Temperature Forecast Project using ML

1.Problem Definition:

Data Set Information:

This data is 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. This data consists of summer data from 2013 to 2017. The input data is largely composed of the LDAPS model's next-day forecast data, in-situ maximum and minimum temperatures of present-day, and geographic auxiliary variables. There are two outputs (i.e. next-day maximum and minimum air temperatures) in this data. Hindcast validation was conducted for the period from 2015 to 2017.

Attribute Information:

For more information, read [Cho et al, 2020].
1. station - used weather station number: 1 to 25
2. Date - Present day: yyyy-mm-dd ('2013-06-30' to '2017-08-30')
3. Present_Tmax - Maximum air temperature between 0 and 21 h on the present day (Â°C): 20 to 37.6
4. Present_Tmin - Minimum air temperature between 0 and 21 h on the present day (Â°C): 11.3 to 29.9
5. LDAPS_RHmin - LDAPS model forecast of next-day minimum relative humidity (%): 19.8 to 98.5
6. LDAPS_RHmax - LDAPS model forecast of next-day maximum relative humidity (%): 58.9 to 100
7. LDAPS_Tmax_lapse - LDAPS model forecast of next-day maximum air temperature applied lapse rate (Â°C): 17.6 to 38.5
8. LDAPS_Tmin_lapse - LDAPS model forecast of next-day minimum air temperature applied lapse rate (Â°C): 14.3 to 29.6
9. LDAPS_WS - LDAPS model forecast of next-day average wind

speed (m/s): 2.9 to 21.9
10. LDAPS_LH - LDAPS model forecast of next-day average latent heat flux (W/m2): -13.6 to 213.4
11. LDAPS_CC1 - LDAPS model forecast of next-day 1st 6-hour split average cloud cover (0-5 h) (%): 0 to 0.97
12. LDAPS_CC2 - LDAPS model forecast of next-day 2nd 6-hour split average cloud cover (6-11 h) (%): 0 to 0.97
13. LDAPS_CC3 - LDAPS model forecast of next-day 3rd 6-hour split average cloud cover (12-17 h) (%): 0 to 0.98
14. LDAPS_CC4 - LDAPS model forecast of next-day 4th 6-hour split average cloud cover (18-23 h) (%): 0 to 0.97
15. LDAPS_PPT1 - LDAPS model forecast of next-day 1st 6-hour split average precipitation (0-5 h) (%): 0 to 23.7
16. LDAPS_PPT2 - LDAPS model forecast of next-day 2nd 6-hour split average precipitation (6-11 h) (%): 0 to 21.6
17. LDAPS_PPT3 - LDAPS model forecast of next-day 3rd 6-hour split average precipitation (12-17 h) (%): 0 to 15.8
18. LDAPS_PPT4 - LDAPS model forecast of next-day 4th 6-hour split average precipitation (18-23 h) (%): 0 to 16.7
19. lat - Latitude (Â°): 37.456 to 37.645
20. lon - Longitude (Â°): 126.826 to 127.135
21. DEM - Elevation (m): 12.4 to 212.3
22. Slope - Slope (Â°): 0.1 to 5.2
23. Solar radiation - Daily incoming solar radiation (wh/m2): 4329.5 to 5992.9
24. Next_Tmax - The next-day maximum air temperature (Â°C): 17.4 to 38.9
25. Next_Tmin - The next-day minimum air temperature (Â°C): 11.3 to 29.8T

There are two target variables are present in dataset:

1) Next_Tmax: Next day maximum: Next temperature

2) Next_Tmin day minimum temperature

2. Data Analysis:

Data analysis is the process of collecting, modeling, and analyzing data to extract insights that support decision-making. There are several methods and techniques to perform analysis depending on the industry and the aim of the analysis.

In Temperature Forecast Project using ML project There are 2 target variable are present :

i) **Next_Tmax: Next day maximum temperature**
ii) **Next_Tmin: Next day minimum temperature**

So in this target columns all are numeric and continuous values are present so it Regression problem so apply all regression models in this project.

DataSet:

Regression Analysis:

Regression analysis is used to estimate the relationship between a set of variables. When conducting any type of regression analysis, you're looking to see if there's a correlation between a dependent variable (that's the variable or outcome you want to measure or predict) and any number of independent variables (factors which may have an impact on the dependent variable). The aim of regression analysis is to estimate how one or more variables might impact the dependent variable, in order to identify trends and patterns. This is especially useful for making predictions and forecasting future trends.

Regression analysis is a type of predictive modeling technique which is used to find the relationship between a dependent variable (usually known as the "Y" variable) and either one independent variable (the "X"

variable) or a series of independent variables. When two or more independent variables are used to predict or explain the outcome of the dependent variable, this is known as multiple regression

Types of Regressor models:

- Linear Regression.
- Logistic Regression.
- Ridge Regression.
- Lasso Regression.
- Polynomial Regression.
- Bayesian Linear Regression.

3.EDA :

Exploratory Data Analysis, or EDA, is an important step in any Data Analysis or Data Science project. EDA is the process of investigating the dataset to discover patterns, and anomalies (outliers), and form hypotheses based on our understanding of the dataset.

EDA involves generating summary statistics for numerical data in the dataset and creating various graphical representations to understand the data better. In this article, we will understand EDA with the help of an example dataset. We will use **Python** language (**Pandas** library) for this purpose.

Importing libraries

We will start by importing the libraries we will require for performing EDA. These include NumPy, Pandas, Matplotlib, and Seaborn.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading Library

First need to import different necessary libraries.

Reading data:

We will now read the data from a CSV file into a Pandas DataFrame. You can download the dataset for your reference.

```
df=pd.read_csv(r'C:\Users\Ankita\Desktop\Temp.csv')
```

Then need to read and load Temperature dataset.

```
pd.set_option('display.max_columns', None)
```

Then this above code is use for display all the columns in dataset.

Let us have a look at how our dataset looks like using df.head(). The output should look like this:

```
df.head()
```

APS_CC3	LDAPS_CC4	LDAPS_PPT1	LDAPS_PPT2	LDAPS_PPT3	LDAPS_PPT4	lat	lon	DEM	Slope	Solar radiation	Next_Tmax	Next_Tmin
61697	0.130928	0.0	0.0	0.0	0.0	37.6046	126.991	212.3350	2.7850	5992.895996	29.1	21.2
59444	0.127727	0.0	0.0	0.0	0.0	37.6046	127.032	44.7624	0.5141	5869.312500	30.5	22.5
04091	0.142125	0.0	0.0	0.0	0.0	37.5776	127.058	33.3068	0.2661	5863.555664	31.1	23.9
61157	0.134249	0.0	0.0	0.0	0.0	37.6450	127.022	45.7160	2.5348	5856.964844	31.7	24.3
78892	0.170021	0.0	0.0	0.0	0.0	37.5507	127.135	35.0380	0.5055	5859.552246	31.2	22.5

Loading the data First 5 rows in dataset using df.head().

```
df.tail()
```

S_CC3	LDAPS_CC4	LDAPS_PPT1	LDAPS_PPT2	LDAPS_PPT3	LDAPS_PPT4	lat	lon	DEM	Slope	Solar radiation	Next_Tmax	Next_Tmin
000	0.000000	0.000000	0.000000	0.000000	0.000000	37.5372	126.891	15.5876	0.155400	4443.313965	28.3	18.1
000	0.000000	0.000000	0.000000	0.000000	0.000000	37.5237	126.909	17.2956	0.222300	4438.373535	28.6	18.8
000	0.000796	0.000000	0.000000	0.000000	0.000000	37.5237	126.970	19.5844	0.271300	4451.345215	27.8	17.4
000	0.000000	0.000000	0.000000	0.000000	0.000000	37.4562	126.826	12.3700	0.098475	4329.520508	17.4	11.3
789	0.974710	23.701544	21.621661	15.841235	16.655469	37.6450	127.135	212.3350	5.178230	5992.895996	38.9	29.8

Loading the data Last 5 rows in dataset using df.head().

```
df.info()
```

```
 0   station            7750 non-null   float64
 1   Date               7750 non-null   object
 2   Present_Tmax       7682 non-null   float64
 3   Present_Tmin       7682 non-null   float64
 4   LDAPS_RHmin        7677 non-null   float64
 5   LDAPS_RHmax        7677 non-null   float64
 6   LDAPS_Tmax_lapse   7677 non-null   float64
 7   LDAPS_Tmin_lapse   7677 non-null   float64
 8   LDAPS_WS           7677 non-null   float64
 9   LDAPS_LH           7677 non-null   float64
 10  LDAPS_CC1          7677 non-null   float64
 11  LDAPS_CC2          7677 non-null   float64
 12  LDAPS_CC3          7677 non-null   float64
 13  LDAPS_CC4          7677 non-null   float64
 14  LDAPS_PPT1         7677 non-null   float64
 15  LDAPS_PPT2         7677 non-null   float64
 16  LDAPS_PPT3         7677 non-null   float64
 17  LDAPS_PPT4         7677 non-null   float64
 18  lat                7752 non-null   float64
 19  lon                7752 non-null   float64
 20  DEM                7752 non-null   float64
 21  Slope              7752 non-null   float64
 22  Solar radiation    7752 non-null   float64
 23  Next_Tmax          7725 non-null   float64
 24  Next_Tmin          7725 non-null   float64
```

In above using df.info() is used to get the information all th columns in dataset and its datatypes. In above dataset there are 25 columns in that there are 2 target variables.

```
df.columns
```

```
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 above df.colums is use to get all the column names in dataset.

```
df.dtypes
```

```
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
```

In above code df.dtypes is used to check the data types of all the column In above dataset all the values are numeric only date dataset is object means categorical value are present so convert need convert into numeric.

4. Pre-Processing Pipeline:

```
df.shape
```

```
(7752, 25)
```

Looking at the shape, we see that we now have half of the data point than original data and the same number of features. Before processing and cleaning manually, let's do some general data processing steps first:

- Remove features associated with >85% missing values
- Remove constant features
- Remove duplicates features
- Remove duplicate rows
- Remove highly collinear features)

Alright, let's get started with the typical data processing:

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

```
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
```

```
df.isnull().sum() * 100 / len(df)
```

```
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
```

Check the how much missing values and percentage of are present in dataset using df.shape() is use to check the missing values in dataset.

In Present_Tmax, Present_Tmin, LDAPS_RHmin, LDAPS_RHmax, LDAPS_RHmin_lapse, LDAPS_RHmax, LDAPS_WS, LDAPS_LH,

LDAPS_CC1, LDAPS_CC2, LDAPS_CC3, LDAPS_CC4, LDAPS_PPT1, LDAPS_PPT2, LDAPS_PPT3, LDAPS_PPT4 near about 70-75 missing values are present. And target columns Next_Tmax, NextTmin columns are 27 missing values are present and station and date column 2 missing values are present near about every columns missing values are present.

1.Remove features associated with 90% missing values: In the code below I first use pandas' built-in method 'isnull()' to find the rows associated with missing values. Then I sum them up to get the count for each feature. Finally, I sort the features according to the number of missing values and create a data frame for further analysis.

2.Remove constant features: At this step, we remove features that have a single unique value. A feature associated with one unique value does not help the model to generalize well since it's variance is zero. A tree-based model cannot take advantage of these type of features since the model can not split these features.

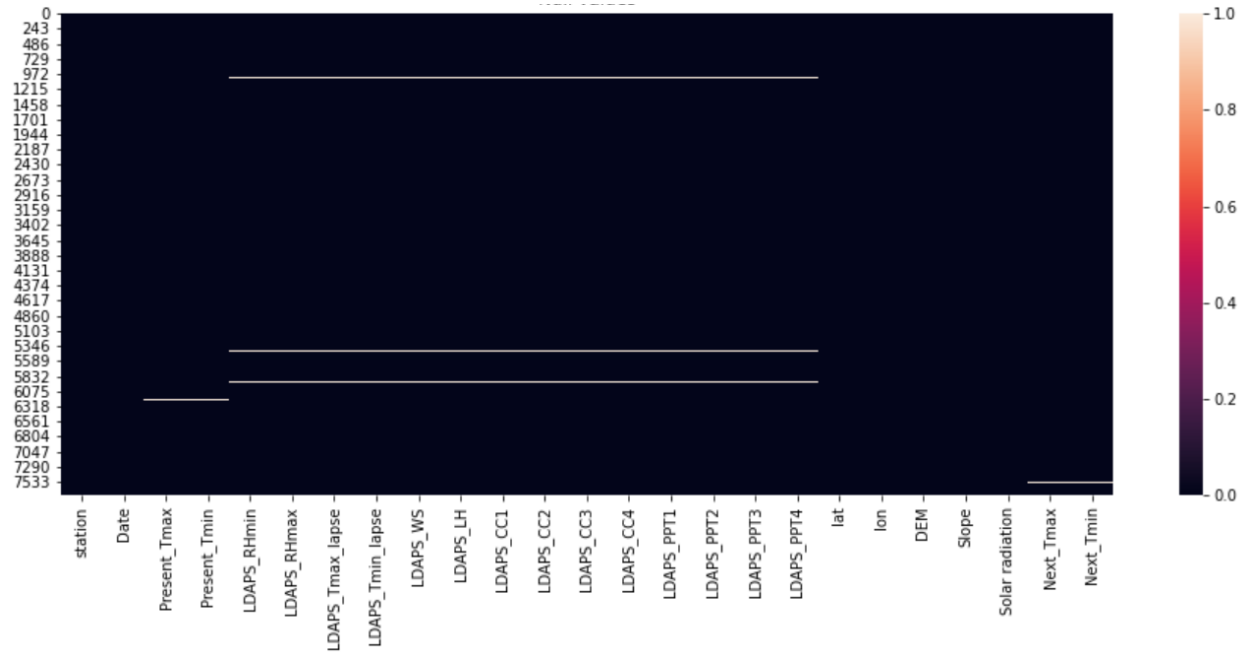
In the above result, we see that there are 16 features which have 75 % percent missing values.

```
df.describe()
```

_CC4	LDAPS_PPT1	LDAPS_PPT2	LDAPS_PPT3	LDAPS_PPT4	lat	lon	DEM	Slope	Solar radiation	Next_Tmax	Next_Tmin
0000	7677.000000	7677.000000	7677.000000	7677.000000	7752.000000	7752.000000	7752.000000	7752.000000	7752.000000	7725.000000	7725.000000
9191	0.591995	0.485003	0.278200	0.269407	37.544722	126.991397	61.867972	1.257048	5341.502803	30.274887	22.932220
4348	1.945768	1.762807	1.161809	1.206214	0.050352	0.079435	54.279780	1.370444	429.158867	3.128010	2.487613
0000	0.000000	0.000000	0.000000	0.000000	37.456200	126.826000	12.370000	0.098475	4329.520508	17.400000	11.300000
1532	0.000000	0.000000	0.000000	0.000000	37.510200	126.937000	28.700000	0.271300	4999.018555	28.200000	21.300000
7664	0.000000	0.000000	0.000000	0.000000	37.550700	126.995000	45.716000	0.618000	5436.345215	30.500000	23.100000
9489	0.052525	0.018364	0.007896	0.000041	37.577600	127.042000	59.832400	1.767800	5728.316406	32.600000	24.600000
4710	23.701544	21.621661	15.841235	16.655469	37.645000	127.135000	212.335000	5.178230	5992.895996	38.900000	29.800000

In above describe dataset there are 30 and 20 percent temperature values are present in target variables like Next_Tmax, Next_Tmin.

