Practical 5 :-

Source Code with Explanation-

```
import numpy as np import pandas as pd
from sklearn.datasets import load_boston boston =
load_boston()
data = pd.DataFrame(boston.data)
 data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
data.columns = boston.feature_names
data['PRICE'] = boston.target
data.head(n=10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

```
print(data.shape)
data.isnull().sum()
```

```
CRIM        0 ZN          0 INDUS       0
CHAS        0 NOX         0 RM          0
AGE         0 DIS         0 RAD         0
TAX         0 PTRATIO     0 B           0
LSTAT       0 PRICE       0 dtype: int64
```

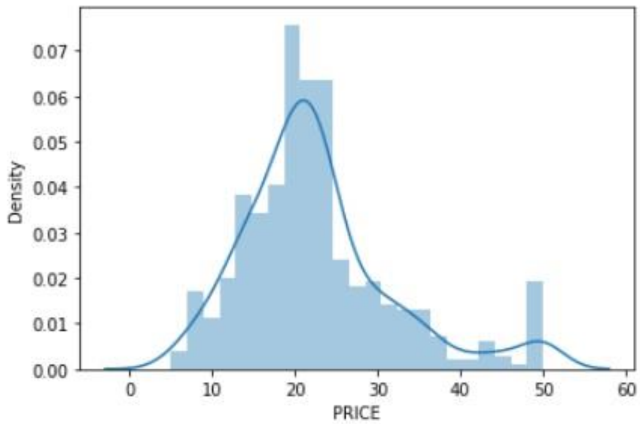
```
data.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

