

# Import Library

```
In [1]: # 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 [2]: (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
    )
```

# Exploratory Data Analysis

## Initial Observation

```
In [3]: # 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))
```

```
Out[3]: (((404, 13), numpy.ndarray),
          ((102, 13), numpy.ndarray),
          ((404,), numpy.ndarray),
          ((102,), numpy.ndarray))
```

```
In [4]: # 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[4]:
```

	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 [5]: # 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 [6]: # distribution of numerical feature values across the samples
X_train_df.describe()
```

```
Out[6]:
```

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

## Preprocessing

```
In [7]: # 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[7]:

	0	1	2	3	4	5	6	7	8	9
<b>count</b>	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000	404.000000
<b>mean</b>	0.042528	0.115681	0.394210	0.348815	0.521905	0.681970	0.241618	0.376560	0.423589	0.625737
<b>std</b>	0.102650	0.242696	0.253866	0.239522	0.138678	0.288719	0.194973	0.379829	0.323827	0.229502
<b>min</b>	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
<b>25%</b>	0.000850	0.000000	0.173387	0.137860	0.444098	0.438466	0.087361	0.130435	0.179389	0.510638
<b>50%</b>	0.002881	0.000000	0.338343	0.314815	0.507569	0.768280	0.184767	0.173913	0.272901	0.691489
<b>75%</b>	0.041717	0.125000	0.646628	0.491770	0.586223	0.942585	0.362255	1.000000	0.914122	0.808511
<b>max</b>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

## Model, Predict, Evaluation

In [8]: `# 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`

Out[8]: ((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 [9]: `# 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)`  
`)`

```

12/12 [=====] - 0s 3ms/step - loss: 13.7454 - mse: 13.7454 - val_loss: 15.7261 - val_
mse: 15.7261
Epoch 49/50
12/12 [=====] - 0s 4ms/step - loss: 14.7645 - mse: 14.7645 - val_loss: 9.9898 - val_m
se: 9.9898
Epoch 50/50
12/12 [=====] - 0s 3ms/step - loss: 15.1951 - mse: 15.1951 - val_loss: 11.3679 - val_
mse: 11.3679

```

## Model Evaluation

```

In [10]: # Preview the mean value of training and validation data
y_train.mean(), y_val.mean()

```

```

Out[10]: (22.235537, 24.89756)

```

```

In [11]: # 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 2ms/step - loss: 15.5170 - mse: 15.5170

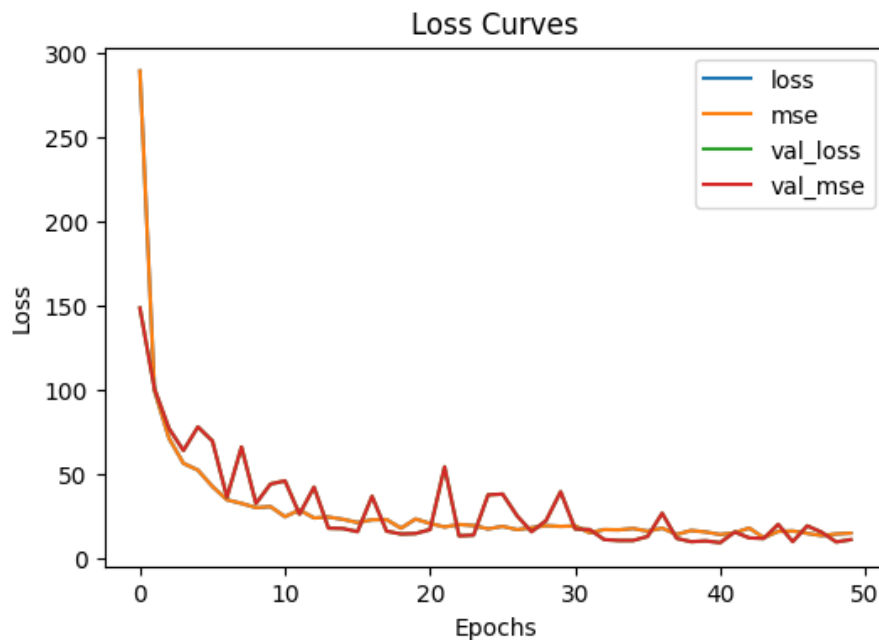
```

Model loss on test set: 15.517014503479004  
Model mean squared error on test set: 15.52

```

In [12]: # Plot the Loss curves
pd.DataFrame(history.history).plot(figsize=(6, 4), xlabel="Epochs", ylabel="Loss", title='Loss Curves')
plt.show()

```



## Model Prediction

```

In [13]: # Make predictions
y_pred = model.predict(X_test)

# View the first prediction
y_pred[0]

```

```

4/4 [=====] - 0s 1ms/step

```

```

Out[13]: array([19.093914], dtype=float32)

```