```
plt.figure(figsize=[16,6])
sns.heatmap(df.isnull())
plt.title("Null values")
plt.show()
```



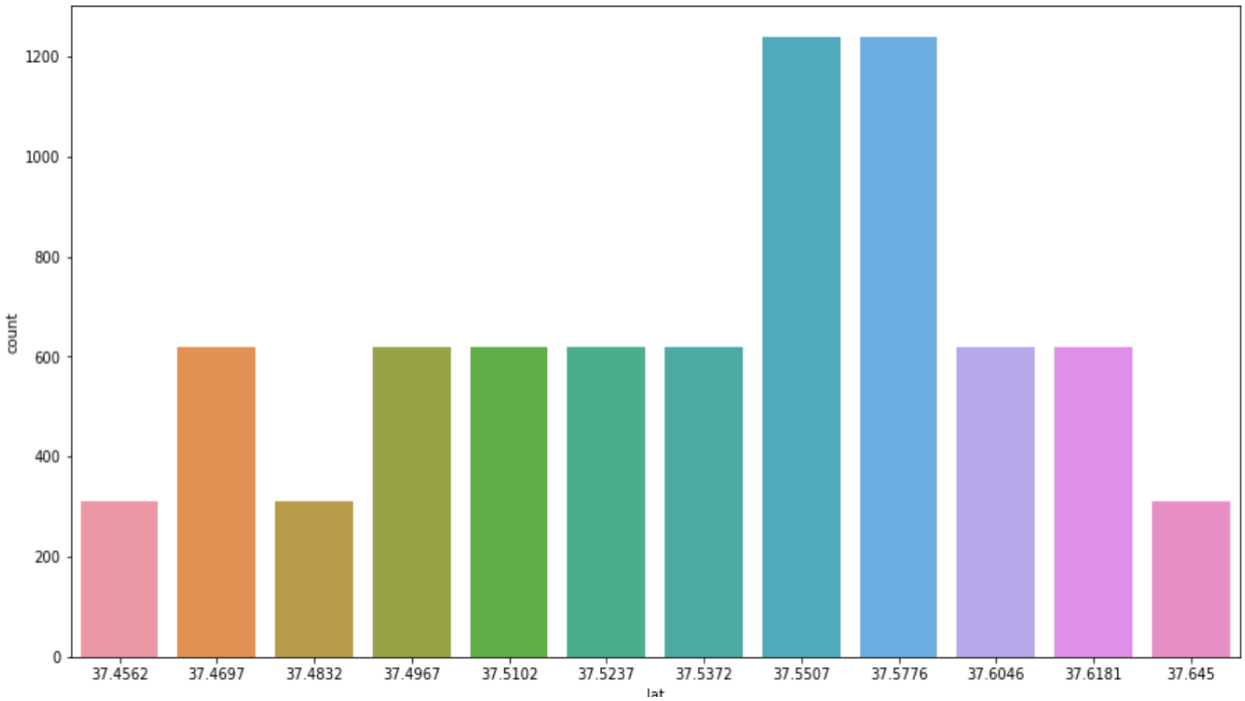
Heamap shows the null Values are present in dataset.

Graphical representation

We will start with Univariate Analysis. We will be using a bar graph for this purpose. We will look at the distribution of students across gender, race/ethnicity, their lunch status, and whether they have a test preparation course or not

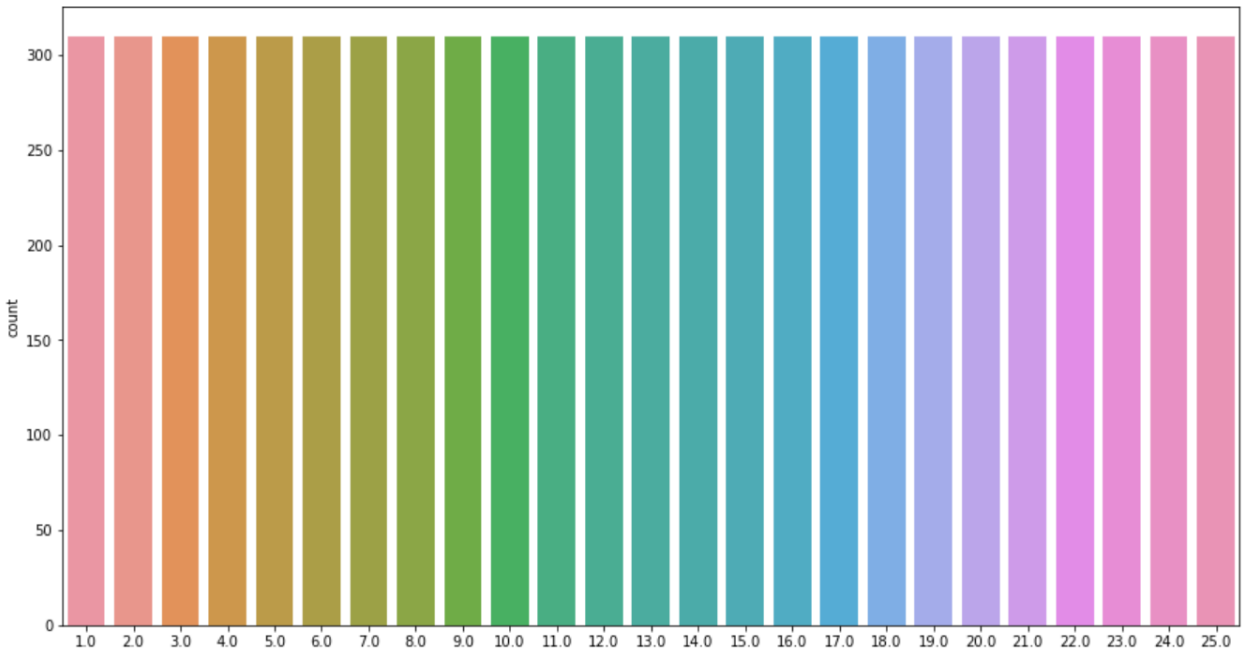
Univarient Analysis:

Univariate analysis **explores each variable in a data set**, separately. It looks at the range of values, as well as the central tendency of the values. It describes the pattern of response to the variable. It describes each variable on its own.



Univarient analysis of Lat column.

```
sns.countplot(df['station']);
```



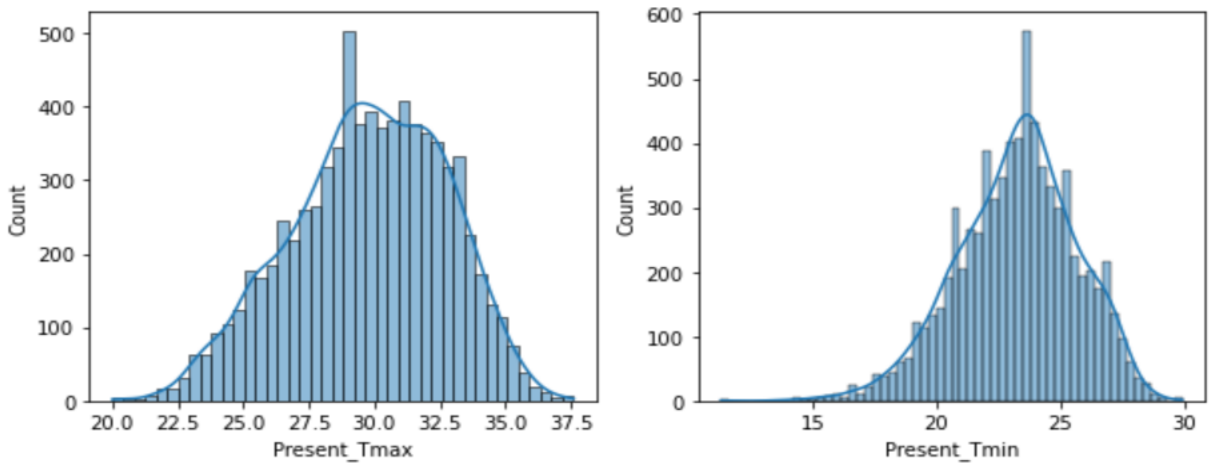
Univarient analysis of Station column.

Checking Distribution:

We will now make a **distribution plot** of the math score of the students. A distribution plot tells us how the data is distributed. We will use the distplot function.

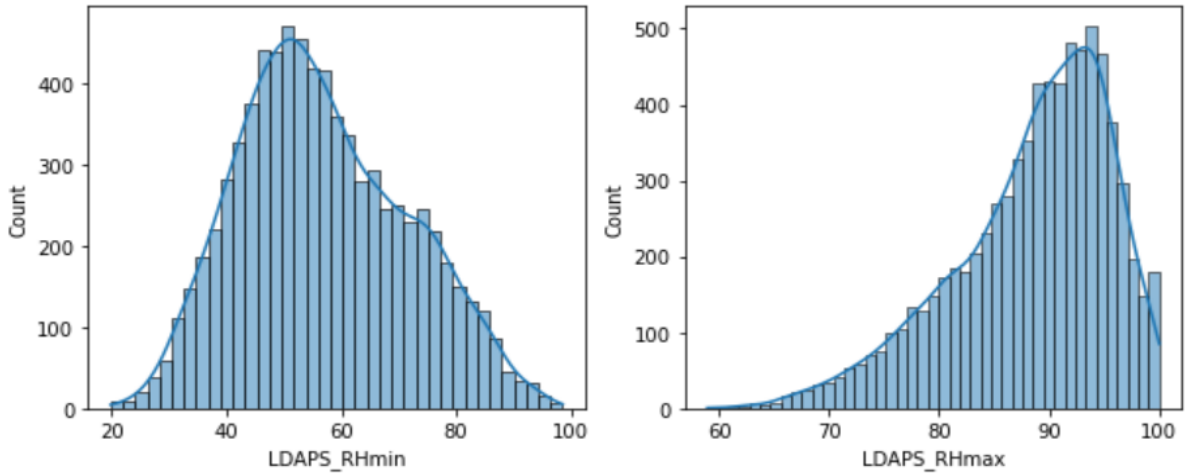
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.histplot(df['Present_Tmax'],kde=True)
plt.subplot(1,2,2)
sns.histplot(df['Present_Tmin'],kde=True)
```

```
<AxesSubplot:xlabel='Present_Tmin', ylabel='Count'>
```



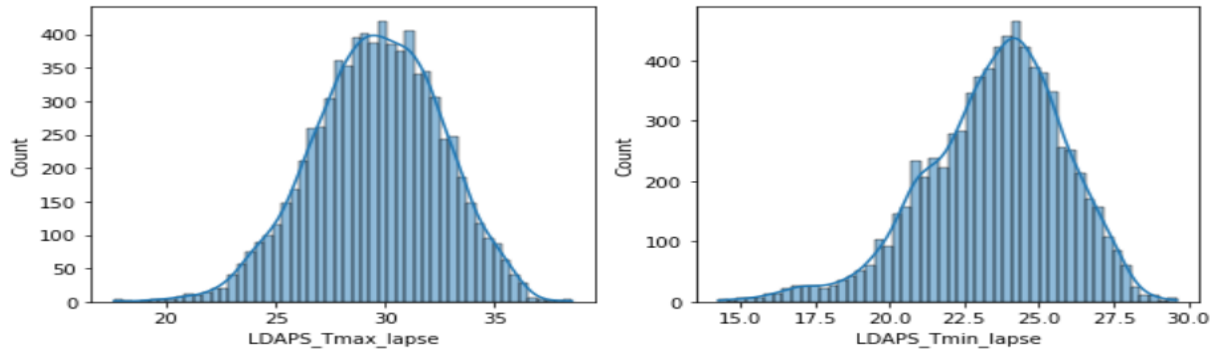

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.histplot(df['LDAPS_RHmin'],kde=True)
plt.subplot(1,2,2)
sns.histplot(df['LDAPS_RHmax'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_RHmax', ylabel='Count'>
```



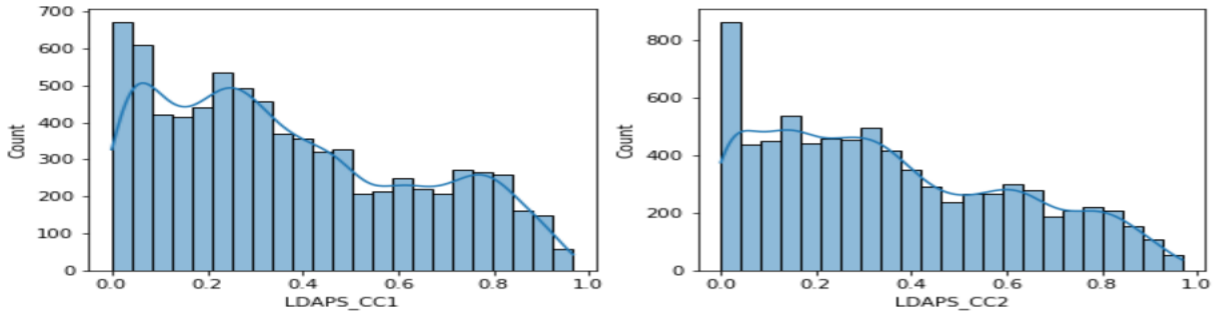
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.histplot(df['LDAPS_Tmax_lapse'],kde=True)
plt.subplot(1,2,2)
sns.histplot(df['LDAPS_Tmin_lapse'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_Tmin_lapse', ylabel='Count'>
```



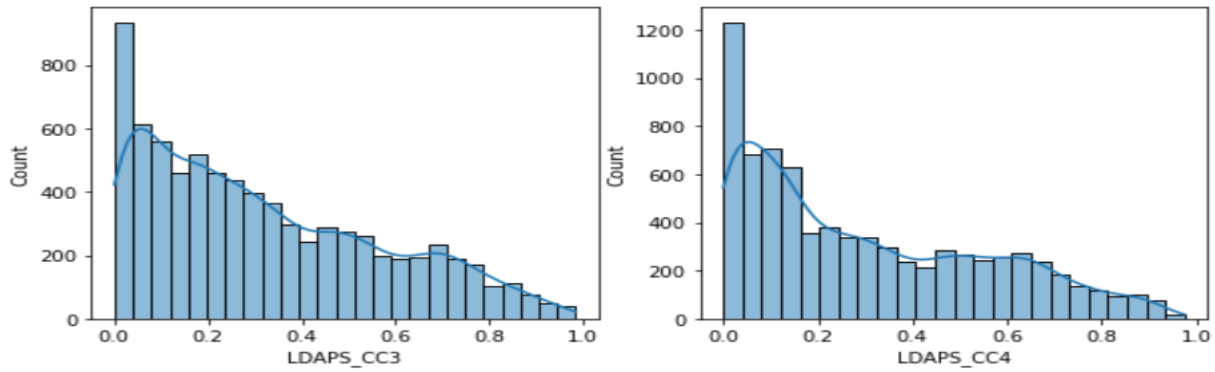

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.histplot(df['LDAPS_CC1'],kde=True)
plt.subplot(1,2,2)
sns.histplot(df['LDAPS_CC2'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_CC2', ylabel='Count'>
```



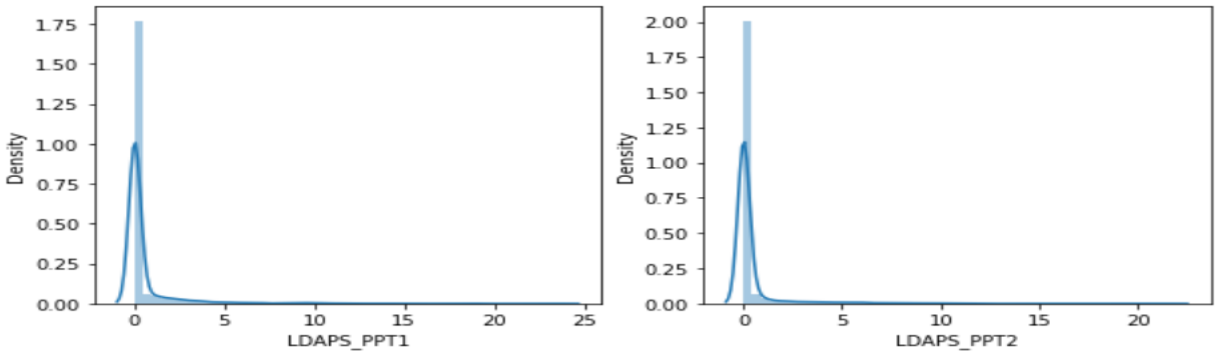

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.histplot(df['LDAPS_CC3'],kde=True)
plt.subplot(1,2,2)
sns.histplot(df['LDAPS_CC4'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_CC4', ylabel='Count'>
```



