In [1]:

# ARTICLE REGARDING PREDICTION OF TEMPERATURE

We were given the data for the purpose of bias correction of next-day maximum and minimum air temperatures forecast of the LDAPS model operated by the Korea Meteorological Administration over Seoul, South Korea. That data consisted of summer data from 2013 to 2017. The input data largely composed of the LDAPS model's next-day forecast data, in-situ maximum and minimum temperatures of present-day, and geographic auxiliary variables.

PROBLEM DEFINITION :

The goal is to predict the next-day minimum and maximum temperatures using machine learning models based on the given dataset. This involves regression tasks where the input features are various meteorological factors, and the output targets are the next-day minimum and maximum temperatures.

**DATA ANALYSIS**

**Data Preparation (Loading and Cleaning)**

Let’s begin first with importing necessary libraries for our analysis and EDA.

# Importing necessary libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

In [2]:

# Loading the dataset.

df=pd.read\_csv('https://raw.githubusercontent.com/FlipRoboTechnologies/ML\_-Datasets/main/Temperature%20Forecast/temperature.csv')

In [3]:

df.head(10)

Out[3]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **...** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.0 | 30-06-2013 | 28.7 | 21.4 | 58.255688 | 91.116364 | 28.074101 | 23.006936 | 6.818887 | 69.451805 | ... | 0.0 | 0.0 | 0.0 | 37.6046 | 126.991 | 212.3350 | 2.7850 | 5992.895996 | 29.1 | 21.2 |
| **1** | 2.0 | 30-06-2013 | 31.9 | 21.6 | 52.263397 | 90.604721 | 29.850689 | 24.035009 | 5.691890 | 51.937448 | ... | 0.0 | 0.0 | 0.0 | 37.6046 | 127.032 | 44.7624 | 0.5141 | 5869.312500 | 30.5 | 22.5 |
| **2** | 3.0 | 30-06-2013 | 31.6 | 23.3 | 48.690479 | 83.973587 | 30.091292 | 24.565633 | 6.138224 | 20.573050 | ... | 0.0 | 0.0 | 0.0 | 37.5776 | 127.058 | 33.3068 | 0.2661 | 5863.555664 | 31.1 | 23.9 |
| **3** | 4.0 | 30-06-2013 | 32.0 | 23.4 | 58.239788 | 96.483688 | 29.704629 | 23.326177 | 5.650050 | 65.727144 | ... | 0.0 | 0.0 | 0.0 | 37.6450 | 127.022 | 45.7160 | 2.5348 | 5856.964844 | 31.7 | 24.3 |
| **4** | 5.0 | 30-06-2013 | 31.4 | 21.9 | 56.174095 | 90.155128 | 29.113934 | 23.486480 | 5.735004 | 107.965535 | ... | 0.0 | 0.0 | 0.0 | 37.5507 | 127.135 | 35.0380 | 0.5055 | 5859.552246 | 31.2 | 22.5 |
| **5** | 6.0 | 30-06-2013 | 31.9 | 23.5 | 52.437126 | 85.307251 | 29.219342 | 23.822613 | 6.182295 | 50.231389 | ... | 0.0 | 0.0 | 0.0 | 37.5102 | 127.042 | 54.6384 | 0.1457 | 5873.780762 | 31.5 | 24.0 |
| **6** | 7.0 | 30-06-2013 | 31.4 | 24.4 | 56.287189 | 81.019760 | 28.551859 | 24.238467 | 5.587135 | 125.110007 | ... | 0.0 | 0.0 | 0.0 | 37.5776 | 126.838 | 12.3700 | 0.0985 | 5849.233398 | 30.9 | 23.4 |
| **7** | 8.0 | 30-06-2013 | 32.1 | 23.6 | 52.326218 | 78.004539 | 28.851982 | 23.819054 | 6.104417 | 42.011547 | ... | 0.0 | 0.0 | 0.0 | 37.4697 | 126.910 | 52.5180 | 1.5629 | 5863.992188 | 31.1 | 22.9 |
| **8** | 9.0 | 30-06-2013 | 31.4 | 22.0 | 55.338791 | 80.784607 | 28.426975 | 23.332373 | 6.017135 | 85.110971 | ... | 0.0 | 0.0 | 0.0 | 37.4967 | 126.826 | 50.9312 | 0.4125 | 5876.901367 | 31.3 | 21.6 |
| **9** | 10.0 | 30-06-2013 | 31.6 | 20.5 | 56.651203 | 86.849632 | 27.576705 | 22.527018 | 6.518841 | 63.006075 | ... | 0.0 | 0.0 | 0.0 | 37.4562 | 126.955 | 208.5070 | 5.1782 | 5893.608398 | 30.5 | 21.0 |

10 rows × 25 columns

In [4]:

df.tail()

Out[4]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **...** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7747** | 23.0 | 30-08-2017 | 23.3 | 17.1 | 26.741310 | 78.869858 | 26.352081 | 18.775678 | 6.148918 | 72.058294 | ... | 0.000000 | 0.000000 | 0.000000 | 37.5372 | 126.891 | 15.5876 | 0.155400 | 4443.313965 | 28.3 | 18.1 |
| **7748** | 24.0 | 30-08-2017 | 23.3 | 17.7 | 24.040634 | 77.294975 | 27.010193 | 18.733519 | 6.542819 | 47.241457 | ... | 0.000000 | 0.000000 | 0.000000 | 37.5237 | 126.909 | 17.2956 | 0.222300 | 4438.373535 | 28.6 | 18.8 |
| **7749** | 25.0 | 30-08-2017 | 23.2 | 17.4 | 22.933014 | 77.243744 | 27.939516 | 18.522965 | 7.289264 | 9.090034 | ... | 0.000000 | 0.000000 | 0.000000 | 37.5237 | 126.970 | 19.5844 | 0.271300 | 4451.345215 | 27.8 | 17.4 |
| **7750** | NaN | NaN | 20.0 | 11.3 | 19.794666 | 58.936283 | 17.624954 | 14.272646 | 2.882580 | -13.603212 | ... | 0.000000 | 0.000000 | 0.000000 | 37.4562 | 126.826 | 12.3700 | 0.098475 | 4329.520508 | 17.4 | 11.3 |
| **7751** | NaN | NaN | 37.6 | 29.9 | 98.524734 | 100.000153 | 38.542255 | 29.619342 | 21.857621 | 213.414006 | ... | 21.621661 | 15.841235 | 16.655469 | 37.6450 | 127.135 | 212.3350 | 5.178230 | 5992.895996 | 38.9 | 29.8 |

5 rows × 25 columns

# Exploratory Data Analysis[¶](#Exploratory-Data-Analysis)

*Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover PATTERNS, TO spot ANOMALIES, TO test hypothesis and to check assumptions with the help of summary statistics and graphical representations. In* [5]:

# Checking the dimensions of the dataset.

df.shape

Out[5]:

(7752, 25)

Here we can see that there are 7752 rows and 25 columns in our dataset regarding temperature analysis. In [6]:

df.columns

Out[6]:

Index(['station', 'Date', 'Present\_Tmax', 'Present\_Tmin', 'LDAPS\_RHmin',

'LDAPS\_RHmax', 'LDAPS\_Tmax\_lapse', 'LDAPS\_Tmin\_lapse', 'LDAPS\_WS',

'LDAPS\_LH', 'LDAPS\_CC1', 'LDAPS\_CC2', 'LDAPS\_CC3', 'LDAPS\_CC4',

'LDAPS\_PPT1', 'LDAPS\_PPT2', 'LDAPS\_PPT3', 'LDAPS\_PPT4', 'lat', 'lon',

'DEM', 'Slope', 'Solar radiation', 'Next\_Tmax', 'Next\_Tmin'],

dtype='object')

In [7]:

# Checking types of columns

df.dtypes

Out[7]:

station float64

Date object

Present\_Tmax float64

Present\_Tmin float64

LDAPS\_RHmin float64

LDAPS\_RHmax float64

LDAPS\_Tmax\_lapse float64

LDAPS\_Tmin\_lapse float64

LDAPS\_WS float64

LDAPS\_LH float64

LDAPS\_CC1 float64

LDAPS\_CC2 float64

LDAPS\_CC3 float64

LDAPS\_CC4 float64

LDAPS\_PPT1 float64

LDAPS\_PPT2 float64

LDAPS\_PPT3 float64

LDAPS\_PPT4 float64

lat float64

lon float64

DEM float64

Slope float64

Solar radiation float64

Next\_Tmax float64

Next\_Tmin float64

dtype: object

Here we have only 1 column as object datatype that is (Date). But its datatype is wrong so we will later change its datatype to date and time type. Rest of our columns are float numeric type. In [8]:

df.isnull().sum()

Out[8]:

station 2

Date 2

Present\_Tmax 70

Present\_Tmin 70

LDAPS\_RHmin 75

LDAPS\_RHmax 75

LDAPS\_Tmax\_lapse 75

LDAPS\_Tmin\_lapse 75

LDAPS\_WS 75

LDAPS\_LH 75

LDAPS\_CC1 75

LDAPS\_CC2 75

LDAPS\_CC3 75

LDAPS\_CC4 75

LDAPS\_PPT1 75

LDAPS\_PPT2 75

LDAPS\_PPT3 75

LDAPS\_PPT4 75

lat 0

lon 0

DEM 0

Slope 0

Solar radiation 0

Next\_Tmax 27

Next\_Tmin 27

dtype: int64

Here we can see that there are many columns with missing values. Let’s check out the percentage value missing out. In [9]:

missing\_values = df.isnull().sum()

(missing\_values/len(df))\*100

Out[9]:

station 0.025800

Date 0.025800

Present\_Tmax 0.902993

Present\_Tmin 0.902993

LDAPS\_RHmin 0.967492

LDAPS\_RHmax 0.967492

LDAPS\_Tmax\_lapse 0.967492

LDAPS\_Tmin\_lapse 0.967492

LDAPS\_WS 0.967492

LDAPS\_LH 0.967492

LDAPS\_CC1 0.967492

LDAPS\_CC2 0.967492

LDAPS\_CC3 0.967492

LDAPS\_CC4 0.967492

LDAPS\_PPT1 0.967492

LDAPS\_PPT2 0.967492

LDAPS\_PPT3 0.967492

LDAPS\_PPT4 0.967492

lat 0.000000

lon 0.000000

DEM 0.000000

Slope 0.000000

Solar radiation 0.000000

Next\_Tmax 0.348297

Next\_Tmin 0.348297

dtype: float64

Since we have such a large dataset and the missing values are less than 1%, so we can drop these values.

In [10]:

df.dropna(inplace=True

In [11]:

df.shape

Out[11]:

(7588, 25)

We have now reduced the size of our dataset. Now there are 7588 rows and 25 columns. In [12]:

df.info()

<class 'pandas.core.frame.DataFrame'>

Index: 7588 entries, 0 to 7749

Data columns (total 25 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 station 7588 non-null float64

1 Date 7588 non-null object

2 Present\_Tmax 7588 non-null float64

3 Present\_Tmin 7588 non-null float64

4 LDAPS\_RHmin 7588 non-null float64

5 LDAPS\_RHmax 7588 non-null float64

6 LDAPS\_Tmax\_lapse 7588 non-null float64

7 LDAPS\_Tmin\_lapse 7588 non-null float64

8 LDAPS\_WS 7588 non-null float64

9 LDAPS\_LH 7588 non-null float64

10 LDAPS\_CC1 7588 non-null float64

11 LDAPS\_CC2 7588 non-null float64

12 LDAPS\_CC3 7588 non-null float64

13 LDAPS\_CC4 7588 non-null float64

14 LDAPS\_PPT1 7588 non-null float64

15 LDAPS\_PPT2 7588 non-null float64

16 LDAPS\_PPT3 7588 non-null float64

17 LDAPS\_PPT4 7588 non-null float64

18 lat 7588 non-null float64

19 lon 7588 non-null float64

20 DEM 7588 non-null float64

21 Slope 7588 non-null float64

22 Solar radiation 7588 non-null float64

23 Next\_Tmax 7588 non-null float64

24 Next\_Tmin 7588 non-null float64

dtypes: float64(24), object(1)

memory usage: 1.5+ MB

**Data Integrity Check:** Dataset can have duplicated entries and whitespaces. Now we will perform this integrity check of dataset. In [13]:

df.duplicated().sum()

Out[13]:

0

Luckily there is no duplicate value, now we should now should convert the datatype of Date column and split it into Day Month and Year and let’s use head to check if we get it correctly. In [14]:

# Converting Date datatypes and spliting date into date, month and year.

df['Date']=pd.to\_datetime(df['Date'])

df['Day']=df['Date'].apply(lambda x:x.day)

df['Month']=df['Date'].apply(lambda x:x.month)

df['Year']=df['Date'].apply(lambda x:x.year)

df.head()

Out[14]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **...** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.0 | 2013-06-30 | 28.7 | 21.4 | 58.255688 | 91.116364 | 28.074101 | 23.006936 | 6.818887 | 69.451805 | ... | 37.6046 | 126.991 | 212.3350 | 2.7850 | 5992.895996 | 29.1 | 21.2 | 30 | 6 | 2013 |
| **1** | 2.0 | 2013-06-30 | 31.9 | 21.6 | 52.263397 | 90.604721 | 29.850689 | 24.035009 | 5.691890 | 51.937448 | ... | 37.6046 | 127.032 | 44.7624 | 0.5141 | 5869.312500 | 30.5 | 22.5 | 30 | 6 | 2013 |
| **2** | 3.0 | 2013-06-30 | 31.6 | 23.3 | 48.690479 | 83.973587 | 30.091292 | 24.565633 | 6.138224 | 20.573050 | ... | 37.5776 | 127.058 | 33.3068 | 0.2661 | 5863.555664 | 31.1 | 23.9 | 30 | 6 | 2013 |
| **3** | 4.0 | 2013-06-30 | 32.0 | 23.4 | 58.239788 | 96.483688 | 29.704629 | 23.326177 | 5.650050 | 65.727144 | ... | 37.6450 | 127.022 | 45.7160 | 2.5348 | 5856.964844 | 31.7 | 24.3 | 30 | 6 | 2013 |
| **4** | 5.0 | 2013-06-30 | 31.4 | 21.9 | 56.174095 | 90.155128 | 29.113934 | 23.486480 | 5.735004 | 107.965535 | ... | 37.5507 | 127.135 | 35.0380 | 0.5055 | 5859.552246 | 31.2 | 22.5 | 30 | 6 | 2013 |

5 rows × 28 columns

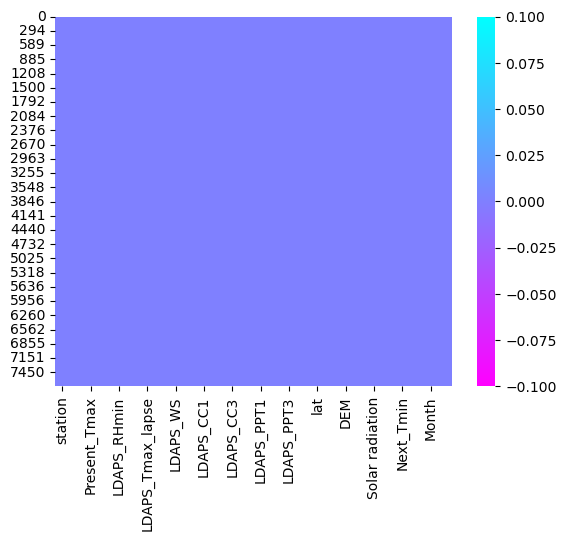
We will now visualize so there is really no null values in our dataset. In [15]:

# Now visualize using heatmap.

sns.heatmap(df.isnull(), cmap = "cool\_r")

Out[15]:

<Axes: >



Our heatmap turns out to be clean, showing there is no null values. our data is clean. Let’s check the value count of each column. In [16]:

# Checking value count for each column

for i in df.columns:

print(df[i].value\_counts())

print("\n")

station

25.0 307

10.0 307

23.0 307

18.0 307

17.0 307

2.0 307

3.0 306

4.0 306

13.0 305

16.0 304

20.0 304

21.0 304

14.0 303

1.0 303

9.0 302

12.0 302

15.0 302

19.0 302

22.0 302

24.0 302

11.0 301

7.0 301

6.0 301

8.0 300

5.0 296

Name: count, dtype: int64

Date

2013-06-30 25

2016-07-23 25

2015-08-22 25

2015-08-23 25

2015-08-24 25

..

2015-08-13 22

2015-07-24 22

2017-07-06 22

2017-08-23 22

2017-06-30 16

Name: count, Length: 307, dtype: int64

Present\_Tmax

31.4 112

29.4 108

29.1 106

29.2 105

30.6 105

...

21.2 1

20.1 1

36.4 1

20.3 1

20.0 1

Name: count, Length: 167, dtype: int64

Present\_Tmin

24.0 160

23.8 153

23.5 143

23.1 143

23.3 140

...

14.3 1

29.9 1

29.7 1

29.1 1

15.0 1

Name: count, Length: 155, dtype: int64

LDAPS\_RHmin

77.030350 2

51.810596 2

71.658089 2

58.255688 1

46.582764 1

..

58.950947 1

58.566174 1

59.790348 1

51.062355 1

22.933014 1

Name: count, Length: 7585, dtype: int64

LDAPS\_RHmax

92.531029 2

88.876610 2

85.863731 2

96.525200 2

96.058418 2

..

84.423607 1

81.810715 1

92.785057 1

91.507935 1

77.243744 1

Name: count, Length: 7577, dtype: int64

LDAPS\_Tmax\_lapse

28.074101 1

31.134897 1

33.447754 1

32.631448 1

30.795575 1

..

27.737869 1

28.321177 1

27.629996 1

28.775293 1

27.939516 1

Name: count, Length: 7588, dtype: int64

LDAPS\_Tmin\_lapse

23.006936 1

21.156704 1

24.222358 1

23.736497 1

23.064826 1

..

