Problem Statement:

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.

Import Required Libarary

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import datetime
from scipy.stats import zscore
pd.set_option('display.max_columns', None) # For display maximum columns
from sklearn.preprocessing import StandardScaler,OrdinalEncoder, MinMaxScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor
from sklearn.linear_model import LinearRegression
import xgboost as xgb
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Reading Data

In [2]:
```
df = pd.read_csv(r"C:\Users\Kushal Arya\Desktop\csv file\temperature.csv")
df.head()
```

Out[2]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	30-06-2013	28.7	21.4	58.255688	91.116364	28.074101
1	2.0	30-06-2013	31.9	21.6	52.263397	90.604721	29.850689
2	3.0	30-06-2013	31.6	23.3	48.690479	83.973587	30.091292
3	4.0	30-06-2013	32.0	23.4	58.239788	96.483688	29.704629
4	5.0	30-06-2013	31.4	21.9	56.174095	90.155128	29.113934

Columns Information:

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

Check no of row and column

In [3]:
```
print('No of Rows and Columns ----->', df.shape )
```

```
No of Rows and Columns -----> (7752, 25)
```

Checking for Null values

```
In [4]: print('================================\n')
        print(df.isnull().sum())
        print('\n================================')
```

```
================================

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

================================
```

There is null value

Information about dataset

```
In [5]: print('==================================================\n')
        print(df.info())
        print('====================================================')
```

```
==================================================

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7752 entries, 0 to 7751
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 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
dtypes: float64(24), object(1)
memory usage: 1.5+ MB
None
====================================================
```

Fill NaN

```
In [6]: df = df.apply(lambda x:x.fillna(x.mean())if x.dtype == 'float64' else x.fillna(x.
```

In [7]:
```
print('================================\n')
print(df.isnull().sum())
print('\n================================')
```

```
================================

station             0
Date                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

================================
```

There is no null value left

Statistics of Data

In [8]:
```
df.describe()
```

Out[8]:

	station	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_la
count	7752.000000	7752.000000	7752.000000	7752.000000	7752.000000	7752.000
mean	13.000000	29.768211	23.225059	56.759372	88.374804	29.613
std	7.210637	2.956557	2.403036	14.596973	7.157124	2.932
min	1.000000	20.000000	11.300000	19.794666	58.936283	17.624
25%	7.000000	27.800000	21.700000	46.046162	84.316923	27.693
50%	13.000000	29.900000	23.400000	55.313244	89.699505	29.662
75%	19.000000	32.000000	24.900000	67.038254	93.704500	31.683
max	25.000000	37.600000	29.900000	98.524734	100.000153	38.542

Outliers are present in our data set

Features Engineering

Date column

In [9]:
```
df['Date'].value_counts()
```

Out[9]:
```
02-08-2016    27
05-07-2016    25
20-08-2014    25
31-07-2017    25
08-07-2014    25
              ..
24-07-2015    25
05-08-2015    25
08-08-2014    25
17-07-2013    25
03-08-2014    25
Name: Date, Length: 310, dtype: int64
```

In [10]:
```
df['Date'] = pd.to_datetime(df['Date'])
```

Add Year Column

In [11]:
```python
df['Years'] = df['Date'].dt.year
df['Years'] = df['Years'].astype('int')
df.head(2)
```

Out[11]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	28.07410
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	29.85068

In [12]:
```python
df['Years'].value_counts()
```

Out[12]:
```
2016    1552
2013    1550
2017    1550
2014    1550
2015    1550
Name: Years, dtype: int64
```

In [13]:
```python
df['Months'] = df['Date'].dt.month
df['Months'] = df['Months'].astype('int')
df.head(2)
```

Out[13]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	28.07410
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	29.85068

Join Years and Months

In [14]:
```python
cols=["Years","Months"]
df['Years&Months'] = df[cols].apply(lambda x: '-'.join(x.values.astype(str)), axi
df.head(2)
```

Out[14]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	28.07410
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	29.85068

