Problem Statement

Linear regression by using Deep Neural network: Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

Import Library

In [65]:
```
# Data analysis and visualization
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Preprocessing and evaluation
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler
```

Load Data

In [66]:
```
(X_train , y_train), (X_test , y_test) = tf.keras.datasets.boston_housing.load_d
                                           path = 'boston_housing_npz',
                                           test_split = 0.2,
                                           seed = 42
                                        )
```

In [67]:
```
# Checking the data shape and type
(X_train.shape, type(X_train)), (X_test.shape, type(X_test)), (y_train.shape, ty
```

Out[67]:
```
(((404, 13), numpy.ndarray),
 ((102, 13), numpy.ndarray),
 ((404,), numpy.ndarray),
 ((102,), numpy.ndarray))
```

In [68]:
```
# Converting Data to DataFrame
X_train_df = pd.DataFrame(X_train)
y_train_df = pd.DataFrame(y_train)

# Preview the training data
X_train_df.head(10)
```

Out[68]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.09178	0.0	4.05	0.0	0.510	6.416	84.1	2.6463	5.0	296.0	16.6	395.50	9.04
1	0.05644	40.0	6.41	1.0	0.447	6.758	32.9	4.0776	4.0	254.0	17.6	396.90	3.53
2	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390.11	18.07
3	0.09164	0.0	10.81	0.0	0.413	6.065	7.8	5.2873	4.0	305.0	19.2	390.91	5.52
4	5.09017	0.0	18.10	0.0	0.713	6.297	91.8	2.3682	24.0	666.0	20.2	385.09	17.27
5	0.10153	0.0	12.83	0.0	0.437	6.279	74.5	4.0522	5.0	398.0	18.7	373.66	11.97
6	0.31827	0.0	9.90	0.0	0.544	5.914	83.2	3.9986	4.0	304.0	18.4	390.70	18.33
7	0.29090	0.0	21.89	0.0	0.624	6.174	93.6	1.6119	4.0	437.0	21.2	388.08	24.16
8	4.03841	0.0	18.10	0.0	0.532	6.229	90.7	3.0993	24.0	666.0	20.2	395.33	12.87
9	0.22438	0.0	9.69	0.0	0.585	6.027	79.7	2.4982	6.0	391.0	19.2	396.90	14.33

In [69]:

```
# View summary of datasets
X_train_df.info()
print('_'*40)
y_train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404 entries, 0 to 403
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       404 non-null    float64
 1   1       404 non-null    float64
 2   2       404 non-null    float64
 3   3       404 non-null    float64
 4   4       404 non-null    float64
 5   5       404 non-null    float64
 6   6       404 non-null    float64
 7   7       404 non-null    float64
 8   8       404 non-null    float64
 9   9       404 non-null    float64
 10  10      404 non-null    float64
 11  11      404 non-null    float64
 12  12      404 non-null    float64
dtypes: float64(13)
memory usage: 41.2 KB
________________________________________
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404 entries, 0 to 403
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       404 non-null    float64
dtypes: float64(1)
memory usage: 3.3 KB
```

In [70]:

```
X_train_df.describe()
```

Out[70]:

	0	1	2	3	4	5	6
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
mean	3.789989	11.568069	11.214059	0.069307	0.554524	6.284824	69.119307
std	9.132761	24.269648	6.925462	0.254290	0.116408	0.723759	28.034606
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.081960	0.000000	5.190000	0.000000	0.452000	5.878750	45.475000
50%	0.262660	0.000000	9.690000	0.000000	0.538000	6.210000	77.500000
75%	3.717875	12.500000	18.100000	0.000000	0.624000	6.620500	94.425000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

Preprocessing

In [71]:
```
# Create column transformer
ct = make_column_transformer(
    (MinMaxScaler(), [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12])
)

# Normalization and data type change
X_train = ct.fit_transform(X_train).astype('float32')
X_test = ct.transform(X_test).astype('float32')
y_train = y_train.astype('float32')
y_test = y_test.astype('float32')

# Distribution of X_train feature values after normalization
pd.DataFrame(X_train).describe()
```