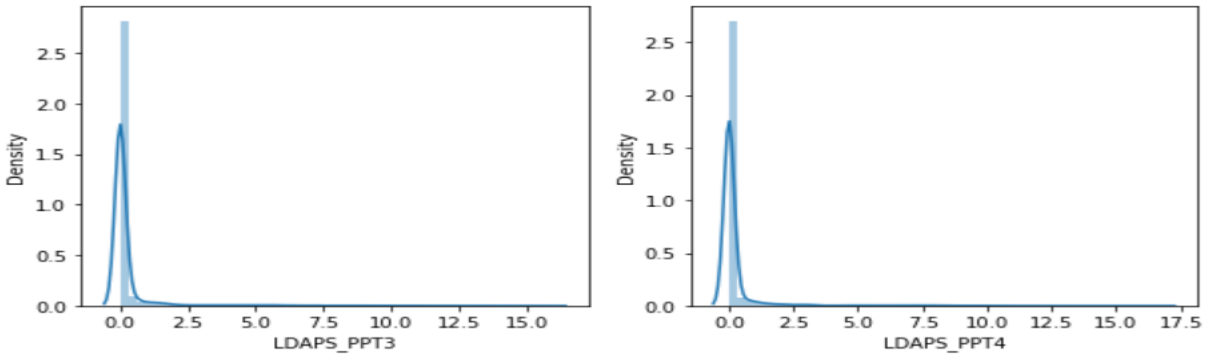
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(df['LDAPS_PPT1'],kde=True)
plt.subplot(1,2,2)
sns.distplot(df['LDAPS_PPT2'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_PPT2', ylabel='Density'>
```



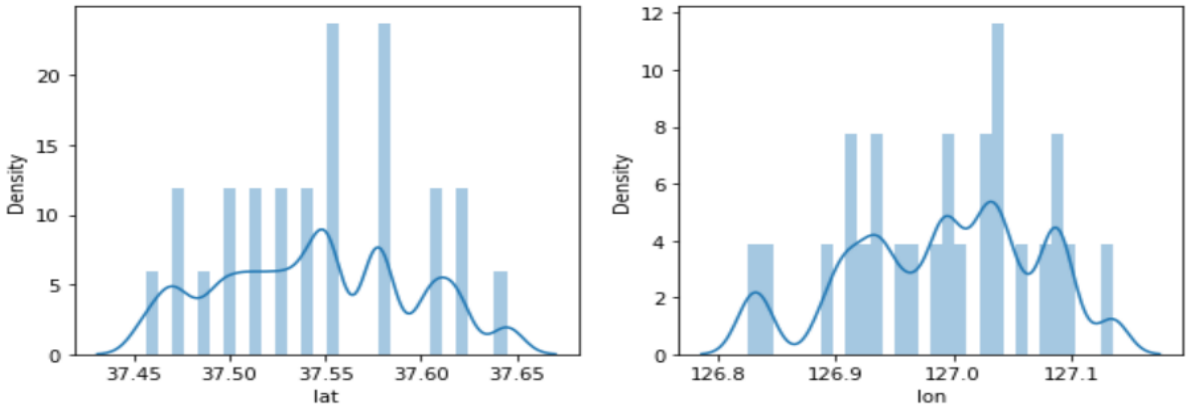

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(df['LDAPS_PPT3'],kde=True)
plt.subplot(1,2,2)
sns.distplot(df['LDAPS_PPT4'],kde=True)
```

```
<AxesSubplot:xlabel='LDAPS_PPT4', ylabel='Density'>
```



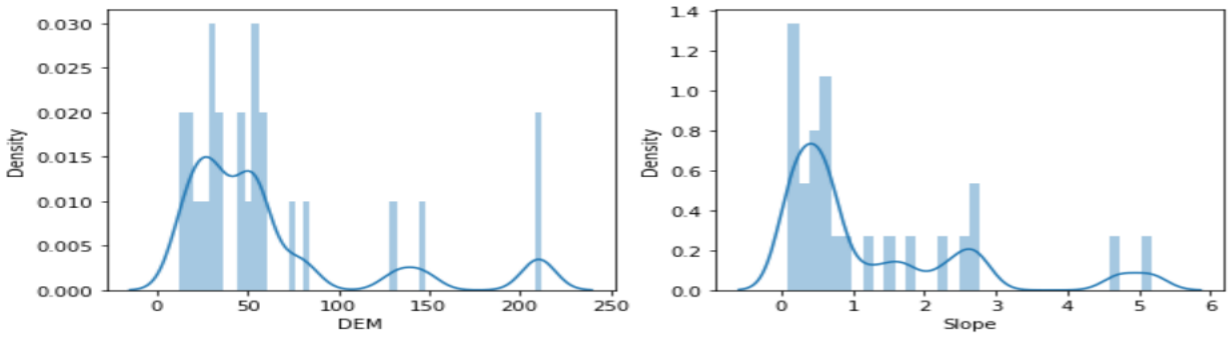

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(df['lat'],kde=True)
plt.subplot(1,2,2)
sns.distplot(df['lon'],kde=True)
```

```
<AxesSubplot:xlabel='lon', ylabel='Density'>
```



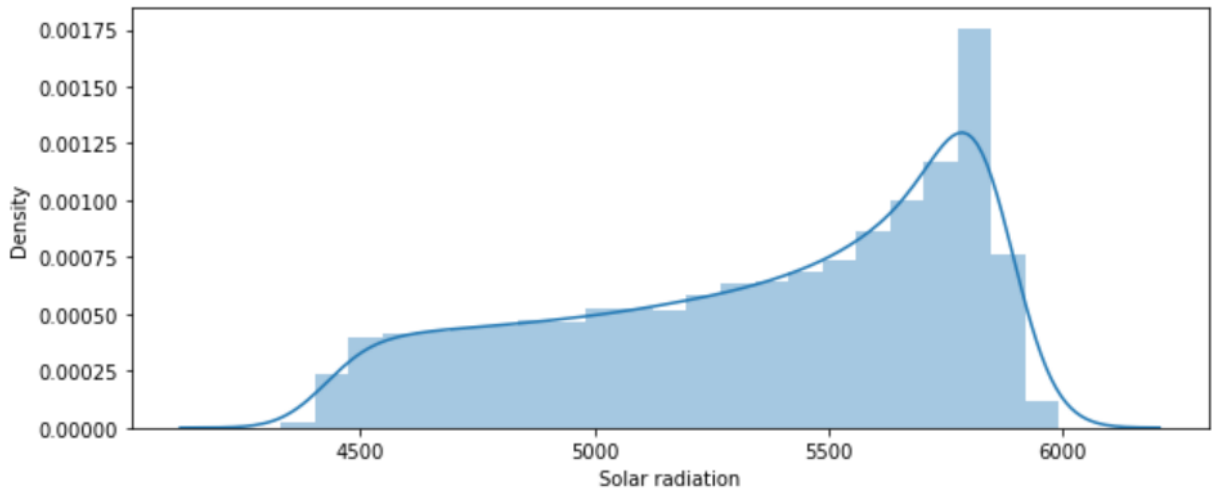
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(df['DEM'],kde=True)
plt.subplot(1,2,2)
sns.distplot(df['Slope'],kde=True)
```

```
<AxesSubplot:xlabel='Slope', ylabel='Density'>
```



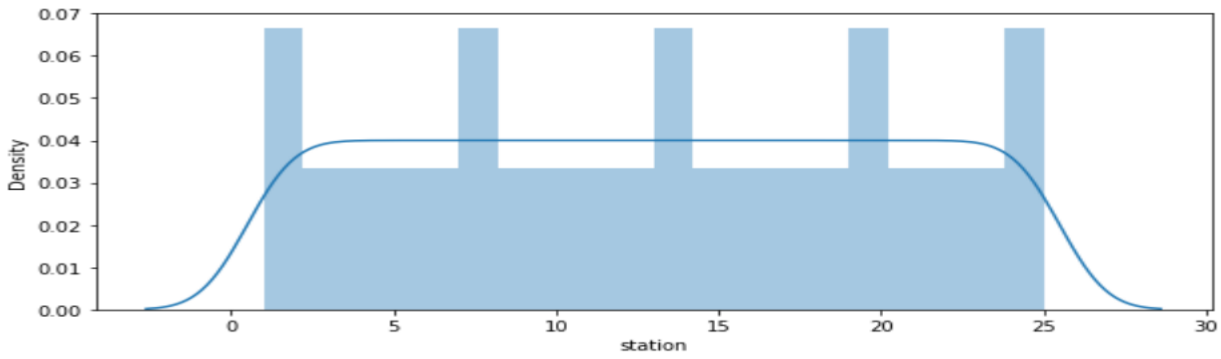

```
plt.figure(figsize=(10,4))#plt.subplot(1,2,1)
sns.distplot(df['Solar radiation'],kde=True)
```

```
<AxesSubplot:xlabel='Solar radiation', ylabel='Density'>
```



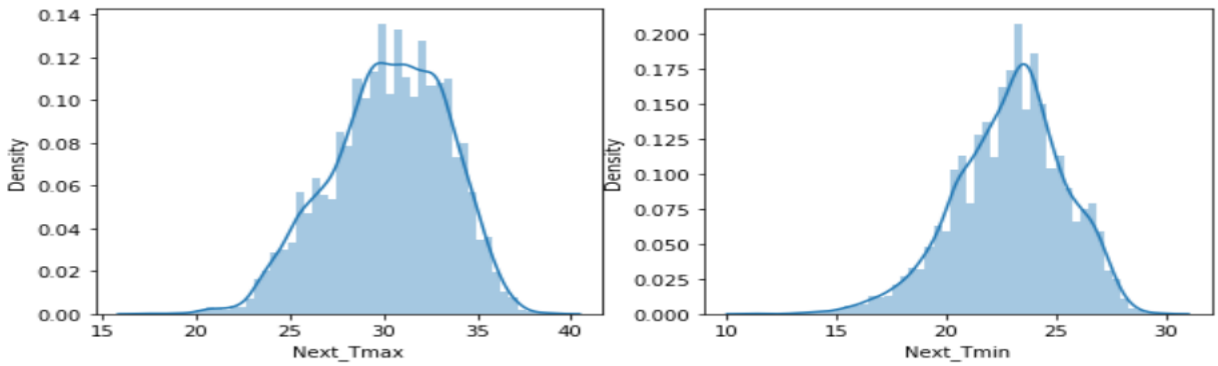

```
plt.figure(figsize=(10,4))#plt.subplot(1,2,1)
sns.distplot(df['station'],kde=True)
```

```
<AxesSubplot:xlabel='station', ylabel='Density'>
```



```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(df['Next_Tmax'],kde=True)
plt.subplot(1,2,2)
sns.distplot(df['Next_Tmin'],kde=True)
```

```
<AxesSubplot:xlabel='Next_Tmin', ylabel='Density'>
```



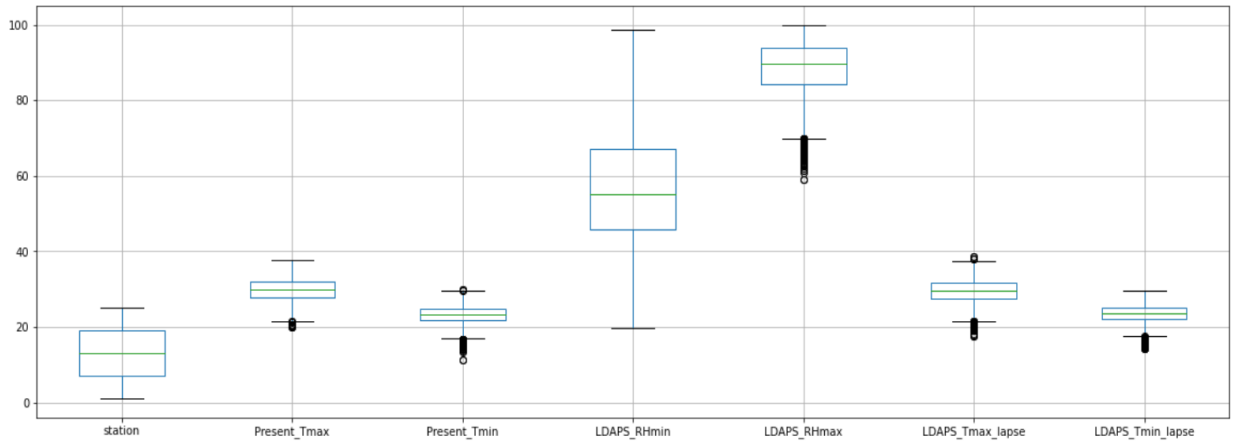
Here data some column is normally distributed and some column are not normally distributed.

Checking Outliers:

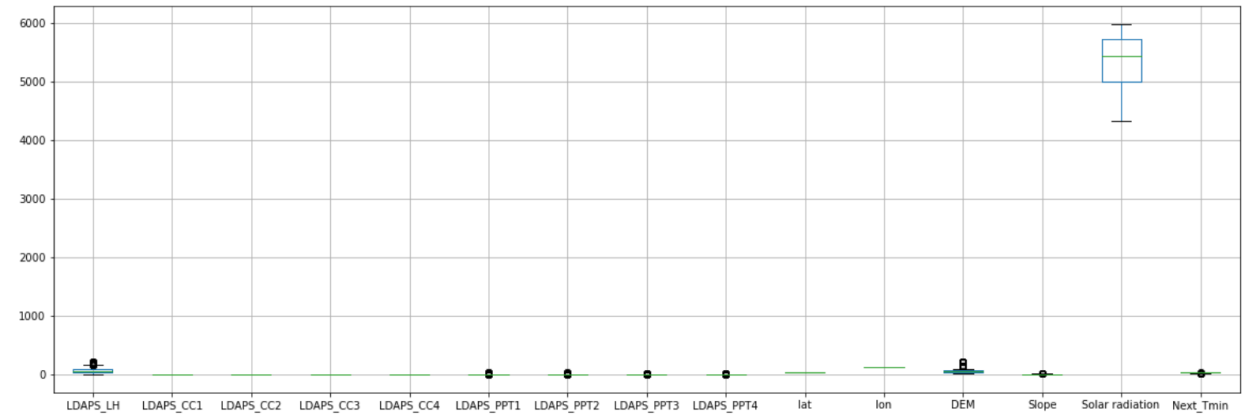
An outlier is an **object(s) that deviates significantly from the rest of the object collection**. It is an abnormal observation during the Data Analysis stage, that data point lies far away from other values. An outlier is an observation that diverges from well-structured data

```
x=df.drop(["Next_Tmax"],axis=1)
y=df["Next_Tmax"]
```

```
x.iloc[:,0:8].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



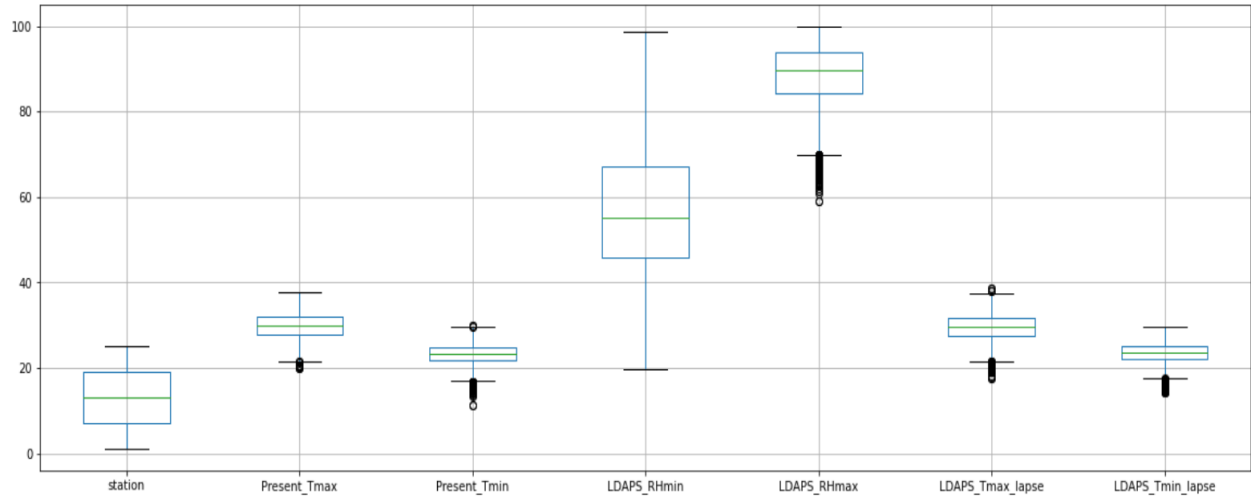
```
x.iloc[:,9:25].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



