import numpy as np

import pandas as pd

from sklearn.datasets import load\_boston boston = load\_boston()

data = pd.DataFrame(boston.data)

data.head()

**//first output**

data.columns = boston.feature\_names

data['PRICE'] = boston.target

data.head(n=10)

**//second output**

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()

**//3rd output**

data.info() <class

'pandas.core.frame.DataFrame'>

RangeIndex: 506 entries, 0 to 505

Data columns (total 14 columns):**//output 4**

import seaborn as sns

sns.distplot(data.PRICE)

**//output 5**

sns.boxplot(data.PRICE)

**//output 6**

correlation = data.corr() correlation.loc['PRICE']

**//output 7**

import matplotlib.pyplot as plt fig,axes = plt.subplots(figsize=(15,12))

sns.heatmap(correlation,square = True,annot = True)**//output 8**

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"')

**//output 9**

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

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

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

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)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler() X\_train = sc.fit\_transform(X\_train)

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

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 ='mean\_squared\_error',metrics=['mae'])

!pip install ann\_visualizer

!pip install graphviz

from ann\_visualizer.visualize import ann\_viz;

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,

SNJB’s Late Sau. K.B. Jain College Of Engineering

xaxis\_title='Epoch',

yaxis\_title='Loss')

fig.show()

**//output**

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()

**//output**

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)**//output**

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)

**//output**

sklearn.metrics import mean\_squared\_error rmse = (np.sqrt(mean\_squared\_error(y\_test, y\_pred)))

print(rmse)

**//output**

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)

**// maybe output Predicted house price: [[11.104753]]**