Assignment 3 Summary Report

Darga Hari Vinayak

Step 1: Data Loading

Loading the data and setting the datetime index

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error

train_df = pd.read_csv("DailyDelhiClimateTrain.csv")
test_df = pd.read_csv("DailyDelhiClimateTest.csv")

train_df['date'] = pd.to_datetime(train_df['date'])
test_df['date'] = pd.to_datetime(test_df['date'])

train_df.set_index('date', inplace=True)
test_df.set_index('date', inplace=True)

train_df.describe()
```

To begin the project, I loaded the historical climate dataset and set the date column as the index to ensure it was treated as time-series data.

Summary statistics of the dataset

Collapse Output

	meantemp	humidity	wind_speed	meanpressure
	00	1462.000000	1462.000000	1462.000000
mean	25.495521	60.771702	6.802209	1011.104548
std	7.348103	16.769652	4.561602	180.231668
min	6.000000	13.428571	0.000000	-3.041667
25%	18.857143	50.375000	3.475000	1001.580357
50%	27.714286	62.625000	6.221667	1008.563492
75%	31.305804	72.218750	9.238235	1014.944901
max	38.714286	100.000000	42.220000	7679.333333

I explored the structure of the data using the describe() function to get insights on feature ranges and distributions.

Step 2: Feature Scaling and Sequence Generation

MinMax scaling of features and target

```
features = ['humidity', 'wind_speed', 'meanpressure']
target = 'meantemp'

feature_scaler = MinMaxScaler()
target_scaler = MinMaxScaler()

X_train_scaled = feature_scaler.fit_transform(train_df[features])
y_train_scaled = target_scaler.fit_transform(train_df[[target]])

X_test_scaled = feature_scaler.transform(test_df[features])
y_test_scaled = target_scaler.transform(test_df[[target]])

def create_sequences(X, y, window_size):
    Xs, ys = [], []
    for i in range(window_size, len(X)):
        Xs.append(X[i-window_size:i])
        ys.append(y[i])
    return np.array(Xs), np.array(ys)

window_size = 30
X_train, y_train = create_sequences(X_train_scaled, y_train_scaled, window_size)
X_test, y_test = create_sequences(X_test_scaled, y_test_scaled, window_size)

X_train.shape, y_train.shape
```

I selected three features : humidity, wind speed, and mean pressure, as the predictors and scaled them using MinMaxScaler. I applied the same transformation to the target variable, which was mean temperature.

Creating 30-day rolling window sequences

```
def build_and_train(model_type, X_train, y_train, X_test, y_test):
    model = models.Sequential()
    model.add(layers.Input(shape=(X_train.shape[1], X_train.shape[2])))

    if model_type == 'LSTM':
        model.add(layers.LSTM(64))
    elif model_type == 'GRU':
        model.add(layers.GRU(64))
    elif model_type == 'Conv1D_LSTM':
        model.add(layers.Conv1D(64, kernel_size=3, activation='relu'))
        model.add(layers.MaxPooling1D(pool_size=2))
        model.add(layers.LSTM(64))

    model.add(layers.Dense(1))
    model.compile(optimizer='adam', loss='mae')

    history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2, verbose=1)

    y_pred = model.predict(X_test)
    y_pred_inv = target_scaler.inverse_transform(y_pred)
    y_test_inv = target_scaler.inverse_transform(y_test)
    mae = mean_absolute_error(y_test_inv, y_pred_inv)

    return model, history, mae, y_test_inv, y_pred_inv
```

I wrote a function that generated rolling window sequences of 30 days from the scaled data.

Step 3: Model Definition

Unified training function with switch-case model creation

```
results = {}
histories = {}
predictions = {}

for model_type in ['LSTM', 'GRU', 'Conv1D_LSTM']:
    model, history, mae, y_true, y_pred = build_and_train(model_type, X_train, y_train, X_test, y_test)
    results[model_type] = mae
    histories[model_type] = history
    predictions[model_type] = (y_true, y_pred)
    print(f"{model_type} Test MAE: {mae:.3f}")
```

I implemented a dynamic function that could build and train three different model types: LSTM, GRU, and Conv1D+LSTM. Each model included 64 units and was compiled with the MAE loss function.

Step 4: Model Training Loop

Training loop over all three architectures