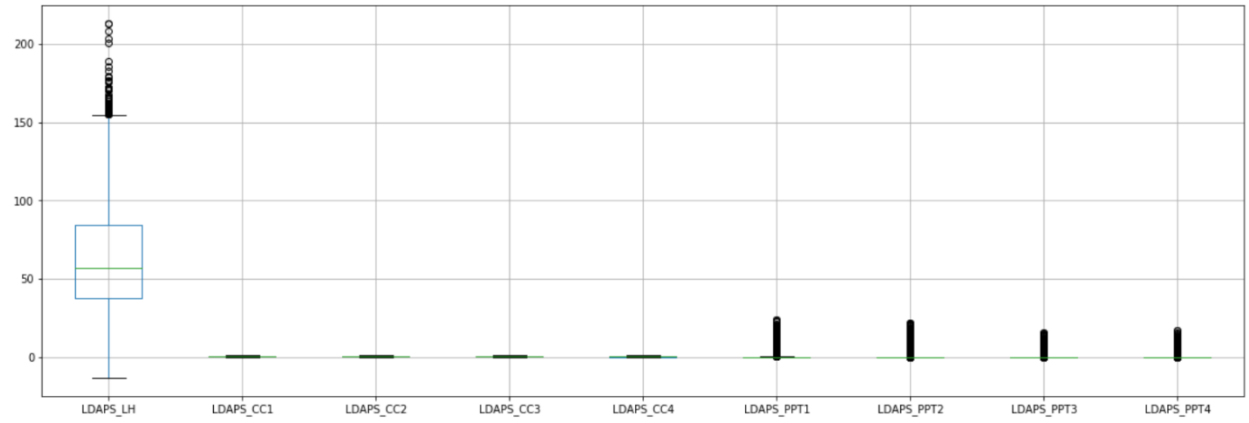
Now use Next_Tmax as a target variable and check outliers of all the columns.

```
x=df.drop(["Next_Tmin"],axis=1)
y=df["Next_Tmin"]
```

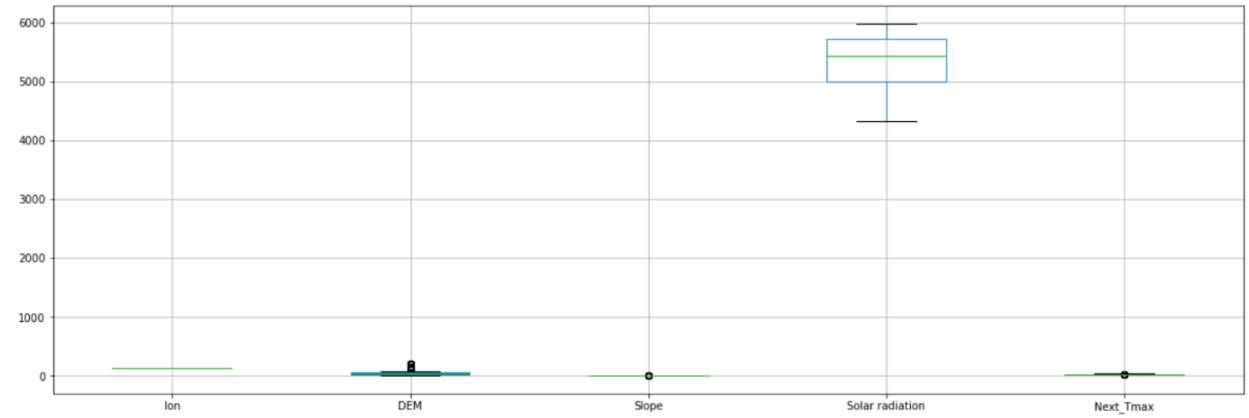
```
x.iloc[:,0:8].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



```
x.iloc[:,9:18].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```




```
x.iloc[:,19:25].boxplot(figsize=[20,8])
plt.subplots_adjust(bottom=0.25)
plt.show()
```



Now use Next_Tmin as a target variable and check outliers of all the columns.

Here outliers are present in many Columns Station, Date, Present_Tmax Present_Tmin, 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, DEM, Slope, Next_Tmax, Next_Tmin columns. All the column in dataset

In above Next_Tin use as target variable and check the outliers of all the columns in dataset.

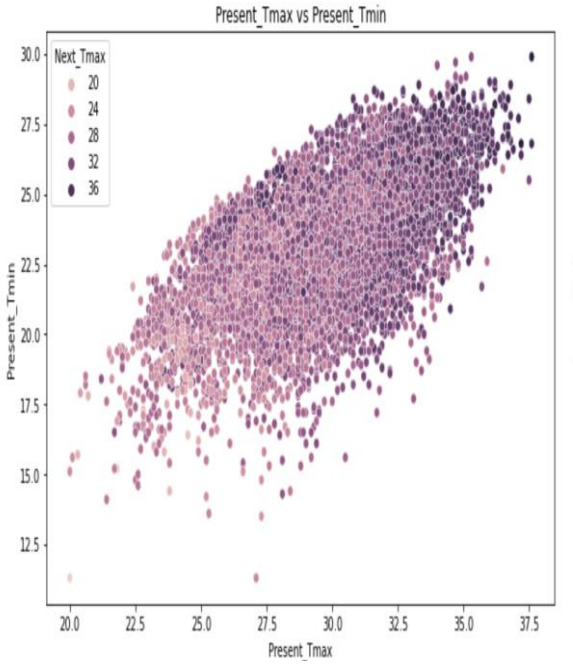
Here outliers are present in many Columns Station, Date, Present_Tmax Present_Tmin, LDAPS_RHmax, LDAPS_Tmax_lapse, LDAPS_Tmin_lapse, LDAPS_WS, LDAPS_LH, LDAPS_CC1, LDAPS_CC2, LDAPS_CC3, LDAPS_CC4, LDAPS_PPT1, LDAPS_PPT2, LDAPS_PPT3, LDAPS_PPT4, lat, DEM, Slope, Next_Tmax, Next_Tmin columns. All the column in dataset.

Bi varient Analysis:

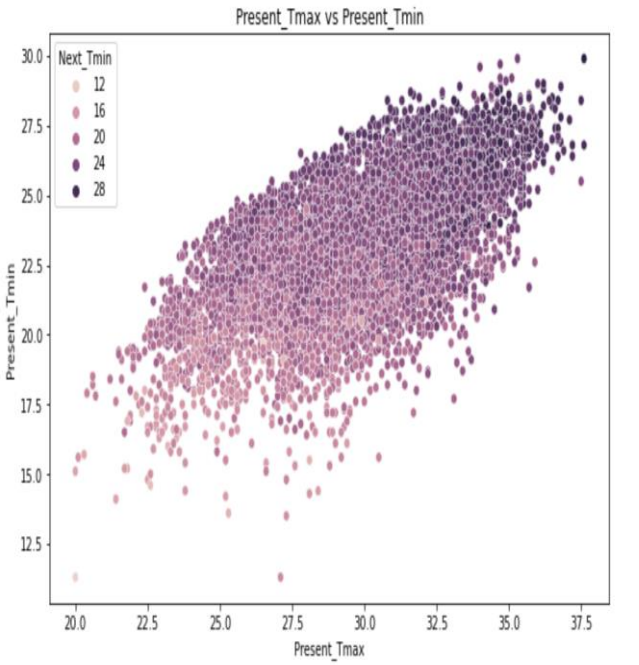
Scatter Plot

A scatter plot represents individual pieces of data using dots. These plots make it easier to see if two variables are related to each other. The resulting pattern indicates the type (linear or non-linear) and strength of the relationship between two variables.

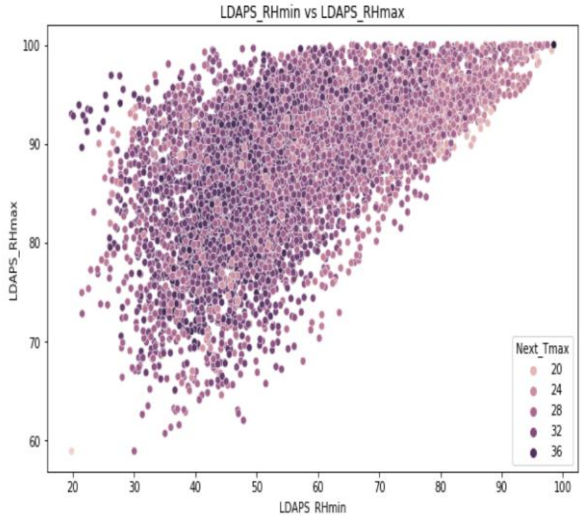
```
plt.figure(figsize=[10,6])
plt.title('Present_Tmax vs Present_Tmin')
sns.scatterplot(df["Present_Tmax"],df["Present_Tmin"],hue=df["Next_Tmax"])
plt.show()
```



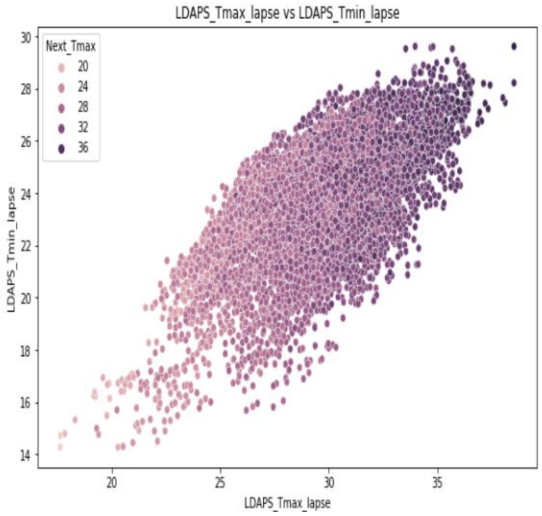

```
plt.figure(figsize=[10,6])
plt.title('Present_Tmax vs Present_Tmin')
sns.scatterplot(df["Present_Tmax"],df["Present_Tmin"],hue=df["Next_Tmin"])
plt.show()
```



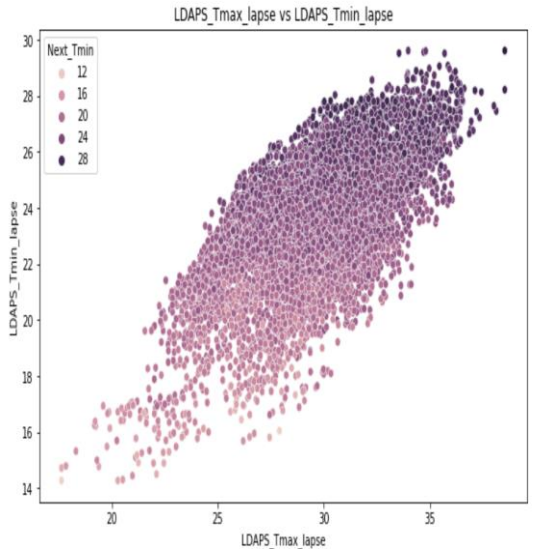
```
plt.figure(figsize=[10,6])
plt.title('LDAPS_RHmin vs LDAPS_RHmax')
sns.scatterplot(df["LDAPS_RHmin"],df["LDAPS_RHmax"],hue=df["Next_Tmax"])
plt.show()
```



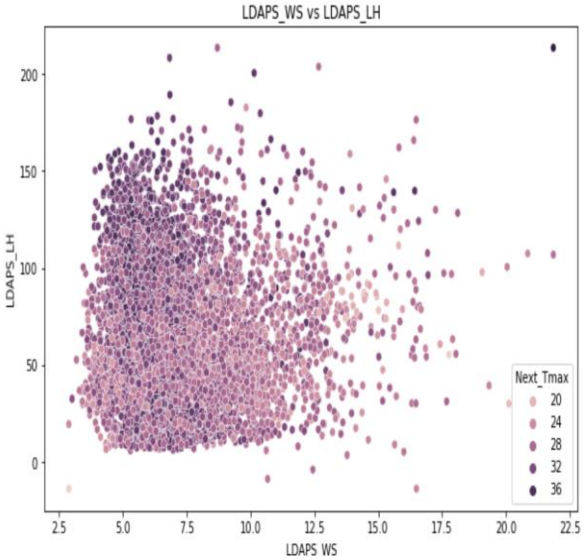

```
plt.figure(figsize=[10,6])
plt.title('LDAPS_Tmax_lapse vs LDAPS_Tmin_lapse')
sns.scatterplot(df["LDAPS_Tmax_lapse"],df["LDAPS_Tmin_lapse"],hue=df["Next_Tmax"]
plt.show()
```



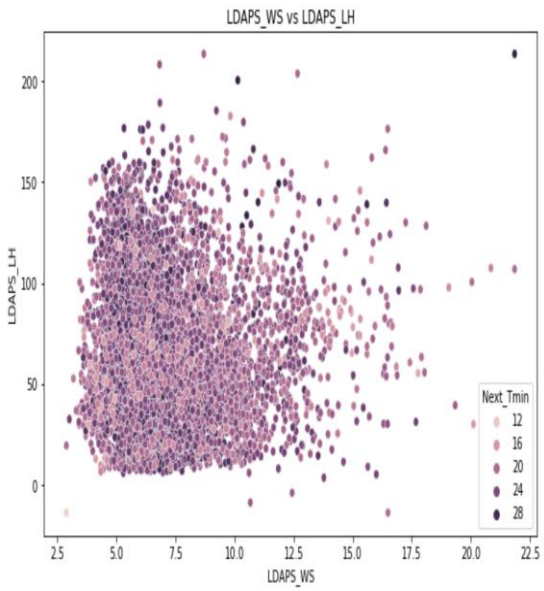

```
plt.figure(figsize=[10,6])
plt.title('LDAPS_Tmax_lapse vs LDAPS_Tmin_lapse')
sns.scatterplot(df["LDAPS_Tmax_lapse"],df["LDAPS_Tmin_lapse"],hue=df["Next_Tmin"])
plt.show()
```



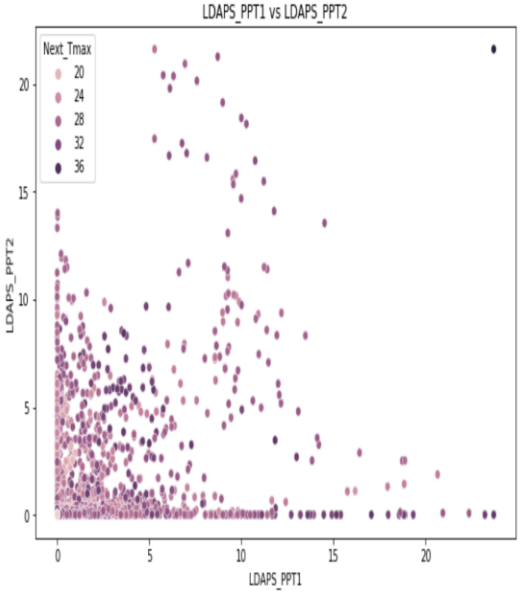

```
plt.figure(figsize=[10,6])
plt.title('LDAPS_WS vs LDAPS_LH')
sns.scatterplot(df["LDAPS_WS"],df["LDAPS_LH"],hue=df["Next_Tmax"])
plt.show()
```



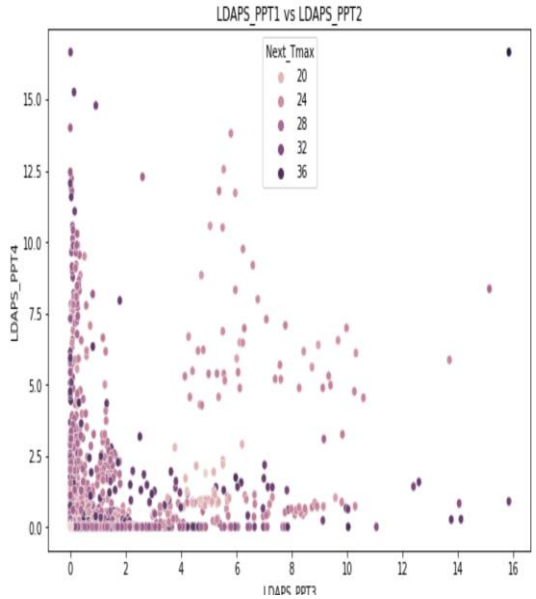
```
plt.figure(figsize=[10,6])
plt.title('LDAPS_WS vs LDAPS_LH')
sns.scatterplot(df["LDAPS_WS"],df["LDAPS_LH"],hue=df["Next_Tmin"])
plt.show()
```



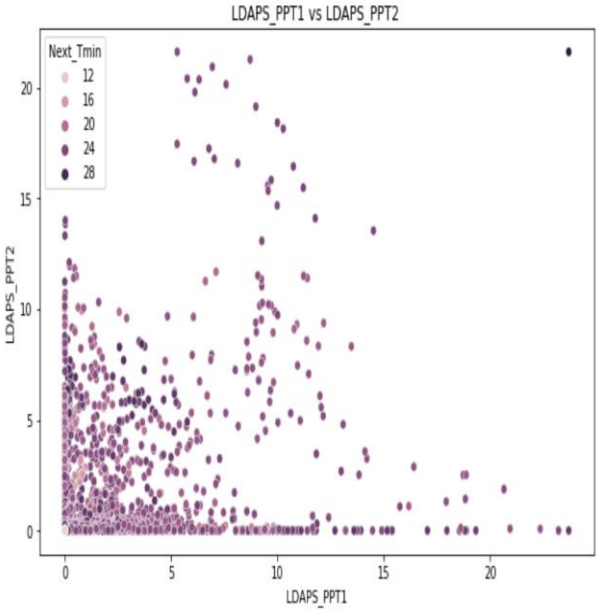

```
plt.figure(figsize=[10,6])
plt.title('LDAPS_PPT1 vs LDAPS_PPT2')
sns.scatterplot(df["LDAPS_PPT1"],df["LDAPS_PPT2"],hue=df["Next_Tmax"])
plt.show()
```