Convert Date column into object

In [15]:
```
df['Date'] = df['Date'].astype('str')
```

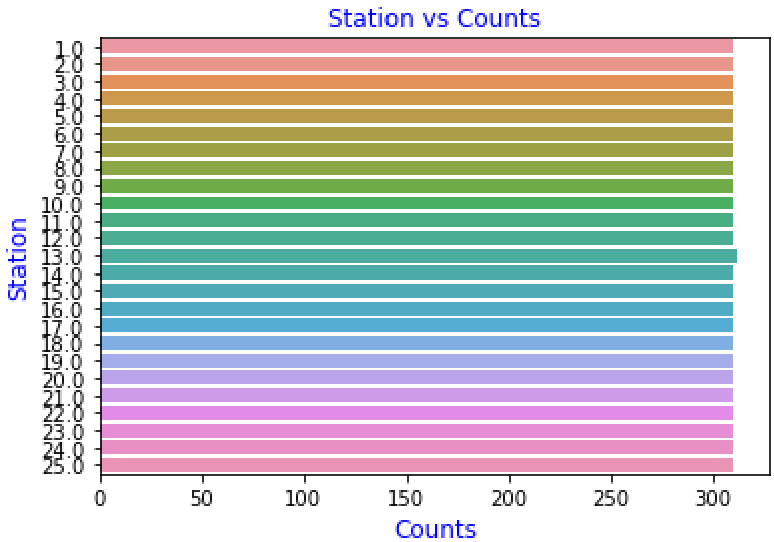
Analysis of data respect to next-day maximum temperatures

Station column

In [16]:
```
df['station'].value_counts()
```

Out[16]:
```
13.0    312
15.0    310
21.0    310
7.0     310
23.0    310
8.0     310
3.0     310
10.0    310
16.0    310
11.0    310
2.0     310
12.0    310
9.0     310
24.0    310
4.0     310
25.0    310
22.0    310
17.0    310
19.0    310
20.0    310
1.0     310
5.0     310
18.0    310
6.0     310
14.0    310
Name: station, dtype: int64
```

In [17]:
```
sns.countplot( y="station", data=df)
plt.ylabel('Station', c = 'b', fontsize = 12)
plt.xlabel('Counts', c = 'b', fontsize = 12)
plt.title('Station vs Counts', c = 'b', fontsize = 12)
plt.show()
```



Above plot shows station no and there counts

Date column

In [18]:
```
df['Years&Months'].value_counts()
```

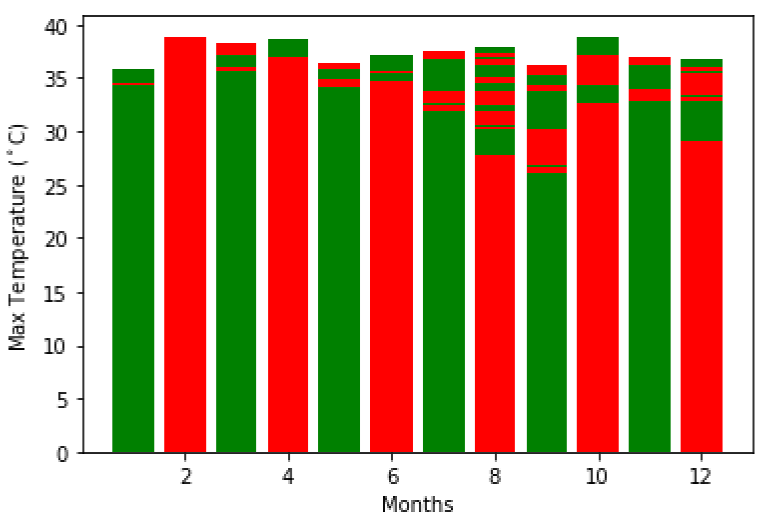
Out[18]:
```
2014-7     525
2013-7     525
2017-7     525
2015-7     525
2016-7     525
2014-8     500
2017-8     500
2015-8     500
2013-8     500
2016-8     500
2015-6      75
2016-6      75
2013-6      75
2014-6      75
2017-6      75
2016-2      52
2014-3      50
2014-12     50
2013-4      50
2013-11     50
2013-9      50
2016-11     50
2015-9      50
2017-10     50
2017-12     50
2013-10     50
2017-9      50
2014-2      50
2013-2      50
2015-12     50
2016-1      50
2013-3      50
2016-9      50
2016-3      50
2015-4      50
2015-2      50
2017-11     50
2014-4      50
2013-5      50
2017-4      50
2015-10     50
2017-1      50
2014-9      50
2014-5      50
2014-10     50
2013-1      50
2014-1      50
2016-12     50
2016-10     50
2014-11     50
2015-1      50
2017-5      50
2017-3      50
2015-3      50
```

```
2017-2      50
2016-4      50
2013-12     50
2015-11     50
2016-5      50
2015-5      50
Name: Years&Months, dtype: int64
```

