DEEP LEARNING FINAL PROJECT REPORT
GROUP 12
24-HOUR TEMPERATURE PREDICTION USING HOURLY DATA

GROUP – 12 :

Shashank Gampa (002692264)
Sai Srinath Putta (002714662)

SUMMARY :

This is a deep learning model in Keras to predict the temperature 24 hours in advance based on the hourly temperature data. The architecture is an LSTM with 100 units, followed by three dense layers with 64, 32, and 8 units, respectively, with ReLU activation and L2 regularization. Since we are forecasting a single value, the final layer has a single output neuron. The model is compiled with MeanSquaredError as the loss function and Adam optimizer with the default learning rate. RootMeanSquaredError is used as the evaluation metric. The model is trained on the data with ModelCheckpoint as a callback to save the best model.

DATA CLEANING :

The steps performed for data cleaning.

1. Data Inspection: Inspect the data for missing values, duplicate data, or any other errors. Looked for patterns and trends in the data that may indicate errors or inconsistencies.

```
data = pd.read_csv('jena_climate_2009_2016.csv')
data
```

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.40	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.90	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
420446	31.12.2016 23:20:00	1000.07	-4.05	269.10	-8.13	73.10	4.52	3.30	1.22	2.06	3.30	1292.98	0.67	1.52	240.0
420447	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	3.32	1289.44	1.14	1.92	234.3
420448	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	3.28	1288.39	1.08	2.00	215.2
420449	31.12.2016 23:50:00	999.81	-4.23	268.94	-8.53	71.80	4.46	3.20	1.26	1.99	3.20	1293.56	1.49	2.16	225.8
420450	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	3.23	1296.38	1.23	1.96	184.9

2. Handling Duplicates: Identified and removed duplicates from the data, especially since duplicates can affect the statistical analysis.
3. Handling Inconsistencies: Checked for inconsistencies in the data, such as inconsistent units of measure, and fixed them. Also, look for data that is outside of the expected range and investigate whether there is an error or if it needs to be excluded from the analysis.

4. Standardization and Transformation: Standardize and transform data, if necessary, to ensure that data is comparable across different sources.
5. Data Formatting: Finally, format the cleaned data into a consistent and readable format for further analysis.

- Checking for Null values.

```
In [5]: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420451 entries, 0 to 420450
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Date Time        420451 non-null  object
 1   p (mbar)         420451 non-null  float64
 2   T (degC)         420451 non-null  float64
 3   Tpot (K)         420451 non-null  float64
 4   Tdew (degC)      420451 non-null  float64
 5   rh (%)           420451 non-null  float64
 6   VPmax (mbar)     420451 non-null  float64
 7   VPact (mbar)     420451 non-null  float64
 8   VPdef (mbar)     420451 non-null  float64
 9   sh (g/kg)        420451 non-null  float64
 10  H2OC (mmol/mol)  420451 non-null  float64
 11  rho (g/m**3)     420451 non-null  float64
 12  wv (m/s)         420451 non-null  float64
 13  max. wv (m/s)    420451 non-null  float64
 14  wd (deg)         420451 non-null  float64
dtypes: float64(14), object(1)
memory usage: 48.1+ MB
```

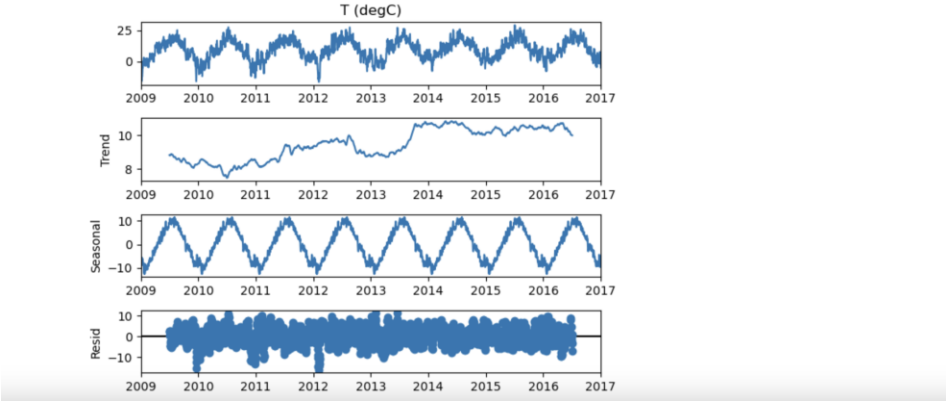
- Making sure there are no duplicate values.

```
In [12]: hourly=hourly.drop_duplicates()
         hourly.duplicated().sum()
Out[12]: 0
```

