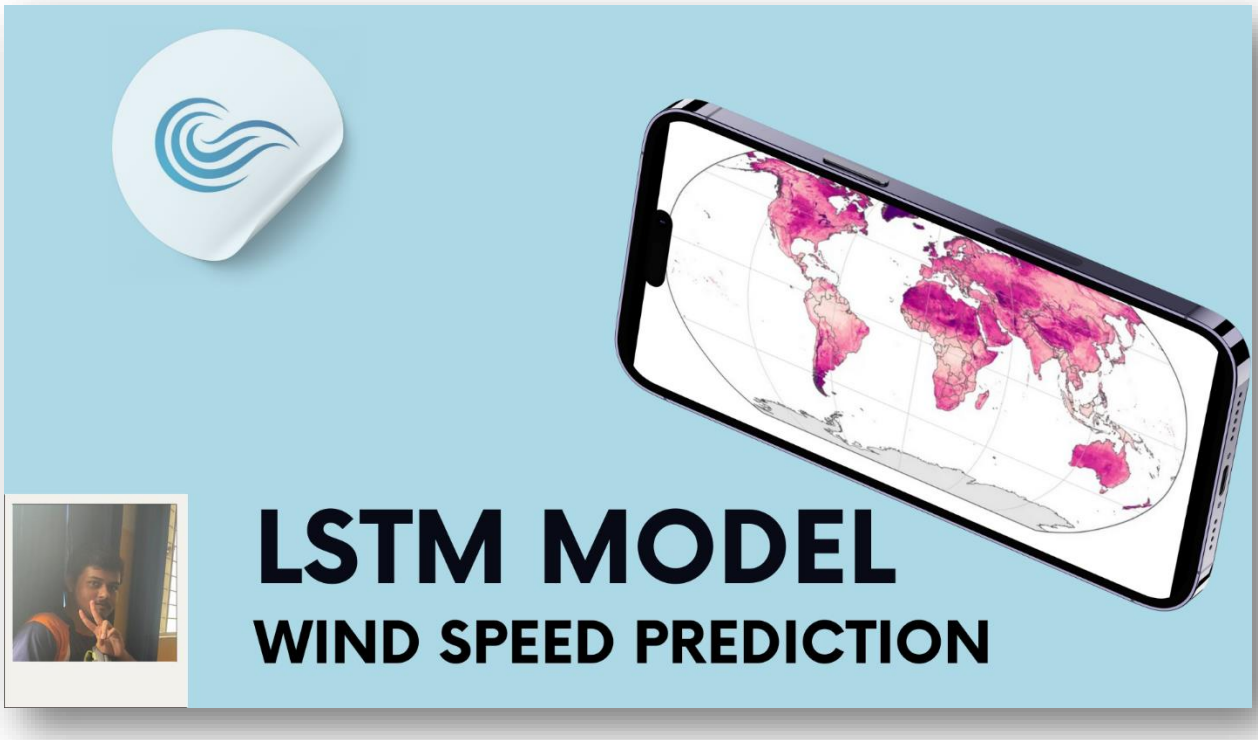
LSTM MODEL
WIND SPEED PREDICTION

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras import metrics
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
# Load data
df = pd.read_csv("wind-speed.csv")
L = len(df)
print(L)
# Prepare Y
Y = df.iloc[:, 3].values  # Convert to 1D array
plt.plot(Y)
plt.show(block=False)
# Create X1, X2, X3
X1 = Y[0:L-5]
X2 = Y[1:L-4]
X3 = Y[2:L-3]
# Stack and transpose X
X = np.vstack((X1, X2, X3)).T  # Stack vertically and transpose
# Prepare Y for LSTM
Y = Y[3:L-2]
# Scale X
sc = MinMaxScaler()  # No parameters
sc.fit(X)
X = sc.transform(X)
# Scale Y
sc1 = MinMaxScaler()
Y = Y.reshape(-1, 1)  # Reshape Y to be 2D
sc1.fit(Y)
Y = sc1.transform(Y)
# Reshape X for LSTM
X = np.reshape(X, (X.shape[0], 1, X.shape[1]))
# Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
# Define the LSTM model
model = Sequential()
model.add(LSTM(10, activation='tanh', input_shape=(1, 3),
recurrent_activation='hard_sigmoid'))
model.add(Dense(1))
# Compile the model
model.compile(loss="mean_squared_error", optimizer='rmsprop',
metrics=[metrics.mean_squared_error])
# Fit the model
model.fit(X_train, Y_train, epochs=25, verbose=2)
# Make predictions
predict = model.predict(X_test)
# Plot predictions
plt.figure(2)
```

```
plt.scatter(Y_test, predict)
plt.xlabel("Real Data")
plt.ylabel("Predicted Data")
plt.show(block=False)
plt.figure(3)
plt.plot(Y_test, label="Real Data")
plt.plot(predict, label="Predicted Data")
plt.legend()
plt.show()
```

Code Explanation

1. Importing Libraries

- **NumPy**: A library for numerical operations, particularly useful for handling arrays and matrices.