22.034331 1

21.008695 1

21.839880 1

22.247555 1

18.522965 1

Name: count, Length: 7588, dtype: int64

LDAPS\_WS

6.818887 1

5.483732 1

6.749639 1

6.404110 1

8.349506 1

..

7.595801 1

8.195843 1

7.852853 1

7.991329 1

7.289264 1

Name: count, Length: 7588, dtype: int64

LDAPS\_LH

69.451805 1

101.307648 1

16.204221 1

77.699182 1

119.922983 1

..

68.288728 1

72.907605 1

65.992804 1

35.616482 1

9.090034 1

Name: count, Length: 7588, dtype: int64

LDAPS\_CC1

0.000000 104

0.233947 1

0.280824 1

0.347618 1

0.372637 1

...

0.011633 1

0.020428 1

0.021683 1

0.008280 1

0.048954 1

Name: count, Length: 7485, dtype: int64

LDAPS\_CC2

0.000000 93

0.203896 1

0.196689 1

0.021183 1

0.001433 1

..

0.091687 1

0.086090 1

0.102609 1

0.076528 1

0.059869 1

Name: count, Length: 7496, dtype: int64

LDAPS\_CC3

0.000000e+00 75

8.680560e-04 2

5.810000e-07 2

1.616969e-01 1

8.667610e-02 1

..

6.616012e-01 1

6.588265e-01 1

5.463407e-01 1

5.650474e-01 1

5.850000e-07 1

Name: count, Length: 7512, dtype: int64

LDAPS\_CC4

0.000000 135

0.002604 7

0.001736 3

0.005208 3

0.373025 2

...

0.048167 1

0.081127 1

0.092123 1

0.124582 1

0.000796 1

Name: count, Length: 7438, dtype: int64

LDAPS\_PPT1

0.000000 4789

0.001953 11

0.002604 5

0.001734 3

0.002199 2

...

0.266814 1

0.262156 1

0.288735 1

0.279363 1

2.040502 1

Name: count, Length: 2779, dtype: int64

LDAPS\_PPT2

0.000000 5094

0.001953 7

0.000040 3

0.000781 3

0.000086 2

...

0.007427 1

0.087218 1

0.000695 1

0.047651 1

0.247997 1

Name: count, Length: 2479, dtype: int64

LDAPS\_PPT3

0.000000 5237

0.001953 10

0.000852 4

0.002604 3

0.001196 2

...

0.038018 1

0.028196 1

0.003034 1

0.121254 1

0.021030 1

Name: count, Length: 2326, dtype: int64

LDAPS\_PPT4

0.000000 5690

0.001953 3

0.001105 2

0.001909 2

0.000046 2

...

0.005363 1

0.001990 1

0.002798 1

0.016436 1

3.093815 1

Name: count, Length: 1892, dtype: int64

lat

37.5776 1214

37.5507 1204

37.6181 611

37.6046 610

37.5237 609

37.5372 608

37.4967 605

37.4697 604

37.5102 603

37.4562 307

37.4832 307

37.6450 306

Name: count, dtype: int64

lon

126.970 307

126.955 307

126.891 307

127.024 307

127.099 307

127.032 307

127.058 306

127.022 306

127.083 305

126.995 304

127.004 304

127.040 304

126.927 303

126.991 303

126.826 302

126.988 302

126.937 302

126.938 302

127.086 302

126.909 302

127.085 301

126.838 301

127.042 301

126.910 300

127.135 296

Name: count, dtype: int64

DEM

19.5844 307

208.5070 307

15.5876 307

56.4448 307

53.4712 307

44.7624 307

33.3068 306

45.7160 306

59.8324 305

82.2912 304

146.5540 304

26.2980 304

30.9680 303

212.3350 303

50.9312 302

132.1180 302

30.0464 302

75.0924 302

21.9668 302

17.2956 302

28.7000 301

12.3700 301

54.6384 301

52.5180 300

35.0380 296

Name: count, dtype: int64

Slope

0.2713 307

5.1782 307

0.1554 307

1.2313 307

0.6970 307

0.5141 307

0.2661 306

2.5348 306

2.6865 305

2.2579 304

4.7296 304

0.5721 304

0.6180 303

2.7850 303

0.4125 302

0.5931 302

0.8552 302

1.7678 302

0.1332 302

0.2223 302

0.6233 301

0.0985 301

0.1457 301

1.5629 300

0.5055 296

Name: count, dtype: int64

Solar radiation

5455.486816 5

5356.832520 5

5209.735352 5

4543.530273 5

5232.047852 5

..

4408.542480 1

4418.823730 1

4423.769531 1

4575.028809 1

4411.375977 1

Name: count, Length: 1575, dtype: int64

Next\_Tmax

29.3 113

33.0 104

31.3 100

29.6 98

31.2 97

...

37.5 1

38.7 1

37.4 1

20.9 1

17.4 1

Name: count, Length: 181, dtype: int64

Next\_Tmin

23.5 157

24.0 152

23.4 150

23.2 148

23.8 146

...

29.8 1

28.5 1

29.1 1

29.4 1

12.9 1

Name: count, Length: 157, dtype: int64

Day

30 366

27 250

19 250

1 249

21 249

8 249

17 249

16 249

29 248

28 248

22 248

15 248

4 248

7 248

12 247

14 247

9 247

3 247

26 247

11 246

2 246

18 246

5 246

13 245

23 245

24 245

25 245

6 244

20 225

10 222

31 99

Name: count, dtype: int64

Month

7 3820

8 3652

6 116

Name: count, dtype: int64

Year

2014 1547

2015 1533

2013 1510

2017 1506

2016 1492

Name: count, dtype: int64

# Description of Dataset[¶](#Description-of-Dataset)

Let’s see the description of our dataset. In [17]:

df.describe()

Out[17]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **...** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 7588.000000 | 7588 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | ... | 7588.000000 | 7588.00000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 | 7588.000000 |
| **mean** | 13.014101 | 2015-07-27 18:22:12.208750592 | 29.748366 | 23.195809 | 56.724969 | 88.360823 | 29.620128 | 23.511786 | 7.094097 | 62.492606 | ... | 37.544792 | 126.99142 | 61.918136 | 1.259755 | 5343.724208 | 30.241526 | 22.910820 | 15.939510 | 7.465999 | 2014.991697 |
| **min** | 1.000000 | 2013-06-30 00:00:00 | 20.000000 | 11.300000 | 19.794666 | 58.936283 | 17.624954 | 14.272646 | 2.882580 | -13.603212 | ... | 37.456200 | 126.82600 | 12.370000 | 0.098500 | 4329.520508 | 17.400000 | 11.300000 | 1.000000 | 6.000000 | 2013.000000 |
| **25%** | 7.000000 | 2014-07-15 00:00:00 | 27.800000 | 21.600000 | 45.960243 | 84.203724 | 27.673756 | 22.086820 | 5.675358 | 37.206201 | ... | 37.510200 | 126.93700 | 28.700000 | 0.271300 | 5001.485717 | 28.200000 | 21.300000 | 8.000000 | 7.000000 | 2014.000000 |
| **50%** | 13.000000 | 2015-07-29 00:00:00 | 29.900000 | 23.400000 | 55.023199 | 89.784122 | 29.709537 | 23.758249 | 6.547838 | 56.898324 | ... | 37.550700 | 126.99500 | 45.716000 | 0.618000 | 5441.987305 | 30.400000 | 23.100000 | 16.000000 | 7.000000 | 2015.000000 |
| **75%** | 19.000000 | 2016-08-14 00:00:00 | 32.000000 | 24.800000 | 67.115099 | 93.742725 | 31.711109 | 25.155660 | 8.028960 | 84.235666 | ... | 37.577600 | 127.04200 | 59.832400 | 1.767800 | 5729.485840 | 32.600000 | 24.600000 | 24.000000 | 8.000000 | 2016.000000 |
| **max** | 25.000000 | 2017-08-30 00:00:00 | 37.600000 | 29.900000 | 98.524734 | 100.000153 | 38.542255 | 29.619342 | 21.857621 | 213.414006 | ... | 37.645000 | 127.13500 | 212.335000 | 5.178200 | 5992.895996 | 38.900000 | 29.800000 | 31.000000 | 8.000000 | 2017.000000 |
| **std** | 7.217858 | NaN | 2.967401 | 2.400880 | 14.626559 | 7.199456 | 2.943496 | 2.342579 | 2.177034 | 33.686158 | ... | 0.050428 | 0.07922 | 54.323529 | 1.372748 | 429.782561 | 3.111807 | 2.482256 | 8.906521 | 0.528635 | 1.410877 |

8 rows × 28 columns We can see that all the values are present in all the columns. Maximum temperature was found out to be 37 degrees and mean temperature in this case was 29 degree Celsius. The Minimum temperature at highest point was found out to be 29 degrees while least temperature was 11 degrees. Also, average minimum temperature was 23 degree Celsius.

# Data Visualization[¶](#Data-Visualization)

Lets check the variation of Present\_Tmax by plotting a graph. In [18]:

# Plot histogram for Present\_Tmax

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

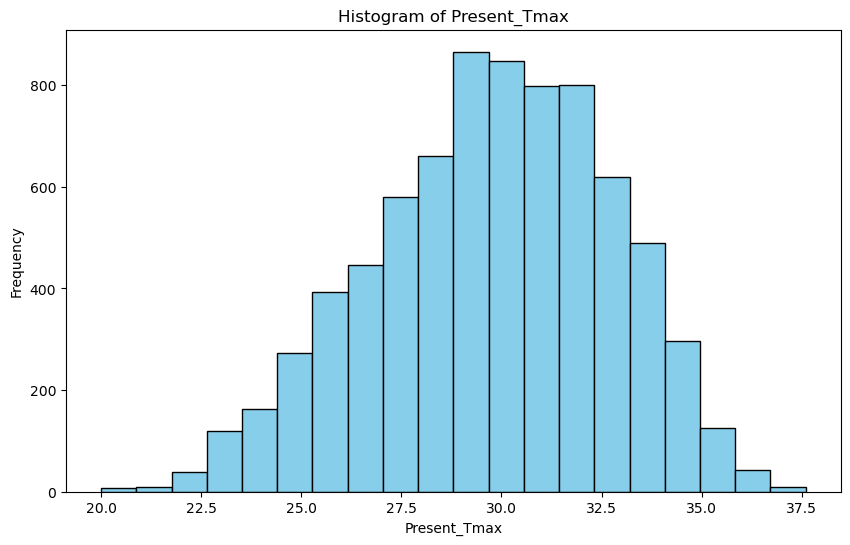
plt.hist(df['Present\_Tmax'], bins=20, color='skyblue', edgecolor='black')

plt.title('Histogram of Present\_Tmax')

plt.xlabel('Present\_Tmax')

plt.ylabel('Frequency')

plt.show()



We can see that the temperature ranges between 25 to 35 degrees. The distribution of data also looks evenly distributed. In [19]:

# Plot histogram for Present\_Tmin

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

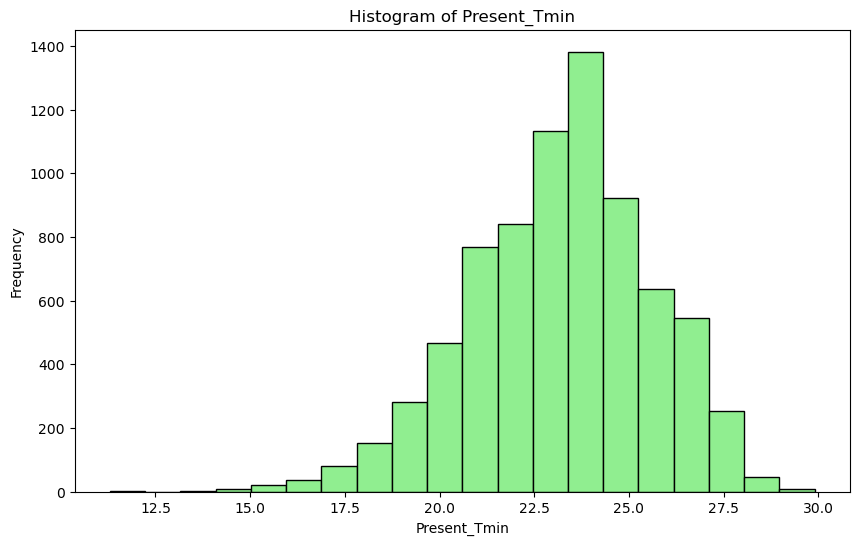
plt.hist(df['Present\_Tmin'], bins=20, color='lightgreen', edgecolor='black')

plt.title('Histogram of Present\_Tmin')

plt.xlabel('Present\_Tmin')

plt.ylabel('Frequency')

plt.show()



Here too the data looks evenly distributed. The temperature ranges from nearly 19 to 26 degree celsius. In [20]:

print(df.Present\_Tmax.max())

print(df.Present\_Tmax.min())

print(df.Present\_Tmax.mean())

37.6

20.0

29.748365840801263

In [21]:

print(df.Present\_Tmin.max())

print(df.Present\_Tmin.min())

print(df.Present\_Tmin.mean())

29.9

11.3

23.195809172377437

### Let's check where these maximum and minimum are present.[¶](#Let's-check-where-these-maximum-and-min)

In [22]:

pd.set\_option('display.max\_columns', None)

df.loc[df.Present\_Tmax==df.Present\_Tmax.max()]

Out[22]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **LDAPS\_CC1** | **LDAPS\_CC2** | **LDAPS\_CC3** | **LDAPS\_CC4** | **LDAPS\_PPT1** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5717** | 18.0 | 2016-08-11 | 37.6 | 26.8 | 44.254253 | 87.745514 | 34.794021 | 27.150764 | 6.366598 | 111.225118 | 0.218892 | 0.094288 | 0.004283 | 0.000343 | 0.0 | 0.0 | 0.0 | 0.0 | 37.4832 | 127.024 | 56.4448 | 1.2313 | 5082.563477 | 37.0 | 27.8 | 11 | 8 | 2016 |

Present maximum temperature Maxima was recorded on station 18 on 2016-08-11 with temperature of 37.6 degrees.

In [23]:

df.loc[df.Present\_Tmax==df.Present\_Tmax.min()]

Out[23]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **LDAPS\_CC1** | **LDAPS\_CC2** | **LDAPS\_CC3** | **LDAPS\_CC4** | **LDAPS\_PPT1** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **7725** | 1.0 | 2017-08-30 | 20.0 | 15.1 | 35.652172 | 89.97319 | 24.323737 | 16.128899 | 7.087329 | 108.981108 | 0.046182 | 0.014955 | 0.0 | 0.00063 | 0.0 | 0.0 | 0.0 | 0.0 | 37.6046 | 126.991 | 212.335 | 2.785 | 4614.76123 | 23.8 | 15.1 | 30 | 8 | 2017 |

Present maximum temperature Minima was recorded at station 1 on 2017-08-30 with temperature of 20 degrees.

In [24]:

df.loc[df.Present\_Tmin==df.Present\_Tmin.max()]

Out[24]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **LDAPS\_CC1** | **LDAPS\_CC2** | **LDAPS\_CC3** | **LDAPS\_CC4** | **LDAPS\_PPT1** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2397** | 23.0 | 2014-08-02 | 35.3 | 29.9 | 53.946949 | 85.985161 | 30.912804 | 25.439537 | 13.011129 | 117.837212 | 0.684685 | 0.448827 | 0.762858 | 0.635728 | 1.670126 | 0.005681 | 0.248885 | 0.003176 | 37.5372 | 126.891 | 15.5876 | 0.1554 | 5360.226563 | 31.3 | 24.8 | 2 | 8 | 2014 |

Present minimum temperature Maxima was recorded at station 23 on 2014-08-02 with temperature of 29.9 degrees.

In [25]:

df.loc[df.Present\_Tmin==df.Present\_Tmin.min()]

Out[25]:

|  | **station** | **Date** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **LDAPS\_CC1** | **LDAPS\_CC2** | **LDAPS\_CC3** | **LDAPS\_CC4** | **LDAPS\_PPT1** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **6116** | 17.0 | 2016-08-27 | 27.1 | 11.3 | 62.793823 | 91.726936 | 23.529546 | 17.963487 | 7.984566 | 84.48145 | 0.668264 | 0.410536 | 0.452879 | 0.627238 | 0.181458 | 0.0 | 0.405181 | 1.015573 | 37.6181 | 127.099 | 53.4712 | 0.697 | 4539.616699 | 24.6 | 17.1 | 27 | 8 | 2016 |

Present minimum temperature Minima was recorded at station 17 on 2016-08-27 with the temperature of 11.3 degrees only.

In [26]:

# Plot scatter plot for Station vs Present\_Tmax

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

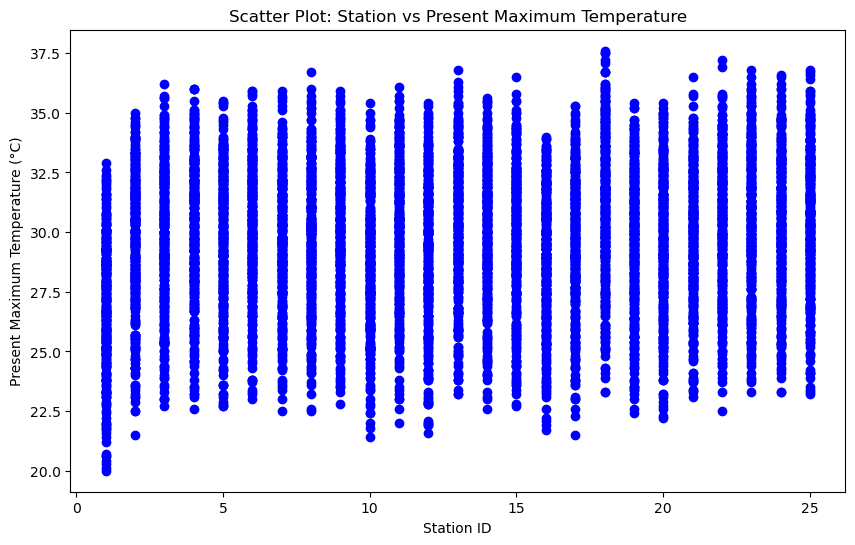
plt.scatter(df['station'], df['Present\_Tmax'], color='blue')

plt.title('Scatter Plot: Station vs Present Maximum Temperature')

plt.xlabel('Station ID')

plt.ylabel('Present Maximum Temperature (°C)')

plt.show()



We can see that the minimum temperature was at station 1 and maximum temperature was at station 18.