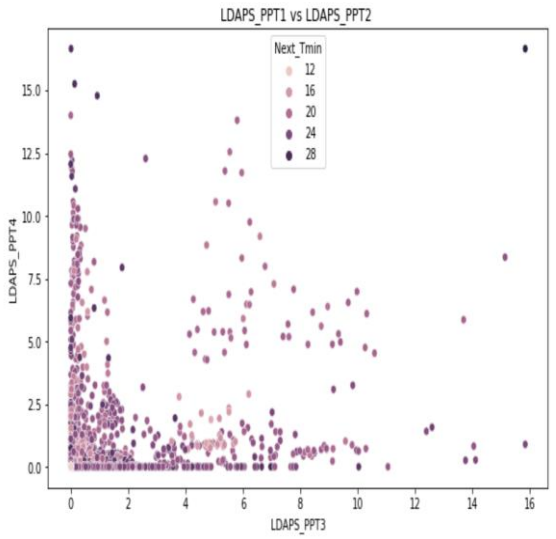

```
plt.title('LDAPS_PPT1 vs LDAPS_PPT2')
sns.scatterplot(df["LDAPS_PPT3"],df["LDAPS_PPT4"],hue=df["Next_Tmax"])
plt.show()
```



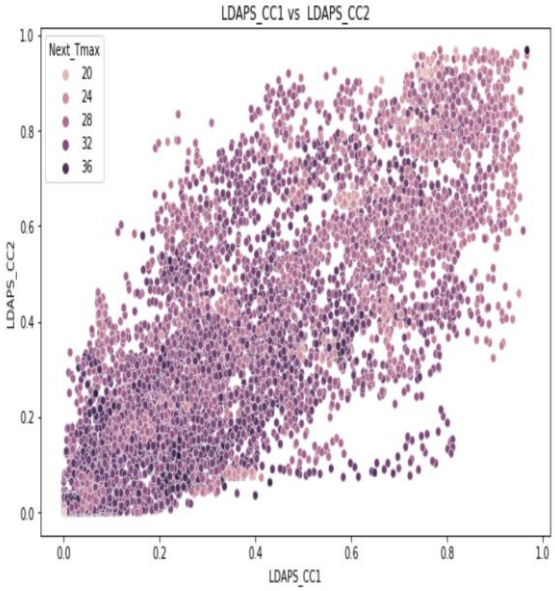

```
plt.figure(figsize=[10,6])
plt.title('LDAPS_PPT1 vs LDAPS_PPT2 ')
sns.scatterplot(df["LDAPS_PPT1"],df["LDAPS_PPT2"],hue=df["Next_Tmin"])
plt.show()
```



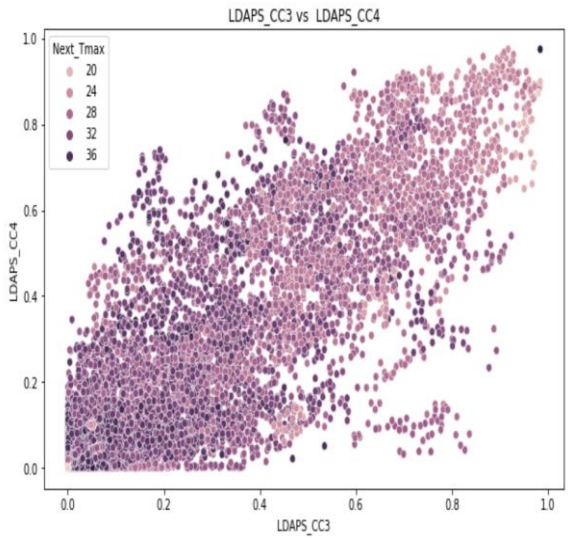
```
plt.figure(figsize=[10,6])
plt.title('LDAPS_PPT1 vs LDAPS_PPT2 ')
sns.scatterplot(df["LDAPS_PPT3"],df["LDAPS_PPT4"],hue=df["Next_Tmin"])
plt.show()
```



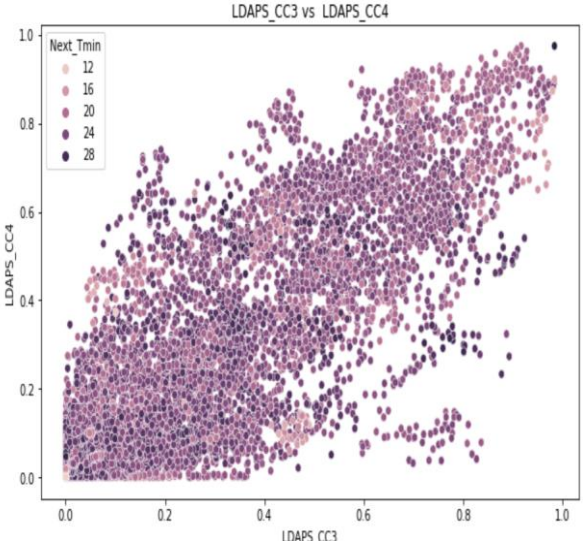

```
plt.figure(figsize=[10,6])
plt.title(' LDAPS_CC1 vs  LDAPS_CC2 ')
sns.scatterplot(df["LDAPS_CC1"],df["LDAPS_CC2"],hue=df["Next_Tmax"])
plt.show()
```




```
plt.figure(figsize=[10,6])
plt.title(' LDAPS_CC3 vs  LDAPS_CC4 ')
sns.scatterplot(df["LDAPS_CC3"],df["LDAPS_CC4"],hue=df["Next_Tmax"])
plt.show()
```




```
plt.figure(figsize=[10,6])
plt.title(' LDAPS_CC3 vs  LDAPS_CC4 ')
sns.scatterplot(df["LDAPS_CC3"],df["LDAPS_CC4"],hue=df["Next_Tmin"])
plt.show()
```



Here Above Scatterplot shows the comparision between two columns In which some column use Next_Tmax as a Target column and Next_Tmin as a Target column.

Multi Varien Analysis:

Multivariate analysis is used **to study more complex sets of data than what univariate analysis methods can handle**. ... Multivariate analysis can reduce the likelihood of Type errors. Sometimes, univariate analysis is preferred as multivariate techniques can result in difficulty interpreting the results of the test.

Histplot:

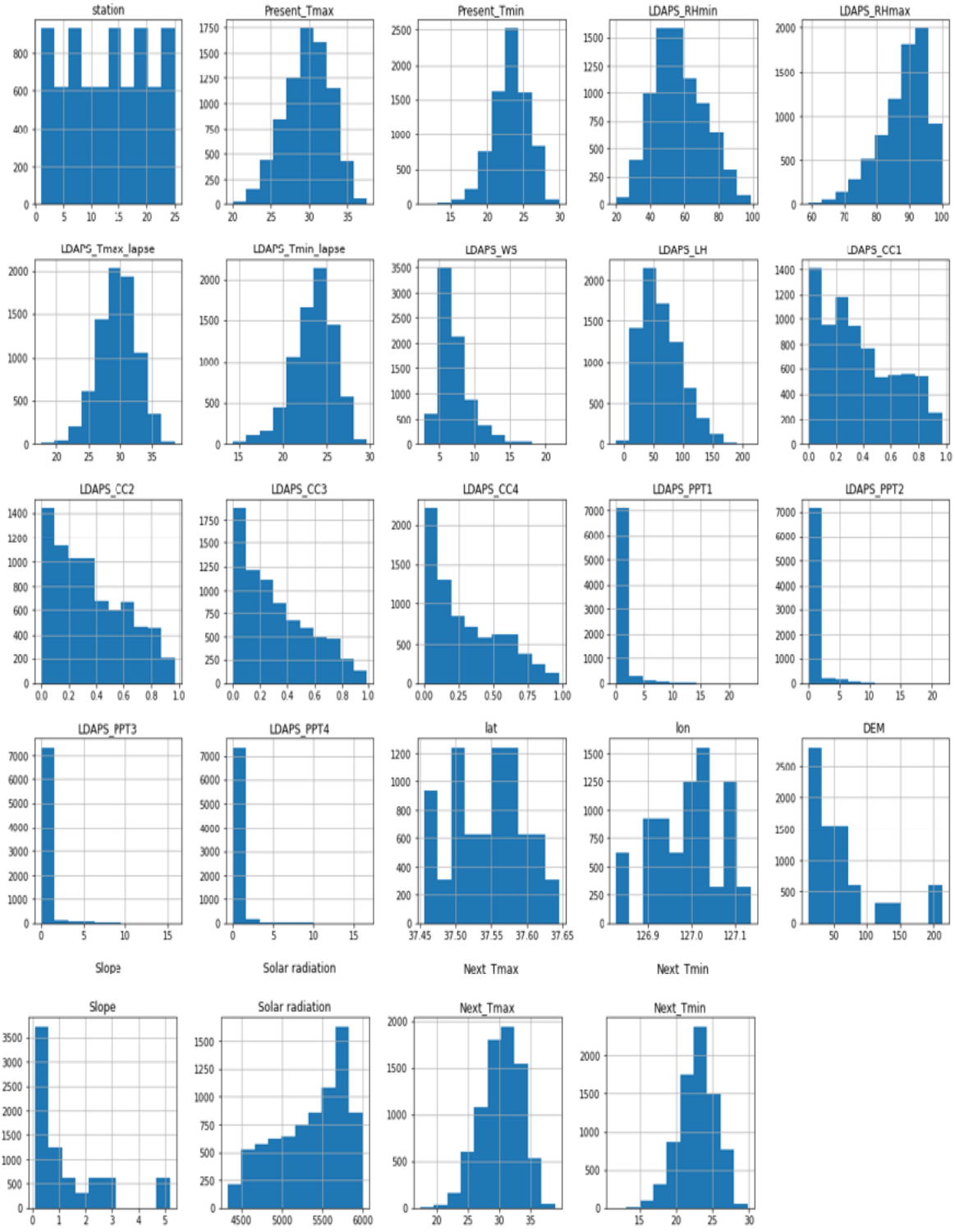
A histogram is a classic visualization tool that represents the .

A histogram provides **a visual representation of the distribution of a dataset**:

location, spread and skewness of the data; it also helps to visualize whether the

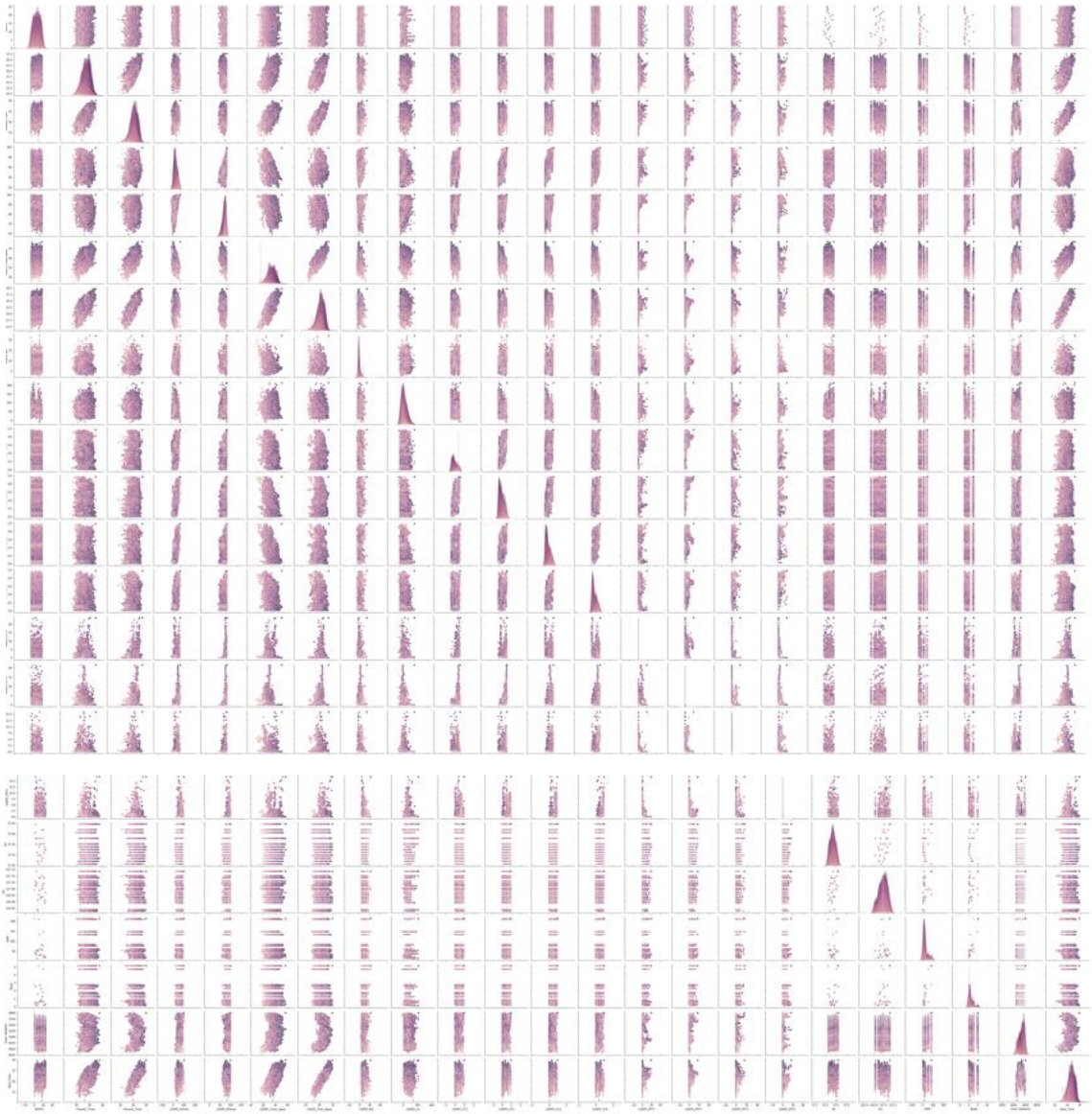
distribution is symmetric or skewed left or right. ... In brief, a histogram summarizes

the distribution properties of a continuous numerical variable.