- **Pandas**: A data manipulation library that provides data structures like DataFrames for handling structured data.
- **Matplotlib**: A plotting library used for visualizing data.
- **Keras**: A high-level neural networks API that simplifies building and training deep learning models.
- **Scikit-learn**: A machine learning library that provides tools for data preprocessing and model evaluation.

2. Loading Data

- This code reads a CSV file named **wind-speed.csv** into a Pandas DataFrame called **df**.
- **len(df)** calculates the number of rows in the DataFrame, which is stored in the variable **L** and printed. This gives an idea of how many data points are available.

3. Preparing the Target Variable (Y)

- The code extracts the fourth column (index 3) from the DataFrame, which is assumed to contain wind speed data, and converts it to a 1D NumPy array called **Y**.
- The **plt.plot(Y)** function visualizes the wind speed data over time, and **plt.show(block=False)** displays the plot without blocking further code execution.

4. Creating Input Features (X1, X2, X3)

- This section creates three feature arrays:
  - **X1**: Contains values from the start of **Y** up to the fifth-to-last value.
  - **X2**: Contains values from the second value to the fourth-to-last value.
  - **X3**: Contains values from the third value to the third-to-last value.
- These arrays represent previous wind speed values used to predict the next value.

5. Stacking and Transposing Features

- The three feature arrays are stacked vertically using **np.vstack**, and then transposed to create a 2D array where each row corresponds to a time step with three features (previous wind speeds).

6. Preparing the Target Variable for LSTM

- The target variable **Y** is adjusted to align with the input features. The first three values are excluded because they do not have corresponding features.

7. Scaling the Data

- **MinMaxScaler**: This scales the data to a range between 0 and 1, which is important for training neural networks.
- The input features **X** and target variable **Y** are both scaled. **Y** is reshaped to be 2D before scaling, as **MinMaxScaler** expects a 2D array.

8. Reshaping X for LSTM

- The input data **X** is reshaped to fit the LSTM layer's expected 3D input format: (samples, time steps, features). Here, each sample has 1 time step and 3 features.

9. Train-Test Split

- The dataset is split into training and testing sets, with 20% of the data reserved for testing. This helps evaluate the model's performance on unseen data.

10. Defining the LSTM Model

- **Sequential Model**: This creates a linear stack of layers for the neural network.
- **LSTM Layer**:
    - The LSTM layer has 10 units (neurons) and uses the **tanh** activation function, which helps in learning complex patterns.
    - The **input_shape=(1, 3)** specifies that each input sample has 1 time step and 3 features.
    - The **recurrent_activation='hard_sigmoid'** is used for the internal gates of the LSTM.
- **Dense Layer**: This is a fully connected layer with 1 output unit, which will produce the predicted wind speed.

11. Compiling the Model

- **Loss Function**: The model uses mean squared error (MSE) as the loss function, which measures the average squared difference between predicted and actual values.
- **Optimizer**: The RMSprop optimizer is used to adjust the weights during training, helping to minimize the loss function.
- **Metrics**: The model will also track mean squared error as a metric during training.

12. Fitting the Model

- The model is trained on the training data (**X_train** and **Y_train**) for 25 epochs (iterations over the entire dataset).
- The **verbose=2** parameter provides detailed output during training, showing the loss and metrics for each epoch.

13. Making Predictions

- After training, the model is used to make predictions on the test set (**X_test**). The predicted values are stored in the **predict** variable.

14. Plotting Predictions

- **Scatter Plot**: The first plot shows a scatter plot of the actual values (**Y_test**) against the predicted values (**predict**). This helps visualize how well the model's predictions match the actual data.
- **Line Plot**: The second plot displays both the actual and predicted values over time, allowing for a visual comparison of the model's performance. The **label** parameter is used to create a legend for clarity.