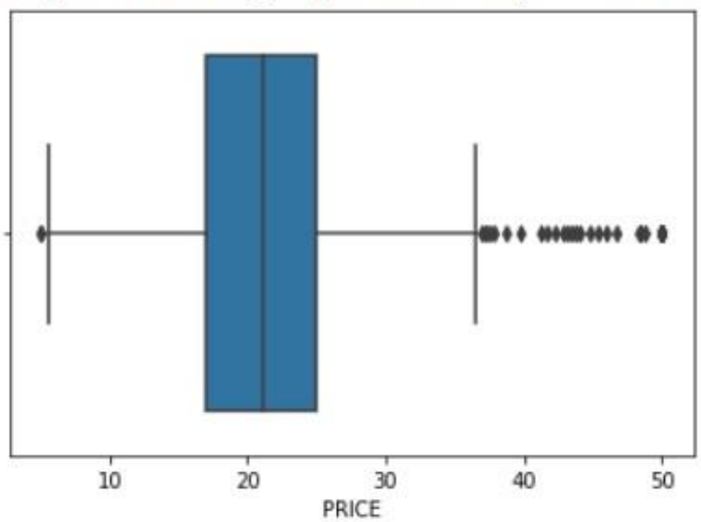
```
data.info() <class
'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505 Data columns
(total 14 columns):

 #   Column   Non-Null Count  Dtype -
--  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
```

```
 11   B         506 non-null    float64
 12   LSTAT     506 non-null    float64
 13   PRICE     506 non-null    float64
dtypes: float64(14) memory usage:
55.5 KB
import seaborn as sns sns.distplot(data.PRICE sns.boxplot(data.PRICE)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f44d082c670>

