

J.VANI SUMEDHA - 23MSD7018

## PROJECT TITLE: LSTM WITH WEATHER FORECASTING DATA


Report of the project explained below

### TASK-1:DATA PROCESSING

#### Step-1:Loading the Data

```
import pandas as pd
```

```
df=pd.read_csv("/content/Weather Data.csv")
df
```




	Date/Time	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa	Weather
0	1/1/2012 0:00	-1.8	-3.9	86	4	8.0	101.24	Fog
1	1/1/2012 1:00	-1.8	-3.7	87	4	8.0	101.24	Fog
2	1/1/2012 2:00	-1.8	-3.4	89	7	4.0	101.26	Freezing Drizzle,Fog
3	1/1/2012 3:00	-1.5	-3.2	88	6	4.0	101.27	Freezing Drizzle,Fog
4	1/1/2012 4:00	-1.5	-3.3	88	7	4.8	101.23	Fog
...	...	...	...	...	...	...	...	...
8779	12/31/2012 19:00	0.1	-2.7	81	30	9.7	100.13	Snow
8780	12/31/2012 20:00	0.2	-2.4	83	24	9.7	100.03	Snow
8781	12/31/2012 21:00	-0.5	-1.5	93	28	4.8	99.95	Snow
8782	12/31/2012 22:00	-0.2	-1.8	89	28	9.7	99.91	Snow
8783	12/31/2012 23:00	0.0	-2.1	86	30	11.3	99.89	Snow


8784 rows × 8 columns

#### Step-2:Understanding the Data

```
df.shape
```

 (8784, 8)

```
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Date/Time              8784 non-null  object
1   Temp_C                 8784 non-null  float64
2   Dew Point Temp_C       8784 non-null  float64
3   Rel Hum_%              8784 non-null  int64
4   Wind Speed_km/h        8784 non-null  int64
5   Visibility_km           8784 non-null  float64
6   Press_kPa              8784 non-null  float64
7   Weather                8784 non-null  object
dtypes: float64(4), int64(2), object(2)
memory usage: 549.1+ KB
```

```
df.describe()
```

	Temp_C	Dew Point	Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
<b>count</b>	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000
<b>mean</b>	8.798144	2.555294	67.431694	14.945469	27.664447	101.051623	
<b>std</b>	11.687883	10.883072	16.918881	8.688696	12.622688	0.844005	
<b>min</b>	-23.300000	-28.500000	18.000000	0.000000	0.200000	97.520000	
<b>25%</b>	0.100000	-5.900000	56.000000	9.000000	24.100000	100.560000	
<b>50%</b>	9.300000	3.300000	68.000000	13.000000	25.000000	101.070000	
<b>75%</b>	18.800000	11.800000	81.000000	20.000000	25.000000	101.590000	
<b>max</b>	33.000000	24.400000	100.000000	83.000000	48.300000	103.650000	

```
df.isnull().sum()
```

	0
<b>Date/Time</b>	0
<b>Temp_C</b>	0
<b>Dew Point Temp_C</b>	0
<b>Rel Hum_%</b>	0
<b>Wind Speed_km/h</b>	0
<b>Visibility_km</b>	0
<b>Press_kPa</b>	0
<b>Weather</b>	0

```
df.columns
```

```
Index(['Date/Time', 'Temp_C', 'Dew Point Temp_C', 'Rel Hum_%',  
      'Wind Speed_km/h', 'Visibility_km', 'Press_kPa', 'Weather'],  
      dtype='object')
```

### ✓ Step-3:Normalizing the data

```
colrmd=df.drop(['Date/Time','Weather'],axis=1)
```

```
colrmd
```

	Temp_C	Dew Point	Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
<b>0</b>	-1.8	-3.9	86	4	8.0	101.24	
<b>1</b>	-1.8	-3.7	87	4	8.0	101.24	
<b>2</b>	-1.8	-3.4	89	7	4.0	101.26	
<b>3</b>	-1.5	-3.2	88	6	4.0	101.27	
<b>4</b>	-1.5	-3.3	88	7	4.8	101.23	
...	...	...	...	...	...	...	...
<b>8779</b>	0.1	-2.7	81	30	9.7	100.13	
<b>8780</b>	0.2	-2.4	83	24	9.7	100.03	
<b>8781</b>	-0.5	-1.5	93	28	4.8	99.95	
<b>8782</b>	-0.2	-1.8	89	28	9.7	99.91	
<b>8783</b>	0.0	-2.1	86	30	11.3	99.89	