In [27]:

plt.figure(figsize=(20,10))

sns.set\_style('whitegrid')

sns.pointplot(x='station', y='Present\_Tmax', data=df, hue='Year',join=False)

plt.title('Present Maximum Temperature Per Year for each Station', fontsize=20, fontweight='bold')

plt.xlabel('Station ID',{'fontsize':15,'fontweight' :'bold'})

plt.ylabel('Present Maximum Temperature',{'fontsize':15,'fontweight' :'bold'})

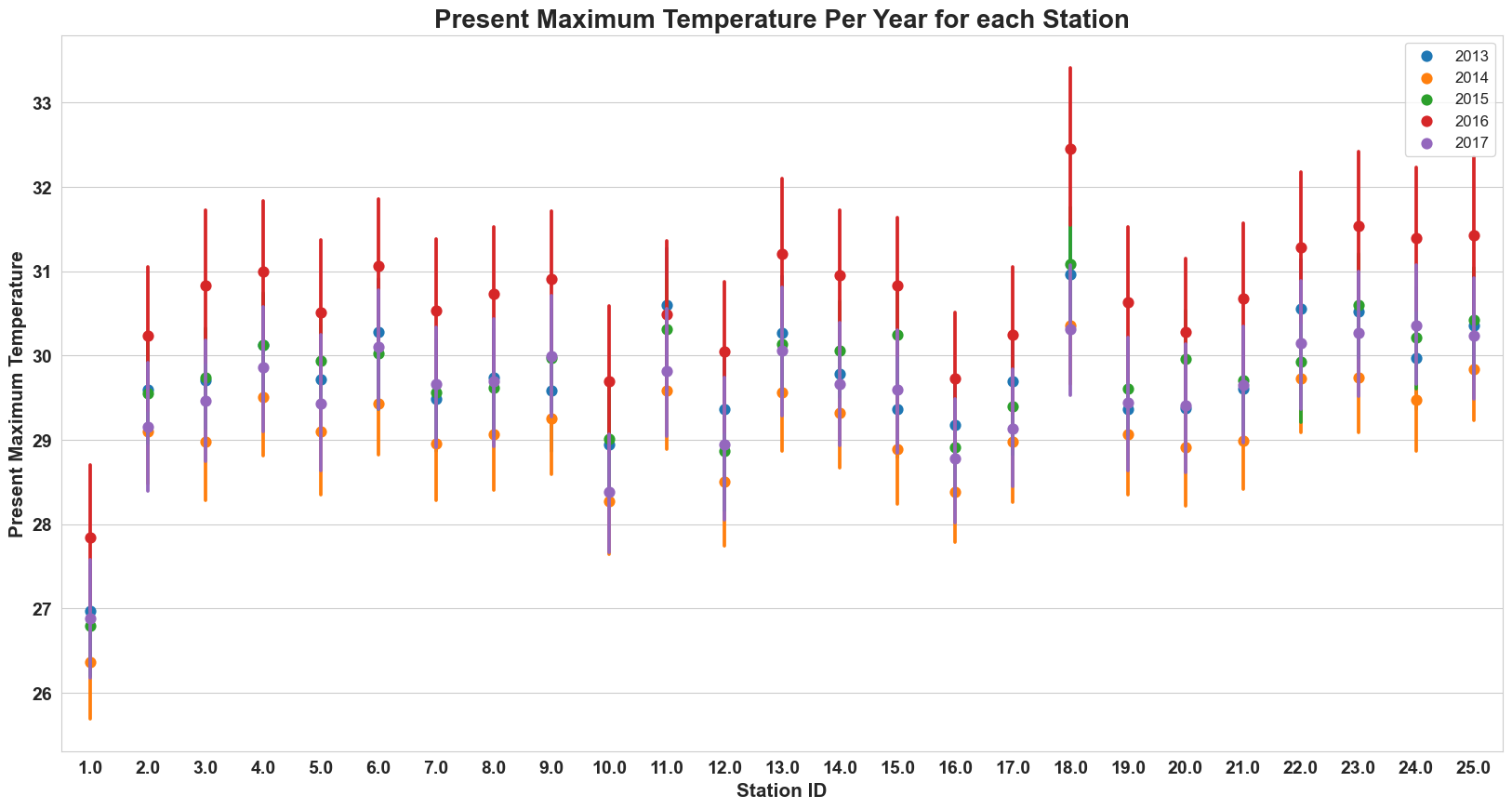
plt.xticks(fontsize=14,fontweight ='bold')

plt.yticks(fontsize=14,fontweight ='bold')

plt.legend(fontsize=12)

Out[27]:

<matplotlib.legend.Legend at 0x1ea764572d0>



We can clearly see that year 2016 is the hottest year. Year 2014 appears to be cooler followed by year 2017.

In [28]:

plt.figure(figsize=(20,10))

sns.set\_style('whitegrid')

sns.pointplot(x='station', y='Present\_Tmin', data=df, hue='Year',join=False)

plt.title('Present Minimum Temperature Per Year for each Station', fontsize=20, fontweight='bold')

plt.xlabel('Station ID',{'fontsize':15,'fontweight' :'bold'})

plt.ylabel('Present Minimum Temperature',{'fontsize':16,'fontweight' :'bold'})

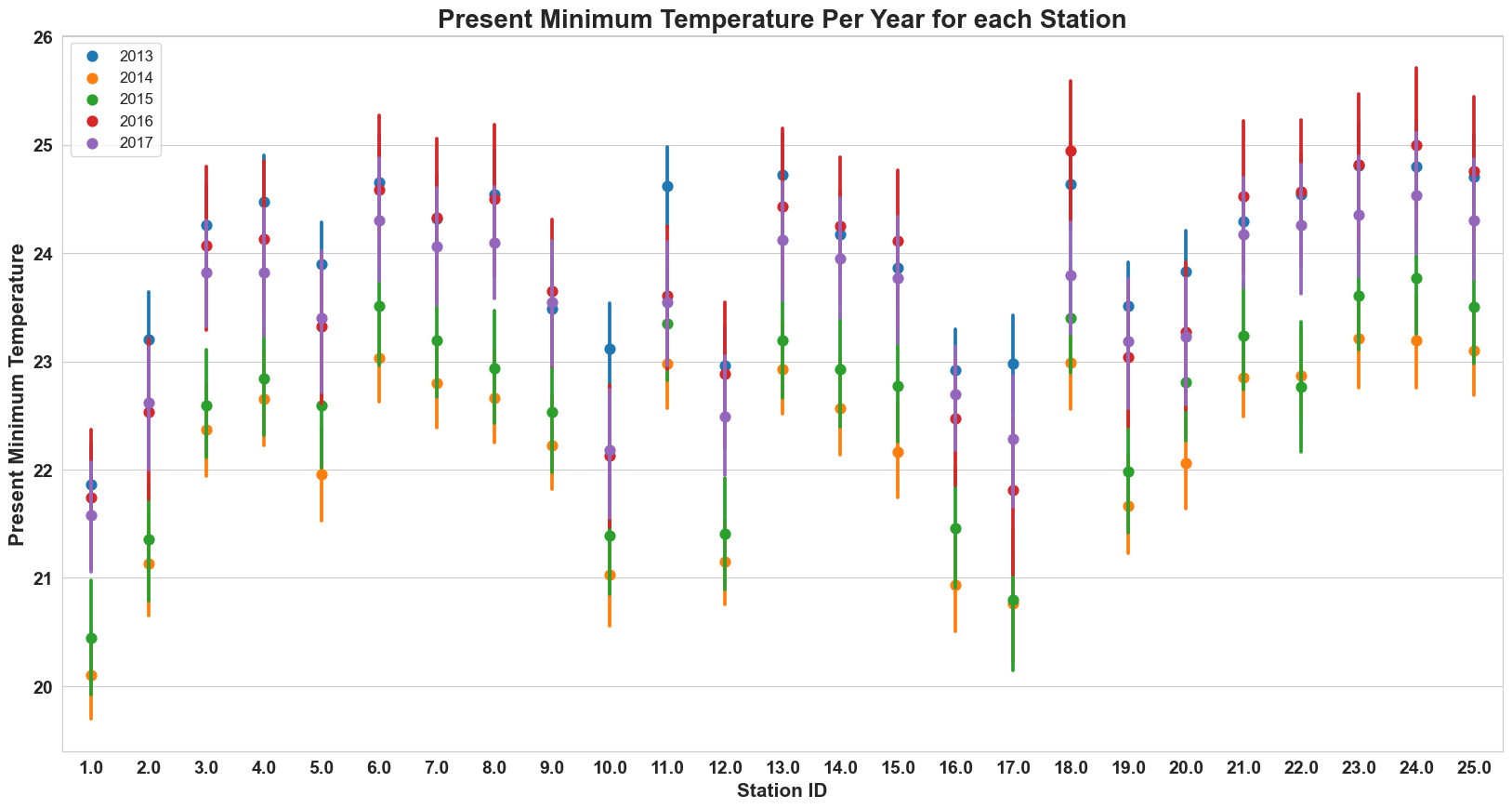
plt.xticks(fontsize=14,fontweight ='bold')

plt.yticks(fontsize=14,fontweight ='bold')

plt.legend(fontsize=12)

Out[28]:

<matplotlib.legend.Legend at 0x1ea79c8ca90>



Year 2014 seems to be the coolest year among the rest.

In [29]:

# Plot histogram for Next\_Tmax

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

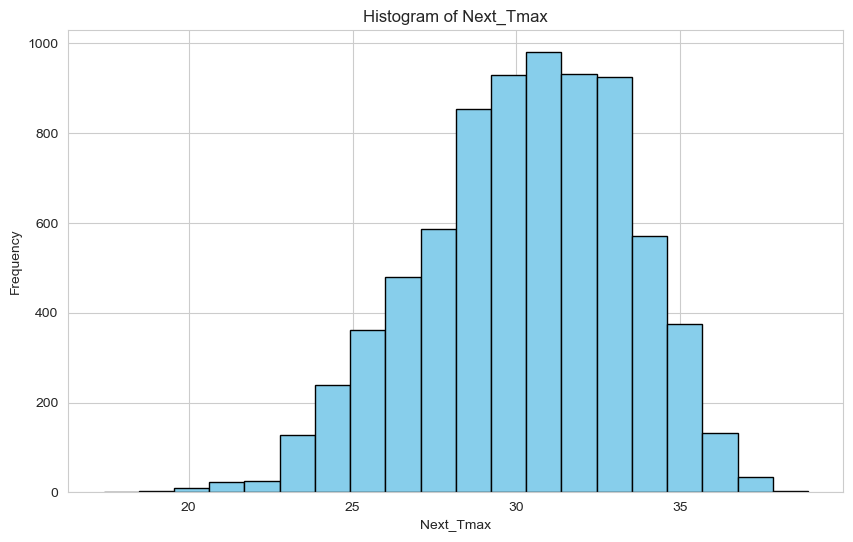
plt.hist(df['Next\_Tmax'], bins=20, color='skyblue', edgecolor='black')

plt.title('Histogram of Next\_Tmax')

plt.xlabel('Next\_Tmax')

plt.ylabel('Frequency')

plt.show()



Here the data distribution of Next\_Tmax looks evenly distributed. In [30]:

# Plot histogram for Next\_Tmin

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

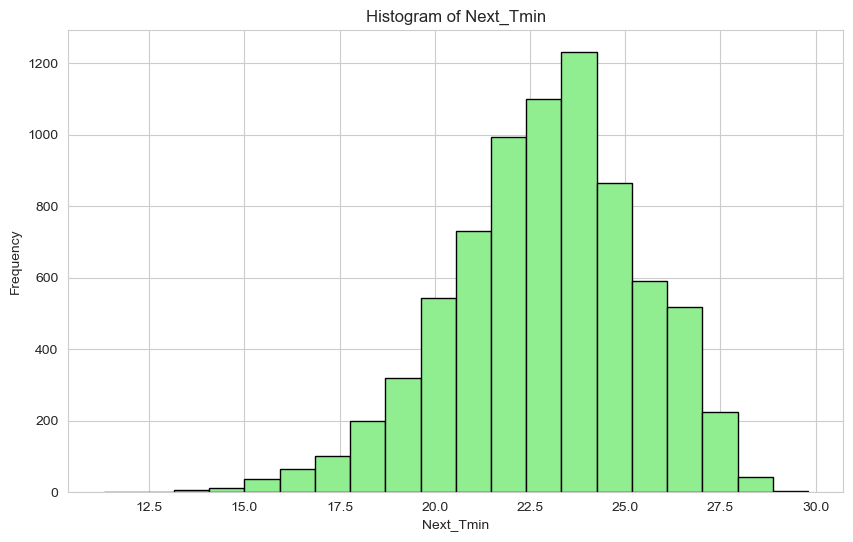
plt.hist(df['Next\_Tmin'], bins=20, color='lightgreen', edgecolor='black')

plt.title('Histogram of Next\_Tmin')

plt.xlabel('Next\_Tmin')

plt.ylabel('Frequency')

plt.show()



Also the plot for Next\_Tmin looks perfectly distributed showing that the outliers are not here. In [31]:

# Plot scatter plot for Station vs Next\_Tmax

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

plt.scatter(df['station'], df['Next\_Tmax'], color='blue')

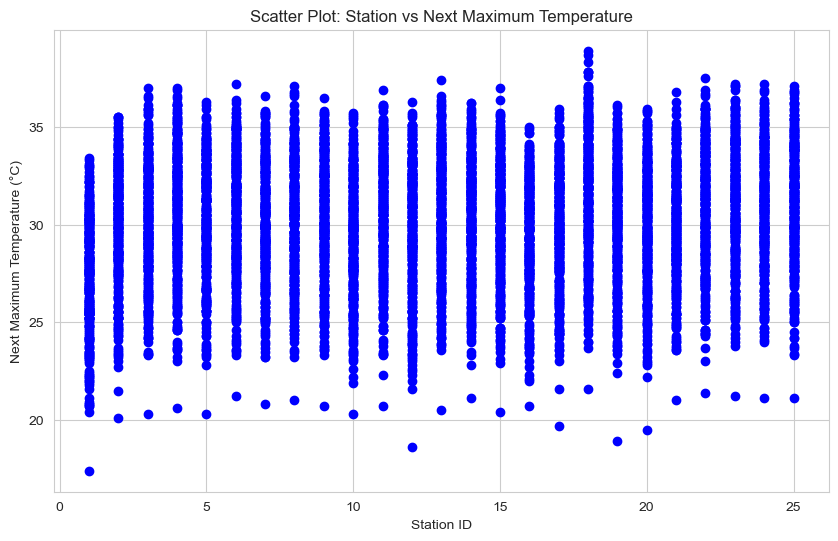
plt.title('Scatter Plot: Station vs Next Maximum Temperature')

plt.xlabel('Station ID')

plt.ylabel('Next Maximum Temperature (°C)')

plt.grid(True)

plt.show()



In [32]:

# Plot scatter plot for Station vs Next\_Tmin

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

plt.scatter(df['station'], df['Next\_Tmin'], color='green')

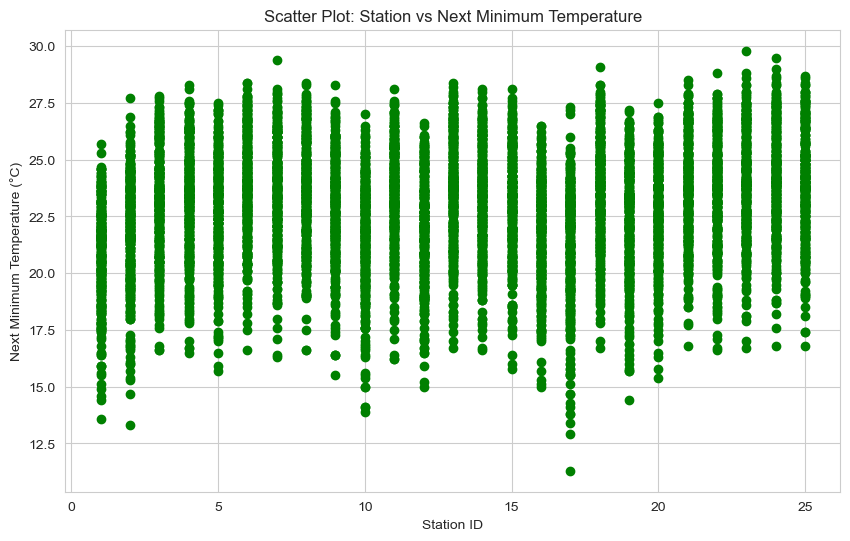
plt.title('Scatter Plot: Station vs Next Minimum Temperature')

plt.xlabel('Station ID')

plt.ylabel('Next Minimum Temperature (°C)')

plt.grid(True)

plt.show()



We can see that station 18 has the maximum temperature in both cases.