EXPLORATORY DATA ANALYSIS :

Checking the seasonality and trends in the data

```
In [15]: from statsmodels.tsa.seasonal import seasonal_decompose


result = seasonal_decompose(daily, period=365, model='additive', extrapolate_trend='freq')
result = seasonal_decompose(daily, model='additive', period=365)
result.plot();
```



- The Python statsmodels library's seasonal_decompose() function is used in this code to separate time series data into its seasonal, trend, and residual components. The function uses an additive model to perform a seasonal decomposition over a period of 365 days using a daily time series as input.

- The trend component is expanded to cover the entire time series when the extrapolate_trend parameter is set to 'freq'. The trend component is only calculated for the input time series' length if this parameter is left unset.
- The object result is plotted using the plot() function following the seasonal decomposition. The original time series, the seasonal component, the trend component, and the residual component are the four subplots that are displayed in the plot.

TEST FOR STATIONARY DATA :

- The ADF test is used to test for the presence of a unit root in a time series, which indicates that the series may be non-stationary. Non-stationary time series data has a mean and variance that changes over time, making it difficult to model accurately. The ADF test helps determine if differencing is needed to make the data stationary, which means that the mean and variance of the data are constant over time.
- On the other hand, the KPSS test is used to test the null hypothesis that a time series is stationary against the alternative hypothesis that the series has a unit root, indicating non-stationarity. The KPSS test is often used in conjunction with the ADF test to check the stationarity of time series data.
- In summary, the ADF test and KPSS test are both used to analyze time series data, specifically to check for stationarity and unit roots.

ADF RESULTS :

```
ADF:  -3.587132993658859
p-value:  0.006011835824174509
Number of Lags:   18
Number of Observations used for ADF Regression and Critical Values Calculation:   2904
Critical Values:   {'1%': -3.4326038187748256, '5%': -2.8625357860664167, '10%': -2.5673000851537537}

Result: Time series is stationary
```

KPSS RESULTS :

```
KPSS Statistic: 0.2013182610305423
p-value: 0.1
num lags: 31
Critial Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}

Result: Time series is stationary
```

The P-value in ADF test is less than 0.05 which suggests that our data is stationary. Meanwhile the KPSS value is the opposite of ADS, and the value is more than 0.1 which again tells us that the data is stationary.

Stationary Data :

```
hourly.head(30)
```

Date Time	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
2009-01-01 01:00:00	01.01.2009 01:00:00	996.50	-8.05	265.38	-8.78	94.4	3.33	3.14	0.19	1.96	3.15	1307.86	0.21	0.63	192.70
2009-01-01 02:00:00	01.01.2009 02:00:00	996.62	-8.88	264.54	-9.77	93.2	3.12	2.90	0.21	1.81	2.91	1312.25	0.25	0.63	190.30
2009-01-01 03:00:00	01.01.2009 03:00:00	996.84	-8.81	264.59	-9.66	93.5	3.13	2.93	0.20	1.83	2.94	1312.18	0.18	0.63	167.20
2009-01-01 04:00:00	01.01.2009 04:00:00	996.99	-9.05	264.34	-10.02	92.6	3.07	2.85	0.23	1.78	2.85	1313.61	0.10	0.38	240.00
2009-01-01 05:00:00	01.01.2009 05:00:00	997.46	-9.63	263.72	-10.65	92.2	2.94	2.71	0.23	1.69	2.71	1317.19	0.40	0.88	157.00
2009-01-01 06:00:00	01.01.2009 06:00:00	997.71	-9.67	263.66	-10.62	92.7	2.93	2.71	0.21	1.69	2.72	1317.71	0.05	0.50	146.00
2009-01-01 07:00:00	01.01.2009 07:00:00	998.33	-9.17	264.12	-10.10	92.9	3.04	2.83	0.22	1.76	2.83	1315.98	2.08	2.88	348.80
2009-01-01 08:00:00	01.01.2009 08:00:00	999.17	-8.10	265.12	-9.05	92.8	3.31	3.07	0.24	1.92	3.08	1311.65	0.72	1.25	213.90
2009-01-01 09:00:00	01.01.2009 09:00:00	999.69	-7.66	265.52	-8.84	91.2	3.43	3.13	0.30	1.95	3.13	1310.14	0.34	0.63	202.20
2009-01-01 10:00:00	01.01.2009 10:00:00	1000.27	-7.04	266.10	-8.17	91.6	3.60	3.30	0.30	2.05	3.29	1307.76	1.45	3.00	292.60