8784 rows × 6 columns

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
norm_data=scaler.fit_transform(colrmd)
norm_df=pd.DataFrame(norm_data,columns=colrmd.columns)
```

norm\_df



	Temp_C	Dew Point	Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
0	0.381883	0.465028	0.829268	0.048193	0.162162	0.606852	
1	0.381883	0.468809	0.841463	0.048193	0.162162	0.606852	
2	0.381883	0.474480	0.865854	0.084337	0.079002	0.610114	
3	0.387211	0.478261	0.853659	0.072289	0.079002	0.611746	
4	0.387211	0.476371	0.853659	0.084337	0.095634	0.605220	
...	...	...	...	...	...	...	...
8779	0.415631	0.487713	0.768293	0.361446	0.197505	0.425775	
8780	0.417407	0.493384	0.792683	0.289157	0.197505	0.409462	
8781	0.404973	0.510397	0.914634	0.337349	0.095634	0.396411	
8782	0.410302	0.504726	0.865854	0.337349	0.197505	0.389886	
8783	0.413854	0.499055	0.829268	0.361446	0.230769	0.386623	

8784 rows × 6 columns

```
df['Date/Time'] = pd.to_datetime(df['Date/Time'])
df['date']=df['Date/Time'].dt.date
df['time']=df['Date/Time'].dt.time
df['month']=df['Date/Time'].dt.month
```

```
df=df.drop(['Date/Time'],axis=1)
```

```
date_time_df = df[['date','time','month','Weather']]
final_df = pd.concat([date_time_df, norm_df], axis=1)
final_df
```



	date	time	month	Weather	Temp_C	Dew Point	Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa
0	2012-01-01	00:00:00	1	Fog	0.381883	0.465028	0.829268	0.048193	0.162162	0.606852	
1	2012-01-01	01:00:00	1	Fog	0.381883	0.468809	0.841463	0.048193	0.162162	0.606852	
2	2012-01-01	02:00:00	1	Freezing Drizzle,Fog	0.381883	0.474480	0.865854	0.084337	0.079002	0.610114	
3	2012-01-01	03:00:00	1	Freezing Drizzle,Fog	0.387211	0.478261	0.853659	0.072289	0.079002	0.611746	
4	2012-01-01	04:00:00	1	Fog	0.387211	0.476371	0.853659	0.084337	0.095634	0.605220	
...	...	...	...	...	...	...	...	...	...	...	...
8779	2012-12-31	19:00:00	12	Snow	0.415631	0.487713	0.768293	0.361446	0.197505	0.425775	
8780	2012-12-31	20:00:00	12	Snow	0.417407	0.493384	0.792683	0.289157	0.197505	0.409462	
8781	2012-12-31	21:00:00	12	Snow	0.404973	0.510397	0.914634	0.337349	0.095634	0.396411	
8782	2012-12-31	22:00:00	12	Snow	0.410302	0.504726	0.865854	0.337349	0.197505	0.389886	
8783	2012-12-31	23:00:00	12	Snow	0.413854	0.499055	0.829268	0.361446	0.230769	0.386623	

8784 rows × 10 columns

## ✓ Step-4:Feature Engineering

```
wtrunique=final_df['Weather'].unique()
wtrunique
```



```
array(['Fog', 'Freezing Drizzle,Fog', 'Mostly Cloudy', 'Cloudy', 'Rain',
      'Rain Showers', 'Mainly Clear', 'Snow Showers', 'Snow', 'Clear',
      'Freezing Rain,Fog', 'Freezing Rain', 'Freezing Drizzle',
      'Rain,Snow', 'Moderate Snow', 'Freezing Drizzle,Snow',
```