In [33]:

# Plot scatter plot for Next Maximum Temperature vs Cloud Cover

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

plt.scatter(df['LDAPS\_CC1'], df['Next\_Tmax'], color='green', label='LDAPS\_CC1')

plt.scatter(df['LDAPS\_CC2'], df['Next\_Tmax'], color='blue', label='LDAPS\_CC2')

plt.scatter(df['LDAPS\_CC3'], df['Next\_Tmax'], color='red', label='LDAPS\_CC3')

plt.scatter(df['LDAPS\_CC4'], df['Next\_Tmax'], color='orange', label='LDAPS\_CC4')

plt.title('Scatter Plot: Next Maximum Temperature vs Cloud Cover')

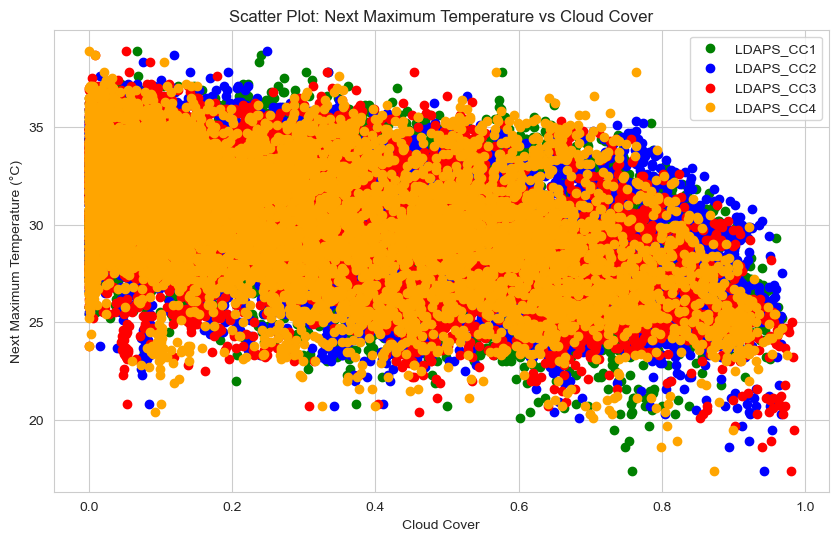
plt.xlabel('Cloud Cover')

plt.ylabel('Next Maximum Temperature (°C)')

plt.legend()

plt.grid(True)

plt.show()



We can see that as the cloud cover increases, the next day maximum temperature decreases.

In [34]:

# Plot scatter plot for Next\_Tmin vs Next-day average cloud cover

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

plt.scatter(df['Next\_Tmin'], df['LDAPS\_CC1'], color='blue', label='LDAPS\_CC1')

plt.scatter(df['Next\_Tmin'], df['LDAPS\_CC2'], color='green', label='LDAPS\_CC2')

plt.scatter(df['Next\_Tmin'], df['LDAPS\_CC3'], color='red', label='LDAPS\_CC3')

plt.scatter(df['Next\_Tmin'], df['LDAPS\_CC4'], color='orange', label='LDAPS\_CC4')

plt.title('Scatter Plot: Next\_Tmin vs Next-day average cloud cover')

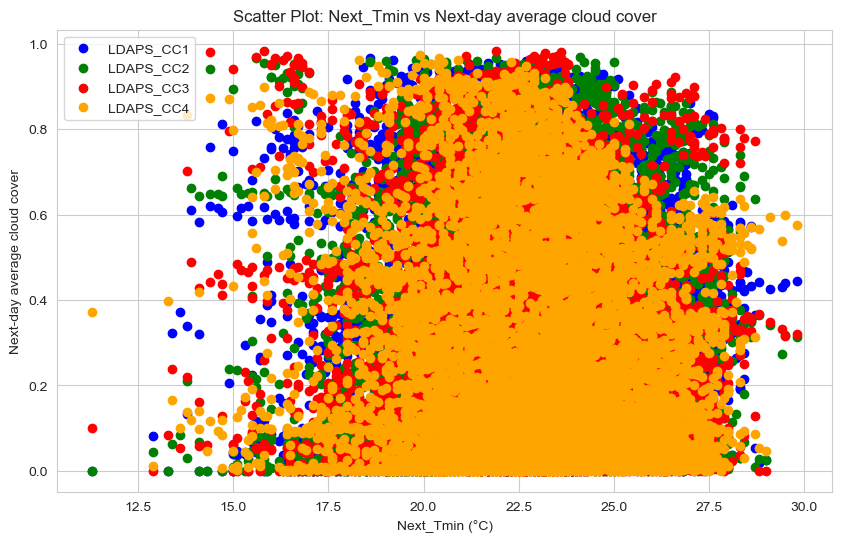
plt.xlabel('Next\_Tmin (°C)')

plt.ylabel('Next-day average cloud cover')

plt.legend()

plt.grid(True)

plt.show()



Not much significant relation is seen.

In [35]:

# Plot scatter plot for Solar Radiation vs Next-day average cloud cover

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

plt.scatter(df['Solar radiation'], df['LDAPS\_CC1'], color='blue', label='LDAPS\_CC1')

plt.scatter(df['Solar radiation'], df['LDAPS\_CC2'], color='green', label='LDAPS\_CC2')

plt.scatter(df['Solar radiation'], df['LDAPS\_CC3'], color='red', label='LDAPS\_CC3')

plt.scatter(df['Solar radiation'], df['LDAPS\_CC4'], color='orange', label='LDAPS\_CC4')

plt.title('Scatter Plot: Solar Radiation vs Next-day average cloud cover')

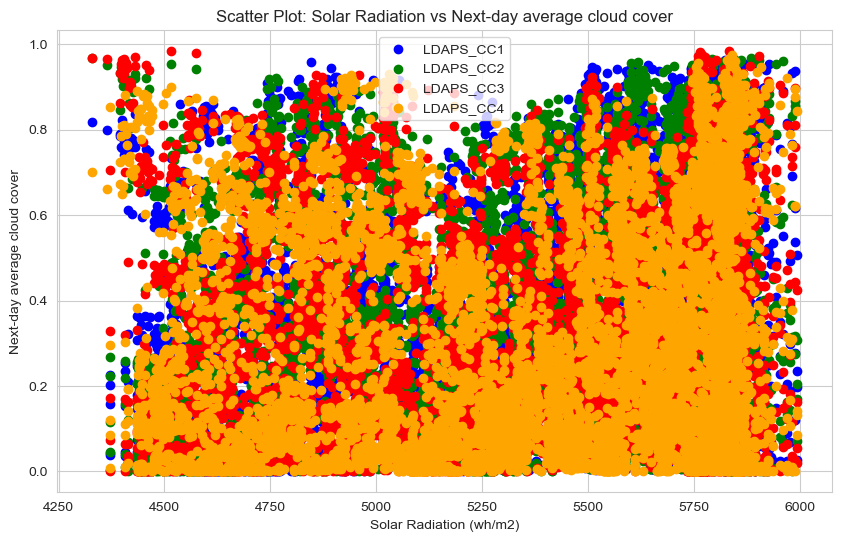
plt.xlabel('Solar Radiation (wh/m2)')

plt.ylabel('Next-day average cloud cover')

plt.legend()

plt.grid(True)

plt.show()



In [36]:

# Plot scatter plot for Solar Radiation vs Next-day Precipitation

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

plt.scatter(df['Solar radiation'], df['LDAPS\_PPT1'], color='blue', label='LDAPS\_PPT1')

plt.scatter(df['Solar radiation'], df['LDAPS\_PPT2'], color='green', label='LDAPS\_PPT2')

plt.scatter(df['Solar radiation'], df['LDAPS\_PPT3'], color='red', label='LDAPS\_PPT3')

plt.scatter(df['Solar radiation'], df['LDAPS\_PPT4'], color='orange', label='LDAPS\_PPT4')

plt.title('Scatter Plot: Solar Radiation vs Next-day Precipitation')

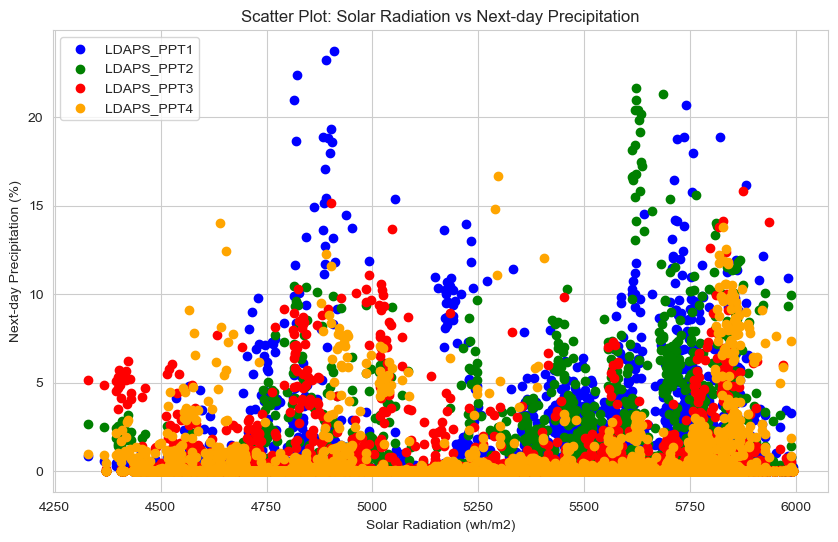
plt.xlabel('Solar Radiation (wh/m2)')

plt.ylabel('Next-day Precipitation (%)')

plt.legend()

plt.grid(True)

plt.show()



We see that the solar radiation value nearly greater than 5500 wh/m2 leads to higher amount of precipitation.

Also Let’s see the plot of distribution of Solar radiation. In [37]:

plt.figure(figsize=(8,6))

sns.histplot(df['Solar radiation'],kde=True,color='y')

print('Minimum Solar radiation :',df['Solar radiation'].min())

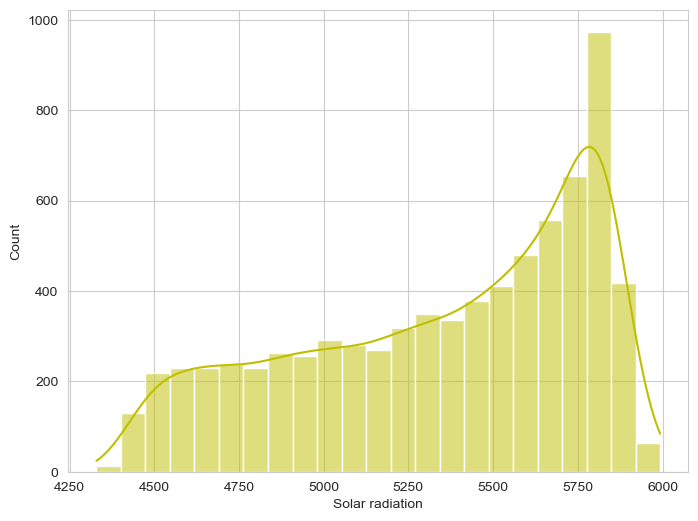
print('Maximum Solar radiation :',df['Solar radiation'].max())

print('Average Solar radiation :',df['Solar radiation'].mean())

Minimum Solar radiation : 4329.520508

Maximum Solar radiation : 5992.895996

Average Solar radiation : 5343.724207856747



In [38]:

plt.figure(figsize=(13,8))

plt.subplot(1,2,1)

sns.histplot(df['LDAPS\_RHmax'],kde=True,color='r')

plt.title('Maximum relative humidity (%)',fontsize=12, fontweight='bold')

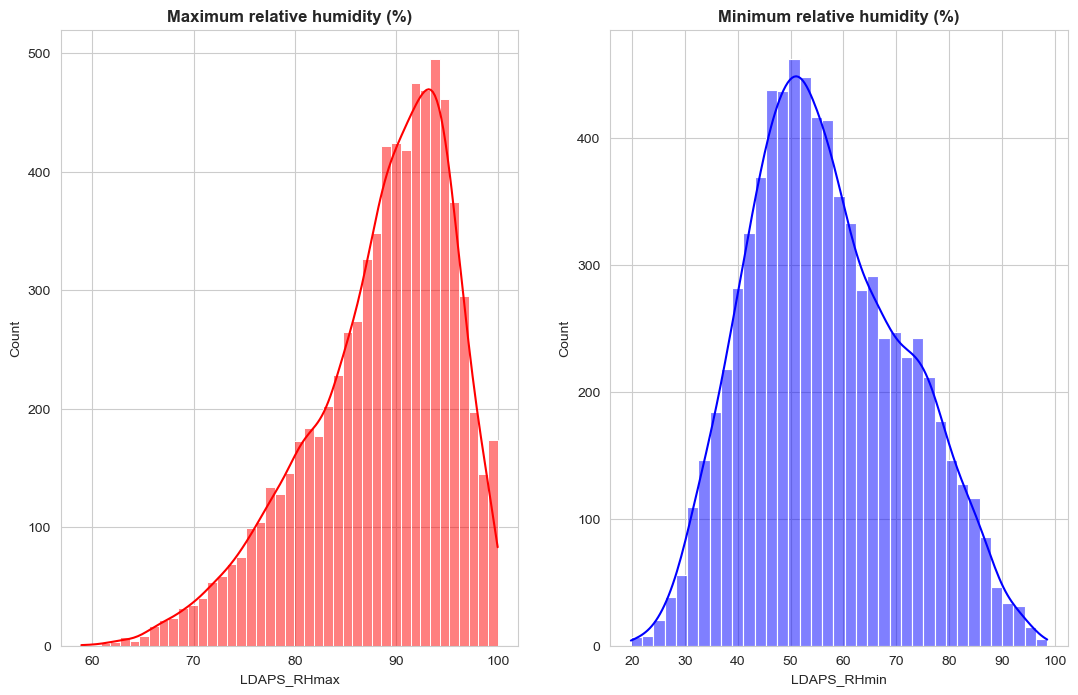
plt.subplot(1,2,2)

sns.histplot(df['LDAPS\_RHmin'],kde=True,color='b')

plt.title('Minimum relative humidity (%)',fontsize=12, fontweight='bold')

Out[38]:

Text(0.5, 1.0, 'Minimum relative humidity (%)')



Maximum relative humidity can be seen in the range of 90-97 and minimum relative humidity lies in the range 40-60.

In [39]:

plt.figure(figsize=(17,8))

plt.subplot(1,2,1)

sns.histplot(df['LDAPS\_Tmax\_lapse'],kde=True,color='purple')

plt.title('Maximum Tmax applied lapse rate',fontsize=16, fontweight='bold')

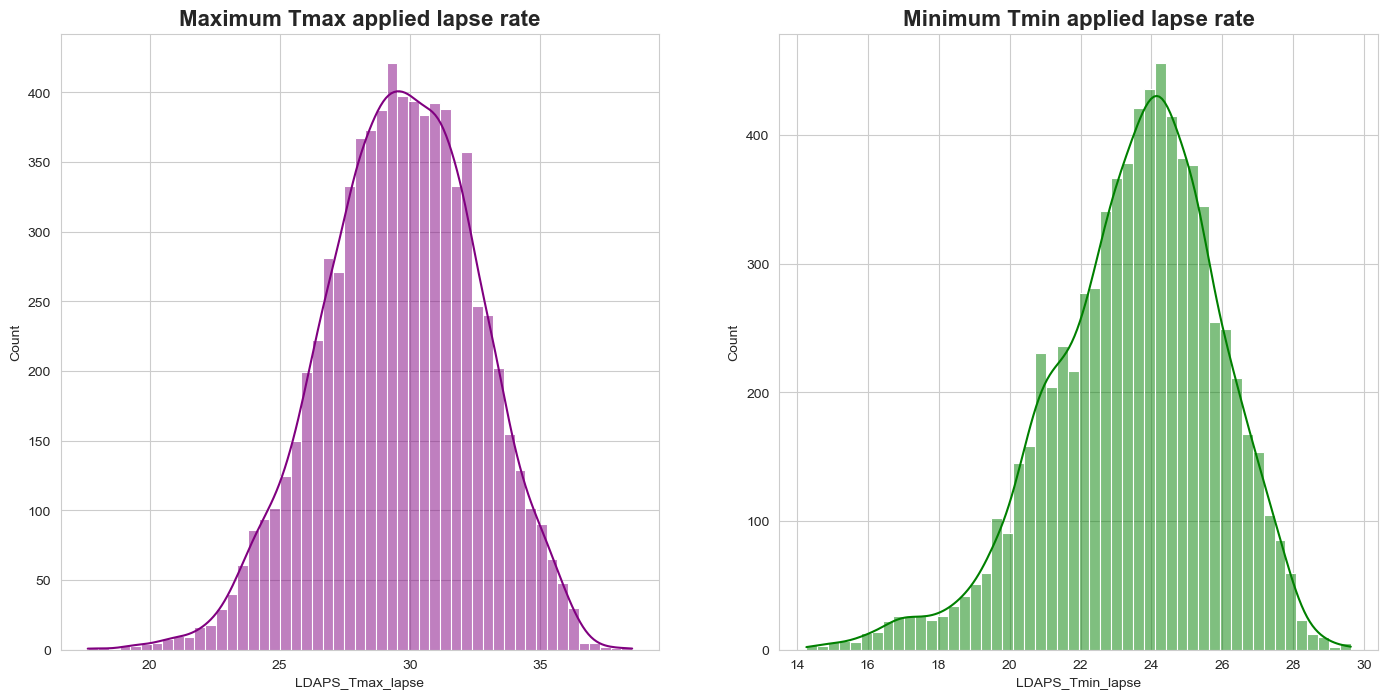
plt.subplot(1,2,2)

sns.histplot(df['LDAPS\_Tmin\_lapse'],kde=True,color='green')

plt.title('Minimum Tmin applied lapse rate',fontsize=16, fontweight='bold')

Out[39]:

Text(0.5, 1.0, 'Minimum Tmin applied lapse rate')



Tmax\_lapse for majority days is between 28-34 degrees, while that for Tmin\_lapse the range lies in between 22-26.