Out[71]:

	0	1	2	3	4	5	6
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
mean	0.042528	0.115681	0.394210	0.348815	0.521905	0.681970	0.241618
std	0.102650	0.242696	0.253866	0.239522	0.138678	0.288719	0.194973
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000850	0.000000	0.173387	0.137860	0.444098	0.438466	0.087361
50%	0.002881	0.000000	0.338343	0.314815	0.507569	0.768280	0.184767
75%	0.041717	0.125000	0.646628	0.491770	0.586223	0.942585	0.362255
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Model, Predict, Evaluation

In [72]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

Out[72]:
```
((363, 12), (41, 12), (363,), (41,))
```

Creating the Model and Optimizing the Learning Rate learning rate = 0.01, batch_size = 32, dense_layers = 2, hidden_units for Dense_1 layer= 10, hidden_units for Dense_2 layer = 100

In [73]:
```
# Set random seed
tf.random.set_seed(42)

# Building the model
model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=10, activation='relu', input_shape=(X_train.shape[
  tf.keras.layers.Dense(units=100, activation='relu', name='Dense_2'),
  tf.keras.layers.Dense(units=1, name='Prediction')
])

# Compiling the model
model.compile(
    loss = tf.keras.losses.mean_squared_error,
    optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.01),
    metrics = ['mse']
)

# Training the model
history = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=50,
    validation_data=(X_val, y_val)
)
```

```
C:\Users\admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras
\src\layers\core\dense.py:88: UserWarning: Do not pass an `input_shape`/`input_
dim` argument to a layer. When using Sequential models, prefer using an `Input
(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - loss: 373.0555 - mse: 373.3045 - val_
loss: 95.6019 - val_mse: 125.1949
Epoch 2/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 90.9924 - mse: 90.7250 - val_los
s: 59.1337 - val_mse: 75.6957
Epoch 3/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 57.1830 - mse: 57.1047 - val_los
s: 48.6360 - val_mse: 58.5284
Epoch 4/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 47.4303 - mse: 47.4261 - val_los
s: 45.1271 - val_mse: 52.3747
Epoch 5/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 43.4081 - mse: 43.4368 - val_los
s: 41.5549 - val_mse: 47.4119
Epoch 6/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 39.8037 - mse: 39.8562 - val_los
s: 36.3672 - val_mse: 41.5954
Epoch 7/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 35.7645 - mse: 35.8292 - val_los
s: 31.5579 - val_mse: 36.1743
Epoch 8/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 32.1831 - mse: 32.2583 - val_los
s: 27.7860 - val_mse: 31.6564
Epoch 9/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 29.3124 - mse: 29.3902 - val_los
s: 25.0090 - val_mse: 28.5063
Epoch 10/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 27.2456 - mse: 27.3230 - val_los
s: 22.5495 - val_mse: 25.6590
Epoch 11/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 25.5360 - mse: 25.6209 - val_los
s: 20.9782 - val_mse: 23.9131
Epoch 12/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 24.5268 - mse: 24.6152 - val_lo
ss: 19.6550 - val_mse: 22.4570
Epoch 13/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 23.6689 - mse: 23.7587 - val_lo
ss: 18.7873 - val_mse: 21.3226
Epoch 14/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 22.9617 - mse: 23.0537 - val_los
s: 17.9019 - val_mse: 20.1441
Epoch 15/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 22.4037 - mse: 22.4961 - val_los
s: 17.3459 - val_mse: 19.4792
Epoch 16/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 21.9437 - mse: 22.0339 - val_los
s: 17.2272 - val_mse: 19.1458
Epoch 17/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.6555 - mse: 21.7453 - val_los
s: 16.9615 - val_mse: 18.7745
Epoch 18/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.3495 - mse: 21.4411 - val_los
s: 16.3717 - val_mse: 18.2899
Epoch 19/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.6726 - mse: 20.7630 - val_los
