Explaination :

**Install Required Libraries**

!pip install pandas numpy scikit-learn tensorflow flask matplotlib joblib

This command installs Python libraries needed:

* pandas, numpy: For data handling.
* scikit-learn: For preprocessing and metrics.
* tensorflow: To build LSTM model.
* flask: For deploying the model (used later).
* matplotlib: For plotting graphs.
* joblib: To save models (used typically for non-deep learning models).

**🧾 Load Training and Testing Datasets**

import pandas as pd

train\_data = pd.read\_csv(r'D:\Aaknaksha\AG2295 vaishnavi\merged training set.csv')

test\_data = pd.read\_csv(r'D:\Aaknaksha\AG2295 vaishnavi\merged testing set.csv')

You are loading your **merged training and testing datasets** using pandas from your system drive.

print("Training Data:")

print(train\_data.head())

print("\nTesting Data:")

print(test\_data.head())

This prints the **first 5 rows** of both datasets to verify they loaded correctly.

**🧹 Clean Column Names**

train\_data.columns = train\_data.columns.str.strip()

test\_data.columns = test\_data.columns.str.strip()

Sometimes column names have **extra spaces**. This line removes those to prevent errors later.

test\_data.rename(columns={'average\_temperature((°C)\_x': 'average\_temperature(°C)\_x'}, inplace=True)

train\_data.rename(columns={'Incident\_Count ': 'Incident\_Count'}, inplace=True)

These two lines **fix specific wrong column names** due to typos or formatting issues.

print("Updated Training Data Columns:")

print(train\_data.columns)

print("\nUpdated Testing Data Columns:")

print(test\_data.columns)

This confirms that the **column names are now correct**.

**📄 Load Preprocessed Data**

file\_path = r'D:\Aaknaksha\AG2295 vaishnavi\preprocessed\_simulated\_road\_data.csv'

data = pd.read\_csv(file\_path)

print(data.head())

Loads the **preprocessed road dataset** and prints the first few rows to verify.

**🔠 Encode Categorical Values**

preprocessed\_data = pd.read\_csv('D:/Aaknaksha/AG2295 vaishnavi/final\_preprocessed\_road\_data.csv')

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

preprocessed\_data['weather\_condition\_x'] = label\_encoder.fit\_transform(preprocessed\_data['weather\_condition\_x'])

print(preprocessed\_data[['weather\_condition\_x']].head())

The column weather\_condition\_x contains text (like "Rainy", "Sunny"). LabelEncoder converts these **to numbers** so models can use them.

**⚖️ Normalize Numerical Columns**

from sklearn.preprocessing import MinMaxScaler

This scaler scales values to a range of **0 to 1**, which is good for models like LSTM.

numerical\_columns = ['Cars', 'Trucks', 'Motorcycles', 'Buses', ...]

scaler = MinMaxScaler(feature\_range=(0, 1))

preprocessed\_data[numerical\_columns] = scaler.fit\_transform(preprocessed\_data[numerical\_columns])

All selected numerical columns are scaled using **Min-Max scaling**. This helps the model train faster and better.

**🔀 Split Data into Train and Test Sets**

X = preprocessed\_data.drop('Wear\_Index', axis=1)

y = preprocessed\_data['Wear\_Index']

* X: All input features (everything except the target).
* y: The output (what we want to predict) — road wear.

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

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

Splits data:

* 80% for training,
* 20% for testing.  
  random\_state=42 ensures the same split every time.

print("Training data shape:", X\_train.shape)

print("Testing data shape:", X\_test.shape)

This confirms the sizes of both sets.

**✅ Encode Again (Precaution)**

categorical\_columns = ['weather\_condition\_x']

label\_encoder = LabelEncoder()

for col in categorical\_columns:

preprocessed\_data[col] = label\_encoder.fit\_transform(preprocessed\_data[col])

You re-encode the categorical column again — though this is already done earlier. It’s safe but **redundant** in this context.

**✅ Normalize Again (Precaution)**

scaler = MinMaxScaler(feature\_range=(0, 1))

preprocessed\_data[numerical\_columns] = scaler.fit\_transform(preprocessed\_data[numerical\_columns])

print(preprocessed\_data.head())

Again you re-normalize, which is **repeated** but ensures consistency before modeling.

**🔄 Reshape Data for LSTM**

X = preprocessed\_data.drop('Wear\_Index', axis=1)

y = preprocessed\_data['Wear\_Index']

Just like before: defining features and target.

