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

```
# 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

```
(X_train , y_train), (X_test , y_test) = tf.keras.datasets.boston_housing.load_data(
                                            path = 'boston_housing_npz',
                                            test_split = 0.2,
                                            seed = 42
                                        )
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/boston_housing.npz
57026/57026 [==============================] - 0s 0us/step
```

```
# Checking the data shape and type
(X_train.shape, type(X_train)), (X_test.shape, type(X_test)), (y_train.shape, type(y_train)), (y_test.shape, type(y_test)),
```

```
(((404, 13), numpy.ndarray),
 ((102, 13), numpy.ndarray),
 ((404,), numpy.ndarray),
 ((102,), numpy.ndarray))
```

```
# 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)
```

	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

```
# 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
```

```
X_train_df.describe()
```

	0	1	2	3	4	5	
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.00000
mean	3.789989	11.568069	11.214059	0.069307	0.554524	6.284824	69.11930
std	9.132761	24.269648	6.925462	0.254290	0.116408	0.723759	28.03460
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.90000
25%	0.081960	0.000000	5.190000	0.000000	0.452000	5.878750	45.47500
50%	0.262660	0.000000	9.690000	0.000000	0.538000	6.210000	77.50000
75%	3.717875	12.500000	18.100000	0.000000	0.624000	6.620500	94.42500
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.00000

Preprocessing

```
# 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()
```

	0	1	2	3	4	5	
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.00000
mean	0.042528	0.115681	0.394210	0.348815	0.521905	0.681970	0.24161
std	0.102650	0.242696	0.253866	0.239522	0.138678	0.288719	0.19497
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000850	0.000000	0.173387	0.137860	0.444098	0.438466	0.08736
50%	0.002881	0.000000	0.338343	0.314815	0.507569	0.768280	0.18476
75%	0.041717	0.125000	0.646628	0.491770	0.586223	0.942585	0.36225
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

Model, Predict, Evaluation

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=42)
X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((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

```
# 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[1],), name='Dense_1'),
  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)
)
```

```
Epoch 1/50
12/12 [==============================] - 1s 23ms/step - loss: 297.5028 - mse: 297.5028 - val_loss: 164.8920 - val_mse: 164.8920
Epoch 2/50
12/12 [==============================] - 0s 5ms/step - loss: 113.7968 - mse: 113.7968 - val_loss: 116.0087 - val_mse: 116.0087
Epoch 3/50
12/12 [==============================] - 0s 7ms/step - loss: 81.1189 - mse: 81.1189 - val_loss: 86.1455 - val_mse: 86.1455
Epoch 4/50
12/12 [==============================] - 0s 7ms/step - loss: 61.2206 - mse: 61.2206 - val_loss: 73.4338 - val_mse: 73.4338
Epoch 5/50
12/12 [==============================] - 0s 6ms/step - loss: 57.2634 - mse: 57.2634 - val_loss: 85.4577 - val_mse: 85.4577
Epoch 6/50
12/12 [==============================] - 0s 5ms/step - loss: 47.9021 - mse: 47.9021 - val_loss: 73.2840 - val_mse: 73.2840
Epoch 7/50
12/12 [==============================] - 0s 5ms/step - loss: 41.9481 - mse: 41.9481 - val_loss: 45.9927 - val_mse: 45.9927
Epoch 8/50
12/12 [==============================] - 0s 6ms/step - loss: 37.5299 - mse: 37.5299 - val_loss: 68.4146 - val_mse: 68.4146
Epoch 9/50
12/12 [==============================] - 0s 7ms/step - loss: 35.5962 - mse: 35.5962 - val_loss: 36.2395 - val_mse: 36.2395
Epoch 10/50
12/12 [==============================] - 0s 5ms/step - loss: 35.3156 - mse: 35.3156 - val_loss: 48.3317 - val_mse: 48.3317
Epoch 11/50
12/12 [==============================] - 0s 6ms/step - loss: 28.4119 - mse: 28.4119 - val_loss: 40.7898 - val_mse: 40.7898
Epoch 12/50
12/12 [==============================] - 0s 5ms/step - loss: 30.3630 - mse: 30.3630 - val_loss: 36.5521 - val_mse: 36.5521
Epoch 13/50
12/12 [==============================] - 0s 7ms/step - loss: 26.1323 - mse: 26.1323 - val_loss: 40.7216 - val_mse: 40.7216
Epoch 14/50
12/12 [==============================] - 0s 7ms/step - loss: 25.8230 - mse: 25.8230 - val_loss: 22.9798 - val_mse: 22.9798
Epoch 15/50
12/12 [==============================] - 0s 5ms/step - loss: 24.5607 - mse: 24.5607 - val_loss: 21.6948 - val_mse: 21.6948
Epoch 16/50
12/12 [==============================] - 0s 5ms/step - loss: 22.1658 - mse: 22.1658 - val_loss: 21.6587 - val_mse: 21.6587
Epoch 17/50
12/12 [==============================] - 0s 6ms/step - loss: 23.5799 - mse: 23.5799 - val_loss: 42.0520 - val_mse: 42.0520
Epoch 18/50
12/12 [==============================] - 0s 6ms/step - loss: 24.2861 - mse: 24.2861 - val_loss: 19.8561 - val_mse: 19.8561
Epoch 19/50
12/12 [==============================] - 0s 6ms/step - loss: 19.8144 - mse: 19.8144 - val_loss: 17.7707 - val_mse: 17.7707
Epoch 20/50
12/12 [==============================] - 0s 6ms/step - loss: 21.0086 - mse: 21.0086 - val_loss: 28.6501 - val_mse: 28.6501
Epoch 21/50
12/12 [==============================] - 0s 7ms/step - loss: 22.7832 - mse: 22.7832 - val_loss: 20.7365 - val_mse: 20.7365
Epoch 22/50
12/12 [==============================] - 0s 7ms/step - loss: 18.8117 - mse: 18.8117 - val_loss: 54.9769 - val_mse: 54.9769
Epoch 23/50
12/12 [==============================] - 0s 5ms/step - loss: 21.4513 - mse: 21.4513 - val_loss: 15.6439 - val_mse: 15.6439
Epoch 24/50
12/12 [==============================] - 0s 5ms/step - loss: 20.6609 - mse: 20.6609 - val_loss: 16.7432 - val_mse: 16.7432
Epoch 25/50
12/12 [==============================] - 0s 5ms/step - loss: 19.0440 - mse: 19.0440 - val_loss: 33.0360 - val_mse: 33.0360
Epoch 26/50
12/12 [==============================] - 0s 6ms/step - loss: 19.6381 - mse: 19.6381 - val_loss: 40.0757 - val_mse: 40.0757
Epoch 27/50
12/12 [==============================] - 0s 6ms/step - loss: 17.8266 - mse: 17.8266 - val_loss: 29.0757 - val_mse: 29.0757
```

