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.

```
# 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
```

Exploratory Data Analysis

Initial Observation

```
# 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

Next steps: Generate code with X_train_df | View recommended plots | New interactive sheet

```
# 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
```

```
# distribution of numerical feature values across the samples
X_train_df.describe()
```

	0	1	2	3	4	5	6	7	8	9	10	
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	40
mean	3.789989	11.568069	11.214059	0.069307	0.554524	6.284824	69.119307	3.792258	9.660891	408.960396	18.481931	35
std	9.132761	24.269648	6.925462	0.254290	0.116408	0.723759	28.034606	2.142651	8.736073	169.685166	2.157322	9
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.137000	1.000000	187.000000	12.600000	
25%	0.081960	0.000000	5.190000	0.000000	0.452000	5.878750	45.475000	2.097050	4.000000	281.000000	17.400000	37
50%	0.262660	0.000000	9.690000	0.000000	0.538000	6.210000	77.500000	3.167500	5.000000	330.000000	19.100000	39
75%	3.717875	12.500000	18.100000	0.000000	0.624000	6.620500	94.425000	5.118000	24.000000	666.000000	20.200000	39
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	39

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	6	7	8	9	10	
count	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	40
mean	0.042528	0.115681	0.394210	0.348815	0.521905	0.681970	0.241618	0.376560	0.423589	0.625737	0.897607	
std	0.102650	0.242696	0.253866	0.239522	0.138678	0.288719	0.194973	0.379829	0.323827	0.229502	0.232131	
min	0.000000	0.000000	0.000000	0.000000	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	0.130435	0.179389	0.510638	0.944992	
50%	0.002881	0.000000	0.338343	0.314815	0.507569	0.768280	0.184767	0.173913	0.272901	0.691489	0.985892	
75%	0.041717	0.125000	0.646628	0.491770	0.586223	0.942585	0.362255	1.000000	0.914122	0.808511	0.997252	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Model, Predict, Evaluation

```
# Reserve data for validation
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)

# Build the model
model = tf.keras.Sequential([
    tf.keras.Input(shape=(X_train.shape[1],), name='Input'),
    tf.keras.layers.Dense(10, activation='relu', name='Dense_1'),
    tf.keras.layers.Dense(100, activation='relu', name='Dense_2'),
    tf.keras.layers.Dense(1, name='Prediction')
])

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

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

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.6700 - mse: 21.6700 - val_loss: 21.1032 - val_mse: 21.1032
Epoch 30/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.6993 - mse: 21.6993 - val_loss: 21.4041 - val_mse: 21.4041
Epoch 31/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 21.5231 - mse: 21.5231 - val_loss: 21.6505 - val_mse: 21.6505
Epoch 32/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.3916 - mse: 21.3916 - val_loss: 21.9054 - val_mse: 21.9054
Epoch 33/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.1697 - mse: 21.1697 - val_loss: 22.1071 - val_mse: 22.1071
Epoch 34/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 21.0834 - mse: 21.0834 - val_loss: 22.3635 - val_mse: 22.3635
Epoch 35/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.9932 - mse: 20.9932 - val_loss: 22.1895 - val_mse: 22.1895
Epoch 36/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.7861 - mse: 20.7861 - val_loss: 22.3764 - val_mse: 22.3764
Epoch 37/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.7359 - mse: 20.7359 - val_loss: 22.2865 - val_mse: 22.2865
Epoch 38/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.5877 - mse: 20.5877 - val_loss: 22.3558 - val_mse: 22.3558
Epoch 39/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.5014 - mse: 20.5014 - val_loss: 22.3156 - val_mse: 22.3156
Epoch 40/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.3520 - mse: 20.3520 - val_loss: 22.5407 - val_mse: 22.5407
Epoch 41/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.3446 - mse: 20.3446 - val_loss: 22.3134 - val_mse: 22.3134
Epoch 42/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.1078 - mse: 20.1078 - val_loss: 22.3771 - val_mse: 22.3771
Epoch 43/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 20.0508 - mse: 20.0508 - val_loss: 22.0095 - val_mse: 22.0095
Epoch 44/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.9031 - mse: 19.9031 - val_loss: 21.8509 - val_mse: 21.8509
Epoch 45/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.7949 - mse: 19.7949 - val_loss: 21.5021 - val_mse: 21.5021
Epoch 46/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.6206 - mse: 19.6206 - val_loss: 21.8219 - val_mse: 21.8219
Epoch 47/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 19.6987 - mse: 19.6987 - val_loss: 21.2313 - val_mse: 21.2313
Epoch 48/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.4991 - mse: 19.4991 - val_loss: 21.1060 - val_mse: 21.1060
Epoch 49/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 19.3398 - mse: 19.3398 - val_loss: 21.1402 - val_mse: 21.1402
Epoch 50/50
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 19.2481 - mse: 19.2481 - val_loss: 21.2756 - val_mse: 21.2756
```

Model Evaluation

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

```
(np.float32(22.235537), np.float32(24.89756))
```

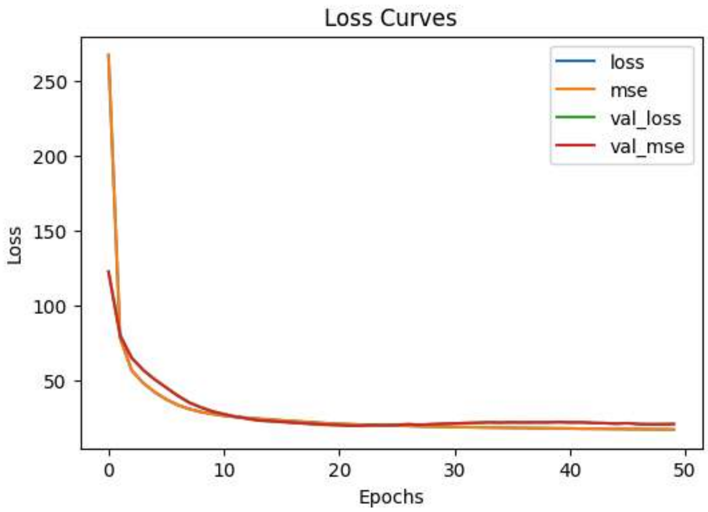
```python
# Evaluate the model on the test data
print("Evaluation on Test data \n")
loss, mse = model.evaluate(X_test, y_test, batch_size=32)
print(f"\nModel loss on test set: {loss}")
print(f"Model mean squared error on test set: {(mse):.2f}")
```

```
Evaluation on Test data

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - loss: 24.4305 - mse: 24.4305

Model loss on test set: 26.986352920532227
Model mean squared error on test set: 26.99
```

```python
# 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 46ms/step
array([21.049059], dtype=float32)
```