Importing libraries and the dataset

Unlike the conventional way, I import the library when it is needed. It will actually help you to understand where the application of the class and it's function is used

In [ ]:
```python
#Importing the pandas for data processing and numpy for numerical computing
import numpy as np
import pandas as pd
```

In [ ]:
```python
# Importing the Boston Housing dataset from the sklearn
from sklearn.datasets import load_boston
boston = load_boston()
```

```
C:\Users\gdalv\AppData\Roaming\Python\Python310\site-packages\sklearn\utils\deprecation.py:8
7: FutureWarning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and
will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

In [ ]:
```python
#Converting the data into pandas dataframe
data = pd.DataFrame(boston.data)
```

First look at the dataset

In [ ]:
```python
#First look at the data
data.head()
```

Out[ ]:

	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

In [ ]:

```
#Adding the feature names to the dataframe
data.columns = boston.feature_names
```

In [ ]:

```
#Adding the target variable to the dataset
data['PRICE'] = boston.target
```

In [ ]:

```
#Looking at the data with names and target variable
data.head()
```

Out[ ]:

	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

In [ ]:

```
#Shape of the data
print(data.shape)
```

```
(506, 14)
```

In [ ]:

```
#Checking the null values in the dataset
data.isnull().sum()
```

Out[ ]:

```
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
```

No null values in the dataset, no missing value treatement needed

In [ ]:

```
#Checking the statistics of the data
data.describe()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	R
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000

This is sometimes very useful, for example if you look at the CRIM the max is 88.97 and 75% of the value is below 3.677083 and mean is 3.613524 so it means the max values is actually an outlier or there are outliers present in the column

In [ ]:

```
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
```

Visualisation

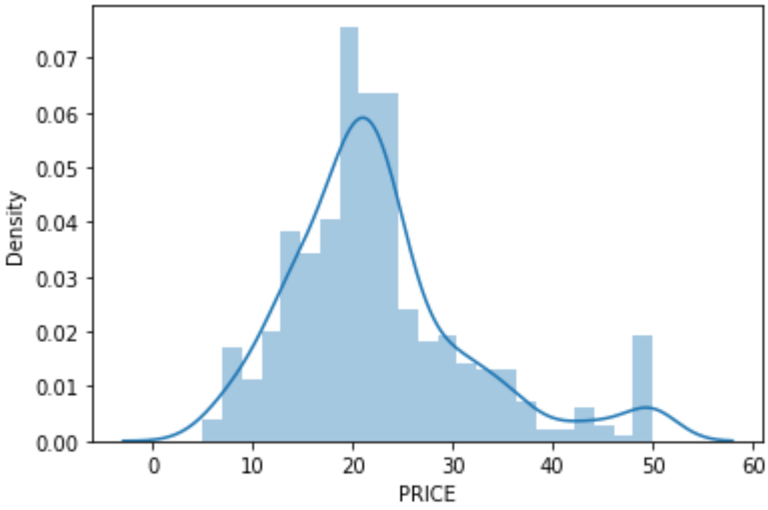
In [ ]:

```
#checking the distribution of the target variable
import seaborn as sns
sns.distplot(data.PRICE)
```

```
C:\Users\gdalv\AppData\Roaming\Python\Python310\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[ ]:

```
<AxesSubplot:xlabel='PRICE', ylabel='Density'>
```

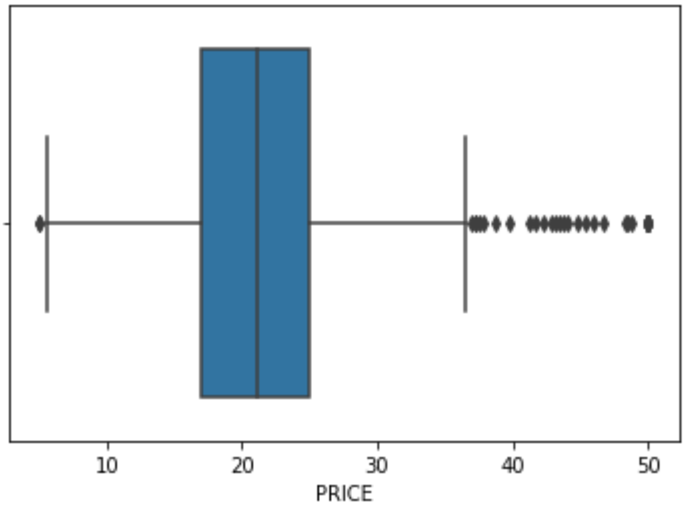


The distribution seems normal, has not be the data normal we would have perform log transformation or took to square root of the data to make the data normal. Normal distribution is need for the machine learning for better predictiblity of the model

In [ ]:
```
#Distribution using box plot
sns.boxplot(data.PRICE)
```