**Feature Engineering: Data Pre-processing**

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

Feature Engineering is very important step in building Machine Learning model. Some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used. In Feature engineering can be done for various reason. Some are:

1. **Feature Importance**: An estimate of the usefulness of a feature
2. **Feature Extraction**: The automatic construction of new features from raw data (Dimensionality reduction Technique like PCA)
3. **Feature Selection**: From many features to a few that are useful
4. **Feature Construction**: The manual construction of new features from raw data (For example, construction of new column for month out date - mm/dd/yy).

Some techniques are important and involves:

1. Handling missing values
2. Handling imbalanced data using SMOTE
3. Outliers’ detection and removal using Z-score, IQR
4. Scaling of data using Standard Scalar or Minmax Scalar
5. Encoding categorical data using one hot encoding, label / ordinal encoding
6. Skewness correction using Boxcox or yeo-Johnson method
7. Handling Multicollinearity among feature using variance inflation factor

### Dropping Unnecessary columns

In [40]:

df.drop(['Date'], axis=1, inplace=True)

# Outlier detection and handling[¶](#Outlier-detection-and-handling)

Machine learning algorithms are sensitive to the range and distribution of attribute values. Data outliers can spoil and mislead the training process resulting in longer training times, less accurate models and ultimately poorer results. Outliers can be seen in boxplot of numerical feature. We did not added boxplot here as it will make this article length, I left it to reader to further investigate. Now we will use Z-score method for outliers’ detection. Now we will check outliers to make our data clean.

In [41]:

plt.figure(figsize=(20,20), facecolor='white')

plotnumber =1

for column in df:

if plotnumber <=31:

ax = plt.subplot(10,3,plotnumber)

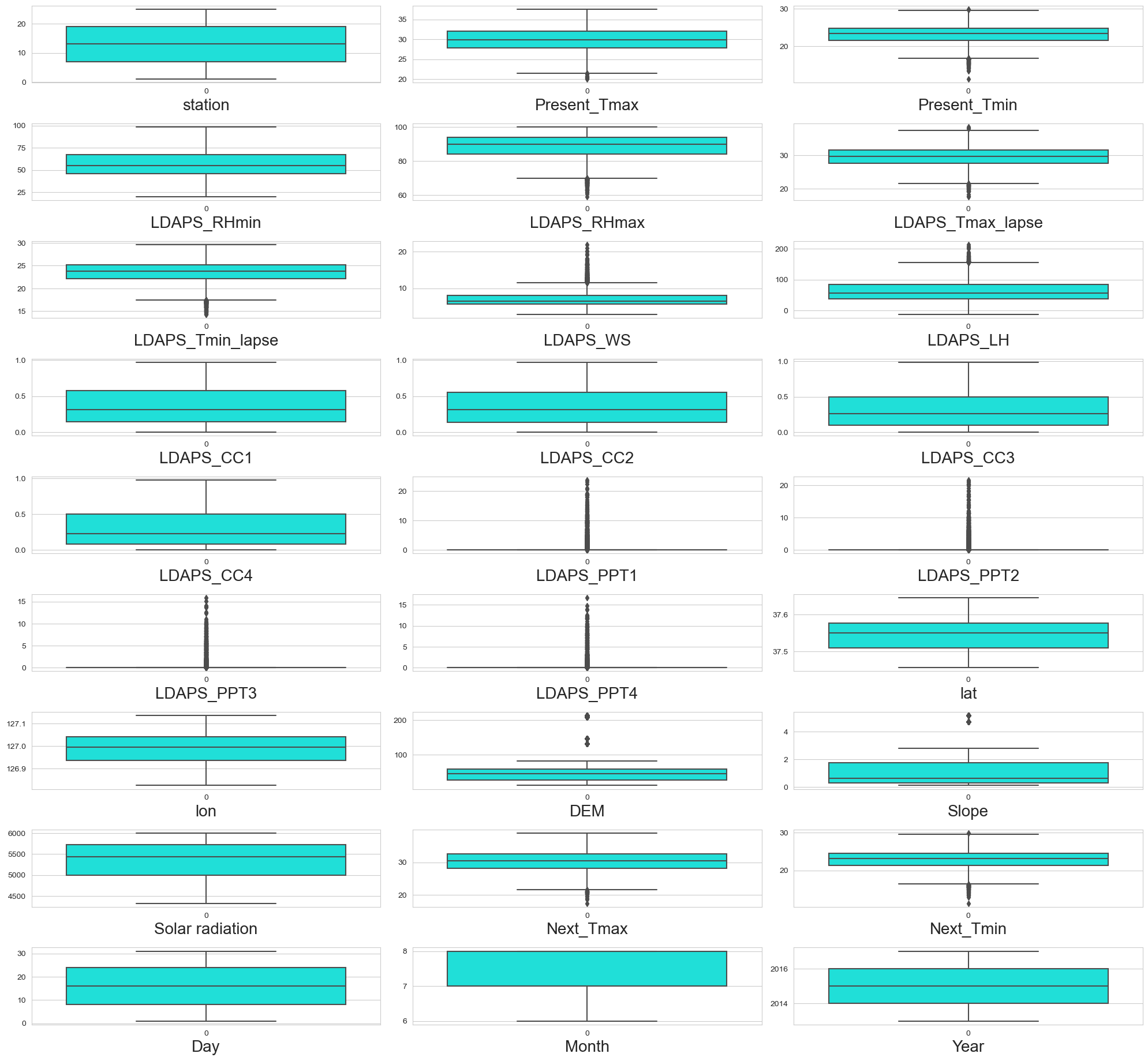
sns.boxplot(df[column], palette='hsv')

plt.xlabel(column,fontsize=20)

plotnumber+=1

plt.tight\_layout()

plt.show()



Let's proceed to remove these outliers.

In [42]:

from scipy.stats import zscore

z = np.abs(zscore(df))

threshold = 3

In [43]:

df=df[(z<3).all(axis = 1)]

df.shape

Out[43]:

(6739, 27)

In [44]:

# Lets calculate percentage data loss.

((7588-6739)/7588)\*100

Out[44]:

11.188719030047443

We can see that we have lost nearly 11% of our data, but since our dataset is large and now too we have many rows present.

### Skewness Check[¶](#Skewness-Check)

We will now check the skewness of our data. In [45]:

df.skew()

Out[45]:

station -0.001594

Present\_Tmax -0.269936

Present\_Tmin -0.221250

LDAPS\_RHmin 0.309567

LDAPS\_RHmax -0.686083

LDAPS\_Tmax\_lapse -0.110563

LDAPS\_Tmin\_lapse -0.379269

LDAPS\_WS 1.085685

LDAPS\_LH 0.567050

LDAPS\_CC1 0.594835

LDAPS\_CC2 0.505774

LDAPS\_CC3 0.701288

LDAPS\_CC4 0.708754

LDAPS\_PPT1 3.724580

LDAPS\_PPT2 4.854967

LDAPS\_PPT3 5.516862

LDAPS\_PPT4 5.924324

lat 0.106983

lon -0.277547

DEM 1.764698

Slope 1.590130

Solar radiation -0.520157

Next\_Tmax -0.267526

Next\_Tmin -0.234328

Day 0.000057

Month -0.168571

Year 0.025066

dtype: float64

Since our skewness value has all positive, negative and zero values, let’s use yeo-johnson power transformation method.

In [46]:

# Removing skewness using yeo-johnson method to get better prediction

skew = ['LDAPS\_RHmax','LDAPS\_Tmin\_lapse','LDAPS\_WS','LDAPS\_LH','LDAPS\_CC3','LDAPS\_CC4','LDAPS\_PPT1','LDAPS\_PPT2','LDAPS\_PPT3','LDAPS\_PPT4','DEM','Slope','Solar radiation','Month']

from sklearn.preprocessing import PowerTransformer

scaler = PowerTransformer(method='yeo-johnson')

In [47]:

df[skew] = scaler.fit\_transform(df[skew].values)

In [48]:

df.skew()

Out[48]:

station -0.001594

Present\_Tmax -0.269936

Present\_Tmin -0.221250

LDAPS\_RHmin 0.309567

LDAPS\_RHmax -0.103320

LDAPS\_Tmax\_lapse -0.110563

LDAPS\_Tmin\_lapse -0.027814

LDAPS\_WS 0.007615

LDAPS\_LH -0.030736

LDAPS\_CC1 0.594835

LDAPS\_CC2 0.505774

LDAPS\_CC3 0.094449

LDAPS\_CC4 0.142061

LDAPS\_PPT1 1.617538

LDAPS\_PPT2 1.820237

LDAPS\_PPT3 1.995359

LDAPS\_PPT4 2.267954

lat 0.106983

lon -0.277547

DEM 0.038830

Slope 0.233799

Solar radiation -0.172156

Next\_Tmax -0.267526

Next\_Tmin -0.234328

Day 0.000057

Month -0.087154

Year 0.025066

dtype: float64

For some values we weren't able to remove skewness completely but we reduced them and looks much better than previous one.

# Correlation between Target variable and independent variable[¶](#Correlation-between-Target-variable-and)

Correlation Heatmap show in a glance which variables are correlated, to what degree, in which direction, and alerts us to potential multicollinearity problems. The bar plot of correlation coefficient of target variable with independent features shown below.

In [49]:

df.corr()

Out[49]:

|  | **station** | **Present\_Tmax** | **Present\_Tmin** | **LDAPS\_RHmin** | **LDAPS\_RHmax** | **LDAPS\_Tmax\_lapse** | **LDAPS\_Tmin\_lapse** | **LDAPS\_WS** | **LDAPS\_LH** | **LDAPS\_CC1** | **LDAPS\_CC2** | **LDAPS\_CC3** | **LDAPS\_CC4** | **LDAPS\_PPT1** | **LDAPS\_PPT2** | **LDAPS\_PPT3** | **LDAPS\_PPT4** | **lat** | **lon** | **DEM** | **Slope** | **Solar radiation** | **Next\_Tmax** | **Next\_Tmin** | **Day** | **Month** | **Year** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **station** | 1.000000 | 0.110291 | 0.133638 | -0.069582 | -0.182354 | 0.066863 | 0.108664 | 0.030231 | -0.135717 | 0.008671 | 0.006380 | 0.006303 | 0.011510 | -0.001774 | -0.006798 | -0.004884 | -0.000285 | -0.241811 | -0.122829 | -0.326019 | -0.146034 | -0.034787 | 0.106378 | 0.129110 | 0.005032 | 0.011309 | 0.006658 |
| **Present\_Tmax** | 0.110291 | 1.000000 | 0.610428 | -0.154266 | -0.320074 | 0.540680 | 0.623443 | -0.071596 | 0.119022 | -0.290078 | -0.157917 | -0.061417 | -0.049978 | -0.190222 | -0.097425 | -0.022348 | -0.021344 | -0.052041 | 0.011132 | -0.130892 | -0.092409 | -0.099696 | 0.586943 | 0.615950 | -0.130708 | 0.171614 | 0.099185 |
| **Present\_Tmin** | 0.133638 | 0.610428 | 1.000000 | 0.151385 | -0.053421 | 0.441577 | 0.764041 | 0.034364 | -0.008556 | 0.090828 | 0.117894 | 0.043629 | 0.000193 | 0.068506 | 0.090113 | 0.035889 | -0.063495 | -0.072707 | -0.043201 | -0.238916 | -0.157839 | -0.047145 | 0.438813 | 0.785946 | -0.082712 | 0.104576 | 0.099222 |
| **LDAPS\_RHmin** | -0.069582 | -0.154266 | 0.151385 | 1.000000 | 0.566069 | -0.555583 | 0.114069 | 0.171684 | -0.002412 | 0.574389 | 0.713238 | 0.636195 | 0.459528 | 0.410985 | 0.510241 | 0.405612 | 0.244139 | 0.090054 | -0.093969 | 0.057460 | 0.083910 | 0.229070 | -0.433872 | 0.134518 | -0.073118 | -0.183976 | 0.001709 |
| **LDAPS\_RHmax** | -0.182354 | -0.320074 | -0.053421 | 0.566069 | 1.000000 | -0.403634 | -0.182113 | 0.060965 | 0.285942 | 0.425827 | 0.372085 | 0.143375 | 0.046555 | 0.420202 | 0.368154 | 0.219383 | 0.118163 | 0.232258 | 0.013059 | 0.174176 | 0.209222 | 0.154031 | -0.323446 | -0.110402 | -0.014136 | -0.134357 | -0.146019 |
| **LDAPS\_Tmax\_lapse** | 0.066863 | 0.540680 | 0.441577 | -0.555583 | -0.403634 | 1.000000 | 0.624572 | -0.154400 | 0.005878 | -0.416255 | -0.492392 | -0.468317 | -0.367547 | -0.307565 | -0.344808 | -0.285371 | -0.207369 | -0.038285 | 0.106354 | -0.110286 | -0.105094 | -0.034699 | 0.816861 | 0.553227 | -0.139339 | 0.115992 | 0.066794 |
| **LDAPS\_Tmin\_lapse** | 0.108664 | 0.623443 | 0.764041 | 0.114069 | -0.182113 | 0.624572 | 1.000000 | -0.002164 | -0.143246 | 0.023128 | 0.087752 | 0.032023 | -0.017483 | -0.054475 | 0.021320 | -0.018357 | -0.084124 | -0.091796 | -0.025047 | -0.174769 | -0.160165 | 0.009890 | 0.550810 | 0.873678 | -0.141873 | 0.083247 | 0.061339 |
| **LDAPS\_WS** | 0.030231 | -0.071596 | 0.034364 | 0.171684 | 0.060965 | -0.154400 | -0.002164 | 1.000000 | -0.056362 | 0.203057 | 0.156336 | 0.065764 | 0.090955 | 0.197126 | 0.166155 | 0.089355 | 0.167950 | -0.027463 | -0.077686 | 0.113157 | 0.095385 | 0.164626 | -0.205979 | 0.023330 | -0.058922 | -0.129418 | -0.119526 |
| **LDAPS\_LH** | -0.135717 | 0.119022 | -0.008556 | -0.002412 | 0.285942 | 0.005878 | -0.143246 | -0.056362 | 1.000000 | -0.113430 | -0.224377 | -0.213416 | -0.149974 | 0.017757 | -0.061339 | 0.010756 | 0.028329 | 0.100486 | 0.009790 | 0.053275 | 0.069892 | -0.032027 | 0.151784 | -0.052365 | -0.021563 | 0.047854 | 0.054981 |
| **LDAPS\_CC1** | 0.008671 | -0.290078 | 0.090828 | 0.574389 | 0.425827 | -0.416255 | 0.023128 | 0.203057 | -0.113430 | 1.000000 | 0.759430 | 0.485809 | 0.332623 | 0.726938 | 0.570382 | 0.300876 | 0.223193 | -0.006352 | -0.013794 | -0.030080 | -0.032476 | 0.235623 | -0.453763 | 0.008400 | -0.046380 | -0.226722 | -0.069276 |
| **LDAPS\_CC2** | 0.006380 | -0.157917 | 0.117894 | 0.713238 | 0.372085 | -0.492392 | 0.087752 | 0.156336 | -0.224377 | 0.759430 | 1.000000 | 0.690774 | 0.488186 | 0.462306 | 0.666230 | 0.387096 | 0.237396 | -0.002066 | -0.013838 | -0.027267 | -0.029457 | 0.173488 | -0.484796 | 0.078871 | -0.031901 | -0.162074 | -0.062914 |
| **LDAPS\_CC3** | 0.006303 | -0.061417 | 0.043629 | 0.636195 | 0.143375 | -0.468317 | 0.032023 | 0.065764 | -0.213416 | 0.485809 | 0.690774 | 1.000000 | 0.765041 | 0.266990 | 0.395289 | 0.459030 | 0.402332 | -0.005810 | 0.012323 | -0.006590 | -0.010127 | 0.125733 | -0.464998 | 0.021743 | -0.039457 | -0.112024 | 0.104130 |
| **LDAPS\_CC4** | 0.011510 | -0.049978 | 0.000193 | 0.459528 | 0.046555 | -0.367547 | -0.017483 | 0.090955 | -0.149974 | 0.332623 | 0.488186 | 0.765041 | 1.000000 | 0.190772 | 0.276920 | 0.308301 | 0.475357 | -0.023834 | -0.016750 | -0.027704 | -0.029013 | 0.110737 | -0.415827 | -0.026091 | -0.025300 | -0.108210 | 0.123391 |
| **LDAPS\_PPT1** | -0.001774 | -0.190222 | 0.068506 | 0.410985 | 0.420202 | -0.307565 | -0.054475 | 0.197126 | 0.017757 | 0.726938 | 0.462306 | 0.266990 | 0.190772 | 1.000000 | 0.505664 | 0.272455 | 0.189534 | 0.017866 | 0.002196 | -0.011255 | -0.005558 | 0.169165 | -0.347241 | -0.053093 | -0.021410 | -0.173612 | -0.068796 |
| **LDAPS\_PPT2** | -0.006798 | -0.097425 | 0.090113 | 0.510241 | 0.368154 | -0.344808 | 0.021320 | 0.166155 | -0.061339 | 0.570382 | 0.666230 | 0.395289 | 0.276920 | 0.505664 | 1.000000 | 0.385266 | 0.180817 | 0.030232 | -0.000990 | 0.002689 | 0.006903 | 0.129413 | -0.357330 | -0.006710 | 0.000754 | -0.131928 | -0.068552 |
| **LDAPS\_PPT3** | -0.004884 | -0.022348 | 0.035889 | 0.405612 | 0.219383 | -0.285371 | -0.018357 | 0.089355 | 0.010756 | 0.300876 | 0.387096 | 0.459030 | 0.308301 | 0.272455 | 0.385266 | 1.000000 | 0.427754 | 0.034607 | 0.009905 | 0.025174 | 0.037009 | 0.006045 | -0.268024 | -0.029908 | -0.047773 | 0.014898 | -0.013736 |
| **LDAPS\_PPT4** | -0.000285 | -0.021344 | -0.063495 | 0.244139 | 0.118163 | -0.207369 | -0.084124 | 0.167950 | 0.028329 | 0.223193 | 0.237396 | 0.402332 | 0.475357 | 0.189534 | 0.180817 | 0.427754 | 1.000000 | 0.000523 | 0.012639 | -0.006950 | -0.005177 | 0.033949 | -0.217651 | -0.100787 | -0.039993 | -0.025582 | 0.022889 |
| **lat** | -0.241811 | -0.052041 | -0.072707 | 0.090054 | 0.232258 | -0.038285 | -0.091796 | -0.027463 | 0.100486 | -0.006352 | -0.002066 | -0.005810 | -0.023834 | 0.017866 | 0.030232 | 0.034607 | 0.000523 | 1.000000 | 0.286414 | 0.034116 | 0.105511 | 0.048285 | -0.048616 | -0.076705 | -0.003366 | -0.009190 | 0.001499 |
| **lon** | -0.122829 | 0.011132 | -0.043201 | -0.093969 | 0.013059 | 0.106354 | -0.025047 | -0.077686 | 0.009790 | -0.013794 | -0.013838 | 0.012323 | -0.016750 | 0.002196 | -0.000990 | 0.009905 | 0.012639 | 0.286414 | 1.000000 | 0.186841 | 0.114267 | 0.011724 | 0.010567 | -0.041574 | -0.002923 | -0.009473 | -0.005863 |
| **DEM** | -0.326019 | -0.130892 | -0.238916 | 0.057460 | 0.174176 | -0.110286 | -0.174769 | 0.113157 | 0.053275 | -0.030080 | -0.027267 | -0.006590 | -0.027704 | -0.011255 | 0.002689 | 0.025174 | -0.006950 | 0.034116 | 0.186841 | 1.000000 | 0.779420 | 0.060194 | -0.117146 | -0.238249 | -0.007399 | -0.000836 | 0.000613 |
| **Slope** | -0.146034 | -0.092409 | -0.157839 | 0.083910 | 0.209222 | -0.105094 | -0.160165 | 0.095385 | 0.069892 | -0.032476 | -0.029457 | -0.010127 | -0.029013 | -0.005558 | 0.006903 | 0.037009 | -0.005177 | 0.105511 | 0.114267 | 0.779420 | 1.000000 | 0.037988 | -0.084174 | -0.158030 | -0.006568 | -0.000824 | 0.000939 |
| **Solar radiation** | -0.034787 | -0.099696 | -0.047145 | 0.229070 | 0.154031 | -0.034699 | 0.009890 | 0.164626 | -0.032027 | 0.235623 | 0.173488 | 0.125733 | 0.110737 | 0.169165 | 0.129413 | 0.006045 | 0.033949 | 0.048285 | 0.011724 | 0.060194 | 0.037988 | 1.000000 | -0.061064 | 0.012532 | -0.372028 | -0.871232 | 0.025585 |
| **Next\_Tmax** | 0.106378 | 0.586943 | 0.438813 | -0.433872 | -0.323446 | 0.816861 | 0.550810 | -0.205979 | 0.151784 | -0.453763 | -0.484796 | -0.464998 | -0.415827 | -0.347241 | -0.357330 | -0.268024 | -0.217651 | -0.048616 | 0.010567 | -0.117146 | -0.084174 | -0.061064 | 1.000000 | 0.580951 | -0.108606 | 0.119298 | 0.101615 |
| **Next\_Tmin** | 0.129110 | 0.615950 | 0.785946 | 0.134518 | -0.110402 | 0.553227 | 0.873678 | 0.023330 | -0.052365 | 0.008400 | 0.078871 | 0.021743 | -0.026091 | -0.053093 | -0.006710 | -0.029908 | -0.100787 | -0.076705 | -0.041574 | -0.238249 | -0.158030 | 0.012532 | 0.580951 | 1.000000 | -0.102557 | 0.054582 | 0.069291 |
| **Day** | 0.005032 | -0.130708 | -0.082712 | -0.073118 | -0.014136 | -0.139339 | -0.141873 | -0.058922 | -0.021563 | -0.046380 | -0.031901 | -0.039457 | -0.025300 | -0.021410 | 0.000754 | -0.047773 | -0.039993 | -0.003366 | -0.002923 | -0.007399 | -0.006568 | -0.372028 | -0.108606 | -0.102557 | 1.000000 | -0.114832 | -0.039453 |
| **Month** | 0.011309 | 0.171614 | 0.104576 | -0.183976 | -0.134357 | 0.115992 | 0.083247 | -0.129418 | 0.047854 | -0.226722 | -0.162074 | -0.112024 | -0.108210 | -0.173612 | -0.131928 | 0.014898 | -0.025582 | -0.009190 | -0.009473 | -0.000836 | -0.000824 | -0.871232 | 0.119298 | 0.054582 | -0.114832 | 1.000000 | -0.010665 |
| **Year** | 0.006658 | 0.099185 | 0.099222 | 0.001709 | -0.146019 | 0.066794 | 0.061339 | -0.119526 | 0.054981 | -0.069276 | -0.062914 | 0.104130 | 0.123391 | -0.068796 | -0.068552 | -0.013736 | 0.022889 | 0.001499 | -0.005863 | 0.000613 | 0.000939 | 0.025585 | 0.101615 | 0.069291 | -0.039453 | -0.010665 | 1.000000 |

