# 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

(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

)

# 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)),

# 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()

print('\_'\*40)

y\_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]