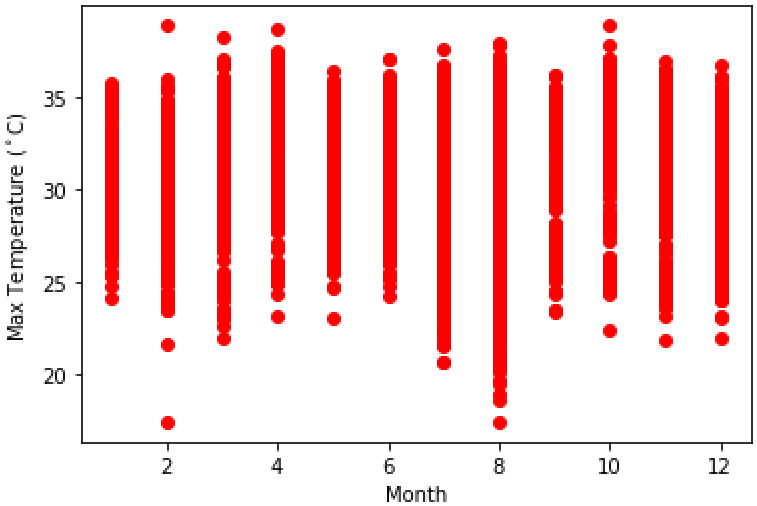
In [19]:
```python
y = df.groupby('Years&Months')['Next_Tmax'].value_counts()
y
```

Out[19]:
```
Years&Months  Next_Tmax
2013-1        28.200000    4
              27.100000    3
              29.700000    3
              26.800000    2
              27.300000    2
                          ..
2017-9        24.500000    1
              24.600000    1
              26.900000    1
              27.100000    1
              30.274887    1
Name: Next_Tmax, Length: 2758, dtype: int64
```

In [20]:
```python
plt.bar(df['Months'],df['Next_Tmax'], color= ('g','r'))
plt.xlabel('Months')
plt.ylabel('Max Temperature ($^\circ$C)')
plt.show()
```



In [21]:
```
plt.figure()
plt.plot(df['Months'], df['Next_Tmax'], 'ro')
plt.xlabel('Month')
plt.ylabel('Max Temperature ($^\circ$C)')
plt.show()
```