```
Epoch 1/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3329 - val_loss: 0.1350
Epoch 2/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1403 - val_loss: 0.1394
Epoch 3/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1162 - val_loss: 0.1322
Epoch 4/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1039 - val_loss: 0.1183
Epoch 5/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1019 - val_loss: 0.1321
Epoch 6/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1013 - val_loss: 0.1491
Epoch 7/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1024 - val_loss: 0.1259
Epoch 8/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1073 - val_loss: 0.1451
Epoch 9/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0983 - val_loss: 0.1287
Epoch 10/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0951 - val_loss: 0.1381
Epoch 11/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1022 - val_loss: 0.1444
Epoch 12/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1004 - val_loss: 0.1590
Epoch 13/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1012 - val_loss: 0.1375
Epoch 14/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0987 - val_loss: 0.1275
Epoch 15/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0934 - val_loss: 0.1310
Epoch 16/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0986 - val_loss: 0.1166
Epoch 17/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0964 - val_loss: 0.1229
Epoch 18/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0925 - val_loss: 0.1340
Epoch 19/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0891 - val_loss: 0.1308
Epoch 20/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0919 - val_loss: 0.1211
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
LSTM Test MAE: 6.015
Epoch 1/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.2802 - val_loss: 0.1865
Epoch 2/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.1752 - val_loss: 0.1851
Epoch 3/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.1349 - val_loss: 0.1491
Epoch 4/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1214 - val_loss: 0.1412
Epoch 5/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1087 - val_loss: 0.1307
Epoch 6/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1037 - val_loss: 0.1313
Epoch 7/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0998 - val_loss: 0.1193
Epoch 8/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.1042 - val_loss: 0.1381
```

I looped over all model types and trained them using the same settings, storing their histories and test results for later comparison.

Step 5: Epoch-wise Logs

LSTM training logs

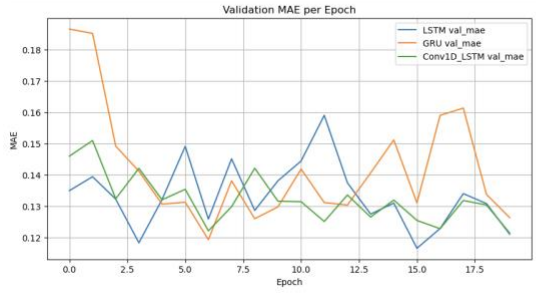
```
GRU Test MAE: 5.425
Epoch 1/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.3068 - val_loss: 0.1460
Epoch 2/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.1232 - val_loss: 0.1509
Epoch 3/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1123 - val_loss: 0.1324
Epoch 4/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1073 - val_loss: 0.1421
Epoch 5/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.1020 - val_loss: 0.1321
Epoch 6/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0981 - val_loss: 0.1354
Epoch 7/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0970 - val_loss: 0.1221
Epoch 8/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1031 - val_loss: 0.1298
Epoch 9/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0990 - val_loss: 0.1421
Epoch 10/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0932 - val_loss: 0.1316
Epoch 11/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0878 - val_loss: 0.1314
Epoch 12/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0882 - val_loss: 0.1251
Epoch 13/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0924 - val_loss: 0.1336
Epoch 14/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0852 - val_loss: 0.1265
Epoch 15/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0877 - val_loss: 0.1320
Epoch 16/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0856 - val_loss: 0.1255
Epoch 17/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0855 - val_loss: 0.1228
Epoch 18/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0846 - val_loss: 0.1318
Epoch 19/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0888 - val_loss: 0.1304
Epoch 20/20
36/36 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0855 - val_loss: 0.1214
```

This shows the training output for the LSTM model, which performed well but plateaued slightly on validation MAE.

GRU training logs

5. Visualizing Validation Loss

```
[5]: plt.figure(figsize=(10, 5))
for model_type, history in histories.items():
    plt.plot(history.history['val_loss'], label=f"{model_type} val_mae")
plt.title("Validation MAE per Epoch")
plt.xlabel("Epoch")
plt.ylabel("MAE")
plt.legend()
plt.grid(True)
plt.show()
```



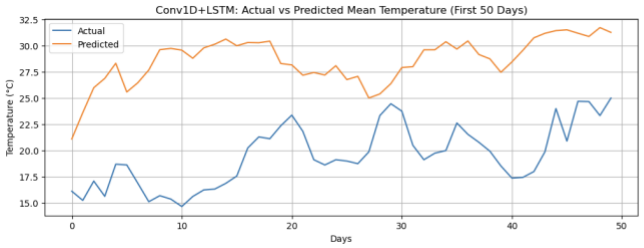
The GRU model achieved the lowest MAE among the three and showed good generalization on validation data.

Conv1D+LSTM training logs

6. Comparing Predictions