Plot Explanation

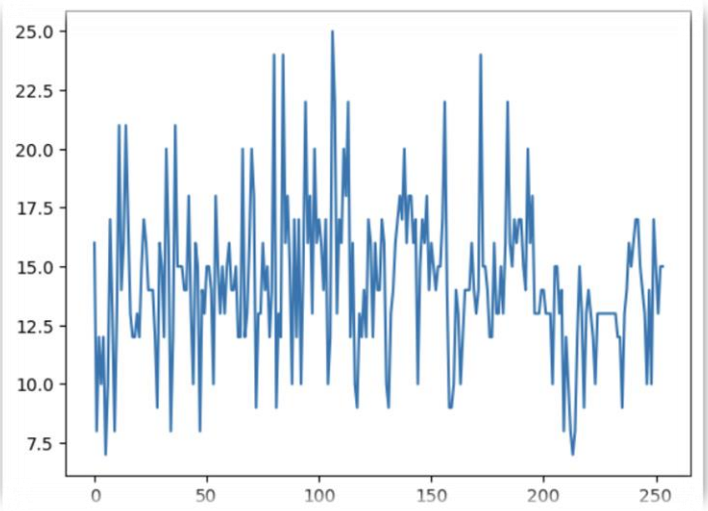

Fig - 1

1. **Wind Speed Plot**: This plot visualizes the historical wind speed data extracted from the CSV file, showing how wind speed varies over time. It provides an initial overview of the dataset, allowing for the identification of trends, patterns, or anomalies in the wind speed measurements.

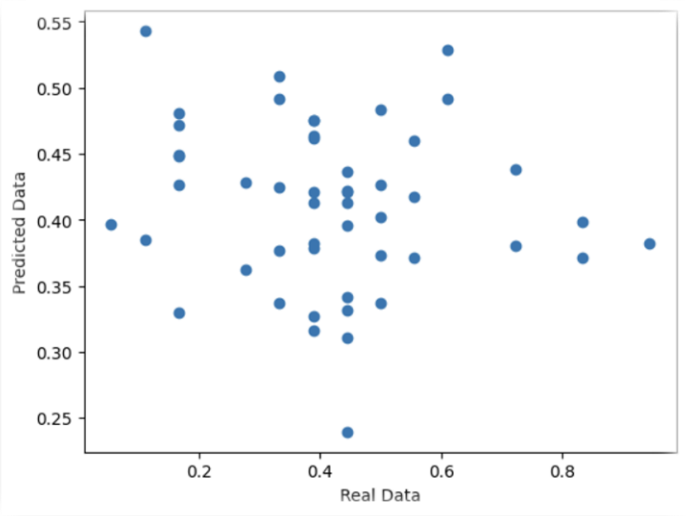

Fig – 2

2. **Scatter Plot of Predictions**: This scatter plot compares the actual wind speed values (**Y_test**) against the predicted values from the LSTM model (**predict**). Ideally, the points should cluster around the diagonal line (y = x), indicating that the model's predictions closely match the actual values, which reflects the model's accuracy.

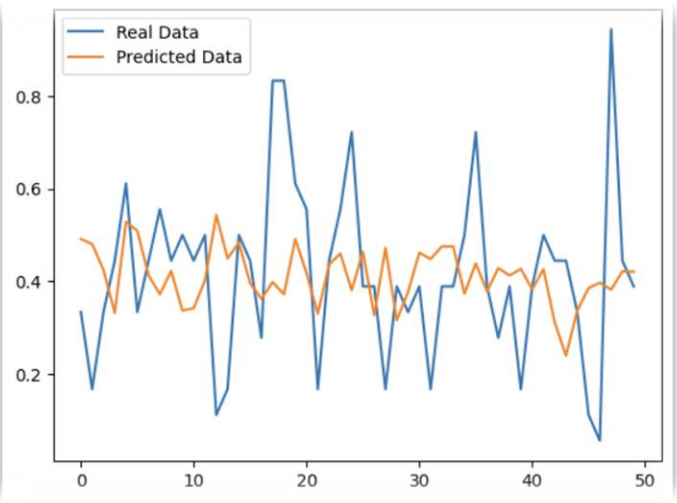

Fig – 3

3. **Line Plot of Real vs. Predicted Data**: This line plot overlays the actual wind speed values and the model's predictions over the same time period, allowing for a visual comparison of the two. It helps to assess how well the model captures the trends and fluctuations in the actual data, with a good fit indicated by the predicted line closely following the actual line.

Summary

This code implements a Long Short-Term Memory (LSTM) neural network to predict wind speed based on historical data. It involves several key steps:

- Importing necessary libraries.
- Loading and preparing the data.
- Creating input features and target variables.
- Scaling the data for better training performance.
- Defining, compiling, and training the LSTM model.
- Making predictions and visualizing the results.