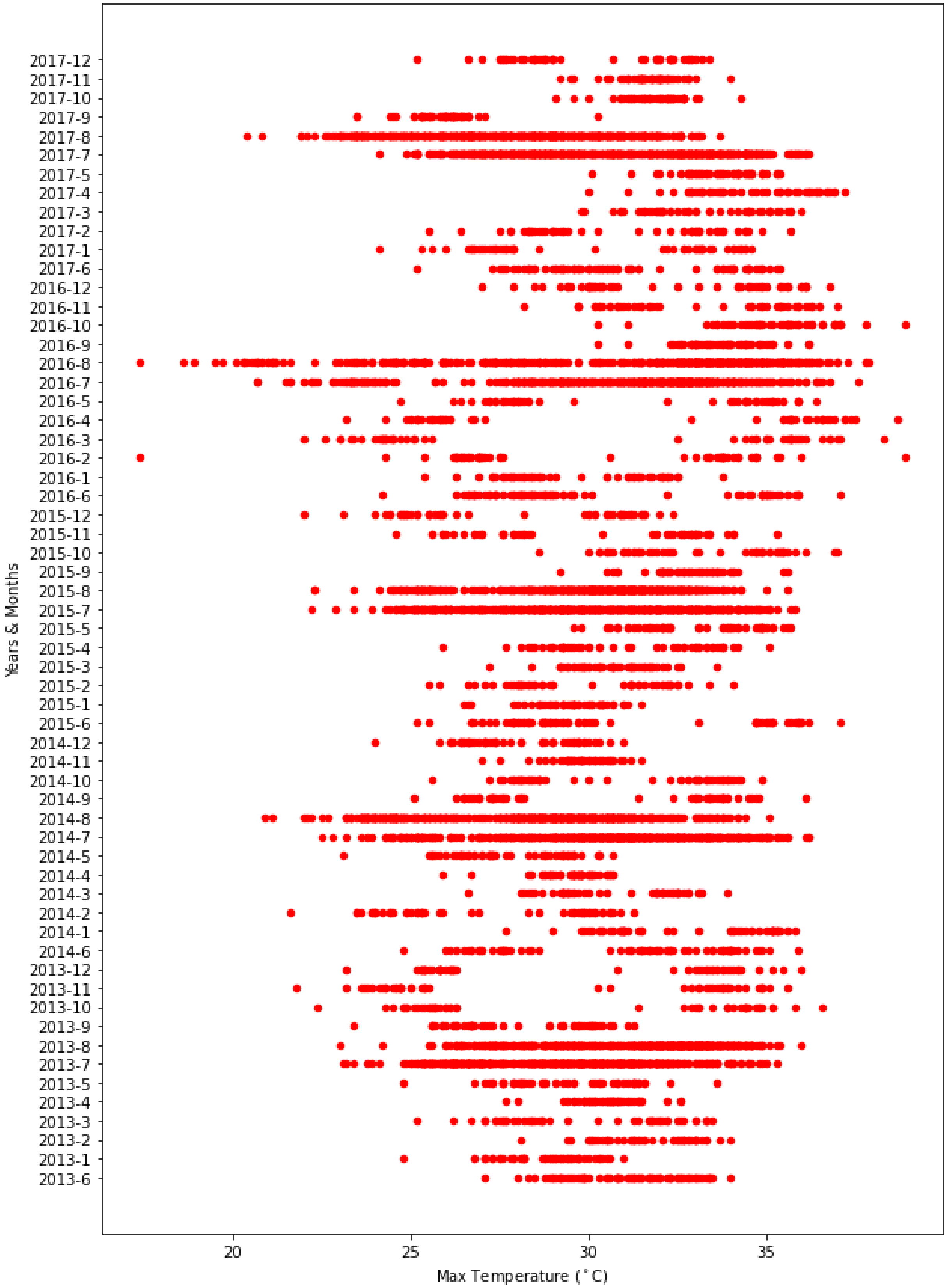
Above both plot shows 8th Month higest disparity in Max Temperature

In [22]:
```
plt.bar( df['Years'],df['Next_Tmax'], color= ('r','g'))
plt.xlabel('Years')
plt.ylabel('Max Temperature ($^\circ$C)')
plt.show()
```

In [23]:
```
plt.figure()
plt.plot(df['Years'], df['Next_Tmax'], 'ro')
plt.xlabel('Years')
plt.ylabel('Max Temperature ($^\circ$C)')
plt.show()
```

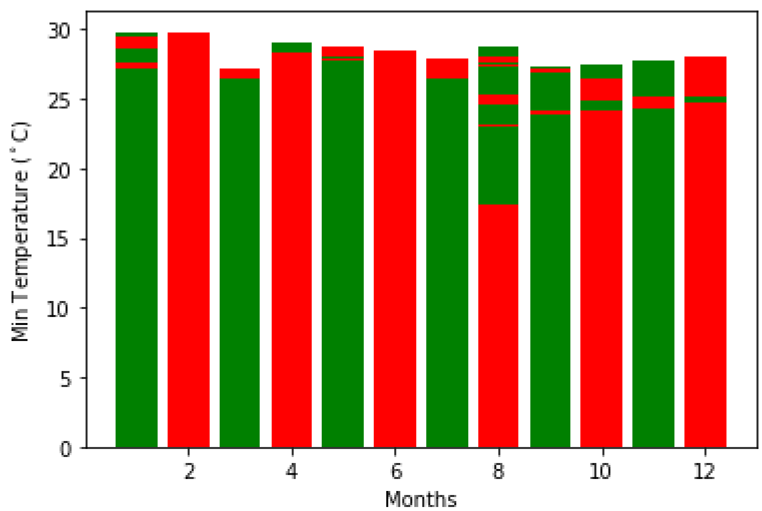
Above both plot shows 2016 Year higest disparity in Max Temperature