X\_lstm = X.values.astype(np.float32).reshape(X.shape[0], 1, X.shape[1])

LSTM needs input in 3D:  
**[samples, timesteps, features]**  
Here:

* samples: number of rows,
* timesteps: 1 (single time point),
* features: number of columns.

print(f"Shape of LSTM input data: {X\_lstm.shape}")

Confirms the reshaped format.

**🤖 Build and Train LSTM Model**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

Import layers needed for the neural network.

lstm\_model = Sequential()

lstm\_model.add(LSTM(units=100, return\_sequences=False, input\_shape=(X\_lstm.shape[1], X\_lstm.shape[2])))

lstm\_model.add(Dropout(0.2))

lstm\_model.add(Dense(units=1))

lstm\_model.compile(optimizer='adam', loss='mean\_squared\_error')

**Explanation of layers:**

* LSTM(100): LSTM layer with 100 memory units.
* Dropout(0.2): Drops 20% of neurons randomly to avoid overfitting.
* Dense(1): Final output layer gives a single road wear prediction.
* Compiled with:
  + Adam optimizer (good for deep learning),
  + mean\_squared\_error loss (used in regression).

lstm\_model.fit(X\_lstm, y, epochs=50, batch\_size=64)

Trains the model for 50 rounds (epochs) using 64 rows at a time.

**💾 Save and Predict**

lstm\_model.save('road\_wear\_lstm\_model.h5')

Saves the trained model to a file.

lstm\_predictions = lstm\_model.predict(X\_lstm)

Predicts the road wear values on the same input data (you can later use test data too).

**📉 Calculate Model Accuracy**

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y, lstm\_predictions)

print(f"Mean Absolute Error (LSTM): {mae}")

MAE measures how far predictions are from true values (on average).  
Here, it’s about **545.79**.

**📊 Plot the Results**

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

plt.plot(y.values, label="True Road Wear")

plt.plot(lstm\_predictions, label="Predicted Road Wear", linestyle="--")

plt.legend()

plt.title("True vs Predicted Road Wear (LSTM Model)")

plt.xlabel("Test Samples")

plt.ylabel("Road Wear Index")

plt.show()

This plot:

* Shows the **actual vs predicted** values.
* Helps visually understand model accuracy.
* Dashed line = prediction, solid = actual.

**. Label Encoding Categorical Data**

from sklearn.preprocessing import LabelEncoder

categorical\_columns = ['weather\_condition\_x']

label\_encoder = LabelEncoder()

for col in categorical\_columns:

preprocessed\_data[col] = label\_encoder.fit\_transform(preprocessed\_data[col])

**✅ What this does:**

* LabelEncoder() converts **text labels** like "Rainy", "Clear skies" into **numbers** like 0, 1, 2.
* You're applying this transformation to the column weather\_condition\_x.

**🖨️ Output example:**

weather\_condition\_x

0 3

1 1

2 0

3 0

4 3

This means:

* "Stormy" → 0
* "Rainy" → 1
* "Cloudy" → 2
* "Clear skies" → 3  
  (Exact mapping depends on label order.)

**📌 2. Converting Columns to Numeric**

numerical\_columns = ['Cars', 'Trucks', 'Motorcycles', 'Buses', 'Car\_Weight(kg)', 'Truck\_Weight(kg)',

'Motorcycle\_Weight(kg)', 'Bus\_Weight(kg)', 'Avg\_Speed(m/s)', 'Avg\_Weight(kg)',

'Incident\_Count', 'Surface Roughness (m/km)', 'Road Age (years)', 'temperature(°C)\_x',

'rainfall(mm)\_x', 'humidity(%)\_x', 'wind\_speed(m/s)\_x', 'average\_temperature(°C)\_x',

'weather\_condition\_x']

preprocessed\_data[numerical\_columns] = preprocessed\_data[numerical\_columns].apply(pd.to\_numeric, errors='coerce')

**✅ What this does:**

* Ensures all these columns are numeric (integers or floats).
* If any value can’t be converted, it's replaced with NaN (missing).

**📌 3. Normalizing Values (0 to 1)**

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature\_range=(0, 1))

preprocessed\_data[numerical\_columns] = scaler.fit\_transform(preprocessed\_data[numerical\_columns])

**✅ What this does:**

* Scales all numbers between 0 and 1.
* Example: if speed varies from 10 to 100, it will now vary from 0.0 to 1.0.