```
C:\Users\gdalv\AppData\Roaming\Python\Python310\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit keyword w
ill result in an error or misinterpretation.
  warnings.warn(
```

Out[ ]: `<AxesSubplot:xlabel='PRICE'>`



Checking the correlation of the independent feature with the dependent feature

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.An intelligent correlation analysis can lead to a greater understanding of your data

In [ ]:
```
#checking Correlation of the data
correlation = data.corr()
correlation.loc['PRICE']
```

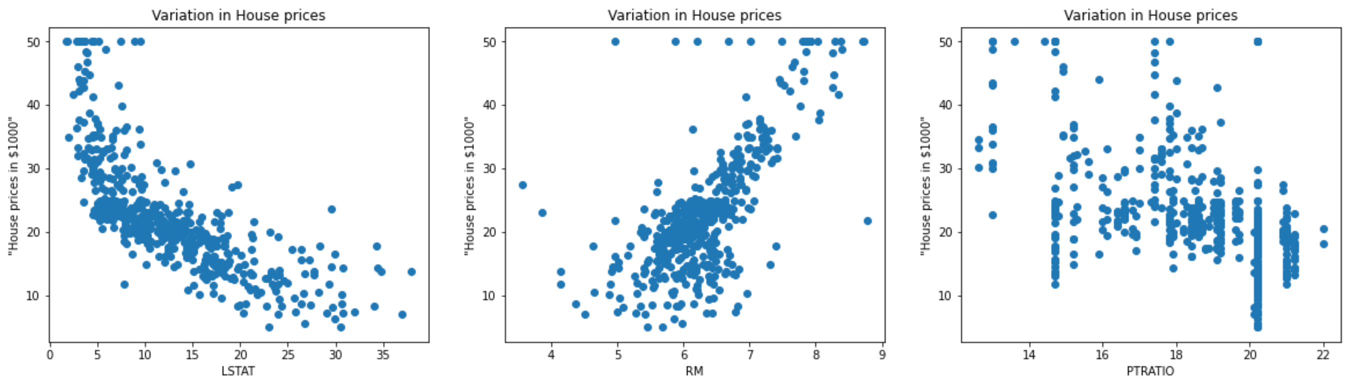
Out[ ]:
```
CRIM      -0.388305
ZN         0.360445
INDUS     -0.483725
CHAS       0.175260
NOX       -0.427321
RM         0.695360
AGE       -0.376955
DIS        0.249929
RAD       -0.381626
TAX       -0.468536
PTRATIO   -0.507787
B          0.333461
LSTAT     -0.737663
PRICE      1.000000
Name: PRICE, dtype: float64
```

In [ ]:
```
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
```

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

By looking at the correlation plot LSAT is negatively correlated with -0.75 and RM is positively correlated to the price and PTRATIO is correlated negatively with -0.51

In [ ]:
```
# 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

In [ ]:
```
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.PRICE
```

Splitting the data for Model Validation

In [ ]:
```
# Splitting the data into train and test for building the model
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, random_state = 4)
```

Building the Model

In [ ]:
```
#Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [ ]:
```
#Fitting the model
regressor.fit(X_train,y_train)
```

Out[ ]:
```
LinearRegression()
```

Model Evaluation

In [ ]:
```
#Prediction on the test dataset
y_pred = regressor.predict(X_test)
```

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

```
5.041784121402057
```

In [ ]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.7263451459702503
```