In [24]:
```
df.plot.scatter(x = 'Next_Tmax', y = 'Years&Months', figsize=(10,15), c = 'r')
plt.ylabel('Years & Months')
plt.xlabel('Max Temperature ($^\circ$C)')
plt.show()
```

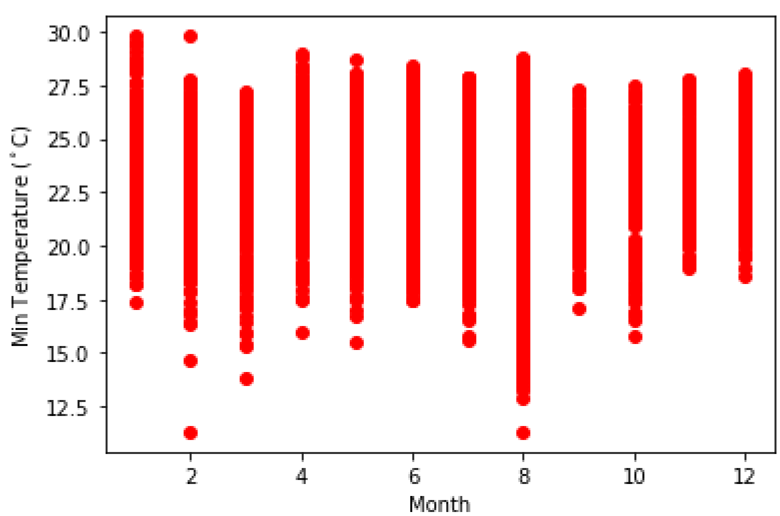


Above plot shows 2016 year 8th month higest disparity in Max Temperature

In [25]:
```
plt.bar(df['Months'],df['Next_Tmin'], color= ('g','r'))
plt.xlabel('Months')
plt.ylabel('Min Temperature ($^\circ$C)')
plt.show()
```



In [26]:
```
plt.figure()
plt.plot(df['Months'], df['Next_Tmin'], 'ro')
plt.xlabel('Month')
plt.ylabel('Min Temperature ($^\circ$C)')
plt.show()
```