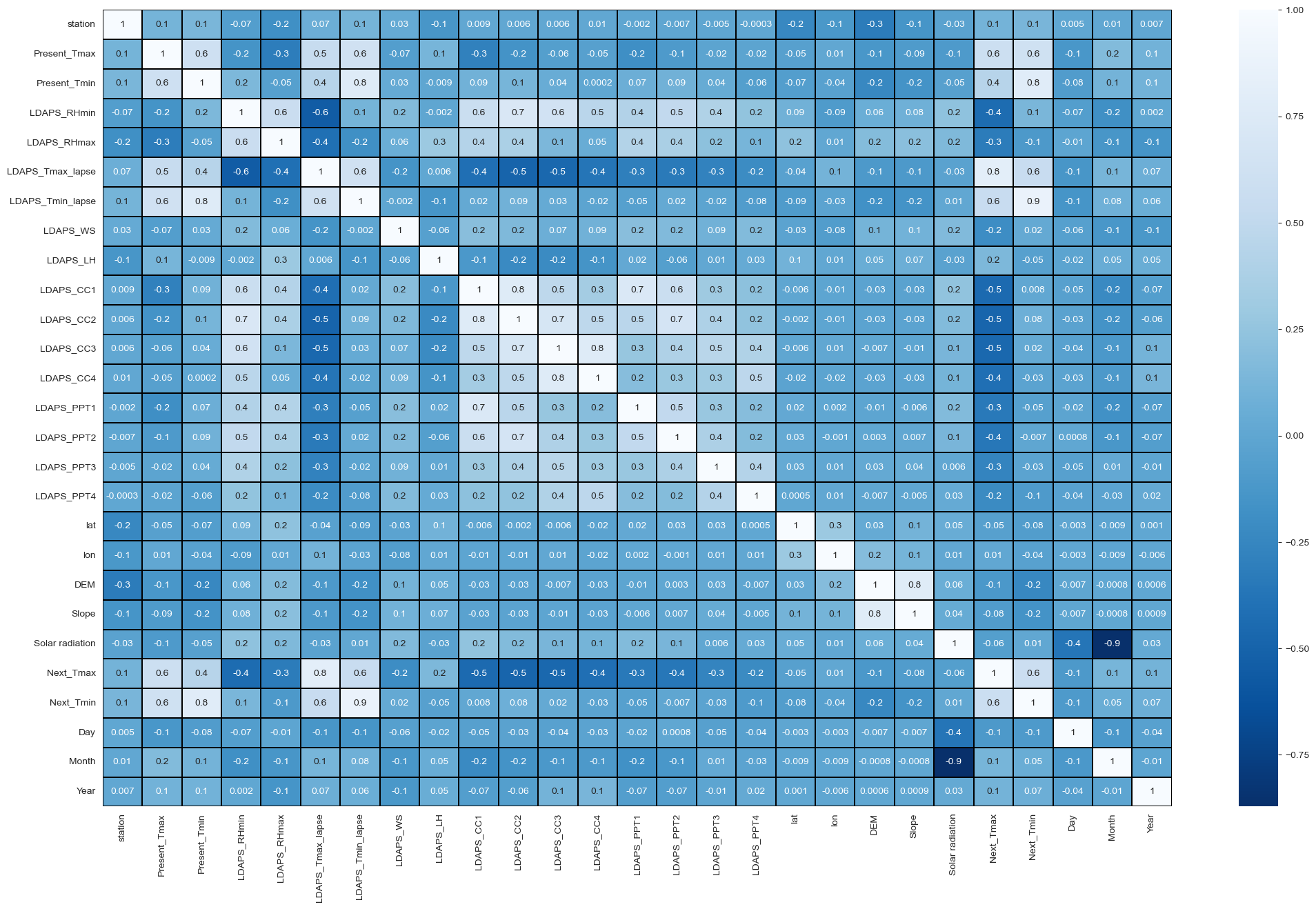
Let’s visualize using heatmap. In [50]:

plt.figure(figsize=(25, 15))

sns.heatmap(df.corr(),linewidth = 0.1, fmt = ".1g", linecolor = "black", annot = True, cmap = "Blues\_r")

Out[50]:

<Axes: >



In [51]:

# Visualizing correlation between label and features using bar plot.

plt.figure(figsize = (20,10))

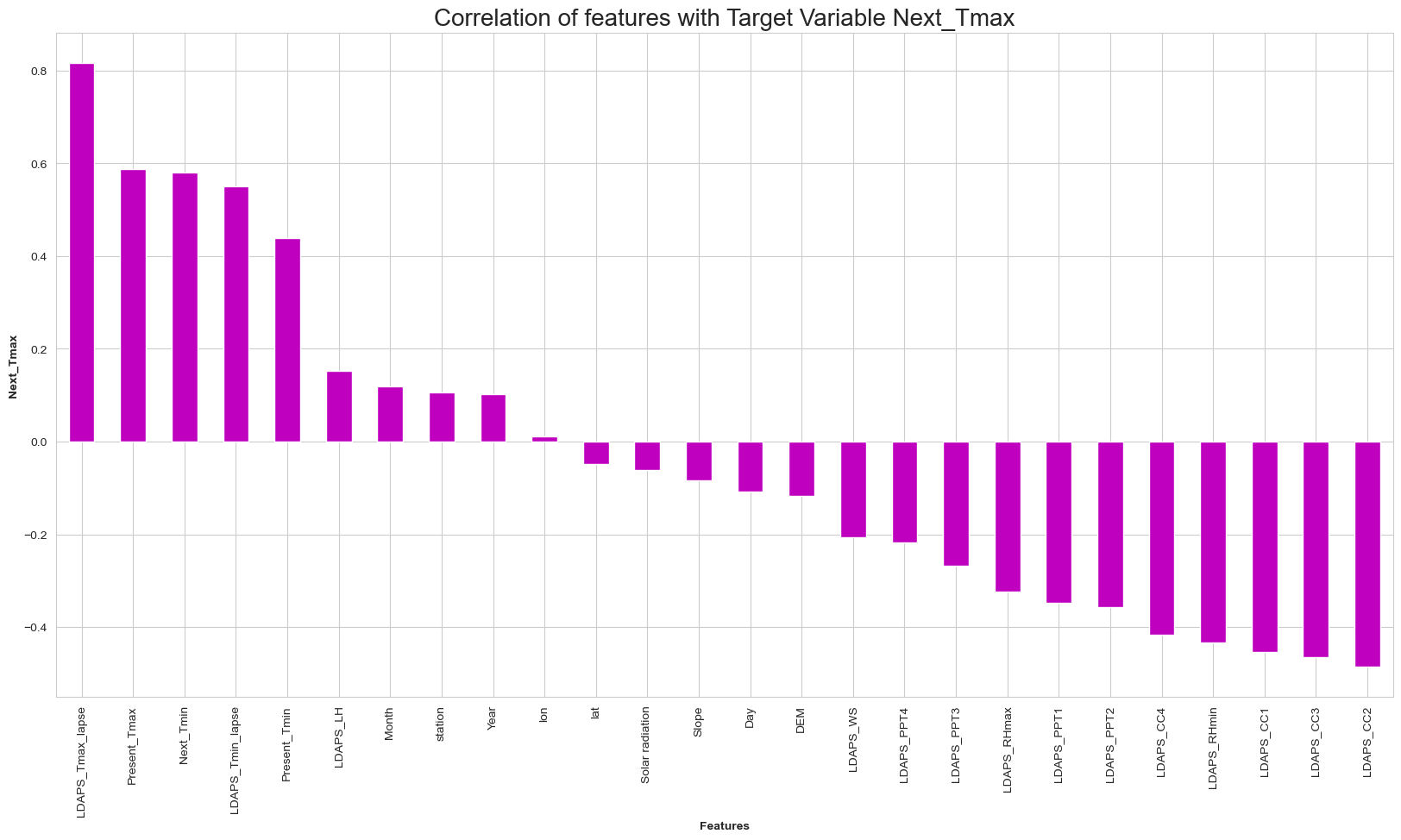
df.corr()['Next\_Tmax'].drop(['Next\_Tmax']).sort\_values(ascending=False).plot(kind='bar',color = 'm')

plt.xlabel('Features',fontsize=10,fontweight='bold')

plt.ylabel('Next\_Tmax',fontsize=10,fontweight='bold')

plt.title('Correlation of features with Target Variable Next\_Tmax',fontsize = 20)

plt.show()



We can see that, here temperature related feature and cloud cover are more correlated with the target feature.

In [52]:

# Visualizing correlation between label and features using bar plot.

plt.figure(figsize = (20,10))

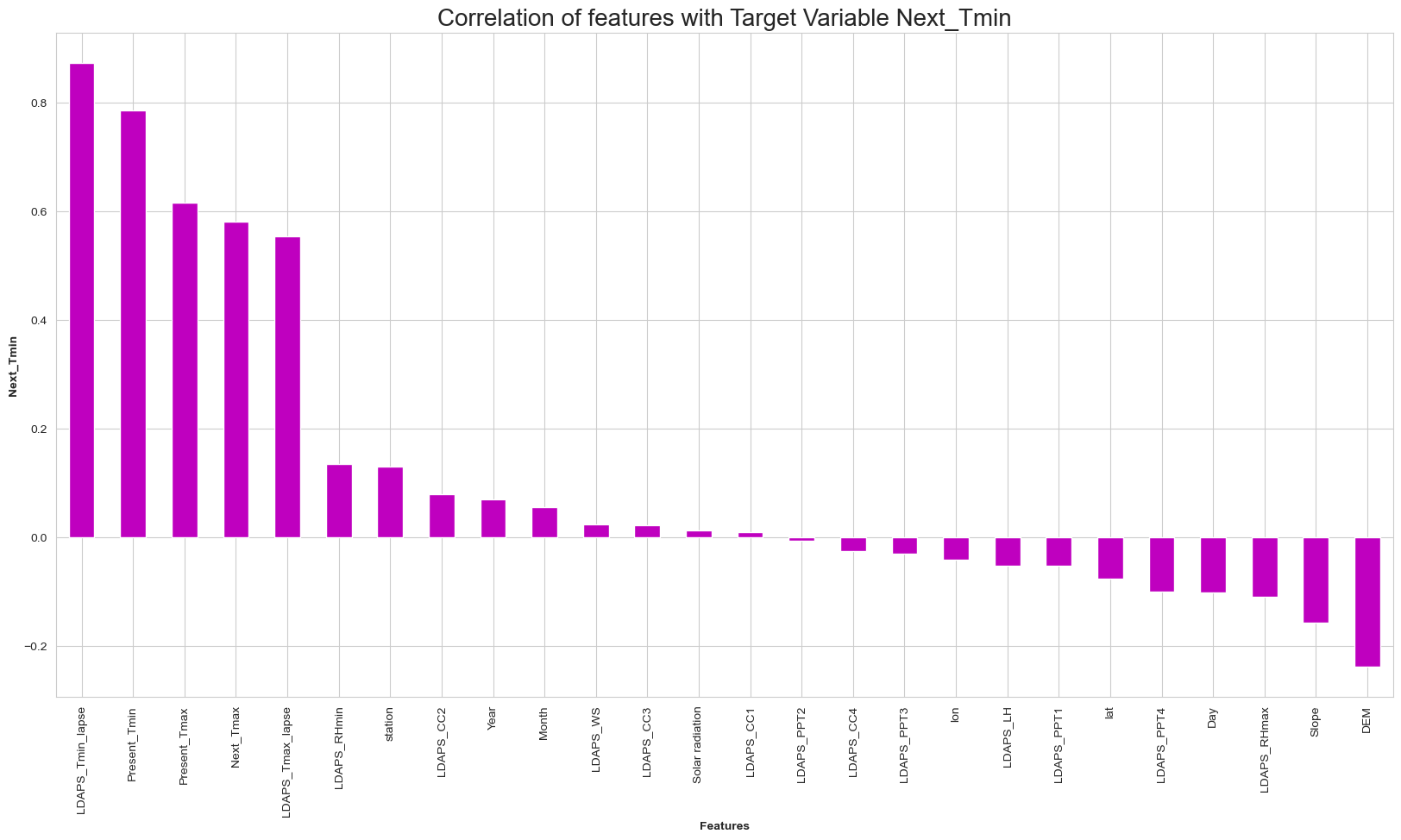
df.corr()['Next\_Tmin'].drop(['Next\_Tmin']).sort\_values(ascending=False).plot(kind='bar',color = 'm')

plt.xlabel('Features',fontsize=10,fontweight='bold')

plt.ylabel('Next\_Tmin',fontsize=10,fontweight='bold')

plt.title('Correlation of features with Target Variable Next\_Tmin',fontsize = 20)

plt.show()



Here temperature related features are more correlated as compared to others.