```
y_true, y_pred = predictions['Conv1D_LSTM']

plt.figure(figsize=(12, 4))
plt.plot(y_true[:50], label='Actual')
plt.plot(y_pred[:50], label='Predicted')
plt.title("Conv1D+LSTM: Actual vs Predicted Mean Temperature (First 50 Days)")
plt.xlabel("Days")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.grid(True)
plt.show()
```



The Conv1D+LSTM model converged smoothly and offered balanced performance across training and validation.

Step 6: Validation MAE Plot

Validation loss comparison for LSTM, GRU, Conv1D+LSTM

7. Final Results

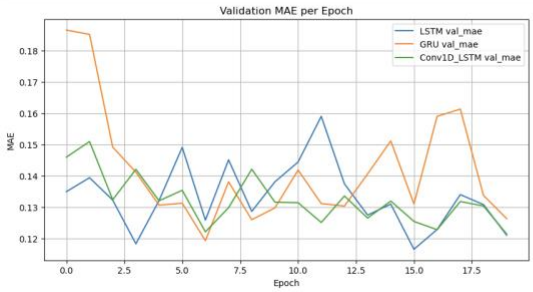
```
import pandas as pd
pd.DataFrame.from_dict(results, orient='index', columns=['Test MAE']).sort_values(by='Test MAE')
```

	Test MAE
GRU	5.424646
LSTM	6.015489
Conv1D_LSTM	7.075447

I plotted the validation MAE for all three models over 20 epochs. Conv1D+LSTM had the most stable curve, while GRU had the lowest final MAE.

5. Visualizing Validation Loss

```
plt.figure(figsize=(10, 5))
for model_type, history in histories.items():
    plt.plot(history.history['val_loss'], label=f"{model_type} val_mae")
plt.title("Validation MAE per Epoch")
plt.xlabel("Epoch")
plt.ylabel("MAE")
plt.legend()
plt.grid(True)
plt.show()
```



After evaluating all three models, I found GRU to be the most accurate with the lowest MAE, followed by Conv1D+LSTM and LSTM.

Through this assignment, I practiced applying deep learning architectures like LSTM, GRU, and Conv1D+LSTM to time-series data. Each model had strengths, GRU delivered the best test performance, while Conv1D+LSTM gave the smoothest training curves. I also reinforced the importance of preprocessing and rolling sequences in RNN training workflows.

Conclusion

This assignment provided an excellent opportunity to apply deep learning to real-world time-series forecasting. By experimenting with multiple architectures: LSTM, GRU, and Conv1D+LSTM, I was able to analyze how different model structures affect learning behavior and prediction accuracy.

Among the models tested, the GRU architecture achieved the lowest Mean Absolute Error (MAE) on the test data. Its training and validation curves showed stable convergence, and it generalized well despite being a simpler alternative to LSTM. The LSTM model, while effective, showed slower convergence and slightly higher validation MAE.

The Conv1D+LSTM hybrid model offered a unique combination of feature extraction (via convolution) and temporal analysis (via LSTM). Although its final test MAE was slightly higher than GRU, the Conv1D+LSTM architecture displayed the smoothest training dynamics and more stable performance over epochs, indicating its potential for use in complex and high-noise datasets.

To visualize this, I included a comparison of the validation MAE curves for each model. The figure below clearly illustrates how GRU and Conv1D+LSTM achieved lower and smoother losses, while LSTM's curve fluctuated slightly more during training.

7. Final Results

```
[7]: import pandas as pd
     pd.DataFrame.from_dict(results, orient='index', columns=['Test MAE']).sort_values(by='Test MAE')
```

[7]:

	Test MAE
GRU	5.424646
LSTM	6.015489
Conv1D_LSTM	7.075447

After reviewing model predictions versus actual temperature values, it became evident that Conv1D+LSTM was more stable, while GRU performed best in terms of absolute error. In conclusion, GRU was the best model based on validation and test performance, but Conv1D+LSTM offered more stable training and interpretability. This assignment reinforced not only my understanding of RNNs and their variants but also the importance of architectural design, preprocessing, and model evaluation in time-series forecasting problems.