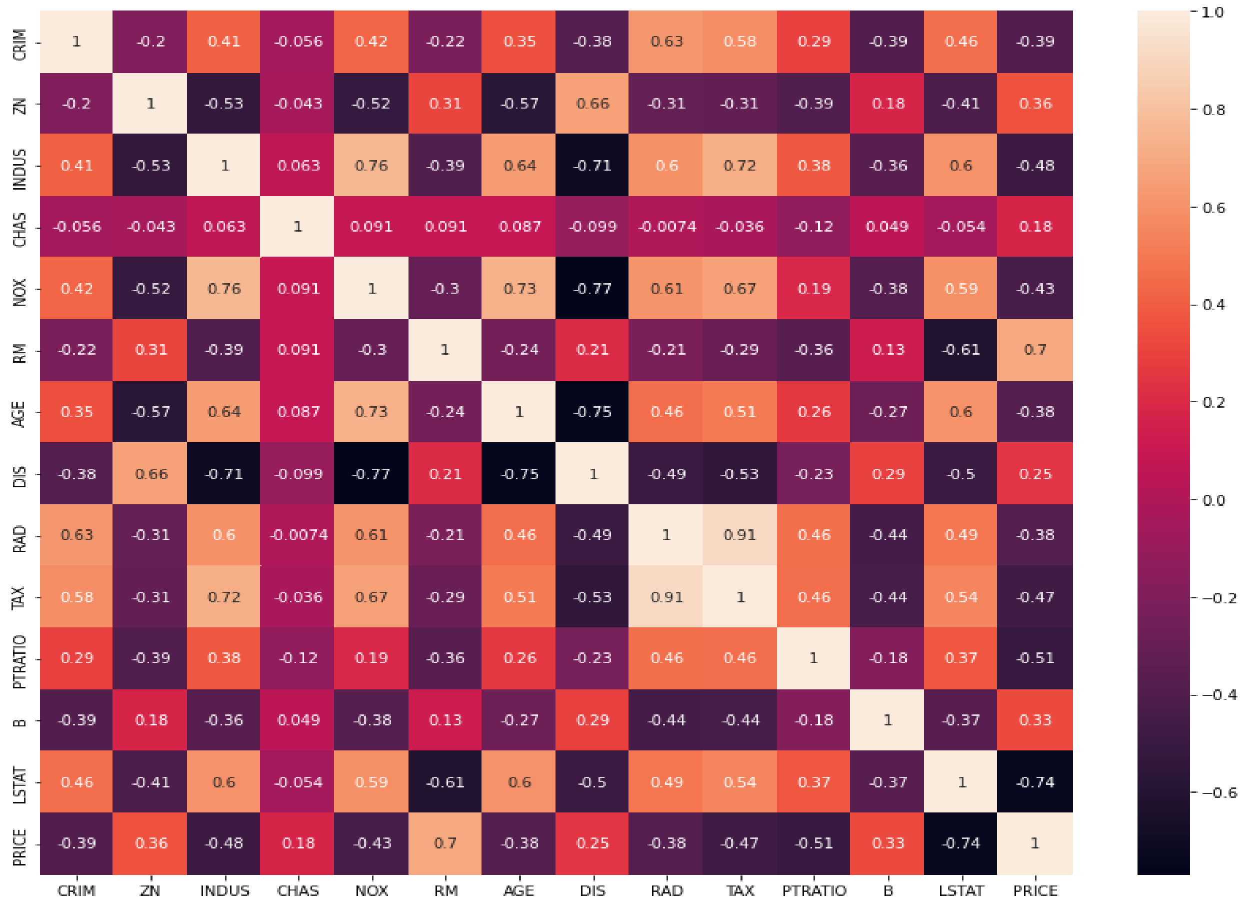


<matplotlib.axes._subplots.AxesSubplot at 0x7f44d077ed60>




```
correlation = data.corr() correlation.loc['PRICE']
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
```

```
RAD        -0.381626
TAX        -0.468536
PTRATIO    -0.507787
B           0.333461
LSTAT      -0.737663
PRICE       1.000000
Name: PRICE, dtype: float64 # plotting the heatmap import

matplotlib.pyplot as plt fig,axes =
plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