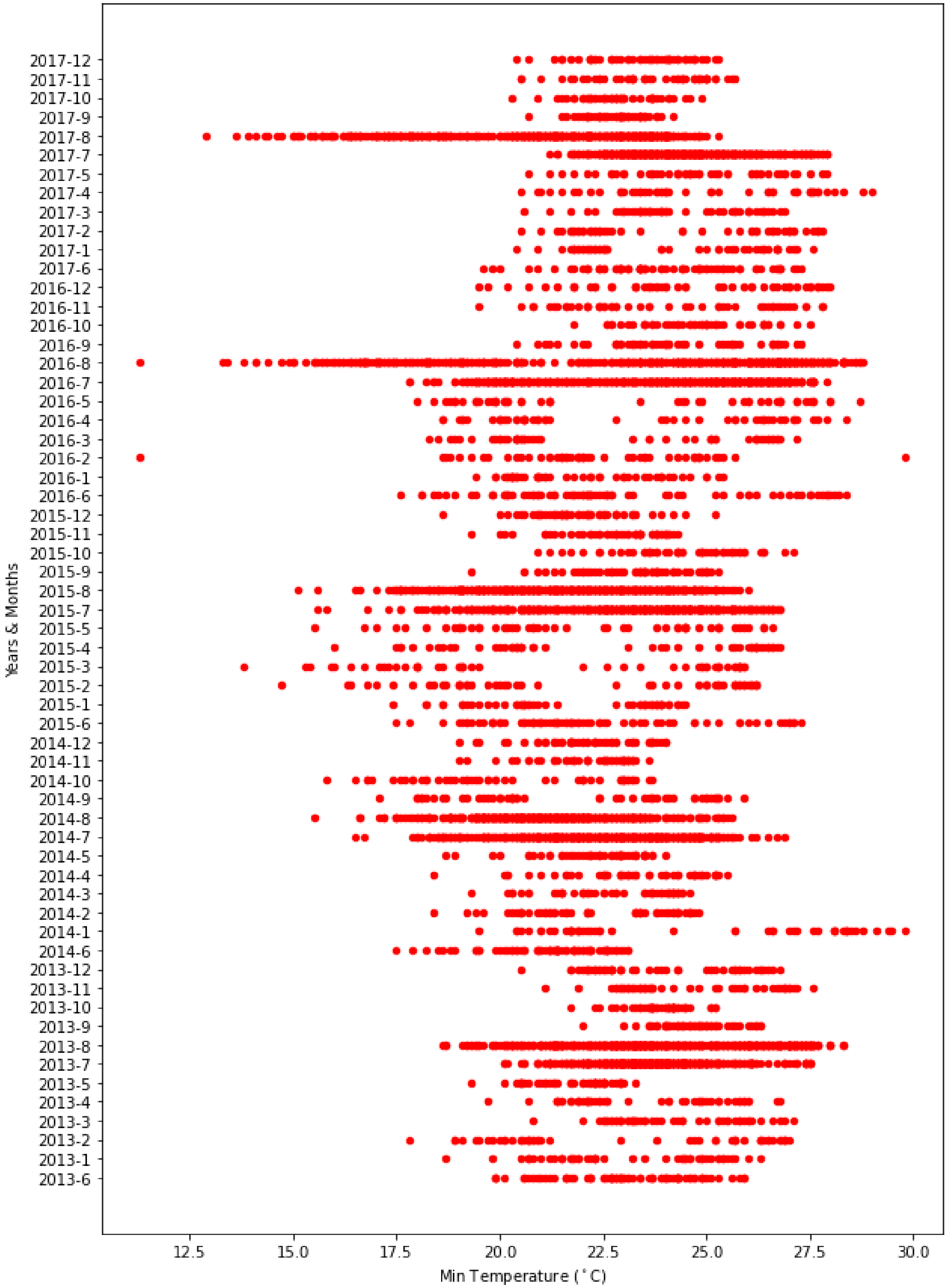
Above both plot shows 3rd Month higest disparity in Min Temperature

In [27]:
```
plt.bar( df['Years'],df['Next_Tmin'], color= ('r','g'))
plt.xlabel('Years')
plt.ylabel('Min Temperature ($^\circ$C)')
plt.show()
```

In [28]:
```
plt.figure()
plt.plot(df['Years'], df['Next_Tmin'], 'ro')
plt.xlabel('Years')
plt.ylabel('Min Temperature ($^\circ$C)')
plt.show()
```

Above both plot shows 2014 Year higest disparity in Min Temperature

In [29]:
```
df.plot.scatter(x = 'Next_Tmin', y = 'Years&Months', figsize=(10,15), c = 'r')
plt.ylabel('Years & Months')
plt.xlabel('Min Temperature ($^\circ$C)')
plt.show()
```