```
'Freezing Rain,Snow Grains', 'Snow,Blowing Snow', 'Freezing Fog',
'Haze', 'Rain,Fog', 'Drizzle,Fog', 'Drizzle',
'Freezing Drizzle,Haze', 'Freezing Rain,Haze', 'Snow,Haze',
'Snow,Fog', 'Snow,Ice Pellets', 'Rain,Haze', 'Thunderstorms,Rain',
'Thunderstorms,Rain Showers', 'Thunderstorms,Heavy Rain Showers',
'Thunderstorms,Rain Showers,Fog', 'Thunderstorms',
'Thunderstorms,Rain,Fog',
'Thunderstorms,Moderate Rain Showers,Fog', 'Rain Showers,Fog',
'Rain Showers,Snow Showers', 'Snow Pellets', 'Rain,Snow,Fog',
'Moderate Rain,Fog', 'Freezing Rain,Ice Pellets,Fog',
'Drizzle,Ice Pellets,Fog', 'Drizzle,Snow', 'Rain,Ice Pellets',
'Drizzle,Snow,Fog', 'Rain,Snow Grains', 'Rain,Snow,Ice Pellets',
'Snow Showers,Fog', 'Moderate Snow,Blowing Snow'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
wtr_encoded=le.fit_transform(final_df['Weather'])
wtr_encoded_df=pd.DataFrame(wtr_encoded,columns=['Weather'])
wtr_encoded_df
```



Weather

0	7
1	7
2	9
3	9
4	7
...	...
8779	35
8780	35
8781	35
8782	35
8783	35

8784 rows x 1 columns

