import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

# Load the data

df = pd.read\_csv('C:\\Users\\muthu vignesh\\Desktop\\Tire\_Force\_Data.csv')

# Preprocess the data

scaler\_X\_Fx = MinMaxScaler()

scaler\_y\_Fx = MinMaxScaler()

scaler\_X\_Fy = MinMaxScaler()

scaler\_y\_Fy = MinMaxScaler()

scaled\_X\_Fx = scaler\_X\_Fx.fit\_transform(df[['Fx\_L1', 'Fx\_Brush']])

scaled\_y\_Fx = scaler\_y\_Fx.fit\_transform(df[['Fx\_Real']])

scaled\_X\_Fy = scaler\_X\_Fy.fit\_transform(df[['Fy\_L1', 'Fy\_Brush']])

scaled\_y\_Fy = scaler\_y\_Fy.fit\_transform(df[['Fy\_Real']])

# Split the data into inputs (X) and target/output (y)

X\_Fx = scaled\_X\_Fx

y\_Fx = scaled\_y\_Fx

X\_Fy = scaled\_X\_Fy

y\_Fy = scaled\_y\_Fy

# Split the data into training and test sets

X\_train\_Fx, X\_test\_Fx, y\_train\_Fx, y\_test\_Fx = train\_test\_split(X\_Fx, y\_Fx, test\_size=0.2, random\_state=42)

X\_train\_Fy, X\_test\_Fy, y\_train\_Fy, y\_test\_Fy = train\_test\_split(X\_Fy, y\_Fy, test\_size=0.2, random\_state=42)

# Define the model

model\_Fx = tf.keras.models.Sequential([

tf.keras.layers.Dense(10, activation='relu'),

tf.keras.layers.Dense(10, activation='relu'),

tf.keras.layers.Dense(1)

])

model\_Fy = tf.keras.models.Sequential([

tf.keras.layers.Dense(10, activation='relu'),

tf.keras.layers.Dense(10, activation='relu'),

tf.keras.layers.Dense(1)

])

# Compile the model

model\_Fx.compile(loss='mean\_squared\_error', optimizer='adam')

model\_Fy.compile(loss='mean\_squared\_error', optimizer='adam')

# Train the model

model\_Fx.fit(X\_train\_Fx, y\_train\_Fx, epochs=50, batch\_size=32)

model\_Fy.fit(X\_train\_Fy, y\_train\_Fy, epochs=50, batch\_size=32)

# Use the model to make predictions

predictions\_Fx = model\_Fx.predict(X\_test\_Fx)

predictions\_Fy = model\_Fy.predict(X\_test\_Fy)

# Rescale the predictions

predictions\_Fx = scaler\_y\_Fx.inverse\_transform(predictions\_Fx).flatten()

predictions\_Fy = scaler\_y\_Fy.inverse\_transform(predictions\_Fy).flatten()

# Rescale the actual values

actual\_Fx = scaler\_y\_Fx.inverse\_transform(y\_test\_Fx).flatten()

actual\_Fy = scaler\_y\_Fy.inverse\_transform(y\_test\_Fy).flatten()

# Calculate accuracy for Fx

accuracies\_Fx = [max(0, (1 - abs(actual - pred) / actual)) \* 100 for actual, pred in zip(actual\_Fx, predictions\_Fx)]

average\_accuracy\_Fx = sum(accuracies\_Fx) / len(accuracies\_Fx)

# Calculate accuracy for Fy

accuracies\_Fy = [max(0, (1 - abs(actual - pred) / actual)) \* 100 for actual, pred in zip(actual\_Fy, predictions\_Fy)]

average\_accuracy\_Fy = sum(accuracies\_Fy) / len(accuracies\_Fy)

print("Average Accuracy for Fx:", average\_accuracy\_Fx)

print("Average Accuracy for Fy:", average\_accuracy\_Fy)

# Create a DataFrame for the predicted values

df\_pred\_Fx = pd.DataFrame(predictions\_Fx, columns=['Predicted\_Fx\_real'])

df\_pred\_Fy = pd.DataFrame(predictions\_Fy, columns=['Predicted\_Fy\_real'])

# Create a DataFrame for the actual values

df\_actual\_Fx = pd.DataFrame(actual\_Fx, columns=['Actual\_Fx\_real'])

df\_actual\_Fy = pd.DataFrame(actual\_Fy, columns=['Actual\_Fy\_real'])

# Concatenate the 'Fx' and 'Fy' dataframes along the column axis

df\_comparison = pd.concat([df\_actual\_Fx, df\_pred\_Fx, df\_actual\_Fy, df\_pred\_Fy], axis=1)

# Save the comparison dataframe to a csv file

df\_comparison.to\_csv('C:\\Users\\muthu vignesh\\Desktop\\comparison.csv', index=False)