s: 15.9535 - val_mse: 17.9287
Epoch 20/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.2702 - mse: 20.3593 - val_los
s: 16.0443 - val_mse: 17.9347
```

```
Epoch 21/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 20.2047 - mse: 20.2942 - val_los
s: 16.1980 - val_mse: 18.0644
Epoch 22/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 19.9128 - mse: 20.0025 - val_los
s: 16.0767 - val_mse: 18.0009
Epoch 23/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 19.6004 - mse: 19.6896 - val_los
s: 16.2789 - val_mse: 18.1460
Epoch 24/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 19.2893 - mse: 19.3797 - val_los
s: 16.0885 - val_mse: 17.9629
Epoch 25/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 18.7077 - mse: 18.7943 - val_los
s: 16.8306 - val_mse: 18.6285
Epoch 26/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 18.9461 - mse: 19.0330 - val_los
s: 16.4194 - val_mse: 18.2690
Epoch 27/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 18.3644 - mse: 18.4507 - val_los
s: 16.5490 - val_mse: 18.3886
Epoch 28/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 18.2240 - mse: 18.3101 - val_los
s: 16.5265 - val_mse: 18.4681
Epoch 29/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 17.9240 - mse: 18.0101 - val_los
s: 16.4153 - val_mse: 18.4507
Epoch 30/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 17.6055 - mse: 17.6908 - val_lo
ss: 16.4671 - val_mse: 18.5082
Epoch 31/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 17.4538 - mse: 17.5392 - val_los
s: 16.2644 - val_mse: 18.3329
Epoch 32/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 17.1733 - mse: 17.2598 - val_los
s: 16.0222 - val_mse: 18.1793
Epoch 33/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 16.9461 - mse: 17.0327 - val_los
s: 15.7303 - val_mse: 17.9443
Epoch 34/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 16.6271 - mse: 16.7137 - val_lo
ss: 15.5304 - val_mse: 17.8173
Epoch 35/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.3307 - mse: 16.4184 - val_los
s: 14.6386 - val_mse: 16.8968
Epoch 36/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 15.9605 - mse: 16.0490 - val_los
s: 14.3735 - val_mse: 16.6316
Epoch 37/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 15.7633 - mse: 15.8517 - val_los
s: 14.3241 - val_mse: 16.6089
Epoch 38/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 15.4102 - mse: 15.4995 - val_los
s: 13.9847 - val_mse: 16.2224
Epoch 39/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 15.0752 - mse: 15.1640 - val_los
s: 13.9605 - val_mse: 16.2534
Epoch 40/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 15.0411 - mse: 15.1314 - val_los
s: 13.0942 - val_mse: 15.3455
```

```
Epoch 41/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 14.7562 - mse: 14.8464 - val_lo
ss: 13.2051 - val_mse: 15.5548
Epoch 42/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 14.4934 - mse: 14.5852 - val_los
s: 12.5080 - val_mse: 14.7156
Epoch 43/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 14.1859 - mse: 14.2789 - val_los
s: 11.8898 - val_mse: 13.9533
Epoch 44/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 13.7463 - mse: 13.8391 - val_los
s: 11.6927 - val_mse: 13.6303
Epoch 45/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 13.5959 - mse: 13.6895 - val_los
s: 11.2100 - val_mse: 13.0022
Epoch 46/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 13.3295 - mse: 13.4239 - val_los
s: 10.6677 - val_mse: 12.0708
Epoch 47/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 12.6864 - mse: 12.7808 - val_lo
ss: 10.3908 - val_mse: 11.6074
Epoch 48/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 12.3872 - mse: 12.4816 - val_los
s: 10.1643 - val_mse: 11.2407
Epoch 49/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 12.0128 - mse: 12.1064 - val_lo
ss: 9.9178 - val_mse: 10.8233
Epoch 50/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 11.7226 - mse: 11.8161 - val_los
s: 9.7806 - val_mse: 10.5111
```