Neural Networks

In [ ]:
```
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- We are using Keras for developing the neural network.
- Models in Keras are defined as a sequence of layers
- We create a Sequential model and add layers one at a time with activation function
- Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.The activation we are using is relu
- As this is a regression problem, the output layer has no activation function
- Elements of neural network has input layer, hidden layer and output layer
- input layer:- This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
- Hidden layer:- Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.
- Output layer:- This layer bring up the information learned by the network to the outer world.
- Model Compilation:- The compilation is the final step in creating a model. Once the compilation is done, we can move on to training phase.
- Optimizer : - The optimizer we are using is adam. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
- Loss - mean square error

In [ ]:
```
#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 = 'mean_squared_error')
```

In [ ]:
```
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 1/100
13/13 [==============================] - 1s 3ms/step - loss: 563.3159
Epoch 2/100
13/13 [==============================] - 0s 2ms/step - loss: 501.5890
Epoch 3/100
13/13 [==============================] - 0s 2ms/step - loss: 366.6654
Epoch 4/100
13/13 [==============================] - 0s 2ms/step - loss: 155.0419
Epoch 5/100
13/13 [==============================] - 0s 2ms/step - loss: 71.9058
Epoch 6/100
13/13 [==============================] - 0s 2ms/step - loss: 40.7743
Epoch 7/100
13/13 [==============================] - 0s 2ms/step - loss: 28.4232
Epoch 8/100
13/13 [==============================] - 0s 2ms/step - loss: 23.0547
Epoch 9/100
13/13 [==============================] - 0s 2ms/step - loss: 20.4405
Epoch 10/100
13/13 [==============================] - 0s 2ms/step - loss: 18.7347
Epoch 11/100
13/13 [==============================] - 0s 2ms/step - loss: 17.1926
Epoch 12/100
13/13 [==============================] - 0s 2ms/step - loss: 16.2740
Epoch 13/100
13/13 [==============================] - 0s 2ms/step - loss: 15.3652
Epoch 14/100
13/13 [==============================] - 0s 2ms/step - loss: 14.6631
Epoch 15/100
13/13 [==============================] - 0s 2ms/step - loss: 14.4074
Epoch 16/100
13/13 [==============================] - 0s 2ms/step - loss: 13.4164
Epoch 17/100
13/13 [==============================] - 0s 2ms/step - loss: 12.9791
Epoch 18/100
13/13 [==============================] - 0s 2ms/step - loss: 12.7508
Epoch 19/100
13/13 [==============================] - 0s 2ms/step - loss: 12.2624
Epoch 20/100
13/13 [==============================] - 0s 2ms/step - loss: 11.9284
Epoch 21/100
13/13 [==============================] - 0s 2ms/step - loss: 11.7339
Epoch 22/100
13/13 [==============================] - 0s 3ms/step - loss: 11.2518
Epoch 23/100
13/13 [==============================] - 0s 3ms/step - loss: 11.0228
Epoch 24/100
13/13 [==============================] - 0s 2ms/step - loss: 10.8607
Epoch 25/100
13/13 [==============================] - 0s 2ms/step - loss: 10.4705
Epoch 26/100
13/13 [==============================] - 0s 2ms/step - loss: 10.5055
Epoch 27/100
13/13 [==============================] - 0s 2ms/step - loss: 10.2065
Epoch 28/100
13/13 [==============================] - 0s 2ms/step - loss: 9.9856
Epoch 29/100
13/13 [==============================] - 0s 2ms/step - loss: 9.7969
Epoch 30/100
13/13 [==============================] - 0s 2ms/step - loss: 9.9357
Epoch 31/100
13/13 [==============================] - 0s 2ms/step - loss: 9.7355
Epoch 32/100
13/13 [==============================] - 0s 2ms/step - loss: 9.1473
Epoch 33/100
13/13 [==============================] - 0s 2ms/step - loss: 9.3084
Epoch 34/100
13/13 [==============================] - 0s 2ms/step - loss: 9.0142
Epoch 35/100
```

```
13/13 [==============================] - 0s 2ms/step - loss: 8.7820
Epoch 36/100
13/13 [==============================] - 0s 2ms/step - loss: 8.5546
Epoch 37/100
13/13 [==============================] - 0s 2ms/step - loss: 8.6269
Epoch 38/100
13/13 [==============================] - 0s 2ms/step - loss: 8.4091
Epoch 39/100
13/13 [==============================] - 0s 2ms/step - loss: 8.2848
Epoch 40/100
13/13 [==============================] - 0s 2ms/step - loss: 7.9578
Epoch 41/100
13/13 [==============================] - 0s 2ms/step - loss: 7.7612
Epoch 42/100
13/13 [==============================] - 0s 2ms/step - loss: 7.7153
Epoch 43/100
13/13 [==============================] - 0s 2ms/step - loss: 7.7634
Epoch 44/100
13/13 [==============================] - 0s 2ms/step - loss: 7.4754
Epoch 45/100
13/13 [==============================] - 0s 2ms/step - loss: 7.2618
Epoch 46/100
13/13 [==============================] - 0s 2ms/step - loss: 7.1263
Epoch 47/100
13/13 [==============================] - 0s 2ms/step - loss: 7.0390
Epoch 48/100
13/13 [==============================] - 0s 2ms/step - loss: 6.8745
Epoch 49/100
13/13 [==============================] - 0s 2ms/step - loss: 6.6266
Epoch 50/100
13/13 [==============================] - 0s 2ms/step - loss: 6.7758
Epoch 51/100
13/13 [==============================] - 0s 2ms/step - loss: 6.6196
Epoch 52/100
13/13 [==============================] - 0s 2ms/step - loss: 6.2501
Epoch 53/100
13/13 [==============================] - 0s 2ms/step - loss: 6.2896
Epoch 54/100
13/13 [==============================] - 0s 2ms/step - loss: 6.3704
Epoch 55/100
13/13 [==============================] - 0s 2ms/step - loss: 6.0671
Epoch 56/100
13/13 [==============================] - 0s 2ms/step - loss: 5.8727
Epoch 57/100
13/13 [==============================] - 0s 2ms/step - loss: 6.1572
Epoch 58/100
13/13 [==============================] - 0s 2ms/step - loss: 5.7986
Epoch 59/100
13/13 [==============================] - 0s 2ms/step - loss: 5.6451
Epoch 60/100
13/13 [==============================] - 0s 2ms/step - loss: 5.5917
Epoch 61/100
13/13 [==============================] - 0s 2ms/step - loss: 6.0675
Epoch 62/100
13/13 [==============================] - 0s 2ms/step - loss: 5.3905
Epoch 63/100
13/13 [==============================] - 0s 2ms/step - loss: 5.4172
Epoch 64/100
13/13 [==============================] - 0s 2ms/step - loss: 5.1796
Epoch 65/100
13/13 [==============================] - 0s 2ms/step - loss: 4.9888
Epoch 66/100
13/13 [==============================] - 0s 2ms/step - loss: 5.3156
Epoch 67/100
13/13 [==============================] - 0s 2ms/step - loss: 5.1464
Epoch 68/100
13/13 [==============================] - 0s 2ms/step - loss: 5.0524
Epoch 69/100
13/13 [==============================] - 0s 2ms/step - loss: 4.8972
```