```
final_df = final_df.drop('Weather', axis=1)
```

```
final_df=pd.concat([final_df,wtr_encoded_df],axis=1)
final_df
```



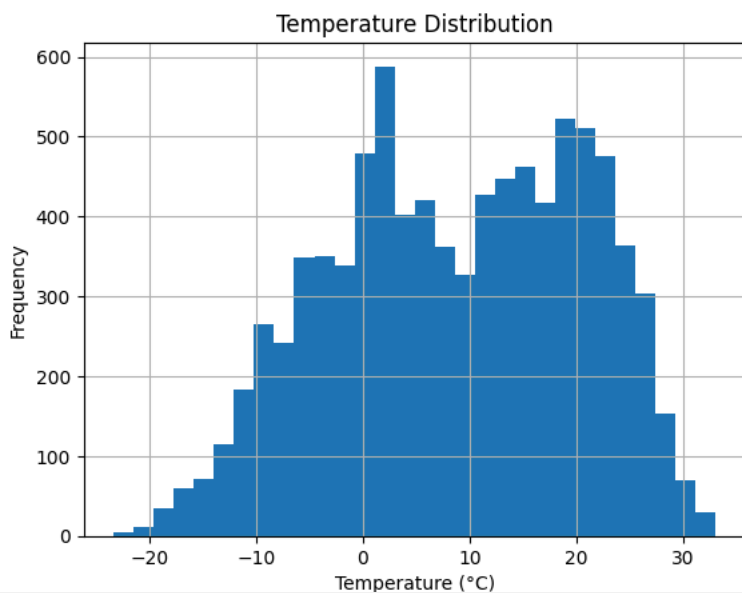
	date	time	month	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa	Weather
0	2012-01-01	00:00:00	1	0.381883	0.465028	0.829268	0.048193	0.162162	0.606852	7
1	2012-01-01	01:00:00	1	0.381883	0.468809	0.841463	0.048193	0.162162	0.606852	7
2	2012-01-01	02:00:00	1	0.381883	0.474480	0.865854	0.084337	0.079002	0.610114	9
3	2012-01-01	03:00:00	1	0.387211	0.478261	0.853659	0.072289	0.079002	0.611746	9
4	2012-01-01	04:00:00	1	0.387211	0.476371	0.853659	0.084337	0.095634	0.605220	7
...	...	...	...	...	...	...	...	...	...	...
8779	2012-12-31	19:00:00	12	0.415631	0.487713	0.768293	0.361446	0.197505	0.425775	35
8780	2012-12-31	20:00:00	12	0.417407	0.493384	0.792683	0.289157	0.197505	0.409462	35
8781	2012-12-31	21:00:00	12	0.404973	0.510397	0.914634	0.337349	0.095634	0.396411	35
8782	2012-12-31	22:00:00	12	0.410302	0.504726	0.865854	0.337349	0.197505	0.389886	35
8783	2012-12-31	23:00:00	12	0.413854	0.499055	0.829268	0.361446	0.230769	0.386623	35

8784 rows x 10 columns

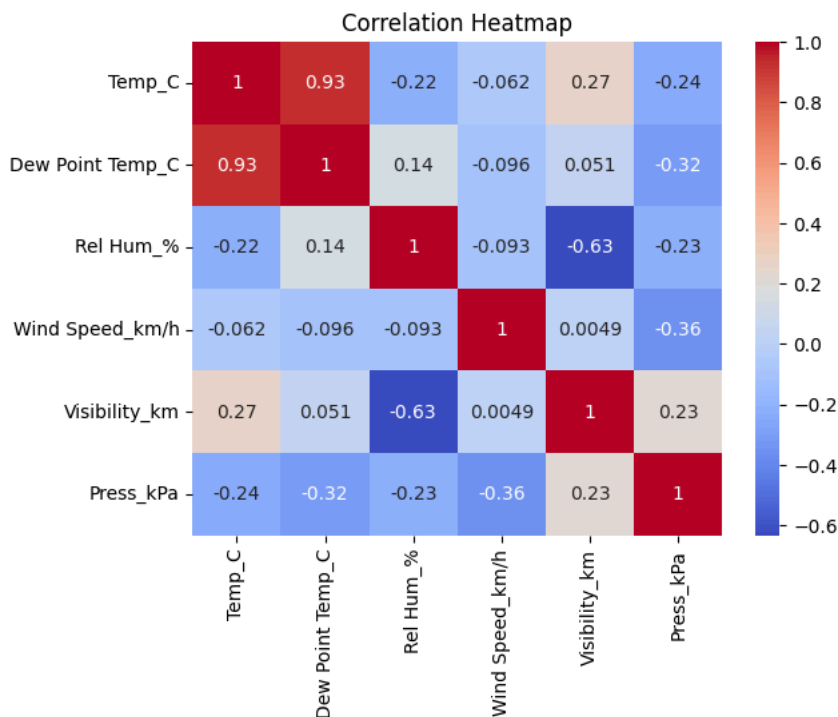
## ✓ TASK-2:EXPLORATORY DATA ANALYSIS

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df['Temp_C'].hist(bins=30)
plt.title('Temperature Distribution')
plt.xlabel('Temperature (°C)')
plt.ylabel('Frequency')
plt.show()
```