# 1. Model Building for Next\_Tmax[¶](" \l "1.-Model-Building-for-Next_Tmax)

### Using Standard Scalarization[¶](#Using-Standard-Scalarization)

Let’s split our data here for model building purposes. We will then import Standard Scaler to standardize our data. In [53]:

# Splitting data in target and dependent feature

X = df.drop(['Next\_Tmax'], axis =1)

Y = df['Next\_Tmax']

In [54]:

from sklearn.preprocessing import StandardScaler

scaler= StandardScaler()

X\_scale = scaler.fit\_transform(X)

## Using Variance Inflation Factor[¶](#Using-Variance-Inflation-Factor)

In [55]:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

vif = pd.DataFrame()

vif["VIF values"] = [variance\_inflation\_factor(X\_scale,i) for i in range(len(X.columns))]

vif["Features"] = X.columns

vif

Out[55]:

|  | **VIF values** | **Features** |
| --- | --- | --- |
| **0** | 1.308903 | station |
| **1** | 2.902352 | Present\_Tmax |
| **2** | 3.404251 | Present\_Tmin |
| **3** | 6.055263 | LDAPS\_RHmin |
| **4** | 2.606760 | LDAPS\_RHmax |
| **5** | 6.637758 | LDAPS\_Tmax\_lapse |
| **6** | 8.233459 | LDAPS\_Tmin\_lapse |
| **7** | 1.231453 | LDAPS\_WS |
| **8** | 1.516456 | LDAPS\_LH |
| **9** | 4.866499 | LDAPS\_CC1 |
| **10** | 5.499697 | LDAPS\_CC2 |
| **11** | 4.570603 | LDAPS\_CC3 |
| **12** | 2.787034 | LDAPS\_CC4 |
| **13** | 2.601762 | LDAPS\_PPT1 |
| **14** | 2.168333 | LDAPS\_PPT2 |
| **15** | 1.599724 | LDAPS\_PPT3 |
| **16** | 1.602099 | LDAPS\_PPT4 |
| **17** | 1.605219 | lat |
| **18** | 1.290864 | lon |
| **19** | 4.618869 | DEM |
| **20** | 3.082338 | Slope |
| **21** | 127.486970 | Solar radiation |
| **22** | 5.652553 | Next\_Tmin |
| **23** | 29.613013 | Day |
| **24** | 109.734007 | Month |
| **25** | 1.127585 | Year |

Nearly all the vif values are within the permissible range, lets proceed further. Also we cannot remove the 2 features as they are correlated poorly or moderately. Let’s use PCA to solve this problem.

## Using PCA Technique[¶](#Using-PCA-Technique)

PCA used find patterns and extract the latent features from our dataset.

In [56]:

from sklearn.decomposition import PCA

pca = PCA()

#plot the graph to find the principal components

x\_pca = pca.fit\_transform(X\_scale)

plt.figure(figsize=(10,8))

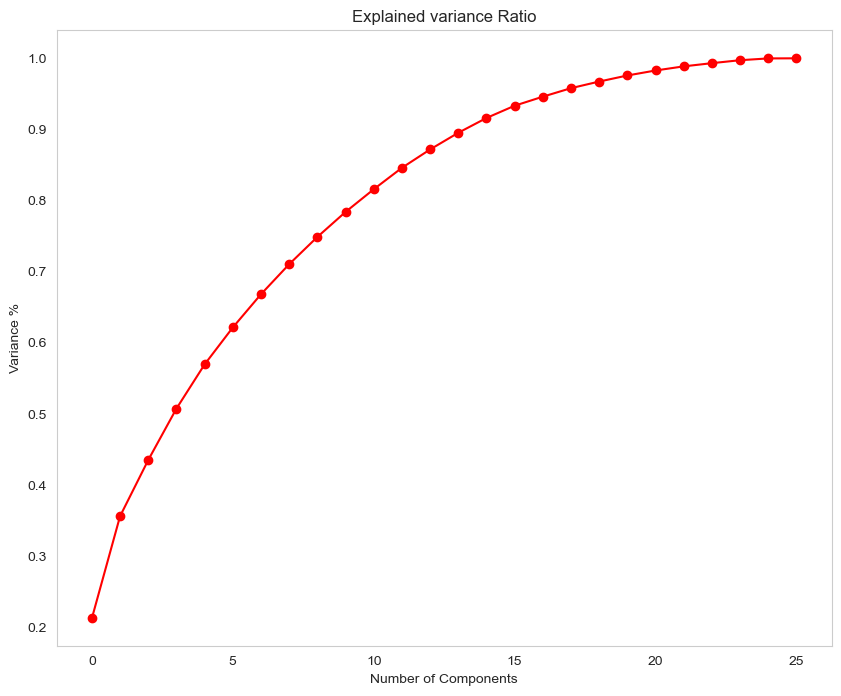
plt.plot(np.cumsum(pca.explained\_variance\_ratio\_), 'ro-')

plt.xlabel('Number of Components')

plt.ylabel('Variance %')

plt.title('Explained variance Ratio')

plt.grid()



Here we can see that nearly 15 principal components contribute for 90% of the variation in our data. Let’s consider taking 15 components for our analysis.

In [57]:

pca\_new = PCA(n\_components=15)

x\_new = pca\_new.fit\_transform(X\_scale)

In [58]:

Principle\_x=pd.DataFrame(x\_new,columns=np.arange(15))

In [59]:

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.svm import SVR

from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.metrics import r2\_score

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

from sklearn.linear\_model import Ridge

from sklearn.linear\_model import Lasso

In this section we will build Supervised learning ML model-based classification algorithm. As objective is to predict the temperature, thus it is a regression problem. train\_test\_split used to split data with size of 0.25.

In [60]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(Principle\_x, Y, random\_state=70, test\_size=.25)

print('Training feature size:',X\_train.shape)

print('Training target size:',Y\_train.shape)

print('Test feature size:',X\_test.shape)

print('Test target size:',Y\_test.shape)

Training feature size: (5054, 15)

Training target size: (5054,)

Test feature size: (1685, 15)

Test target size: (1685,)

### Finding Best Random State[¶](#Finding-Best-Random-State)

In [61]:

maxR2\_score=0

maxRS=0

for i in range(1,200):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scale, Y, random\_state=i, test\_size=.25)

lin\_reg=LinearRegression()

lin\_reg.fit(X\_train,Y\_train)

y\_pred=lin\_reg.predict(X\_test)

R2=r2\_score(Y\_test,y\_pred)

if R2>maxR2\_score:

maxR2\_score=R2

maxRS=i

print('Best R2 Score is', maxR2\_score ,'on Random\_state', maxRS)

Best R2 Score is 0.8093566394440032 on Random\_state 43

# Linear Regression Model[¶](#Linear-Regression-Model)

In [62]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scale, Y, random\_state=43, test\_size=.25)

lin\_reg=LinearRegression()

lin\_reg.fit(X\_train,Y\_train)

lin\_reg.score(X\_train,Y\_train)

y\_pred=lin\_reg.predict(X\_test)

print('\033[1m'+'Predicted Wins:'+'\033[0m\n',y\_pred)

print('\n')

print('\033[1m'+'Actual Wins:'+'\033[0m\n',Y\_test)

Predicted Wins:

[29.08570519 32.36851434 28.23362157 ... 30.32265043 23.98747582

29.61891944]

Actual Wins:

5006 28.6

947 33.4

3726 29.8

715 29.3

4242 33.0

...

7073 35.1

5959 33.9

4402 29.5

7354 26.3

7610 29.3

Name: Next\_Tmax, Length: 1685, dtype: float64

In [63]:

print('\033[1m'+' Error :'+'\033[0m')

print('Mean absolute error :', mean\_absolute\_error(Y\_test,y\_pred))

print('Mean squared error :', mean\_squared\_error(Y\_test,y\_pred))

print('Root Mean Squared Error:', np.sqrt(mean\_squared\_error(Y\_test,y\_pred)))

print('\n')

print('\033[1m'+' R2 Score :'+'\033[0m')

print(r2\_score(Y\_test,y\_pred,multioutput='variance\_weighted'))

Error :

Mean absolute error : 0.9921799141886302

Mean squared error : 1.691753461961296

Root Mean Squared Error: 1.3006742336039783

R2 Score :

0.8093566394440032

In [64]:

# Cross Validation

from sklearn.model\_selection import cross\_val\_score

score = cross\_val\_score(lin\_reg, X\_scale, Y, cv =3)

print('\033[1m'+'Cross Validation Score :',lin\_reg,":"+'\033[0m\n')

print("Mean CV Score :",score.mean())

Cross Validation Score : LinearRegression() :

Mean CV Score : 0.6628440776815404

In [65]:

from sklearn.ensemble import BaggingRegressor

In [66]:

RFR = RandomForestRegressor()

RFR.fit(X\_train, Y\_train)

pred\_RFR = RFR.predict(X\_test)

pred\_train = RFR.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_RFR))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_RFR))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_RFR))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_RFR)))

R2\_score: 0.9068287490078398

R2\_score on training data: 98.5812584742304

Mean Absolute Error: 0.6812106824925814

Mean squared error: 0.8267939988130559

Root Mean Squared Error: 0.9092821337808502

In [67]:

dtc = DecisionTreeRegressor()

dtc.fit(X\_train, Y\_train)

pred\_dtc = dtc.predict(X\_test)

pred\_train = dtc.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_dtc))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_dtc))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_dtc))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_dtc)))

R2\_score: 0.7850052484955443

R2\_score on training data: 100.0

Mean Absolute Error: 0.9958456973293769

Mean squared error: 1.9078456973293767

Root Mean Squared Error: 1.381247876859681

In [68]:

XT = ExtraTreesRegressor()

XT.fit(X\_train, Y\_train)

pred\_XT = XT.predict(X\_test)

pred\_train = XT.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_XT))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_XT))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_XT))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_XT)))

R2\_score: 0.9301721273748734

R2\_score on training data: 100.0

Mean Absolute Error: 0.6007329376854601

Mean squared error: 0.6196467839762616

Root Mean Squared Error: 0.7871764630476838

In [69]:

BR = BaggingRegressor()

BR.fit(X\_train, Y\_train)

pred\_BR = BR.predict(X\_test)

pred\_train = BR.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_BR))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_BR))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_BR))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_BR)))

R2\_score: 0.8913990132364318

R2\_score on training data: 97.7605501048536

Mean Absolute Error: 0.7327715133531156

Mean squared error: 0.9637162017804153

Root Mean Squared Error: 0.981690481659273

In [70]:

adb=AdaBoostRegressor()

adb.fit(X\_train, Y\_train)

pred\_adb = adb.predict(X\_test)

pred\_train = adb.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_adb))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_adb))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_adb))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_adb)))

R2\_score: 0.7703171511120019

R2\_score on training data: 76.5696716136435

Mean Absolute Error: 1.1615342633844166

Mean squared error: 2.038186662395053

Root Mean Squared Error: 1.427650749446465

## Cross Validation Score[¶](#Cross-Validation-Score)

**Cross-validation** is a technique used in machine learning to evaluate the performance of a model on unseen data. In [71]:

from sklearn.model\_selection import cross\_val\_score

In [72]:

score = cross\_val\_score(lin\_reg,X\_scale,Y,cv=5,scoring='r2')

print(score)

print(score.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,y\_pred) - score.mean())\*100)

[0.67684166 0.6846022 0.68339839 0.77323599 0.58315995]

0.6802476385180654

Difference between R2 score and cross validation score is- 12.91090009259378

In [73]:

score\_1 = cross\_val\_score(RFR,X\_scale,Y,cv=5,scoring='r2')

print(score\_1)

print(score\_1.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_RFR) - score\_1.mean())\*100)

[0.74524353 0.67421476 0.67118689 0.73016921 0.62054271]

0.688271416506791

Difference between R2 score and cross validation score is- 21.85573325010488

In [74]:

score\_2 = cross\_val\_score(dtc,X\_scale,Y,cv=5,scoring='r2')

print(score\_2)

print(score\_2.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_dtc) - score\_2.mean())\*100)

[0.38740756 0.37505356 0.30558852 0.48413023 0.36479853]

0.38339567738136926

Difference between R2 score and cross validation score is- 40.1609571114175

In [75]:

score\_3 = cross\_val\_score(XT,X\_scale,Y,cv=5,scoring='r2')

print(score\_3)

print(score\_3.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_XT) - score\_3.mean())\*100)

[0.72576138 0.64272169 0.71302534 0.76611681 0.61287469]

0.6920999829745093

Difference between R2 score and cross validation score is- 23.807214440036418

In [76]:

score\_4 = cross\_val\_score(BR,X\_scale,Y,cv=5,scoring='r2')

print(score\_4)

print(score\_4.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_BR) - score\_4.mean())\*100)

[0.71011484 0.65475581 0.65390352 0.72407665 0.61881871]

0.6723339047825857

Difference between R2 score and cross validation score is- 21.90651084538461

In [77]:

score\_5 = cross\_val\_score(adb,X\_scale,Y,cv=5,scoring='r2')

print(score\_5)

print(score\_5.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_adb) - score\_5.mean())\*100)

[0.68034073 0.61346376 0.6562935 0.68194142 0.62480695]

0.6513692718025796

Difference between R2 score and cross validation score is- 11.894787930942229

Since the r2 score of Random forest regressor and Extra tree regressor looks good, we can consider them . But the difference in the cross validation score and r2 score in case of random forest regressor appears to be less as compared to the extra tree regressor. We will thus proceed with the Random Forest Regressor model and perform hyperparameter tuning.

# Hyperparameter Tuning[¶](#Hyperparameter-Tuning)

Hyperparameter tuning is the process of selecting the optimal values for a Machine Learning model’s hyperparameters. Hyperparameters are settings that control the learning process of the model, such as the learning rate, the number of neurons in a neural network, or the kernel size in a support vector machine. The goal of hyperparameter tuning is to find the values that lead to the best performance on a given task. We used Grid Search CV for the purpose. In [78]:

from sklearn.model\_selection import GridSearchCV

In [79]:

# Define the parameter grid

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'max\_features': ['auto', 'sqrt', 'log2'],

'bootstrap': [True, False]

}

In [80]:

# Create GridSearchCV object

GSCV = GridSearchCV(RandomForestRegressor(), param\_grid, cv=5, n\_jobs=-1)

In [81]:

# Fit the GridSearchCV object to the training data

GSCV.fit(X\_train, Y\_train)

Out[81]:

GridSearchCV(cv=5, estimator=RandomForestRegressor(), n\_jobs=-1,

param\_grid={'bootstrap': [True, False],

'max\_depth': [None, 10, 20],

'max\_features': ['auto', 'sqrt', 'log2'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],

'n\_estimators': [50, 100, 200]})

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.   
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

  GridSearchCV[?Documentation for GridSearchCV](https://scikit-learn.org/1.4/modules/generated/sklearn.model_selection.GridSearchCV.html)iFitted

GridSearchCV(cv=5, estimator=RandomForestRegressor(), n\_jobs=-1,

param\_grid={'bootstrap': [True, False],

'max\_depth': [None, 10, 20],

'max\_features': ['auto', 'sqrt', 'log2'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],

'n\_estimators': [50, 100, 200]})

estimator: RandomForestRegressor

RandomForestRegressor()

 RandomForestRegressor[?Documentation for RandomForestRegressor](https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

RandomForestRegressor()

In [82]:

GSCV.best\_params\_

Out[82]:

{'bootstrap': False,

'max\_depth': 20,

'max\_features': 'sqrt',

'min\_samples\_leaf': 1,

'min\_samples\_split': 2,

'n\_estimators': 200}

## Final Model[¶](#Final-Model)

We will now proceed to build our final model and we can see that the R2 score turned out to be same as above. In [85]:

from sklearn.ensemble import RandomForestRegressor

Final\_model = RandomForestRegressor(n\_estimators=200 ,bootstrap=False, max\_depth= 20, max\_features = 'sqrt',

min\_samples\_leaf = 1, min\_samples\_split = 2)

Final\_model.fit(X\_train,Y\_train)

y\_pred=Final\_model.predict(X\_test)

print('\n')

print('\033[1m'+' Error in Final Model :' +'\033[0m')

print('Mean absolute error :', mean\_absolute\_error(Y\_test,pred\_RFR))

print('Mean squared error :', mean\_squared\_error(Y\_test,pred\_RFR))

print('Root Mean Squared Error:', np.sqrt(mean\_squared\_error(Y\_test,pred\_RFR)))

print('\n')

print('\033[1m'+' R2 Score of Final Model :'+'\033[0m')

print(r2\_score(Y\_test,pred\_RFR))

print('\n')

Error in Final Model :

Mean absolute error : 0.6812106824925814

Mean squared error : 0.8267939988130559

Root Mean Squared Error: 0.9092821337808502

R2 Score of Final Model :

0.9068287490078398

In [87]:

plt.figure(figsize=(12, 7))

y\_pred = Final\_model.predict(X\_test)

plt.scatter(Y\_test.round(2), y\_pred)

plt.plot([Y\_test.min(), Y\_test.max()], [Y\_test.min(), Y\_test.max()], 'k--', lw=2) # Plot the diagonal line

print('\033[1m' + ' True Values Vs Predicted Value plot :' + '\033[0m')