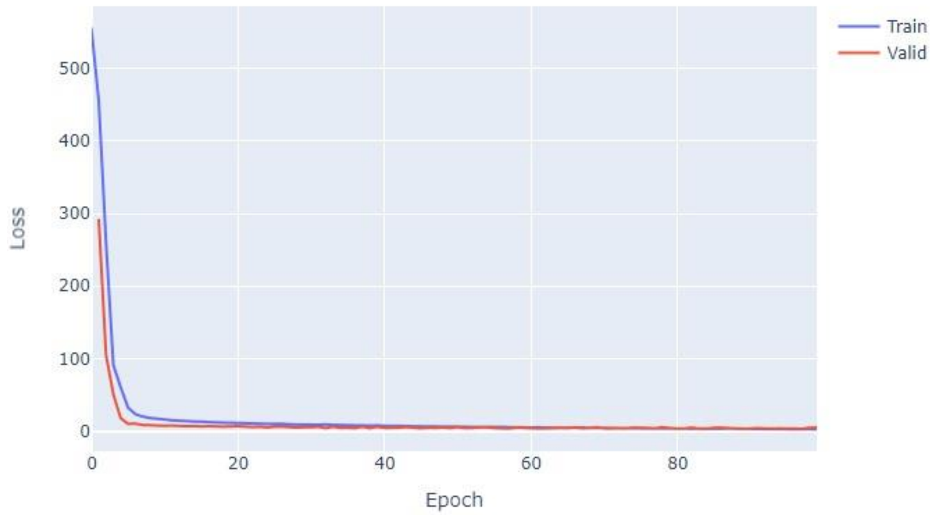



```
# Checking the scatter plot with the most correlated features
plt.figure(figsize = (20,5)) features =
['LSTAT','RM','PTRATIO'] for i, col in
enumerate(features):
    plt.subplot(1, len(features) ,
    i+1) x = data[col] y =
    data.PRICE
```

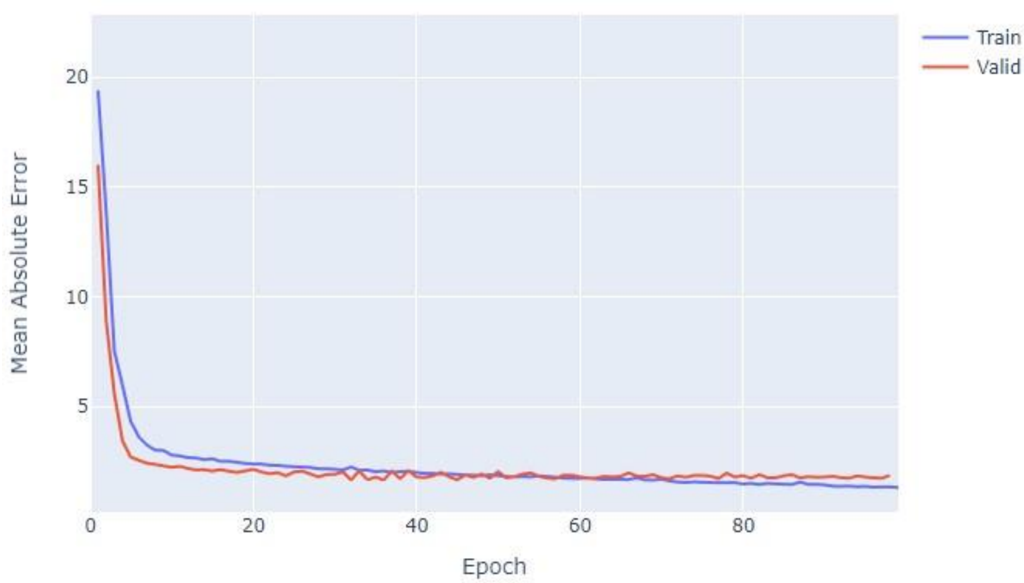
```
plt.scatter(x, y, marker='o')
plt.title("Variation in House prices")
plt.xlabel(col)
plt.ylabel('"House prices in $1000"')
```


```
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO']] X = data.iloc[:,:-1]
y= data.PRICE

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
#Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
#Fitting the model
regressor.fit(X_train,y_train)
# Model Evaluation

#Prediction on the test dataset y_pred =
regressor.predict(X_test) # Predicting RMSE the Test
set results from sklearn.metrics import
mean_squared_error rmse =
(np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
```

```
print(r2) # Neural Networks #Scaling
the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler() X_train =
sc.fit_transform(X_train)
X_test = sc.transform(X_test)

#Creating the neural network model
import keras
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential

model = Sequential()
model.add(Dense(128,activation          =          'relu',input_dim          =13))
model.add(Dense(64,activation  = 'relu')) model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation             =       'relu'))       model.add(Dense(1))
#model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.compile(optimizer = 'adam',loss ='mean_squared_error',metrics=['mae'])
!pip install ann_visualizer
!pip install graphviz
from ann_visualizer.visualize import ann_viz;
#Build your model here
ann_viz(model, title="DEMO ANN");
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)
from plotly.subplots import make_subplots import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['loss'],
                    name='Train'))

fig.add_trace(go.Scattergl(y=history.history['val_loss'],
                    name='Valid'))


fig.update_layout(height=500, width=700,
                  xaxis_title='Epoch', yaxis_title='Loss')
fig.show()
```




```
fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['mae'],name='Train'))
fig.add_trace(go.Scattergl(y=history.history['val_mae'],name='Valid'))
fig.update_layout(height=500, width=700,xaxis_title='Epoch', yaxis_title='Mean Absolute Error')
fig.show()
```




```
#Evaluation of the model
y_pred = model.predict(X_test)
mse_nn, mae_nn = model.evaluate(X_test, y_test)

print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
```