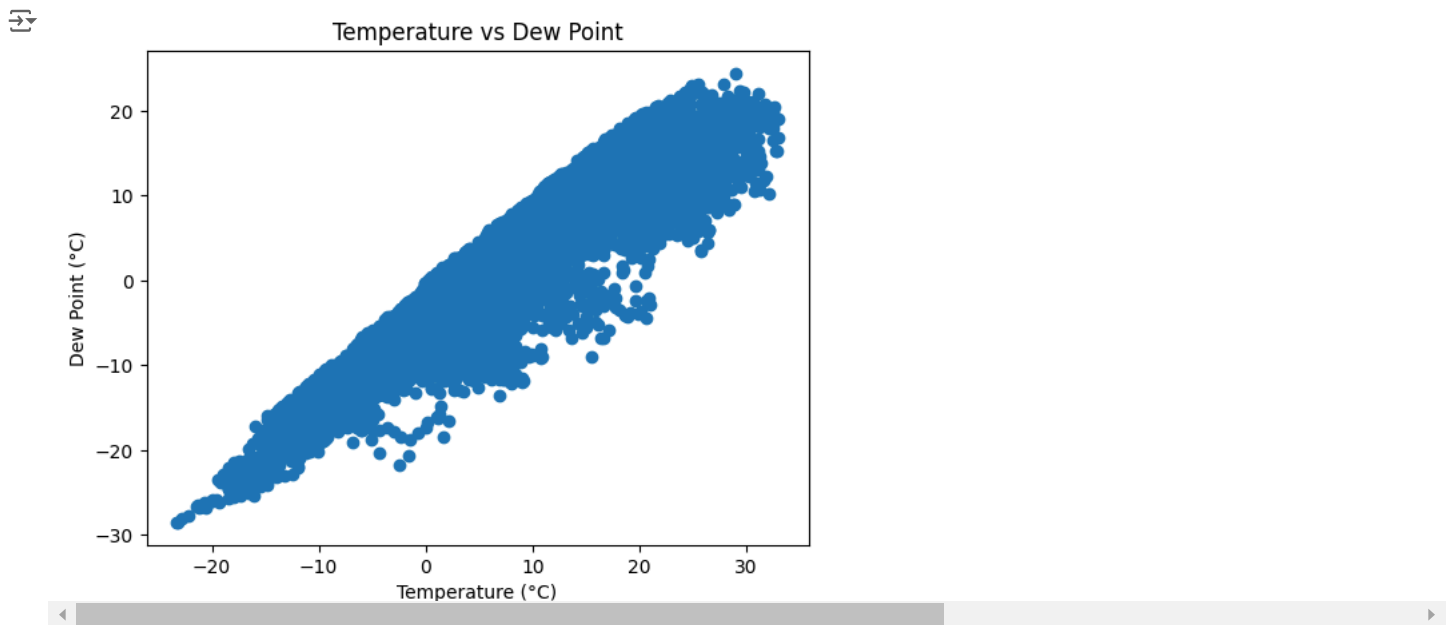
```
sns.heatmap(norm_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



#need to learn all the variables.

```
plt.scatter(df['Temp_C'], df['Dew Point Temp_C'])
plt.title('Temperature vs Dew Point')
plt.xlabel('Temperature (°C)')
plt.ylabel('Dew Point (°C)')
```

```
plt.show()
```



## ✓ TASK-3:MODEL BUILDING

```
from sklearn.preprocessing import MinMaxScaler

# Select relevant columns for modeling
features = final_df[['Temp_C', 'Dew Point Temp_C', 'Rel Hum_%', 'Wind Speed_kmh',
                    'Visibility_kmh', 'Press_kPa']]
target = final_df['Temp_C'] # Predict future temperature

# Scale the features and target between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
features_scaled = scaler.fit_transform(features)
target_scaled = scaler.fit_transform(target.values.reshape(-1, 1))

import numpy as np

def create_sequences(data, target, time_steps=30):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(target[i + time_steps])
    return np.array(X), np.array(y)
```

## ✓ Step-1:Train-Test Splitting

```
# Create input-output sequences
X, y = create_sequences(features_scaled, target_scaled, time_steps=30)

# Train-Test Split (80% training, 20% testing)
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Check shape of input (samples, timesteps, features)
print(X_train.shape) # (samples, 30, 6)
print(y_train.shape) # (samples, 1)

(7003, 30, 6)
(7003, 1)
```

## ✓ Step-2:Model implementation

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
# Define the LSTM model
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(LSTM(64, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1)) # Output layer to predict temperature
```

→ /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument  
super().\_\_init\_\_(\*\*kwargs)

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Summary of the model
model.summary()
```

→ Model: "sequential\_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 30, 64)	18,176
dropout_2 (Dropout)	(None, 30, 64)	0
lstm_3 (LSTM)	(None, 64)	33,024
dropout_3 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 51,265 (200.25 KB)  
Trainable params: 51,265 (200.25 KB)