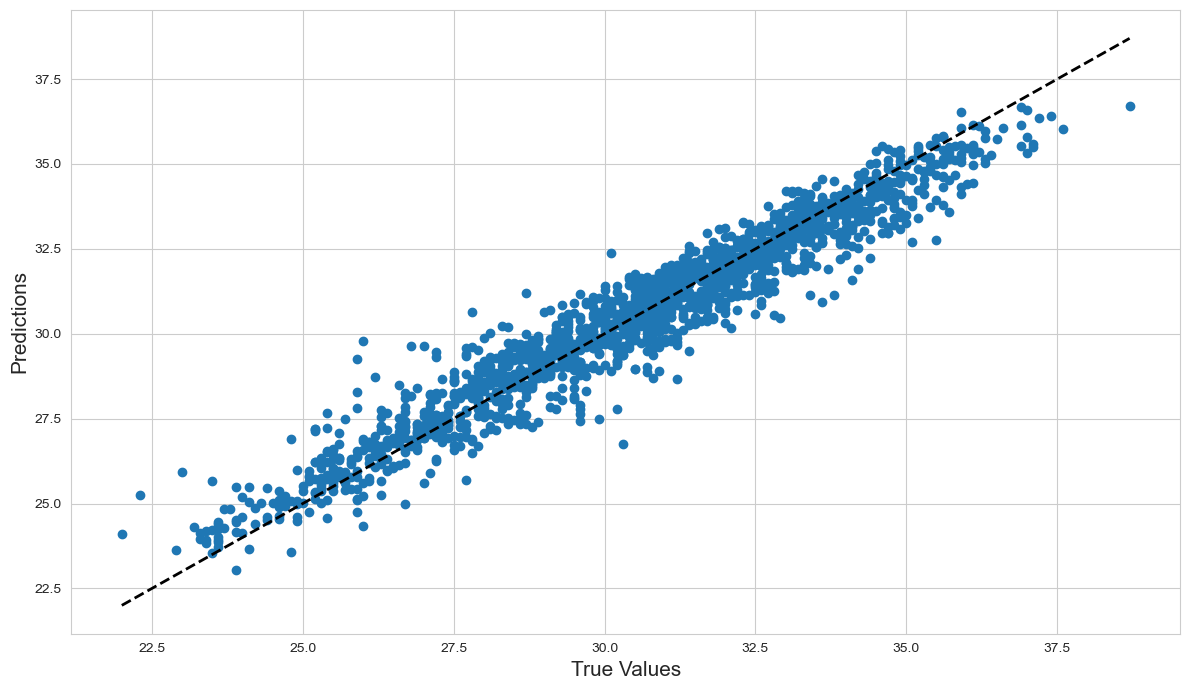
plt.xlabel('True Values', fontsize=15)

plt.ylabel('Predictions', fontsize=15)

plt.tight\_layout()

plt.show()

True Values Vs Predicted Value plot :



#### Saving Model[¶](#Saving-Model)

In [89]:

import joblib

joblib.dump(Final\_model,'Next\_Tmax\_Analysis\_Model.pkl')

Out[89]:

['Next\_Tmax\_Analysis\_Model.pkl']

# Model Building for Next\_Tmin[¶](" \l "Model-Building-for-Next_Tmin)

## Standard Scalarization[¶](#Standard-Scalarization)

In [90]:

# Splitting data in target and dependent feature

X = df.drop(['Next\_Tmin'], axis =1)

Y = df['Next\_Tmin']

In [91]:

from sklearn.preprocessing import StandardScaler

scaler= StandardScaler()

X\_scale = scaler.fit\_transform(X)

In [92]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scale, Y, random\_state=70, test\_size=.25)

print('Training feature size:',X\_train.shape)

print('Training target size:',Y\_train.shape)

print('Test feature size:',X\_test.shape)

print('Test target size:',Y\_test.shape)

Training feature size: (5054, 26)

Training target size: (5054,)

Test feature size: (1685, 26)

Test target size: (1685,)

### Finding Best Random State[¶](#Finding-Best-Random-State)

Here on finding out the random state it turned out to be 176. In [95]:

maxR2\_score=0

maxRS=0

for i in range(1,200):

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scale, Y, random\_state=i, test\_size=.25)

lin\_reg=LinearRegression()

lin\_reg.fit(X\_train,Y\_train)

y\_pred=lin\_reg.predict(X\_test)

R2=r2\_score(Y\_test,y\_pred)

if R2>maxR2\_score:

maxR2\_score=R2

maxRS=i

print('Best R2 Score is', maxR2\_score ,'on Random\_state', maxRS)

Best R2 Score is 0.8528556132857731 on Random\_state 176

# Linear Regression Model[¶](#Linear-Regression-Model)

In [97]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_scale, Y, random\_state=176, test\_size=.25)

lin\_reg=LinearRegression()

lin\_reg.fit(X\_train,Y\_train)

lin\_reg.score(X\_train,Y\_train)

y\_pred=lin\_reg.predict(X\_test)

print('\033[1m'+'Predicted Wins:'+'\033[0m\n',y\_pred)

print('\n')

print('\033[1m'+'Actual Wins:'+'\033[0m\n',Y\_test)

Predicted Wins:

[27.65566182 21.57081897 25.16830362 ... 20.1797222 21.69344604

25.18825181]

Actual Wins:

7110 26.3

6291 23.0

5658 25.4

2310 24.6

3241 15.5

...

4593 19.3

491 22.0

5087 20.1

6866 22.1

6046 24.3

Name: Next\_Tmin, Length: 1685, dtype: float64

In [98]:

print('\033[1m'+' Error :'+'\033[0m')

print('Mean absolute error :', mean\_absolute\_error(Y\_test,y\_pred))

print('Mean squared error :', mean\_squared\_error(Y\_test,y\_pred))

print('Root Mean Squared Error:', np.sqrt(mean\_squared\_error(Y\_test,y\_pred)))

print('\n')

from sklearn.metrics import r2\_score

print('\033[1m'+' R2 Score :'+'\033[0m')

print(r2\_score(Y\_test,y\_pred,multioutput='variance\_weighted'))

Error :

Mean absolute error : 0.739096368713007

Mean squared error : 0.8664746752985479

Root Mean Squared Error: 0.9308462146340544

R2 Score :

0.8528556132857731

In [99]:

# Cross Validation

from sklearn.model\_selection import cross\_val\_score

score = cross\_val\_score(lin\_reg, X\_scale, Y, cv =3)

print('\033[1m'+'Cross Validation Score :',lin\_reg,":"+'\033[0m\n')

print("Mean CV Score :",score.mean())

Cross Validation Score : LinearRegression() :

Mean CV Score : 0.7872619222063713

In [100]:

RFR = RandomForestRegressor()

RFR.fit(X\_train, Y\_train)

pred\_RFR = RFR.predict(X\_test)

pred\_train = RFR.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_RFR))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_RFR))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_RFR))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_RFR)))

R2\_score: 0.9146163980033966

R2\_score on training data: 98.64539282683681

Mean Absolute Error: 0.5412605341246293

Mean squared error: 0.502790017804154

Root Mean Squared Error: 0.7090768772172408

In [101]:

dtc = DecisionTreeRegressor()

dtc.fit(X\_train, Y\_train)

pred\_dtc = dtc.predict(X\_test)

pred\_train = dtc.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_dtc))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_dtc))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_dtc))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_dtc)))

R2\_score: 0.7885982763715513

R2\_score on training data: 100.0

Mean Absolute Error: 0.8214243323442136

Mean squared error: 1.2448605341246293

Root Mean Squared Error: 1.1157331823176315

In [102]:

XT = ExtraTreesRegressor()

XT.fit(X\_train, Y\_train)

pred\_XT = XT.predict(X\_test)

pred\_train = XT.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_XT))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_XT))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_XT))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_XT)))

R2\_score: 0.9257301796385867

R2\_score on training data: 100.0

Mean Absolute Error: 0.5018836795252224

Mean squared error: 0.4373453851632046

Root Mean Squared Error: 0.6613209396073926

In [103]:

BR = BaggingRegressor()

BR.fit(X\_train, Y\_train)

pred\_BR = BR.predict(X\_test)

pred\_train = BR.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_BR))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_BR))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_BR))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_BR)))

R2\_score: 0.9012943856341115

R2\_score on training data: 97.78923627592039

Mean Absolute Error: 0.5850741839762612

Mean squared error: 0.5812380415430267

Root Mean Squared Error: 0.7623896913934676

In [104]:

adb=AdaBoostRegressor()

adb.fit(X\_train, Y\_train)

pred\_adb = adb.predict(X\_test)

pred\_train = adb.predict(X\_train)

print('R2\_score: ', r2\_score(Y\_test, pred\_adb))

print('R2\_score on training data: ',r2\_score(Y\_train, pred\_train)\*100)

print("Mean Absolute Error: ",mean\_absolute\_error(Y\_test, pred\_adb))

print("Mean squared error: ",mean\_squared\_error(Y\_test, pred\_adb))

print("Root Mean Squared Error: ",np.sqrt(mean\_squared\_error(Y\_test, pred\_adb)))

R2\_score: 0.8228859947022988

R2\_score on training data: 81.2393244087698

Mean Absolute Error: 0.8319967810793297

Mean squared error: 1.0429538201086739

Root Mean Squared Error: 1.021251105315766

## Cross Validation Score[¶](#Cross-Validation-Score)

In [105]:

from sklearn.model\_selection import cross\_val\_score

In [106]:

score = cross\_val\_score(lin\_reg,X\_scale,Y,cv=5,scoring='r2')

print(score)

print(score.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,y\_pred) - score.mean())\*100)

[0.75743627 0.67407187 0.79322917 0.86926946 0.86644208]

0.792089770128066

Difference between R2 score and cross validation score is- 6.076584315770706

In [107]:

score\_1 = cross\_val\_score(RFR,X\_scale,Y,cv=5,scoring='r2')

print(score\_1)

print(score\_1.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_RFR) - score\_1.mean())\*100)

[0.71767558 0.69871241 0.80821378 0.87648154 0.85784562]

0.791785787257641

Difference between R2 score and cross validation score is- 12.283061074575553

In [108]:

score\_2 = cross\_val\_score(dtc,X\_scale,Y,cv=5,scoring='r2')

print(score\_2)

print(score\_2.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_dtc) - score\_2.mean())\*100)

[0.48019702 0.44399939 0.61201022 0.78011539 0.66891581]

0.597047565454736

Difference between R2 score and cross validation score is- 19.15507109168153

In [109]:

score\_3 = cross\_val\_score(XT,X\_scale,Y,cv=5,scoring='r2')

print(score\_3)

print(score\_3.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_XT) - score\_3.mean())\*100)

[0.72051088 0.6951393 0.80876252 0.87688496 0.87675427]

0.795610384657137

Difference between R2 score and cross validation score is- 13.011979498144976

In [110]:

score\_4 = cross\_val\_score(BR,X\_scale,Y,cv=5,scoring='r2')

print(score\_4)

print(score\_4.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_BR) - score\_4.mean())\*100)

[0.70822207 0.66539452 0.79211935 0.85869586 0.83597713]

0.7720817860152052

Difference between R2 score and cross validation score is- 12.92125996189063

In [111]:

score\_5 = cross\_val\_score(adb,X\_scale,Y,cv=5,scoring='r2')

print(score\_5)

print(score\_5.mean())

print("Difference between R2 score and cross validation score is- ", (r2\_score(Y\_test,pred\_adb) - score\_5.mean())\*100)

[0.68669674 0.6153691 0.75574531 0.82109214 0.83927891]

0.7436364378660805

Difference between R2 score and cross validation score is- 7.924955683621837

Here both Random forest regressor and Extra tree regressor are looking good. The r2 score of extra tree regressor is good but the difference in the cross val score in case of random forest regressor is less (12) as compared to that of extra tree regressor that is 13. So proceeding with the Random forest Regressor.

# Hyperparameter Tuning[¶](#Hyperparameter-Tuning)

In [112]:

from sklearn.model\_selection import GridSearchCV

In [113]:

# Define the parameter grid

param\_grid = {

'n\_estimators': [10, 50, 100],

'max\_depth': [0, 10, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'max\_features': ['auto', 'sqrt', 'log2'],

}

In [114]:

# Create GridSearchCV object

GSCV = GridSearchCV(RandomForestRegressor(), param\_grid, cv=5, n\_jobs=-1)

In [115]:

# Fit the GridSearchCV object to the training data

GSCV.fit(X\_train, Y\_train)

Out[115]:

GridSearchCV(cv=5, estimator=RandomForestRegressor(), n\_jobs=-1,

param\_grid={'max\_depth': [0, 10, 20],

'max\_features': ['auto', 'sqrt', 'log2'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],

'n\_estimators': [10, 50, 100]})

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.   
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

  GridSearchCV[?Documentation for GridSearchCV](https://scikit-learn.org/1.4/modules/generated/sklearn.model_selection.GridSearchCV.html)iFitted

GridSearchCV(cv=5, estimator=RandomForestRegressor(), n\_jobs=-1,

param\_grid={'max\_depth': [0, 10, 20],

'max\_features': ['auto', 'sqrt', 'log2'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],

'n\_estimators': [10, 50, 100]})

estimator: RandomForestRegressor

RandomForestRegressor()

 RandomForestRegressor[?Documentation for RandomForestRegressor](https://scikit-learn.org/1.4/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

RandomForestRegressor()

In [116]:

GSCV.best\_params\_

Out[116]:

{'max\_depth': 20,

'max\_features': 'sqrt',

'min\_samples\_leaf': 1,

'min\_samples\_split': 2,

'n\_estimators': 100}

## Final Model[¶](#Final-Model)

In [117]:

from sklearn.ensemble import RandomForestRegressor

Final\_model = RandomForestRegressor(n\_estimators=100 , max\_depth= 20, max\_features = 'sqrt',

min\_samples\_leaf = 1, min\_samples\_split = 2)

Final\_model.fit(X\_train,Y\_train)

y\_pred=Final\_model.predict(X\_test)

print('\n')

print('\033[1m'+' Error in Final Model :' +'\033[0m')

print('Mean absolute error :', mean\_absolute\_error(Y\_test,pred\_RFR))

print('Mean squared error :', mean\_squared\_error(Y\_test,pred\_RFR))

print('Root Mean Squared Error:', np.sqrt(mean\_squared\_error(Y\_test,pred\_RFR)))

print('\n')

print('\033[1m'+' R2 Score of Final Model :'+'\033[0m')

print(r2\_score(Y\_test,pred\_RFR))

print('\n')

Error in Final Model :

Mean absolute error : 0.5412605341246293

Mean squared error : 0.502790017804154

Root Mean Squared Error: 0.7090768772172408

R2 Score of Final Model :

0.9146163980033966

We can also now see the plot and tell that most of the values are within the range of the line. In [118]:

plt.figure(figsize=(12, 7))

y\_pred = Final\_model.predict(X\_test)

plt.scatter(Y\_test.round(2), y\_pred)

plt.plot([Y\_test.min(), Y\_test.max()], [Y\_test.min(), Y\_test.max()], 'k--', lw=2) # Plot the diagonal line

print('\033[1m' + ' True Values Vs Predicted Value plot :' + '\033[0m')

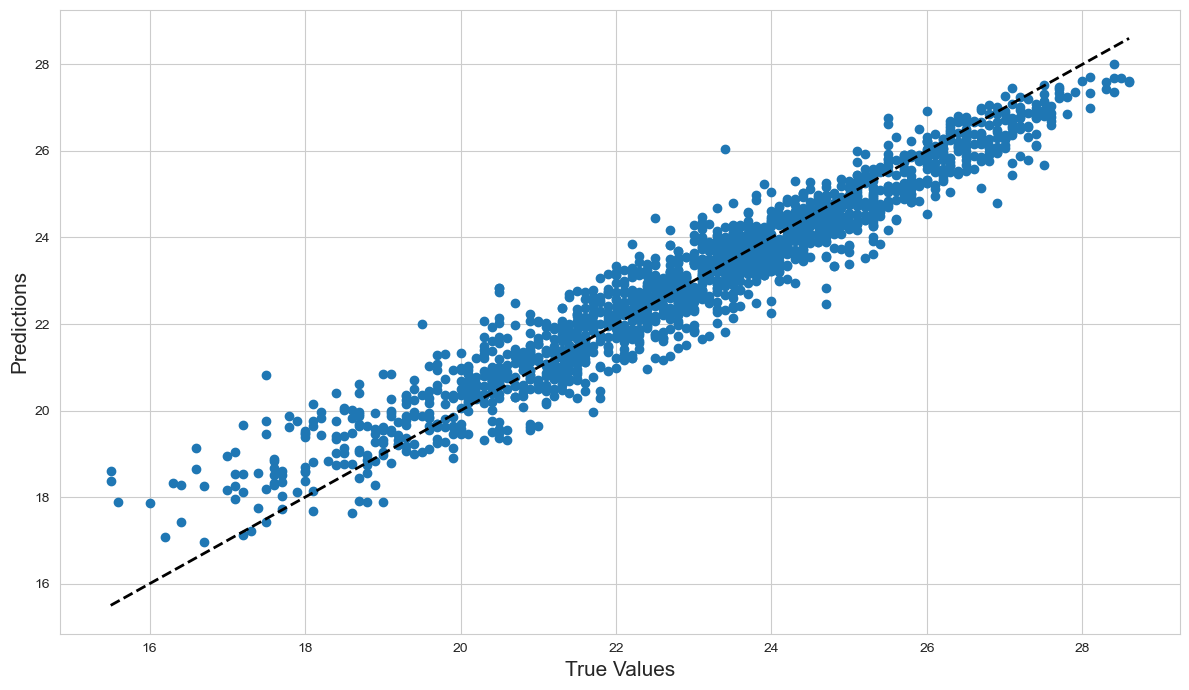
plt.xlabel('True Values', fontsize=15)

plt.ylabel('Predictions', fontsize=15)

plt.tight\_layout()

plt.show()

True Values Vs Predicted Value plot :



In [122]:

import joblib

Model=joblib.dump(Final\_model,'Next\_Tmin\_Analysis\_Final.pkl')

In [ ]: