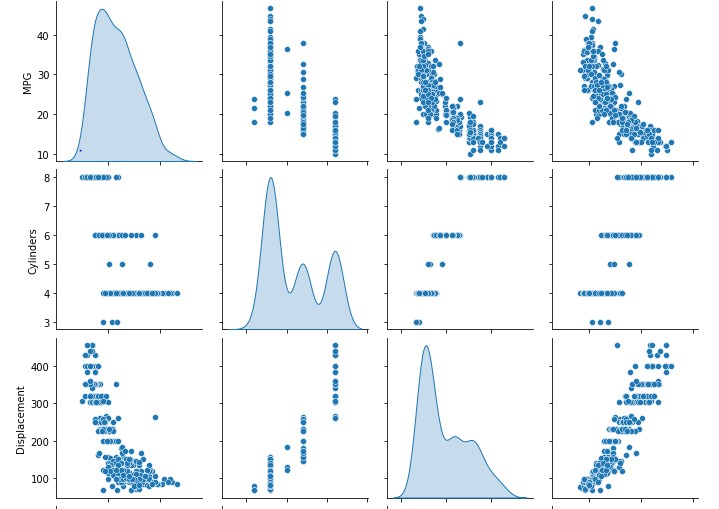
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/autompg/auto-mpg.data'

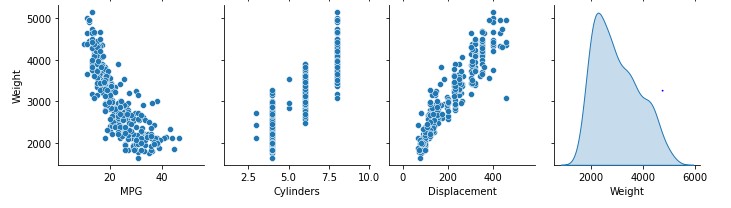
column\_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight' ,

'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: 'Japa n'}) dataset = pd.get\_dummies(dataset, columns=['Origin'], prefix='', prefix\_se p='') dataset.tail()

## Output-

[[ 5.478 195.318 104.869 2990.252 15.559 75.898 0.178 0.197 0.624]]





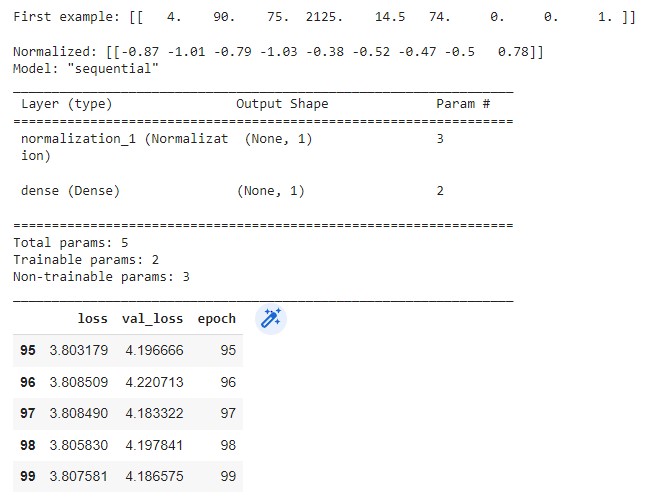
first = np.array(train\_features[:1]) with np.printoptions(precision=2, suppress=True): print('First example:', first) print() print('Normalized:', normalizer(first).numpy())

horsepower = np.array(train\_features['Horsepower']) horsepower\_normalizer = layers.Normalization(input\_shape=[1,], axis=None) horsepower\_normalizer.adapt(horsepower) horsepower\_model = tf.keras.Sequential([ horsepower\_normalizer, layers.Dense(units=1)

]) horsepower\_model.summary() horsepower\_model.predict(horsepower[:10]) horsepower\_model.compile( optimizer=tf.optimizers.Adam(learning\_rate=0.1), loss='mean\_absolute\_error') history = horsepower\_model.fit(train\_features['Horsepower'],train\_labels,e pochs=100, verbose=0, validation\_split = 0.2) hist = pd.DataFrame(history.history) hist['epoch'] = history.epoch

hist.tail()

## Output-

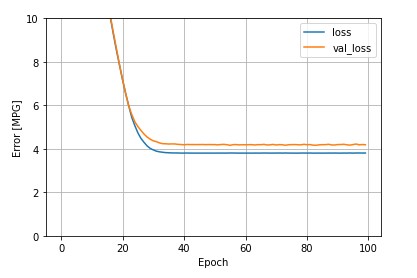


def plot\_loss(history):

plt.plot(history.history['loss'], label='loss') plt.plot(history.history['val\_loss'], label='val\_loss') plt.ylim([0, 10]) plt.xlabel('Epoch') plt.ylabel('Error [MPG]')

plt.legend() plt.grid(True) plot\_loss(history) test\_results = {} test\_results['horsepower\_model'] = horsepower\_model.evaluate( test\_features['Horsepower'], test\_labels, verbose=0) x = tf.linspace(0.0, 250, 251) y = horsepower\_model.predict(x)

## Output-



def plot\_horsepower(x, y):

plt.scatter(train\_features['Horsepower'], train\_labels, label='Data') plt.plot(x, y, color='k', label='Predictions') plt.xlabel('Horsepower') plt.ylabel('MPG') plt.legend()

plot\_horsepower(x, y)

## Output-