```
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
                    validation_data=(X_test, y_test))
```

→ Epoch 1/10  
219/219 ————— 10s 30ms/step - loss: 0.0253 - val\_loss: 0.0024  
Epoch 2/10  
219/219 ————— 10s 29ms/step - loss: 0.0046 - val\_loss: 0.0019  
Epoch 3/10  
219/219 ————— 11s 34ms/step - loss: 0.0037 - val\_loss: 0.0012  
Epoch 4/10  
219/219 ————— 6s 28ms/step - loss: 0.0032 - val\_loss: 0.0013  
Epoch 5/10  
219/219 ————— 8s 35ms/step - loss: 0.0030 - val\_loss: 0.0010  
Epoch 6/10  
219/219 ————— 10s 34ms/step - loss: 0.0027 - val\_loss: 9.2015e-04  
Epoch 7/10  
219/219 ————— 6s 29ms/step - loss: 0.0024 - val\_loss: 8.9483e-04  
Epoch 8/10  
219/219 ————— 12s 38ms/step - loss: 0.0023 - val\_loss: 7.2724e-04  
Epoch 9/10  
219/219 ————— 10s 35ms/step - loss: 0.0020 - val\_loss: 7.1227e-04  
Epoch 10/10  
219/219 ————— 9s 29ms/step - loss: 0.0020 - val\_loss: 6.8572e-04

```
# Predict on the test set
y_pred = model.predict(X_test)
```

→ 55/55 ————— 2s 20ms/step

```
# Inverse transform the scaled predictions and actual values
y_pred_inverse = scaler.inverse_transform(y_pred)
y_test_inverse = scaler.inverse_transform(y_test)
```

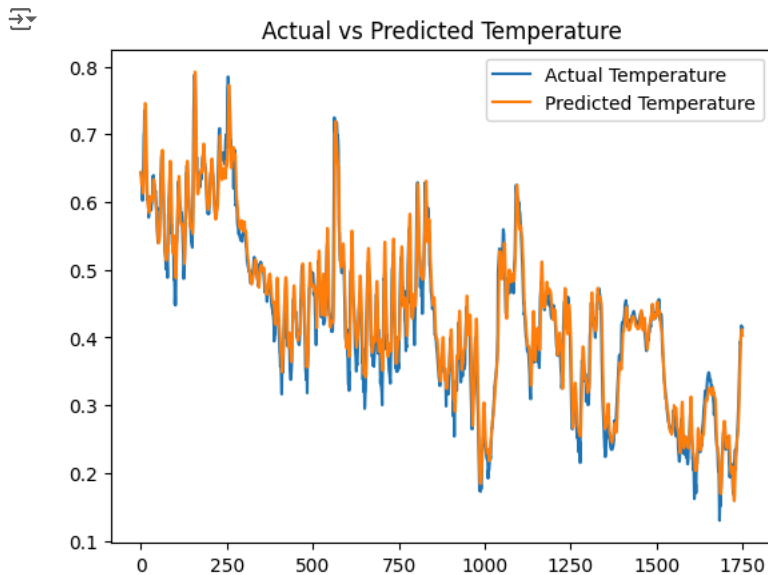
### ✓ Step-3:Evaluating the model

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
mae=mean_absolute_error(y_test_inverse,y_pred_inverse)
mse=mean_squared_error(y_test_inverse,y_pred_inverse)
r2=r2_score(y_test_inverse,y_pred_inverse)
```

```
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("R-squared:", r2)
```

```
→ Mean Absolute Error: 0.019757552029139272
   Mean Squared Error: 0.0006857224193425398
   R-squared: 0.9576572264676144
```

```
# Plot the actual vs predicted temperatures
plt.plot(y_test_inverse, label='Actual Temperature')
plt.plot(y_pred_inverse, label='Predicted Temperature')
plt.legend()
plt.title('Actual vs Predicted Temperature')
plt.show()
```



features\_scaled

```
→ array([[0.38188277, 0.46502836, 0.82926829, 0.04819277, 0.16216216,
          0.60685155],
        [0.38188277, 0.46880907, 0.84146341, 0.04819277, 0.16216216,
          0.60685155],
        [0.38188277, 0.47448015, 0.86585366, 0.08433735, 0.07900208,
          0.61011419],
        ...,
        [0.40497336, 0.51039698, 0.91463415, 0.3373494 , 0.0956341 ,
          0.39641109],
        [0.41030195, 0.5047259 , 0.86585366, 0.3373494 , 0.1975052 ,
          0.38988581],
        [0.41385435, 0.49905482, 0.82926829, 0.36144578, 0.23076923,
          0.38662316]])
```