```
Epoch 28/50
12/12 [==============================] - 0s 5ms/step - loss: 18.8474 - mse: 18.8474 - val_loss: 14.5323 - val_mse: 14.5323
Epoch 29/50
12/12 [==============================] - 0s 7ms/step - loss: 19.4269 - mse: 19.4269 - val_loss: 31.8526 - val_mse: 31.8526
```

Model Evaluation

```
# Preview the mean value of training and validation data
y_train.mean(), y_val.mean()
```

```
(22.235537, 24.89756)
```

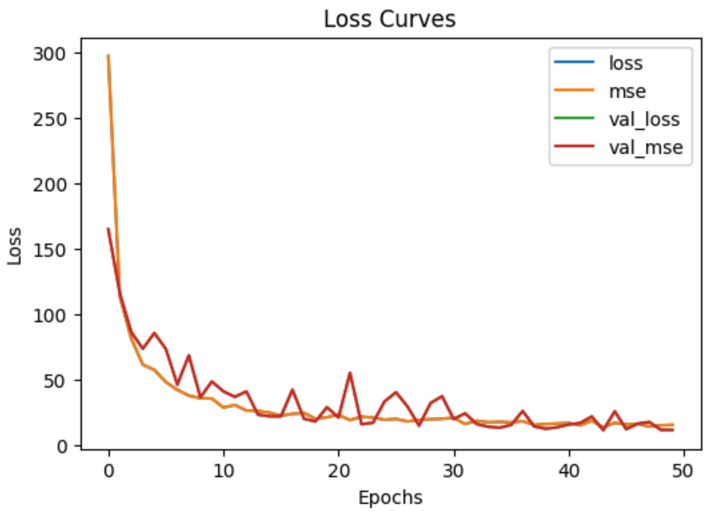
```
# Evaluate the model on the test data
print("Evaluation on Test data \n")
loss, mse = model.evaluate(X_test, y_test, batch_size=32)
print(f"\nModel loss on test set: {loss}")
print(f"Model mean squared error on test set: {(mse):.2f}")
```

```
Evaluation on Test data

4/4 [==============================] - 0s 3ms/step - loss: 14.6317 - mse: 14.6317

Model loss on test set: 14.631721496582031
Model mean squared error on test set: 14.63
```

```
# Plot the loss curves
pd.DataFrame(history.history).plot(figsize=(6, 4), xlabel="Epochs", ylabel="Loss", title='Loss Curves')
plt.show()
```



Model Prediction

```
# Make predictions
y_pred = model.predict(X_test)

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

```
4/4 [==============================] - 0s 3ms/step
array([21.119247], dtype=float32)
```