**📌 4. Train-Test Split for GBM**

from sklearn.ensemble import GradientBoostingRegressor

X = preprocessed\_data.drop('Wear\_Index', axis=1)

y = preprocessed\_data['Wear\_Index']

* X = All columns **except** the target (Wear\_Index)
* y = Target column (what you want to predict)

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

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

**✅ What this does:**

* Splits data: 80% training, 20% testing.
* random\_state=42 makes the split **reproducible**.

**📌 5. Train the Gradient Boosting Model**

gbm\_model = GradientBoostingRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=3)

gbm\_model.fit(X\_train, y\_train)

* n\_estimators=100 → 100 small decision trees
* learning\_rate=0.1 → how fast the model learns
* max\_depth=3 → how deep each tree is

**📌 6. GBM Predictions and Error**

gbm\_predictions = gbm\_model.predict(X\_test)

from sklearn.metrics import mean\_absolute\_error

mae\_gbm = mean\_absolute\_error(y\_test, gbm\_predictions)

print(f"Mean Absolute Error (GBM): {mae\_gbm}")

* mae\_gbm = Average error in GBM predictions

**🖨️ Output:**

Mean Absolute Error (GBM): 8.171820915849024

**📌 7. LSTM Model Input Reshaping**

X\_test\_lstm = X\_test.values.astype(np.float32).reshape(X\_test.shape[0], 1, X\_test.shape[1])

print(f"Shape of X\_test\_lstm: {X\_test\_lstm.shape}")

* LSTM requires 3D input: (samples, timesteps, features)
* If you have 200 rows and 19 features:

**🖨️ Output:**

Shape of X\_test\_lstm: (200, 1, 19)

**📌 8. LSTM Predictions and Flattening**

lstm\_predictions = lstm\_model.predict(X\_test\_lstm)

gbm\_predictions = gbm\_model.predict(X\_test)

lstm\_predictions = lstm\_predictions.flatten()

gbm\_predictions = gbm\_predictions.flatten()

* Ensures both predictions are 1D arrays.

**🖨️ Output:**

Shape of LSTM predictions: (200,)

Shape of GBM predictions: (200,)

**📌 9. Hybrid Predictions (Average and Weighted)**

combined\_predictions = (lstm\_predictions + gbm\_predictions) / 2

mae\_combined = mean\_absolute\_error(y\_test, combined\_predictions)

print(f"Mean Absolute Error (Hybrid Model): {mae\_combined}")

**🖨️ Output:**

Mean Absolute Error (Hybrid Model): 264.47822558301556

⚠️ **Warning**: This is a high error! You may need to fix scaling, LSTM architecture, or weight balancing.

combined\_predictions = (0.6 \* lstm\_predictions + 0.4 \* gbm\_predictions)

* This gives **more importance to LSTM predictions**.

**📌 10. Save Trained Models**

lstm\_model.save('road\_wear\_lstm\_model.h5') # For Keras

import joblib

joblib.dump(gbm\_model, 'road\_wear\_gbm\_model.pkl') # For scikit-learn

* Saves your models so you don’t need to retrain them every time.

**📌 11. Save LabelEncoder and Scaler**

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

import joblib

import pandas as pd

train\_data = pd.DataFrame({

'weather\_condition\_x': ['Clear skies', 'Rainy', 'Stormy', 'Clear skies'],

'Cars': [100, 200, 150, 120],

'Trucks': [50, 70, 60, 55],

})

label\_encoder = LabelEncoder()

train\_data['weather\_condition\_x'] = label\_encoder.fit\_transform(train\_data['weather\_condition\_x'])

scaler = MinMaxScaler(feature\_range=(0, 1))

train\_data\_normalized = scaler.fit\_transform(train\_data[['Cars', 'Trucks']])

joblib.dump(label\_encoder, 'label\_encoder.pkl')

joblib.dump(scaler, 'scaler.pkl')

print("LabelEncoder and Scaler saved successfully.")

**✅ What this does:**

* Encodes weather condition
* Normalizes Cars and Trucks
* Saves both encoder and scaler to disk

**🖨️ Output:**

LabelEncoder and Scaler saved successfully.

**🖼️ Expected Image/Data Output**

Sample of processed DataFrame before model training:

| **Cars** | **Trucks** | **weather\_condition\_x** | **Wear\_Index** |
| --- | --- | --- | --- |
| 0.33 | 0.25 | 1 | 22.3 |
| 0.67 | 0.75 | 2 | 28.4 |
| 1.00 | 1.00 | 0 | 33.9 |
|  |  |  |  |
|  |  |  |  |