4/4 [==============================] - 0s 4ms/step - loss: 10.5717 - mae: 2.2670

Mean squared error on test data: 10.571733474731445

Mean absolute error on test data: 2.2669904232025146

```
#Comparison with traditional approaches #First let's try with a
simple algorithm, the Linear Regression: from sklearn.metrics
import mean_absolute_error

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test,
y_pred_lr) mae_lr =
mean_absolute_error(y_test,
y_pred_lr)

print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)
from sklearn.metrics import r2_score r2 =
r2_score(y_test, y_pred) print(r2)
0.8812832788381159
 # Predicting RMSE the Test set results from
 sklearn.metrics import mean_squared_error rmse =
 (np.sqrt(mean_squared_error(y_test, y_pred)))
 print(rmse) 3.320768607496587
 # Make predictions on new data import sklearn new_data =
 sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0,
 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]))
 prediction = model.predict(new_data)
 print("Predicted house price:", prediction) 1/1
 [==============================] - 0s 70ms/step



 Predicted house price:
 [[11.104753]]
```

Practical 6 :-

Source Code and Output-

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
#loading imdb data with most frequent 10000 words
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
data = np.concatenate((X_train, X_test), axis=0)
label = np.concatenate((y_train, y_test), axis=0)
X_train.shape

(25000,)
X_test.shape
(25000,)
y_train.shape
(25000,)
y_test.shape
(25000,)
print("Review is ",X_train[0]) # series of no converted word to vocabulory associated with index
print("Review is ",y_train[0])
Review is [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394,
20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114,
9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5,
89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4,
1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362,
98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255,
5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64,
1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]
Review is  0
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indices print(vocab)
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders':
16115,
y_train
array([1, 0, 0, ..., 0, 1, 0])
y_test
array([0, 1, 1, ..., 0, 0, 0])
```

def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it with zeros so that it contains exactly 10,000 numbers.

  # Create an all-zero matrix of shape (len(sequences),

    dimension) results = np.zeros((len(sequences), dimension))

    for i, sequence in enumerate(sequences):

        results[i, sequence] = 1

    return results

Now we split our data into a training and a testing set.

The training set will contain reviews and the testing set

Set a VALIDATION set

test_x = data[:10000]

test_y = label[:10000]

train_x =

data[10000:] train_y

= label[10000:]

test_x.shape (10000,)

test_y.shape (10000,)

train_x.shape

(40000,)

train_y.shape

(40000,)

print("Categories:", np.unique(label))

print("Number of unique words:", len(np.unique(np.hstack(data))))

The hstack() function is used to stack arrays in sequence horizontally (column wise).

Categories: [0 1]

Number of unique words: 9998

length = [len(i) for i in data]