```
Epoch 70/100
13/13 [==============================] - 0s 2ms/step - loss: 4.8128
Epoch 71/100
13/13 [==============================] - 0s 2ms/step - loss: 4.7170
Epoch 72/100
13/13 [==============================] - 0s 2ms/step - loss: 4.7933
Epoch 73/100
13/13 [==============================] - 0s 2ms/step - loss: 4.6601
Epoch 74/100
13/13 [==============================] - 0s 2ms/step - loss: 4.5801
Epoch 75/100
13/13 [==============================] - 0s 2ms/step - loss: 4.4648
Epoch 76/100
13/13 [==============================] - 0s 2ms/step - loss: 4.4170
Epoch 77/100
13/13 [==============================] - 0s 2ms/step - loss: 4.2981
Epoch 78/100
13/13 [==============================] - 0s 2ms/step - loss: 4.3322
Epoch 79/100
13/13 [==============================] - 0s 2ms/step - loss: 4.2547
Epoch 80/100
13/13 [==============================] - 0s 2ms/step - loss: 4.1987
Epoch 81/100
13/13 [==============================] - 0s 2ms/step - loss: 4.3049
Epoch 82/100
13/13 [==============================] - 0s 2ms/step - loss: 4.3821
Epoch 83/100
13/13 [==============================] - 0s 2ms/step - loss: 3.9092
Epoch 84/100
13/13 [==============================] - 0s 2ms/step - loss: 4.1693
Epoch 85/100
13/13 [==============================] - 0s 2ms/step - loss: 3.9212
Epoch 86/100
13/13 [==============================] - 0s 2ms/step - loss: 4.0723
Epoch 87/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8634
Epoch 88/100
13/13 [==============================] - 0s 2ms/step - loss: 4.0444
Epoch 89/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8160
Epoch 90/100
13/13 [==============================] - 0s 2ms/step - loss: 3.7094
Epoch 91/100
13/13 [==============================] - 0s 2ms/step - loss: 3.7065
Epoch 92/100
13/13 [==============================] - 0s 2ms/step - loss: 3.6865
Epoch 93/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8382
Epoch 94/100
13/13 [==============================] - 0s 2ms/step - loss: 3.6737
Epoch 95/100
13/13 [==============================] - 0s 2ms/step - loss: 3.6077
Epoch 96/100
13/13 [==============================] - 0s 2ms/step - loss: 3.4857
Epoch 97/100
13/13 [==============================] - 0s 2ms/step - loss: 3.6583
Epoch 98/100
13/13 [==============================] - 0s 2ms/step - loss: 3.8278
Epoch 99/100
13/13 [==============================] - 0s 2ms/step - loss: 3.5358
Epoch 100/100
13/13 [==============================] - 0s 2ms/step - loss: 3.3656
```

Out[ ]:

```
<keras.callbacks.History at 0x1a01e7b6d70>
```

Evaluation of the model

In [ ]:
```
y_pred = model.predict(X_test)
```

```
4/4 [==============================] - 0s 2ms/step
4/4 [==============================] - 0s 2ms/step
```

In [ ]:
```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
```

```
0.9094182609708286
```

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

```
2.9007012044499842
```