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

from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler
```

```
(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
 )
```

```
# 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)
```

```
# View summary of datasets
X_train_df.info()
```

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

# 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()

# 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


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

# 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}")

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

```
# Make predictions
y_pred = model.predict(X_test)
# View the first prediction
y_pred[0]
```