Model Evaluation

In [74]:
```
# Preview the mean value of training and validation data
y_train.mean(), y_val.mean()
```

Out[74]:
```
(22.235537, 24.89756)
```

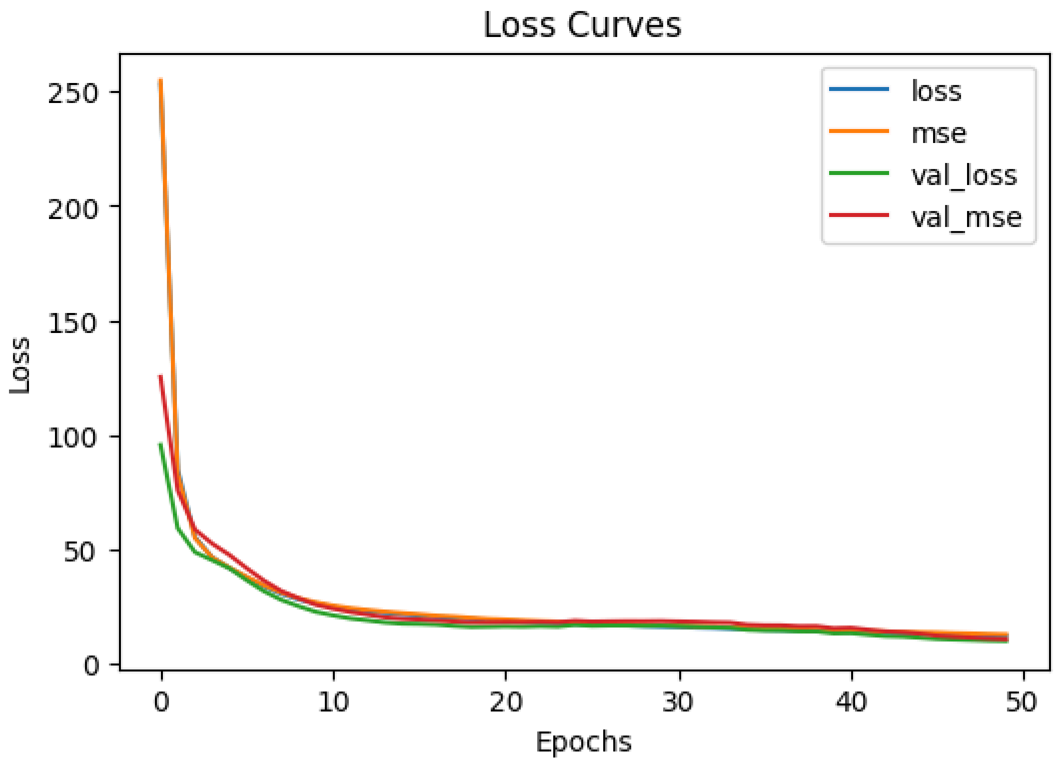
In [75]:
```
# Evaluate the model on the test data
print("Evaluation on Test data \n")
loss, mse = model.evaluate(X_test, y_test, batch_size=32)
print(f"\nModel loss on test set: {loss}")
print(f"Model mean squared error on test set: {(mse):.2f}")
```

```
Evaluation on Test data

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 13.4026 - mse: 14.2381

Model loss on test set: 12.777583122253418
Model mean squared error on test set: 14.87
```

In [76]:
```
# Plot the loss curves
pd.DataFrame(history.history).plot(figsize=(6, 4), xlabel="Epochs", ylabel="Loss
plt.show()
```



Model Prediction

In [77]:
```
# Make predictions
y_pred = model.predict(X_test)

# View the first prediction
y_pred[0]
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```

Out[77]:
```
array([19.273684], dtype=float32)
```