### ✓ TASK-4:FORECASTING FOR THE NEXT 30 DAYS

```
# Use the last 30 days of data to predict the next 30 days
last_30_days = features_scaled[-30:]
```

```
# Reshape to fit the LSTM input format
last_30_days = last_30_days.reshape(1, 30, features_scaled.shape[1])
```



```
# Forecast the next 30 days
forecast = model.predict(last_30_days)
```

1/1 ————— 0s 23ms/step

```
forecast
```

```
array([[0.40327495]], dtype=float32)
```

```
# Inverse transform the forecasted values
forecast_inverse = scaler.inverse_transform(forecast)
```

```
print(forecast_inverse)
```

```
[[0.40327495]]
```

```
# Inverse transform the forecasted value
forecast_actual = scaler.inverse_transform(forecast)
print(f'Forecasted Temperature: {forecast_actual[0][0]:.2f} °C')
```

```
Forecasted Temperature: 0.40 °C
```

```
forecast_30_days = [] # Store predictions
```

```
for _ in range(30):
    # Predict the next day's temperature
    next_pred = model.predict(last_30_days) # Shape: (1, 1)

    # Inverse transform to get the actual temperature
    temp_pred_actual = scaler.inverse_transform(
        np.concatenate([next_pred, np.zeros((1, 5))], axis=1)
    )[0][0] # Get temperature only

    forecast_30_days.append(temp_pred_actual) # Store forecast

    # Scale the new prediction for input
    # Create a new input array for the next prediction
    temp_pred_scaled = scaler.transform([[temp_pred_actual] + [0] * 5]) # 1D array with 6 features

    # Update input data by shifting the window forward
    last_30_days = np.append(last_30_days[:, 1:, :], temp_pred_scaled[np.newaxis, :, :], axis=1)

# Print the 30-day forecast
print("30-Day Forecasted Temperatures:", forecast_30_days)
```

```
# Original forecasted values (scale as needed, here multiplying by 100)
forecasted_temps = [0.4040361, 0.41888714, 0.42003602, 0.40593693, 0.38047856,
                    0.34877962, 0.31535512, 0.28358227, 0.25568527, 0.23289368,
                    0.21565293, 0.20390755, 0.19721927, 0.19487998, 0.1960852,
                    0.19998297, 0.20564243, 0.21226214, 0.21915692, 0.22580981,
                    0.23182373, 0.23693654, 0.24099943, 0.24395283, 0.24585399,
                    0.24679923, 0.24696437, 0.24650863, 0.24560252, 0.2444095]
```

```
# Convert values to degrees (e.g., by multiplying by 100)
forecasted_temps_degrees = [temp * 100 for temp in forecasted_temps]
```

```
# Create DataFrame with column name "30-Day Forecasted Temperature"
df_forecast = pd.DataFrame(forecasted_temps_degrees, columns=['30-Day Forecasted Temperature'])
```

```
# Display DataFrame
print(df_forecast)
```

```
30-Day Forecasted Temperature
0      40.403610
1      41.888714
2      42.003602
3      40.593693
4      38.047856
5      34.877962
6      31.535512
7      28.358227
8      25.568527
```

9	23.289368
10	21.565293
11	20.390755
12	19.721927
13	19.487998
14	19.608520
15	19.998297
16	20.564243
17	21.226214
18	21.915692
19	22.580981
20	23.182373
21	23.693654
22	24.099943
23	24.395283
24	24.585399
25	24.679923
26	24.696437
27	24.650863
28	24.560252
29	24.440950

Start coding or [generate](#) with AI.

## REPORT OF THE PROJECT

---

Let's understand each of the tasks and steps implemented....