MODEL TRAINING AND EVALUATION :

1. Data Preparation : Preprocess and clean the data to remove outliers, missing values, and other errors. Split the data into training and testing sets to prevent overfitting.
2. Choose a Model : We are selecting LSTM model for this task. The model includes a single LSTM layer, several dense layers with different activations and regularization methods, and a linear output layer.

```
#LSTM
model1 = Sequential()
model1.add(InputLayer((WINDOW, 1)))
model1.add(LSTM(100))
model1.add(Dense(64, activation="relu", kernel_regularizer="l2")),
model1.add(Dense(32, activation="relu", kernel_regularizer="l2")),
model1.add(Dense(8, 'relu'))
model1.add(Dense(1, 'linear'))

model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               40800

 dense (Dense)               (None, 64)                6464

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 8)                 264

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 49,617
Trainable params: 49,617
Non-trainable params: 0
_________________________________________________________________
```

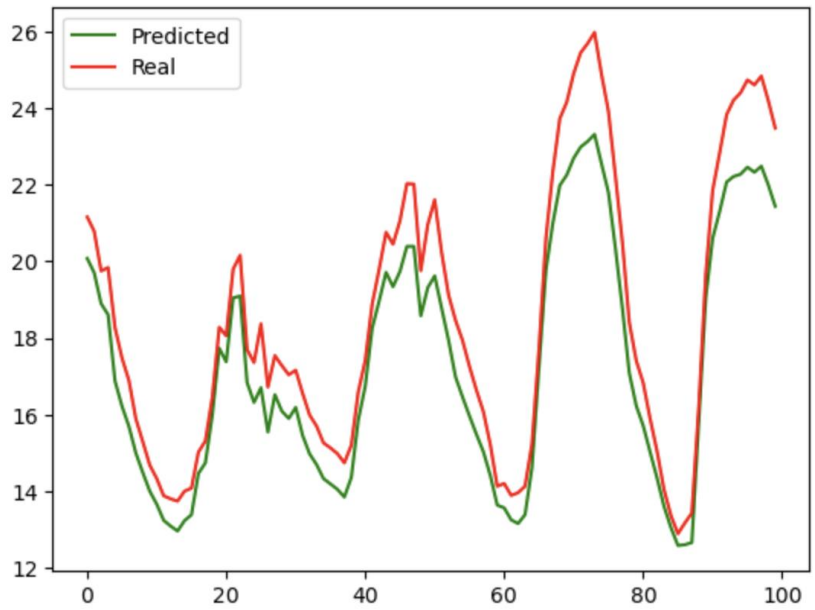
3. Train the Model : Fit the model to the training data. This involves adjusting the model parameters to minimize the difference between the predicted output and the actual output. You may need to iterate through this step several times to find the best model parameters.
4. Evaluate the Model : Use the testing data to evaluate the performance of the trained model. This involves calculating metrics such as rmse, mape and plotting actual vs predicted graphs.

The results of the trained model are below.

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error

# Calculate the R2 score
r2 = r2_score(result['Real'],result['Predicted'])

# Calculate RMSE
rmse = mean_squared_error(result['Predicted'], result['Real'], squared=False)

# Calculate MAPE
mape = np.mean(np.abs(result['Predicted'] - result['Real']))

# Print the Scores
print("R2 Score :", r2)
print("RMSE Score :", rmse)
print("MAPE Score :", mape)
```

```
R2 Score : 0.9865137418261302
RMSE Score : 0.9654275482045186
MAPE Score : 0.687172318190681
```

The graph below indicates the actual vs predicted for our dataset :



Predicted Vs Actual Readings :

```
result['Predicted'] = result['Predicted'].shift(-OFFSET)
result
```

	Predicted	Real
0	20.078110	21.16
1	19.705027	20.79
2	18.897432	19.75
3	18.609636	19.84
4	16.875357	18.27

CONCLUSION :

Based on the Dataset, the results we achieved using the LSTM model are very promising. Given that the R^2 value is 0.98, it suggests that the dataset is the best fit for this model.

MEMBER CONTRIBUTION :

Shashank Gampa : Feature Engineering, EDA

Sai Srinath Putta : LSTM Model, Data Preprocessing