print("Average Review length:", np.mean(length))

print("Standard Deviation:", round(np.std(length)))

Average Review length: 234.75892

Standard Deviation: 173

print("Label:", label[0])

Label: 1

print("Label:",

label[1]) Label: 0

print(data[0])

Retrieves a dict mapping words to their index in the IMDB dataset.

index = imdb.get_word_index() # word to index

Create inverted index from a dictionary with document ids as keys and a list of terms as values for each document

reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word

decoded = " ".join([reverse_index.get(i - 3, "#") for i in data[0]])

The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "start of sequence" and "unknown". print(decoded)

this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert # is an amazing actor and now the same being director # father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film

#Adding sequence to data

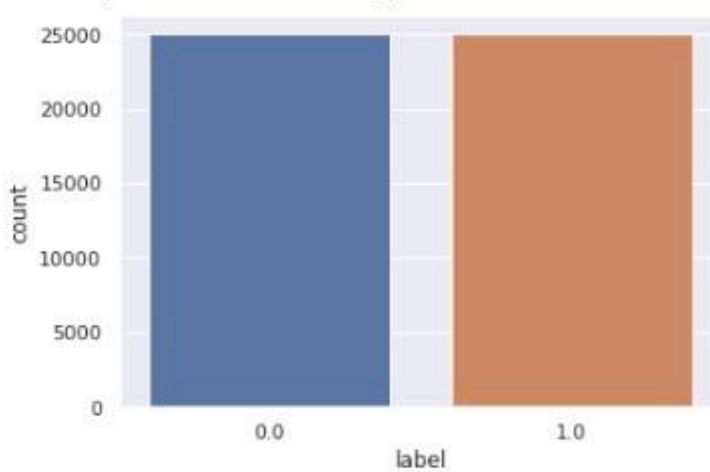
Vectorization is the process of converting textual data into numerical vectors and is a process that is usually applied once the text is cleaned. data = vectorize(data)

label = np.array(label).astype("float32")

labelDF=pd.DataFrame({'label':label})

sns.countplot(x='label', data=labelDF)

<AxesSubplot:xlabel='label', ylabel='count'>



Creating train and test data set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data,label, test_size=0.20, random_state=1)

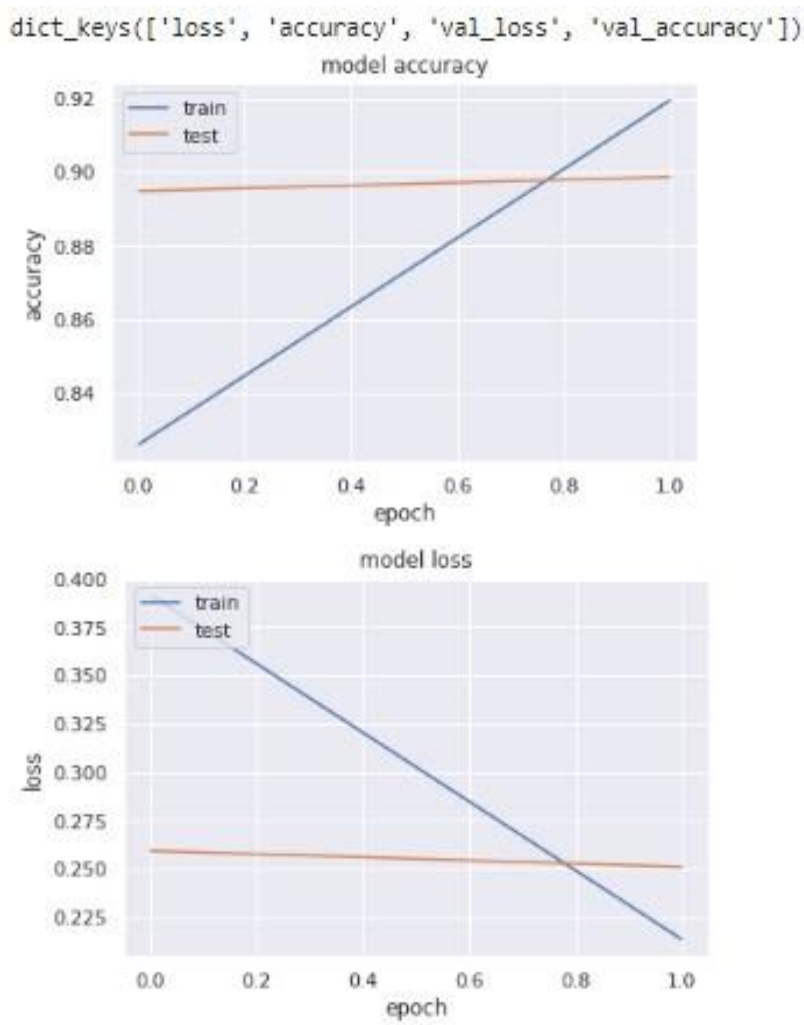
X_train.shape

(40000, 10000)
X_test.shape
(10000, 10000)
Let's create sequential model from
keras.utils import to_categorical
from keras import models from
keras import layers model =
models.Sequential()
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=============================================================
 ===== dense (Dense)              (None, 50)
 500050

 dropout (Dropout)           (None, 50)                0

 dense_1 (Dense)             (None, 50)                2550

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 50)                2550

 dense_3 (Dense)             (None, 1)                 51

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

Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
```

```
import tensorflow as tf callback =
tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
model.compile(
 optimizer = "adam", loss
 = "binary_crossentropy",
 metrics = ["accuracy"]
) from sklearn.model_selection import
train_test_split

results = model.fit(
 X_train, y_train, epochs= 2,
 batch_size = 500,
 validation_data = (X_test,
 y_test), callbacks=[callback]
) # Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"])
) # Evaluate the model score =
model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0]) print('Test
accuracy:', score[1])
20/20 [==============================] - 1s 24ms/step - loss: 0.2511 -
accuracy:
0.8986
Test loss: 0.25108325481414795
Test accuracy: 0.8985999822616577
#Let's plot training history of our
model.

# list all data in history
print(results.history.keys()) #
summarize history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy']
```

```
) plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch') plt.legend(['train',
'test'], loc='upper left') plt.show() #
summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch') plt.legend(['train',
'test'], loc='upper left') plt.show()
```