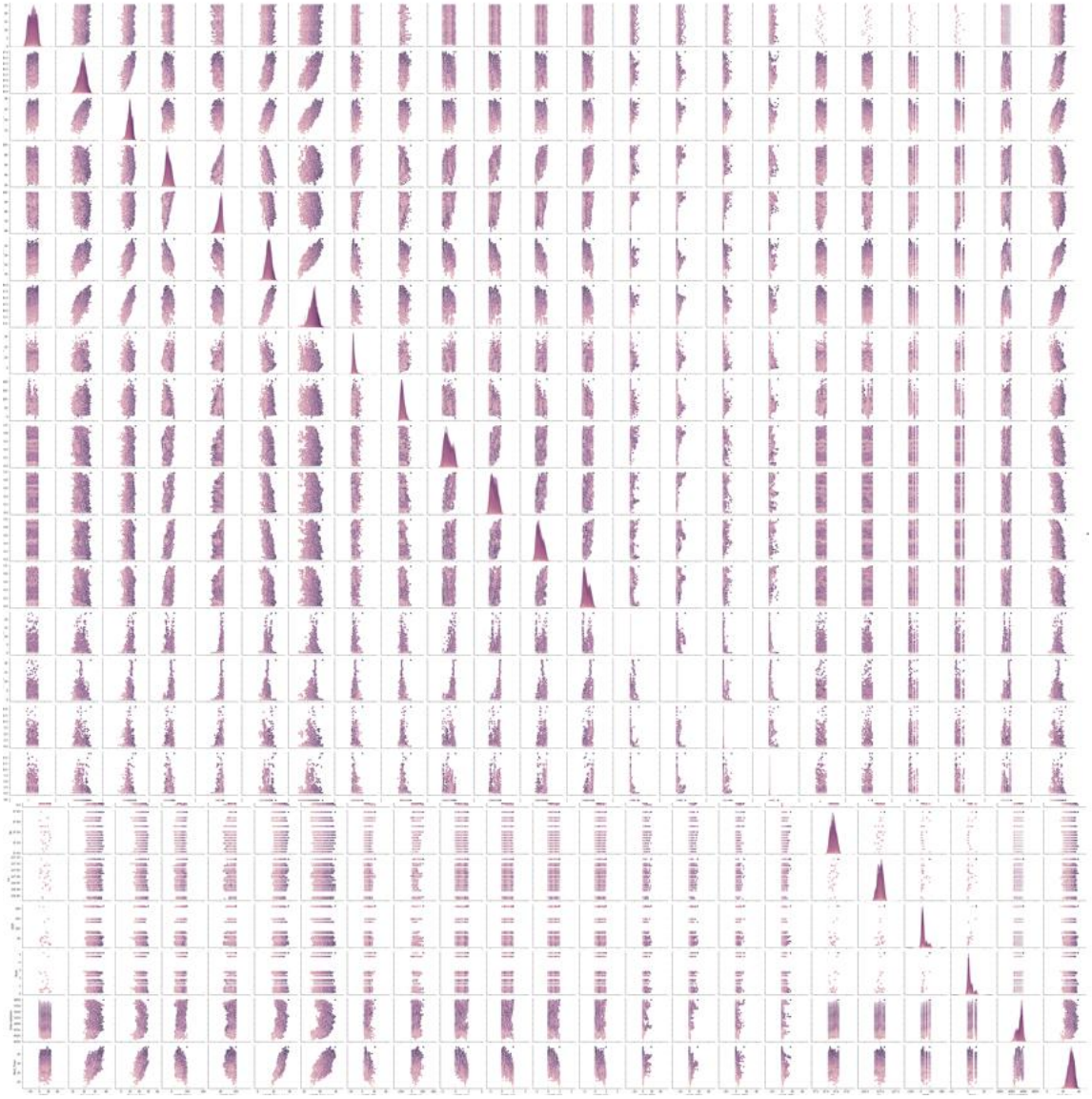

station
Present_Tmax
Present_Tmin
LDAPS_RHmin
LDAPS_RHmax
LDAPS_Tmax_lapse
LDAPS_Tmin_lapse
LDAPS_WS
LDAPS_LH
LDAPS_CC1
LDAPS_CC2
LDAPS_CC3
LDAPS_CC4
LDAPS_PPT1
LDAPS_PPT2
LDAPS_PPT3
LDAPS_PPT4
lat
lon
DEM
Slope
Solar radiation
Next_Tmax
Next_Tmin

PairPlot Hue Target column as Next_Tmax:

Pairplot **visualizes given data to find the relationship between them where the variables can be continuous or categorical**. Plot pairwise relationships in a data-set. Pairplot is a module of seaborn library which provides a high-level interface for drawing attractive and informative statistical graphics.



PairPlot Hue Target column as Next_Tmin:



Correlation:

Correlation heatmap is **graphical representation of correlation matrix representing correlation between different variables**. The value of correlation can take any values from -1 to 1. ... Correlation between two variables can also be determined using scatter plot between these two variables

```
cor=df.corr()
```

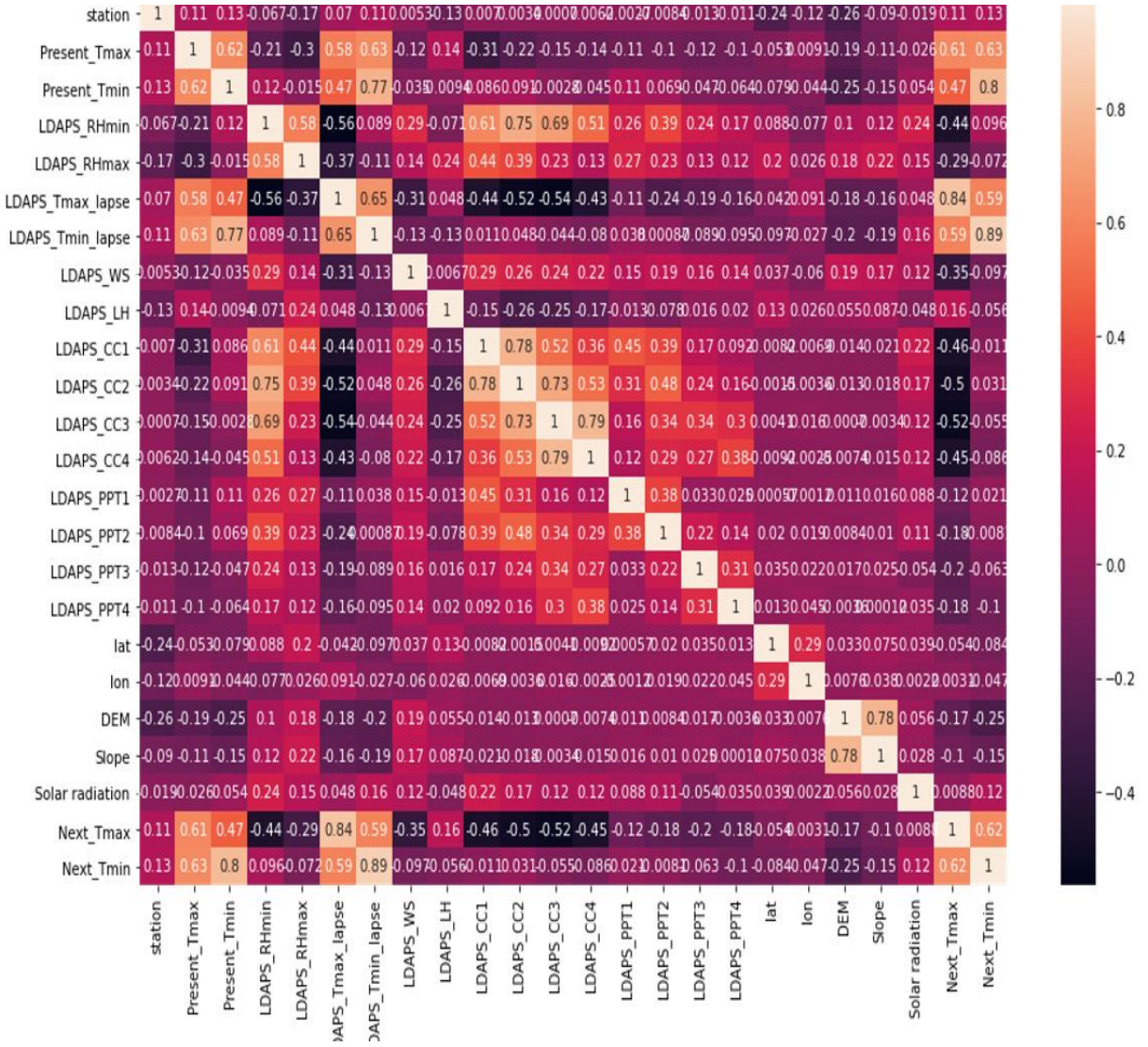
```
cor
```

	station	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS
station	1.000000	0.113211	0.132419	-0.067122	-0.168010	0.069561	0.105422	0.005320
Present_Tmax	0.113211	1.000000	0.618760	-0.206782	-0.304209	0.575289	0.629247	-0.122876
Present_Tmin	0.132419	0.618760	1.000000	0.124921	-0.015334	0.470118	0.772921	-0.035111
LDAPS_RHmin	-0.067122	-0.206782	0.124921	1.000000	0.579141	-0.564580	0.089476	0.294361
LDAPS_RHmax	-0.168010	-0.304209	-0.015334	0.579141	1.000000	-0.373404	-0.114143	0.135333
LDAPS_Tmax_lapse	0.069561	0.575289	0.470118	-0.564580	-0.373404	1.000000	0.654021	-0.311996
LDAPS_Tmin_lapse	0.105422	0.629247	0.772921	0.089476	-0.114143	0.654021	1.000000	-0.130035
LDAPS_WS	0.005320	-0.122876	-0.035111	0.294361	0.135333	-0.311996	-0.130035	1.000000
LDAPS_LH	-0.134226	0.136716	-0.009368	-0.070858	0.238579	0.048010	-0.134761	0.006711
LDAPS_CC1	0.006956	-0.314863	0.085593	0.613818	0.436652	-0.438439	0.010901	0.289445
LDAPS_CC2	0.003414	-0.215816	0.091439	0.745443	0.391330	-0.523619	0.047727	0.261090
LDAPS_CC3	0.000698	-0.145513	-0.002839	0.689679	0.226957	-0.541327	-0.044018	0.242991
LDAPS_CC4	0.006159	-0.142497	-0.045091	0.514075	0.130619	-0.429539	-0.080335	0.220533
LDAPS_PPT1	-0.002748	-0.110001	0.114312	0.262665	0.268449	-0.111065	0.038056	0.152587
LDAPS_PPT2	-0.008369	-0.100420	0.069095	0.390967	0.229050	-0.242122	0.000870	0.191886
LDAPS_PPT3	-0.012671	-0.121271	-0.046645	0.240642	0.134607	-0.188115	-0.088575	0.161684
LDAPS_PPT4	-0.010580	-0.101471	-0.064394	0.168595	0.117853	-0.160273	-0.094655	0.144085
lat	-0.237610	-0.052776	-0.078715	0.087523	0.196751	-0.042298	-0.096726	0.036836
lon	-0.118763	0.009075	-0.043725	-0.076608	0.025792	0.091107	-0.026831	-0.059756
DEM	-0.255970	-0.187855	-0.251257	0.102612	0.178031	-0.179766	-0.196407	0.191983
Slope	-0.090113	-0.106097	-0.146736	0.124346	0.220668	-0.163123	-0.186753	0.172464
Solar radiation	-0.019011	-0.025557	0.053828	0.244795	0.149699	0.048111	0.160443	0.122116
Next_Tmax	0.108306	0.613109	0.473868	-0.442958	-0.286478	0.836144	0.593195	-0.346592
Next_Tmin	0.128719	0.625252	0.799758	0.095830	-0.072490	0.591535	0.886964	-0.097390

HeatMap:

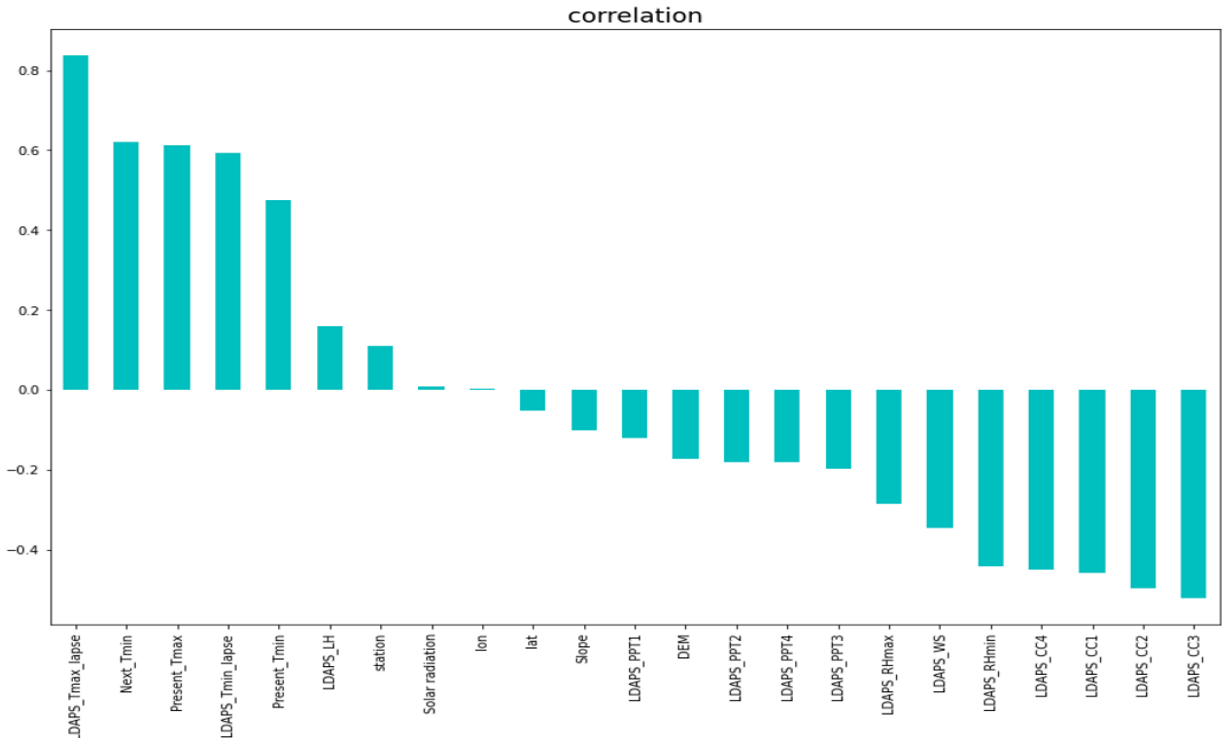
A heatmap is **a graphical representation of data in which data values are represented as colors**. That is, it uses color in order to communicate a value to the reader. This is a great tool to assist the audience towards the areas that matter the most when you have a large volume of data.

A **heat map** (or **heatmap**) is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space. There are two fundamentally different categories of heat maps: the cluster heat map and the spatial heat map. In a cluster heat map, magnitudes are laid out into a matrix of fixed cell size whose rows and columns are discrete phenomena and categories, and the sorting of rows and columns is intentional and somewhat arbitrary, with the goal of suggesting clusters or portraying them as discovered via statistical analysis. The size of the cell is arbitrary but large enough to be clearly visible. By contrast, the position of a magnitude in a spatial heat map is forced by the location of the magnitude in that space, and there is no notion of cells; the phenomenon is considered to vary continuously.



In Above heatmap Check correlation of all the independent column with target column there are two target variables are present here Next_Tmax, Next_Tmin. In above heat map Next_Tmax as atarget variable column , LDAPS_Tmax_lapse is highly correlated with target variable and Next_Tmin as a target variable column LDAPS_HRmin is highly correlated with target variable.

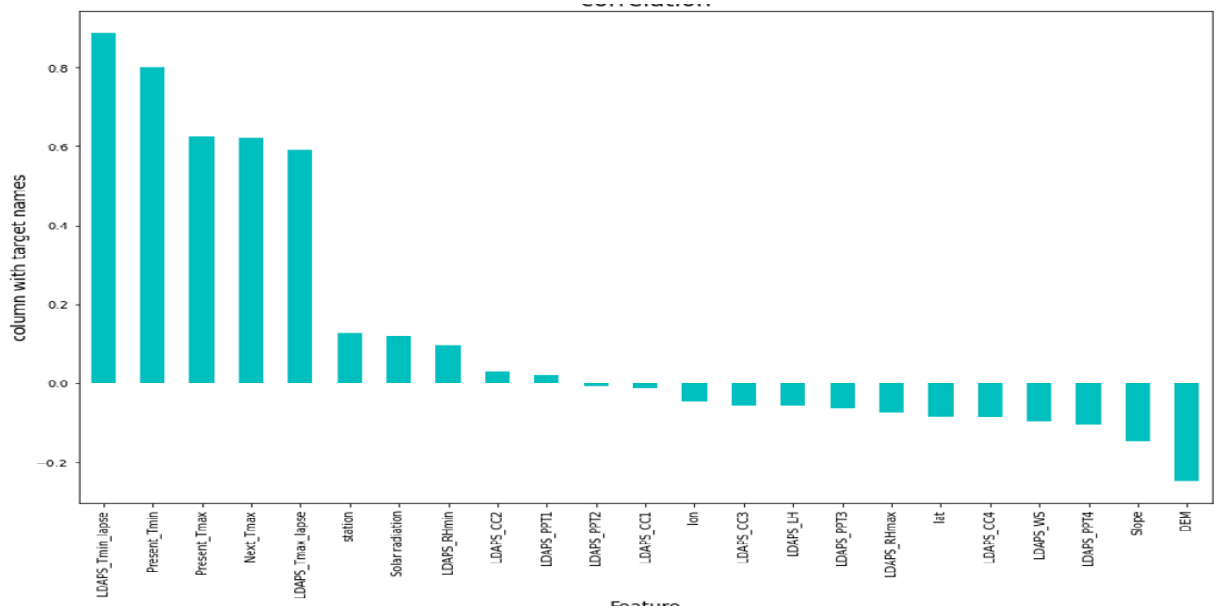
```
plt.figure(figsize=(15,10))
df.corr()['Next_Tmax'].sort_values(ascending=False).drop(['Next_Tmax']).plot(kind='bar',color='c') plt.xlabel('Feature',fontsize=14)
plt.ylabel('column with target names',fontsize=14)
plt.title('correlation',fontsize=18) plt.show()
```



Here Check correlation another way use Next_Tmax as target vriable. In this columns are positively correlated with target variable some columns are negatively correlated with target variable.

1. LDAPS_Tmax_lapse
2. Next_Tmin
3. Present_Tmax
4. LDAPS_Tmin_lapse
5. Present_Tmin
6. LDAPS_LH
7. Station

```
plt.figure(figsize=(15,10))
df.corr()['Next_Tmin'].sort_values(ascending=False).drop(['Next_Tmin']).plot(kind='bar',color='c')
plt.xlabel('Feature',fontsize=14)
plt.ylabel('column with target names',fontsize=14)
plt.title('correlation',fontsize=18)
plt.show()
```



Here Check correlation another way use Next_Tmax as target vriable. In this columns are positively correlated with target variable some columns are negatively correlated with target variable.

Columns which are positively correlated with target variable:

1.Present_Tmax
2.Present_Tmin
3.LDAPS_RHmin
4.LDAPS_RHmin
5.LDAPS_Tmax_lapse
6.LDAPS_Tmin_lapse
7.Solar radiation
8.Next_Tmax
9.LDAPS_CC2
10.LDAPS_PPT1

Drop column:

To drop a single column or multiple columns from pandas dataframe in Python, you can use `df.drop` and other different methods.Columns from a DataFrame are dropped if they are not relevant to your analysis or problem you are trying to solve. When building a machine learning models, they are removed if it is redundant or doesn't help your model.The most common way to remove a column is using df.drop(). Sometimes, del command in Python is also used.

Using the drop method

You can use the drop method of Dataframes to drop single or multiple columns in different ways.

pandas.DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

Purpose: To drop the specified rows or columns from the DataFrame.

Parameters:

labels: single label or list (default: None). Used to specify the row or column index labels which are to be dropped.

axis: 0 or 1 (default: 0). Specify the orientation along which the labels are to be dropped. If the value of this parameter is set to 0 then the labels will be dropped along the rows and if it is set to 1, then the labels will be dropped along the columns.

index: single label or list (default:None). Alternative to the axis parameter for specifying the orientation along which the labels are to be dropped.

columns: single label or list (default: None). Alternative to specifying the orientation along which the labels are to be dropped.

level: int or level name (default: None). Specify the level in case multi-level indices are present from which the labels are to be removed.

inplace: Boolean (default: False). Specify if a copy of the DataFrame is to be returned or not. If the value of this parameter is set to 'False' then a copy of the original DataFrame will be returned. If it is set to 'True' then the changes will be made in the original DataFrame.

errors: 'ignore' or 'raise' (default: raise). Specify if the errors are to be raised or ignored. If the 'ignore' value is passed to this parameter then the error is suppressed and only the existing labels are dropped.

Dropping a single column

For dropping a single column, specify the name of that column in the Date parameter.

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

Here we drop Dtae column because it is not much important and all unique values.

Replace values of column:

Replacing column values in a Pandas dataframe changes every occurrence of that value in the column.

DataFrame.replace()

Access a specific pandas. DataFrame column
using DataFrame[column_name]. To replace values in the column, call DataFrame.replace(to_replace, inplace=True) with to_replace set as a dictionary mapping old values to new values.

```
df['Slope'].unique()
```

```
array([2.785    , 0.5141   , 0.2661   , 2.5348   , 0.5055   , 0.1457   ,
       0.0985   , 1.5629   , 0.4125   , 5.1782   , 0.6233   , 0.5931   ,
       2.6865   , 0.618    , 0.8552   , 2.2579   , 0.697    , 1.2313   ,
       1.7678   , 4.7296   , 0.5721   , 0.1332   , 0.1554   , 0.2223   ,
       0.2713   , 0.0984746, 5.17823  ])
```

```
df['Slope'].replace({0.0984746:0.0985,5.17823:5.1782},inplace=True)
```

Here we replace some value from slope column.

Replace Missing Values with Mean, Median & Mode:

how to impute or replace missing values with **mean, median** and **mode** in one or more numeric feature columns of **Pandas DataFrame** while building machine learning (ML) models with Python programming. You will also learn about **how to decide which technique to use** for **imputing missing values** with **central tendency measures** of feature column such as **mean, median** or **mode**. This is important to understand this technique for data scientists as **handling missing values** one of the key aspects of **data preprocessing** when training ML models.

Replace Missing Values with Mean:

One of the techniques is mean imputation in which the missing values are replaced with the mean value of the entire feature column. In the case of fields like salary, the data may be skewed as shown in the previous section. In such cases, it may not be a good idea to use mean imputation for replacing the missing values. Note that imputing missing data with mean values can only be done with **numerical data**.

```
df.fillna(df.mean())
```

Replace Missing Values with Median:

Another technique is **median imputation** in which the missing values are replaced with the **median value** of the entire feature column. When the data is skewed, it is good to consider using median value for replacing the missing values. Note that imputing missing data with median value can only be done with **numerical data**.

```
df.fillna(df.median())
```

Replace Missing Values with Mode:

Yet another technique is **mode imputation** in which the missing values are replaced with the **mode value** or **most frequent** value of the entire feature column. When the data is skewed, it is good to consider using mode values for replacing the missing values. For data points such as the salary field, you may consider using mode for replacing the values. Note that imputing missing data with **mode** values can be done with numerical and categorical data.

```
df['Column_name']=df['Column_name'].fillna(df['Column_name''].mode()[0])
```

here station column missing values was present so replace station column missing value using mean

```
df["station"]=df["station"].fillna(df["station"].mean())
```

DataFrame-dropna() function:

he dropna() function is used to remove missing values.

Syntax:

```
DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)
```

Name	Description	Type/Default Value	Required / Optional
axis	Determine if rows or columns which contain missing values are removed. • 0, or 'index' : Drop rows which contain missing values. • 1, or 'columns' : Drop columns which contain missing value.	{0 or 'index', 1 or 'columns'} Default Value: 0	Required
how	Determine if row or column is removed from DataFrame, when we have at least one NA or all NA. • 'any' : If any NA values are present, drop that row or column. • 'all' : If all values are NA, drop that row or column.	{'any', 'all'} Default Value: 'any'	Required
thresh	Require that many non-NA values.	int	Optional
subset	Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include.	array-like	Optional
inplace	If True, do operation inplace and return None.	bool Default Value: False	Required

Drop the null values all the columns in the dataset:

```
df.dropna(subset=df.columns, thresh=14,inplace=True)
```

Using this drop statement use threshold using this drop that all the rows in which more than 14 null values are present in dataset drop that all the rows in all the columns.

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

```
station                0
Present_Tmax           0
Present_Tmin           0
LDAPS_RHmin            0
LDAPS_RHmax            0
LDAPS_Tmax_lapse       0
LDAPS_Tmin_lapse       0
LDAPS_WS               0
LDAPS_LH               0
LDAPS_CC1              0
LDAPS_CC2              0
LDAPS_CC3              0
LDAPS_CC4              0
LDAPS_PPT1             0
LDAPS_PPT2             0
LDAPS_PPT3             0
LDAPS_PPT4             0
lat                    0
lon                    0
DEM                    0
Slope                  0
Solar radiation        0
Next_Tmax              0
Next_Tmin              0
dtype: int64
```

After delete the null values the describe dataset and check null values here we see there are no null values are present in columns.

zScore:

Simply put, a z-score (also called a standard score) gives **you an idea of how far from the mean a data point is**. But more technically it's a measure of how many standard deviations below or above the population mean a raw score is. A z-score can be placed on a normal distribution curve.

Take your data point, subtract the mean from the **data point, and then divide by your standard deviation**. That gives you your Z-score. You can use Z-Score to determine outliers.One of the most commonly used tools in determining outliers is the Z-score. Z-score is just the number of standard deviations away from the mean that a certain data point is.

In your future data science life, Z-scores are gonna be a really useful way to think about how usual or how unusual a certain data point is. And that’s going to be really valuable once we start making inferences based on our data. In this story, we will take a deep dive into our notebooks and learn how to detect outliers using Z-Score.

```
from scipy.stats import zscore
z=np.abs(zscore(df))
new_df=df[(z<3).all(axis=1)]
```

```
new_df.shape
```

```
(6748, 24)
```

```
df.shape
```

```
(7590, 24)
```

```
Data_loss=((7590-6748)/7590)*100
```

```
Data_loss
```

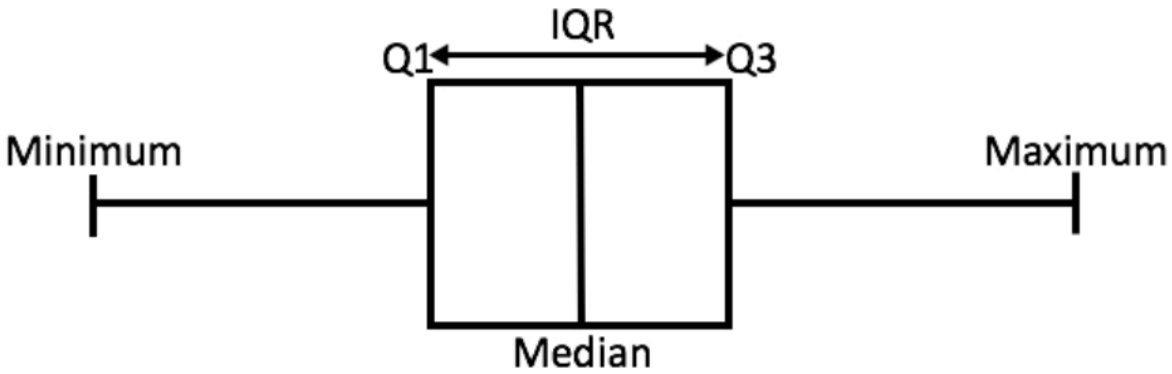
```
11.093544137022397
```

After using zscore there are 11.09 data loss.

Use IQR Check the dataloss:

IQR Method of Outlier Detection

To explain IQR Method easily, let's start with a box plot.



A box plot tells us, more or less, about the distribution of the data. It gives a sense of how much the data is actually spread about, what's its range, and about its skewness. As you might have noticed in the figure, that a box plot enables us to draw inference from it for an ordered data, i.e., it tells us about the various metrics of a data arranged in ascending order.

In the above figure,

- minimum is the minimum value in the dataset,
- and maximum is the maximum value in the dataset.

So the difference between the two tells us about the range of dataset.

- The median is the median (or centre point), also called second quartile, of the data (resulting from the fact that the data is ordered).
- *Q1* is the first quartile of the data, i.e., to say 25% of the data lies between minimum and *Q1*.
- *Q3* is the third quartile of the data, i.e., to say 75% of the data lies between minimum and *Q3*.

The difference between *Q3* and *Q1* is called the **Inter-Quartile Range** or **IQR**.

```
IQR = Q3 - Q1
```

To detect the outliers using this method, we define a new range, let's call it decision range, and any data point lying outside this range is considered as outlier and is accordingly dealt with. The interquartile range shows how the data is spread about the median. It is less susceptible than the range to outliers and can, therefore, be more helpful.

Using the Interquartile Rule to Find Outliers

Though it's not often affected much by them, the interquartile range can be used to detect outliers. This is done using these steps:

1. Calculate the interquartile range for the data.
2. Multiply the interquartile range (IQR) by 1.5 (a constant used to discern outliers).
3. Add 1.5 x (IQR) to the third quartile. Any number greater than this is a suspected outlier.
4. Subtract 1.5 x (IQR) from the first quartile. Any number less than this is a suspected outlier.

Remember that the interquartile rule is only a rule of thumb that generally holds but does not apply to every case. In general, you should always follow up your outlier analysis by studying the resulting outliers to see if they make sense. Any potential outlier obtained by the interquartile method should be examined in the context of the entire set of data.

```
Q1=df.quantile(0.25)
Q3=df.quantile(0.75)
IQR=Q3-Q1
df_new=df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
```

```
df_new.shape
```

```
(3165, 24)
```

Using IQR more data loss.

Checking Skewness:

The skewness is a measure of symmetry or asymmetry of data distribution, and kurtosis measures whether data is heavy-tailed or light-tailed in a normal distribution. Data can be positive-skewed (data-pushed towards the right side) or negative-skewed (data-pushed towards the left side).

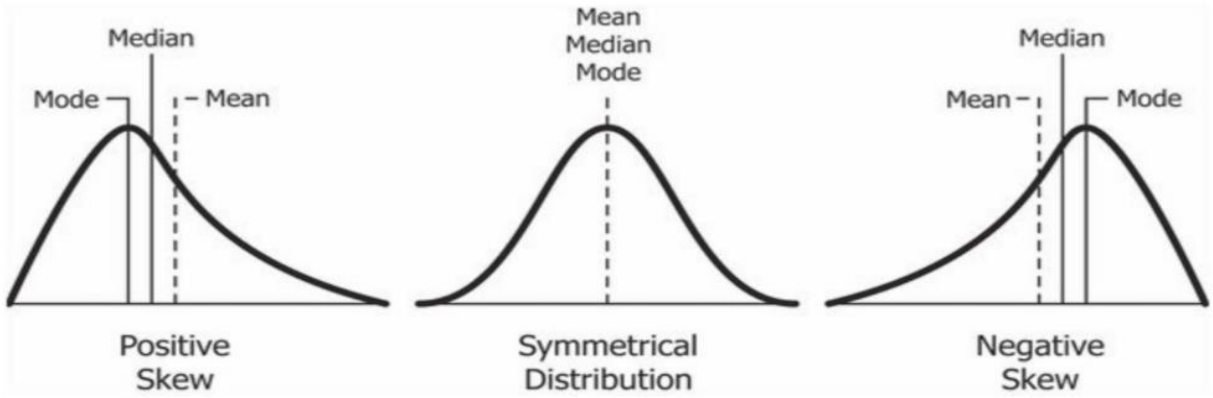
Skewness is the measure of the asymmetry of an ideally symmetric probability distribution and is given by the third standardized moment. If that sounds way too complex, don't worry! Let me break it down for you.

In simple words, skewness is the measure of how much the probability distribution of a random variable deviates from the normal distribution. Now, you might be thinking – why am I talking about normal distribution here?

Well, the normal distribution is the probability distribution without any skewness. You can look at the image below which shows symmetrical distribution that's basically a normal distribution and you can see that it is symmetrical on both sides of the dashed line. Apart from this, there are two types of skewness:

- Positive Skewness
- Negative Skewness

- Negative Skewness



The probability distribution with its tail on the right side is a positively skewed distribution and the one with its tail on the left side is a negatively skewed distribution. If you're finding the above figures confusing, that's alright. We'll understand this in more detail later. Before that, let's understand why skewness is such an important concept for you as a data science prof essional.

```
df.skew()
```

```
station             -0.004178
Present_Tmax        -0.258173
Present_Tmin        -0.363848
LDAPS_RHmin          0.301392
LDAPS_RHmax         -0.849594
LDAPS_Tmax_lapse    -0.226062
LDAPS_Tmin_lapse    -0.576782
LDAPS_WS             1.576841
LDAPS_LH             0.669181
LDAPS_CC1            0.456982
LDAPS_CC2            0.474312
LDAPS_CC3            0.640306
LDAPS_CC4            0.664487
LDAPS_PPT1           5.375597
LDAPS_PPT2           5.772761
LDAPS_PPT3           6.445912
LDAPS_PPT4           6.770184
lat                  0.085498
lon                 -0.288956
DEM                  1.720915
Slope                1.558715
Solar radiation     -0.524522
Next_Tmax           -0.339116
Next_Tmin           -0.401286
dtype: float64
```

Here we check the skewness using df.skew() using this skewness is present many columns.

How to remove skewness:

Since you know how much the skewed data can affect our machine learning model's predicting capabilities, it is better to transform the skewed data to normally distributed data. Here are some of the ways you can transform your skewed data:

- Power Transformation
- Log Transformation
- Exponential Transformation

Here we use Power Transformation to remove skewness:

Power Transformation :

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.

By default, zero-mean, unit-variance normalization is applied to the transformed data.

```
from sklearn.preprocessing import PowerTransformer
pt=PowerTransformer()
dfpt=pt.fit_transform(df)
df=pd.DataFrame(dfpt,columns=df.columns)
```

Here power transformer to remove skewness.

log transformation and index changing in python:

1. Apply log to each column variable.
2. Name this newly generated variable, "log_variable". ...
3. Do log(variable_value +1) for values in df[variables] columns that are zero or missing, to avoid getting "-inf" returned.
4. Find index of original variable

```
df['LDAPS_PPT1']=np.log(df['LDAPS_PPT1'])
```

```
df['LDAPS_PPT2']=np.log(df['LDAPS_PPT2'])
```

```
df['LDAPS_PPT3']=np.log(df['LDAPS_PPT3'])
```

```
df['LDAPS_PPT4']=np.log(df['LDAPS_PPT4'])
```

Here after use power transformer still skewness is precent in some columns so use log transform to remove skewness.

```
df.skew()
```

```
station            -0.003515
Present_Tmax        0.003937
Present_Tmin       -0.007417
LDAPS_RHmin        -0.024182
LDAPS_RHmax         0.012409
LDAPS_Tmax_lapse   -0.007778
LDAPS_Tmin_lapse    0.002819
LDAPS_WS           -0.033065
LDAPS_LH           -0.064278
LDAPS_CC1          -0.004069
LDAPS_CC2          -0.000919
LDAPS_CC3           0.002059
LDAPS_CC4          -0.000522
LDAPS_PPT1         -0.412757
LDAPS_PPT2         -0.379903
LDAPS_PPT3         -0.374866
LDAPS_PPT4         -0.595221
lat                -0.014324
lon                 0.002646
DEM                 0.007795
Slope              -0.017519
Solar radiation     0.001054
Next_Tmax           0.002008
Next_Tmin          -0.006514
dtype: float64
```

After use log transform there was no skewness is present in dataset.

MinMax Scaler Technique:

Here use MinMaxScaler Techique. The MinMax scaler is one of the simplest scalers to understand. It just scales all the data between 0 and 1.

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
```

```
xd=scaler.fit_transform(x)
x=pd.DataFrame(xd,columns=x.columns)
```

5.Building Machine Learning Models:

Now we will several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their dataset we need to use the predictions on the dataset to compare the algorithms with each other. Later on, we will use cross validation.

```
from sklearn.model_selection import train_test_split,cross_val_score
```

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

Loading different library which are necessary.

```
x=df.drop(["Next_Tmax"],axis=1)
y=df["Next_Tmax"]
```

Here we use Next_Tmax as a target variable and use different model.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=12)
```

The initial step involves splitting the training dataset into train and validation sets . The validation set is important for analyzing results on completely unseen data.

we mainly focus on the performance evaluation criteria for classifier comparison of accuracy.

Here we use linear regression model.

Linear Regression:

Regression is a modeling task that involves predicting a numeric value given an input.

Linear regression is the standard algorithm for regression that assumes a linear relationship between inputs and the target variable. An extension to linear regression invokes adding penalties to the loss function during training that encourages simpler models that have smaller coefficient values. These extensions are referred to as regularized linear regression or penalized linear regression.

Lasso Regression:

Regression is a modeling task that involves predicting a numeric value given an input.

Linear regression is the standard algorithm for regression that assumes a linear relationship between inputs and the target variable. An extension to linear regression invokes adding penalties to the loss function during training that encourages simpler models that have smaller coefficient values. These extensions are referred to as regularized linear regression or penalized linear regression.

Lasso Regression is a popular type of regularized linear regression that includes an L1 penalty. This has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task. This penalty allows some coefficient values to go to the value of zero, allowing input variables to be effectively removed from the model, providing a type of automatic feature selection.
In this tutorial, you will discover how to develop and evaluate Lasso Regression models in Python.

After completing this tutorial, you will know:

- Lasso Regression is an extension of linear regression that adds a regularization penalty to the loss function during training.
- How to evaluate a Lasso Regression model and use a final model to make predictions for new data.
- How to configure the Lasso Regression model for a new dataset via grid search and automatically.

```
lr.fit(x_train,y_train)
```

```
pred_test=lr.predict(x_test)
```

```
print(r2_score(y_test,pred_test))
```

```
0.7987325817906769
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
import warnings
warnings.filterwarnings('ignore')
```

```
parameters={'alpha':[.0001, .001, .01, .1, 1, 10],'random_state':list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'alpha': 0.0001, 'random_state': 0}
```

```
ls=Lasso(alpha=0.0001,random_state=0)
ls.fit(x_train,y_train)
ls.score(x_train,y_train)
pred_ls=ls.predict(x_test)
lss=r2_score(y_test,pred_ls)
lss
```

```
0.7986817604876617
```

```
cv_score=cross_val_score(ls,x,y,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

```
0.733771862109234
```

Here we use Lasso Regresson model and apply hyper parameter tunning on lasso because select best parameter list and check cross validaton score of lasso above lasso r2 score is 0.79 and cross validation score is 0.73.

Ridge Regression:

Regression is a modeling task that involves predicting a numeric value given an input.

Linear regression is the standard algorithm for regression that assumes a linear relationship between inputs and the target variable. An extension to linear regression invokes adding penalties to the loss function during training that encourages simpler models that have smaller coefficient values. These extensions are referred to as regularized linear regression or penalized linear regression.

Ridge Regression is a popular type of regularized linear regression that includes an L2 penalty. This has the effect of shrinking the coefficients for those input variables that do not contribute much to the prediction task.
In this tutorial, you will discover how to develop and evaluate Ridge Regression models in Python.

After completing this tutorial, you will know:

- Ridge Regression is an extension of linear regression that adds a regularization penalty to the loss function during training.
- How to evaluate a Ridge Regression model and use a final model to make predictions for new data.
- How to configure the Ridge Regression model for a new dataset via grid search and automatically.

```
parameters={'alpha':[1e-15,1e-10,1e-8,1e-5,1e-3,0.1,1,5,10,15,20,30,35,45,50,55,65,100,110,150,1000
ri=Ridge()
clf=GridSearchCV(ri,parameters)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'alpha': 0.1}
```

```
ri=Ridge(alpha=0.1)
ri.fit(x_train,y_train)
ri.score(x_train,y_train)
pred_ri=ri.predict(x_test)
rii=r2_score(y_test,pred_ri)
rii
```

```
0.7986730217932076
```

```
cv_score=cross_val_score(ri,x,y,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

```
0.7338072460043186
```

Here we use Ridge Regresson model and apply hyper parameter tunning on lasso because select best parameter list and check cross validaton score of Ridge above lasso r2 score is 0.79 and cross validation score is 0.73.

RandomForestRegressor:

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
pred_decision=rf.predict(x_test)

rfs=r2_score(y_test,pred_decision)
print('R2 Score:',rfs*100)
```

```
R2 Score: 90.73966739550853
```

```
rfscore=cross_val_score(rf,x,y,cv=5)
rfc=rfscore.mean()
print('cross cal score:',rfs*100)
```

```
cross cal score: 90.73966739550853
```

Here we use RandomForestRegressor model check cross validaton score of RandomForestRegressor above r2 score is 90 and cross validation score is 90.

SVR:

Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems. Let's spend a few minutes understanding the idea behind SVR.
Support Vector regression is a type of Support vector machine that supports linear and non-linear regression.
Support Vector Regression (SVR) is a regression algorithm and it applies a similar technique of Support Vector Machines (SVM) for regression analysis. As we know, regression data contains continuous real numbers. To fit such type of data, the SVR model approximates the best values with a given margin called ε-tube (epsilon-tube, ε identifies a tube width) with considering the model complexity and error rate.

```
from sklearn.svm import SVR
sv=SVR()
sv.fit(x_train,y_train)
sv.score(x_train,y_train)
pred_sv=sv.predict(x_test)
svv=r2_score(y_test,pred_sv)
svv
```

```
0.8644238689409874
```

```
svscore=cross_val_score(rf,x,y,cv=5)
sv=svscore.mean()
print('cross cal score:',sv*100)
```

```
cross cal score: 71.05765009515709
```

Here we use SVR Regresson model check cross validaton score of SVR above r2 score is 0.86 and cross validation score is 0.71.

DecisionTree Regression:

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all of their possible results, including outcomes, input costs and utility.

DecisionTreeRegression:
Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.
Discrete output example: A weather prediction model that predicts whether or not there'll be rain in a particular day.
Continuous output example: A profit prediction model that states the probable profit that can be generated from the sale of a product.

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x_train,y_train)
dt.score(x_train,y_train)
pred_decision=dt.predict(x_test)

dts=r2_score(y_test,pred_decision)
print('R2 Score:',dts*100)
```

```
R2 Score: 79.9607080790306
```

```
dtscore=cross_val_score(dt,x,y,cv=5)
dt=rfscore.mean()
print('cross cal score:',dts*100)
```

```
cross cal score: 79.9607080790306
```

Here we use DecisionTreeRegression model check cross validaton score of DecisionTreeRegression above r2 score is 79 and cross validation score is 79.

After checking r2 Score and cross validation score of all the models the RandomForestRegression is the best model.

After using different model RandomForestRegression is the best model. After selecting best model use Hyperparameter tunning on best model . hyperparameter tuning of the models using Grid Search Cross-validation for optimized parameters needed for Regression.

GridSearchCv:

GridSearchCV is **a library function that** is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters.

Parameterlist:

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as **Hyperparameters.** These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. Of course, you must select from a specific list of hyperparameters for a given model as it varies from model to model.

```
from sklearn.model_selection import GridSearchCV
```

ort GridsearcgCv

```
import numpy as np
```

```
from sklearn.ensemble import RandomForestRegressor
parameters = {'criterion' : ['mse','mae'],
              'max_features': ["auto","log2"]}
```

Here above are the parameter list of RandomForestRegression model .
The above code block we have the following parameters:. max_features: In this maximum features there are two values auto and log2.
Criterion: In criterion there are two parameter values are present mse and mae.

First, we initialize `RandomizedSearchCV` which will run iterations with 5-fold cross-validation. Then we fit it on the training dataset

```
GCV=GridSearchCV(RandomForestRegressor(),parameters,cv=5)
```

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['mse', 'mae'],
                         'max_features': ['auto', 'log2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
GCV.best_params_
```

```
{'criterion': 'mse', 'max_features': 'log2'}
```

Here after use different parameter list the best parameter list is select. Above are the best parameter list RandomForestRegression.

```
Final_mod2 = RandomForestRegressor(criterion= 'mse',max_features='log2')
Final_mod2.fit(x_train,y_train)
pred = Final_mod2.predict(x_test)
rfs=r2_score(y_test,pred_decision)
print(rfs*100)
```

```
cross cal score: 90.73966739550853
```

Put this parameters into the model so output is finally best score os RandomForestRegression.

```
import joblib
joblib.dump(rfs,"Tempmax.pkl")
```

```
['Tempmax.pkl']
```

orting the model

```
mod=joblib.load("Tempmax.pkl")
```

```
print(Final_mod2.predict(x_test))
```

```
[32.813 32.864 35.036 ... 32.714 28.675 32.107]
```

Finally load the model and predict the values.

After that use 2nd target variable and apply different modeling technique on it.

Now use Next_Tmin as target variable:

```
x1_train, x1_test, y1_train, y1_test = train_test_split(x1,y1,test_size=0.2,random_state=12)
```

different models to select best r2 score and select best model.

```
lr.fit(x1_train,y1_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
pred_test=lr.predict(x1_test)
```

```
print(r2_score(y1_test,pred_test))
```

```
0.8481951310936117
```

Here we use train_test_split and use linear regression use Next_Tmin as target variable.

```
parameters={'alpha':[.0001, .001, .01, .1, 1, 10],'random_state':list(range(0,10))}
ls=Lasso()
clf=GridSearchCV(ls,parameters)
clf.fit(x1_train,y1_train)
print(clf.best_params_)
```

```
{'alpha': 0.0001, 'random_state': 0}
```

```
ls=Lasso(alpha=0.0001,random_state=0)
ls.fit(x1_train,y1_train)
ls.score(x1_train,y1_train)
pred_ls=ls.predict(x1_test)
lss=r2_score(y1_test,pred_ls)
lss
```

```
0.8482005949773606
```

```
cv_score=cross_val_score(ls,x1,y1,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

```
0.8012865251091783
```

Here we use Lasso Regression model and apply hyper parameter tunning on lasso because select best parameter list and check cross validaton score of lasso above lasso r2 score is 0.84 and cross validation score is 0.80.

```
parameters={'alpha':[1e-15,1e-10,1e-8,1e-5,1e-3,0.1,1,5,10,15,20,30,35,45,50,55,65,100,110,150,1000]}
ri=Ridge()
clf=GridSearchCV(ri,parameters)
clf.fit(x1_train,y1_train)
print(clf.best_params_)
```

```
{'alpha': 0.1}
```

```
ri=Ridge(alpha=0.1)
ri.fit(x1_train,y1_train)
ri.score(x1_train,y1_train)
pred_ri=ri.predict(x1_test)
rii=r2_score(y1_test,pred_ri)
rii
```

```
0.8482295137473177
```

```
cv_score=cross_val_score(ri,x1,y1,cv=5)
cv_mean=cv_score.mean()
cv_mean
```

Here we use Lasso Regresson model and apply hyper parameter tunning on lasso because select best parameter list and check cross validaton score of lasso above lasso r2 score is 0.84 and cross validation score is 0.80.

```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(x1_train,y1_train)
rf.score(x1_train,y1_train)
pred_decision=rf.predict(x1_test)

rfs=r2_score(y1_test,pred_decision)
print('R2 Score:',rfs*100)
```

```
R2 Score: 91.14704917529416
```

```
rfscore=cross_val_score(rf,x1,y1,cv=5)
rfc=rfscore.mean()
print('cross cal score:',rfs*100)
```

```
cross cal score: 91.18314052359018
```

Here we use RandomForestRegressor model check cross validaton score of RandomForestRegressor above r2 score is 91 and cross validation score is 91.

```
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(x1_train,y1_train)
dt.score(x1_train,y1_train)
pred_decision=dt.predict(x1_test)

dts=r2_score(y1_test,pred_decision)
print('R2 Score:',dts*100)
```

```
R2 Score: 78.42704720743762
```

```
dtscore=cross_val_score(dt,x1,y1,cv=5)
dt=rfscore.mean()
print('cross cal score:',dts*100)
```

```
cross cal score: 78.42704720743762
```

Here we use DecisionTreeRegression model check cross validaton score of DecisionTreeRegression above r2 score is 78.42 and cross validation score is 78.42.

Hyperparameter tuning:

Hyperparameter tuning is **choosing a set of optimal hyperparameters for** a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning.
In this project RandomForestRegressor is our best model so Appy hyperparameter tuning on it.

Why do We Need Hyperparameter Tuning?

In random forests, there are a number of hyperparameters available. Although we can get good results without any changes to these

parameters, there are some parameters which have great impact on the output of our classifier or regressor. But we do not want to manually search and test for the optimal values of these hyperparameters. Such a trial and error method may take a lot of time.

Therefore, we have methods
like **RandomizedSearchCV** and **GridSearchCV** which help us to fine-tune the hyperparameters by providing us with the best values. On top of that, we can implement these using Scikit-Learn as well.

```
from sklearn.model_selection import GridSearchCV
```

ort GridsearcgCv

```
import numpy as np
```

```
from sklearn.ensemble import RandomForestRegressor
parameters = {'criterion' : ['mse','mae'],
              'max_features': ["auto","log2"]}
```

ort parameters

ılize:

```
GCV=GridSearchCV(RandomForestRegressor(),parameters,cv=5)
```

Parameter list of RandomForestRegressor.

The above code block we have the following parameters:. max_features: In this maximum features there are two values auto and log2.
Criterion: In criterion there are two parameter values are present mse and mae.

First, we initialize `RandomizedSearchCV` which will run iterations with 5-fold cross-validation. Then we fit it on the training dataset.

```
GCV.fit(x1_train,y1_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['mse', 'mae'],
                         'max_features': ['auto', 'log2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
GCV.best_params_
```

```
{'criterion': 'mse', 'max_features': 'log2'}
```

Best parameterlist of RandomForestRegressor.

```
Final_mod2 = RandomForestRegressor(criterion= 'mse',max_features='log2
Final_mod2.fit(x1_train,y1_train)
pred = Final_mod2.predict(x1_test)
rfs=r2_score(y1_test,pred_decision)
print(rfs*100)
```

```
cross cal score: 91.14704917529416
```

ally print best score

orting the model

```
import joblib
joblib.dump(rfs,"Tempmin.pkl")
```

```
['Tempmin.pkl']
```

ding model

```
mod=joblib.load("Tempmin.pkl")
```

```
print(Final_mod2.predict(x1_test))
```

```
[27.405 24.602 26.591 ... 23.145 20.569 25.016]
```

Load the best parameter and predict the parameter list.

6.Conclusion:

By using RandomForest, we can reach best R2 score on both targets like **Next_Tmax and Next_Tmin** up to 90.70 and 91.14 which really good score so model is good performance and is truely decent.