### TASK-1:DATA PROCESSING

1. We have imported the dataset from Kaggle website that has one year data from Jan to Dec 2012.
2. The dataset has around 8000 rows and 8 columns.
  - The columns are Date/Time,Temp\_C,Dew Point Temp\_C,Rel Hum\_%,Wind Speed\_km/h, Visibility\_km,Press\_kPa,Weather.
  - The 'Weather'column has around 50 unique values which are Fog, 'Freezing Drizzle,Fog', 'Mostly Cloudy', 'Cloudy', 'Rain', 'Rain Showers', 'Mainly Clear', 'Snow Showers', 'Snow', 'Clear', 'Freezing Rain,Fog', 'Freezing Rain' etc...
3. We imported the dataset and performed the preprocessing tasks.
4. We checked for the null/missing values.
5. We removed the unnecessary columns for the normalization of the data.
6. We extracted month and time column from the 'date/time' column.
7. We encoded the Weather column with numbers to make it easy to understand.
8. Finally, we made the final dataframe ready for the next steps.

### ✓ TASK-2:EXPLORATORY DATA ANALYSIS

1. We want to analysis different relationships between the data.
2. So, we used different plots and charts to map the relation between the data.
3. In the data, we used histograms, scatter plots and correlation plots to identify the relation between the data.

LSTMs are a type of recurrent neural network (RNN) that uses memory cells and gates to store and update past information. This allows them to learn long-term dependencies in a sequence of data.LSTM models can provide accurate and precise weather forecasts, even over long periods of time. They can also help to model complex weather phenomena.By design, LSTMs are known to store data for a long time. This method is employed when analyzing time-series data, making predictions, and categorizing data.

## TASK-3: MODEL BUILDING

1. We build the LSTM model on the weather data to forecast the weather for next 30 days.
2. We initiated with the train-test split as 80-20% of the data.
3. We further defined the LSTM model along with the dropout layers for the regularization.
4. We have around 51,000 trainable parameters.
5. We next trained and fit the model for the data.
6. Through the metrics, we evaluated the model using R2,MSE,MAE.
7. The metrics MSE,MAE showed the least error after testing the test.
8. R2 is used to determine how well a model fits data and predicts future outcomes.
9. The R2 value gave the value of 95% indicating a best fit for the data.
10. We have plotted between training and validation loss.
11. Further, we have plotted between actual temperature and the predicted temperature.

## TASK-4:FORECASTING THE WEATHER

1. We have considered the last 30 days data to forecast the next days using the model.
2. We have rescaled the data to send as input to the model.
3. We next sent the data into model to predict.
4. We need to get the original values since we get the values as inverse values from the model.
5. For the forecasting,
  - Created an array to store predictions.
  - Scale the new prediction for input.
  - Create a new input array for the next prediction.
  - Update input data by shifting the window forward.
  - finally,printed the final output for next 30 days.
2. At the end, we created a dataframe to show the next 30 days forecast of the weather.

### ✓ Overfitting issues and strategies to handle them.

- LSTMs are well-suited for processing sequences of data with long-range dependencies. They can capture information from earlier time steps and remember it for a more extended period, making them effective for tasks like natural language processing (NLP) and time series analysis.
- LSTMs address the vanishing gradient problem, which is a common issue in training deep networks, particularly RNNs.
- LSTMs have been instrumental in many fields, including NLP, speech recognition, and time series forecasting, due to their ability to capture complex sequential patterns.

But still,

- LSTMs are susceptible to overfitting when there is insufficient training data. Regularization techniques like dropout can help mitigate this issue.
- LSTMs have several hyperparameters to tune, such as the number of LSTM units, the learning rate, and the sequence length.

Some of the strategies to handle the overfittings in the LSTMS:

1. Reduce Model Complexity: We can decrease the number of LSTM units and reduce the number of layers.
2. Use Dropout Regularization: Adding dropout (typically between 20-50%) after LSTM layers can help prevent the network from becoming too reliant on any one node.

3. Increase Training Data: If possible, increase the amount of data available for training. More data helps the model generalize better.
4. Batch Normalization: This can help the model learn a more stable pattern, leading to better generalization.
5. Reduce Learning Rate: Slower training can lead to a more generalized model. Consider using a learning rate scheduler to decrease it over epochs.

These are some of the techniques that lead to reduction in the overfitting for the model. Hence, this concludes the project.

Double-click (or enter) to edit