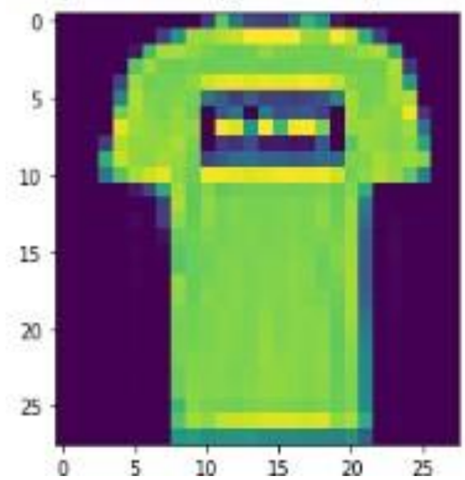
Practical 7 :-

Source Code with Output

```
import tensorflow as tf import
matplotlib.pyplot as plt from
tensorflow import keras import
numpy as np
```

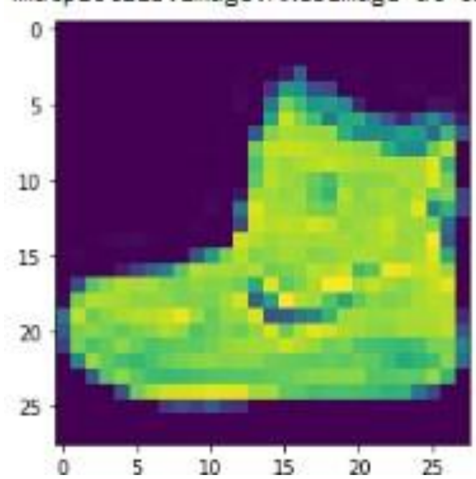
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data() plt.imshow(x_train[1])
```

```
<matplotlib.image.AxesImage at 0x7f85874f3a00>
```




```
plt.imshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f8584b93d00>
```




```
# Next, we will preprocess the data by scaling the pixel values to be between 0 and 1, and then reshaping the images to be 28x28 pixels.

x_train = x_train.astype('float32') / 255.0 x_test =
x_test.astype('float32') / 255.0
```

x_train = x_train.reshape(-1, 28, 28, 1) x_test =
x_test.reshape(-1, 28, 28, 1)

converting the training_images array to 4 dimensional array with sizes 60000, 28, 28, 1 for 0th to 3rd dimension. x_train.shape (60000, 28, 28) x_test.shape

(10000, 28, 28, 1)

y_train.shape (60000,)

y_test.shape (10000,)

model = keras.Sequential([keras.layers.Conv2D(32, (3,3), activation='relu',

input_shape=(28,28,1)),

keras.layers.MaxPooling2D((2,2)),

It shown 13 * 13 size image with 32 channel or filter or depth. keras.layers.Dropout(0.25),

Reduce Overfitting of Training sample drop out 25% Neuron keras.layers.Conv2D(64, (3,3),

activation='relu'),

keras.layers.MaxPooling2D((2,2)),

It shown 5 * 5 size image with 64 channel or filter or depth. keras.layers.Dropout(0.25),

keras.layers.Conv2D(128, (3,3), activation='relu'),

keras.layers.Flatten(),

keras.layers.Dense(128, activation='relu'),

128 Size of Node in Dense Layer

1152*128 = 147584

keras.layers.Dropout(0.25),

keras.layers.Dense(10, activation='softmax')

10 Size of Node another Dense Layer

128*10+10 bias= 1290

])

model.summary()

Model: "sequential"

Layer (type) Output Shape Param #

=== conv2d (Conv2D)

(None, 26, 26, 32) 320

max_pooling2d (MaxPooling2D (None, 13, 13, 32) 0
)

dropout (Dropout) (None, 13, 13, 32) 0

conv2d_1 (Conv2D) (None, 11, 11, 64) 18496

max_pooling2d_1 (MaxPooling (None, 5, 5, 64) 0
2D)

dropout_1 (Dropout) (None, 5, 5, 64) 0

conv2d_2 (Conv2D) (None, 3, 3, 128) 73856

flatten (Flatten) (None, 1152) 0

dense (Dense) (None, 128) 147584

dropout_2 (Dropout) (None, 128) 0

dense_1 (Dense) (None, 10) 1290

===

Total params: 241,546

Trainable params: 241,546

Non-trainable params: 0

```
# Compile and Train the Model
# After defining the model, we will compile it and train it on the training data.
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']) history
= model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test)) # 1875 is a number of batches.
By default batches contain 32 samles.60000 / 32 = 1875 # Finally, we will evaluate the performance of
the model on the test data.
test_loss, test_acc = model.evaluate(x_test, y_test) print('Test
accuracy:', test_acc)
```

313/313 [==============================] - 3s 10ms/step - loss: 0.2606 - accuracy: 0.9031

Test accuracy: 0.9031000137329102