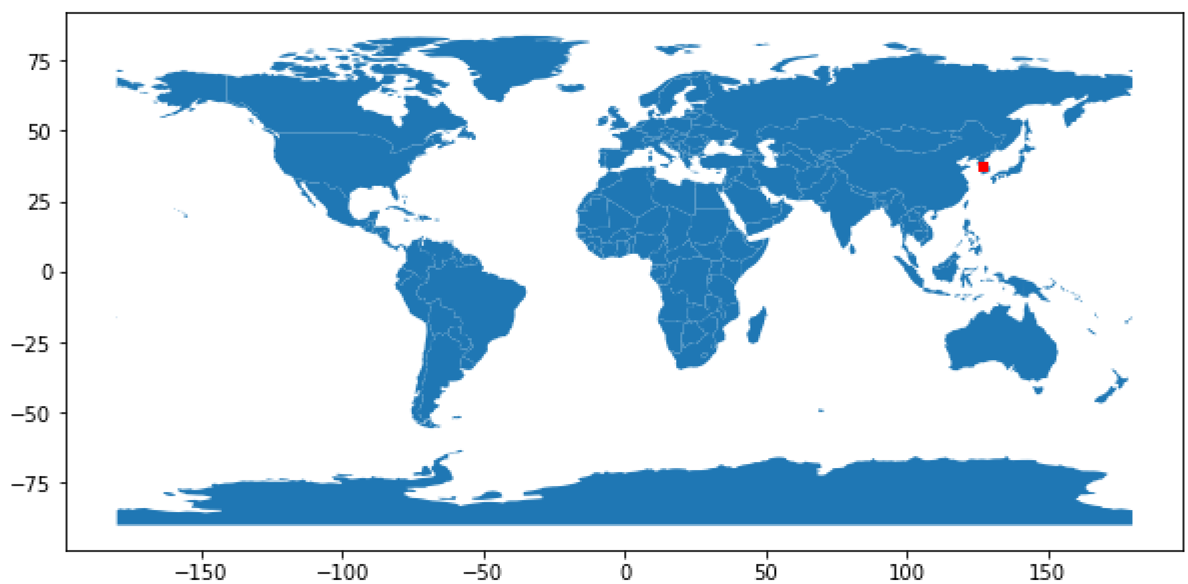
Above plot shows 2014 year 3re month higest disparity in Min Temperature

Co-ordinates

In [30]:
```
import geopandas as gpd
from shapely.geometry import Point, Polygon
import descartes
from geopandas import GeoDataFrame
from pyproj import CRS
```

In [31]:
```
geometry = [Point(xy) for xy in zip(df['lon'], df['lat'])]
gdf = GeoDataFrame(df, geometry=geometry)

#this is a simple map that goes with geopandas
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
gdf.plot(ax=world.plot(figsize=(10, 6)), marker='o', color='red', markersize=15);
```



Above plot shows all cordinate belong to Korea

Drop Unwanted column

In [32]:
```
df.drop('geometry', axis = 1, inplace = True)
df.head(2)
```

Out[32]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	28.07410
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	29.85068

In [33]:
```
df.shape # Here we check shape of remaining data after removal of column.
```

Out[33]: (7752, 28)

Encoding Categorical Column

In [34]:
```
oe = OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes == 'object':
        df[i] = oe.fit_transform(df[i].values.reshape(-1,1))
df.head(2)
```

Out[34]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	12.0	28.7	21.4	58.255688	91.116364	28.074101
1	2.0	12.0	31.9	21.6	52.263397	90.604721	29.850689

In [35]:

```
print('==================================================\n')
print(df.info())
print('====================================================')
```

```
==================================================

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7752 entries, 0 to 7751
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   station           7752 non-null   float64
 1   Date              7752 non-null   float64
 2   Present_Tmax      7752 non-null   float64
 3   Present_Tmin      7752 non-null   float64
 4   LDAPS_RHmin       7752 non-null   float64
 5   LDAPS_RHmax       7752 non-null   float64
 6   LDAPS_Tmax_lapse  7752 non-null   float64
 7   LDAPS_Tmin_lapse  7752 non-null   float64
 8   LDAPS_WS          7752 non-null   float64
 9   LDAPS_LH          7752 non-null   float64
 10  LDAPS_CC1         7752 non-null   float64
 11  LDAPS_CC2         7752 non-null   float64
 12  LDAPS_CC3         7752 non-null   float64
 13  LDAPS_CC4         7752 non-null   float64
 14  LDAPS_PPT1        7752 non-null   float64
 15  LDAPS_PPT2        7752 non-null   float64
 16  LDAPS_PPT3        7752 non-null   float64
 17  LDAPS_PPT4        7752 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         7752 non-null   float64
 24  Next_Tmin         7752 non-null   float64
 25  Years             7752 non-null   int32  
 26  Months            7752 non-null   int32  
 27  Years&Months      7752 non-null   float64
dtypes: float64(26), int32(2)
memory usage: 1.6 MB
None
====================================================
```

Data distribution

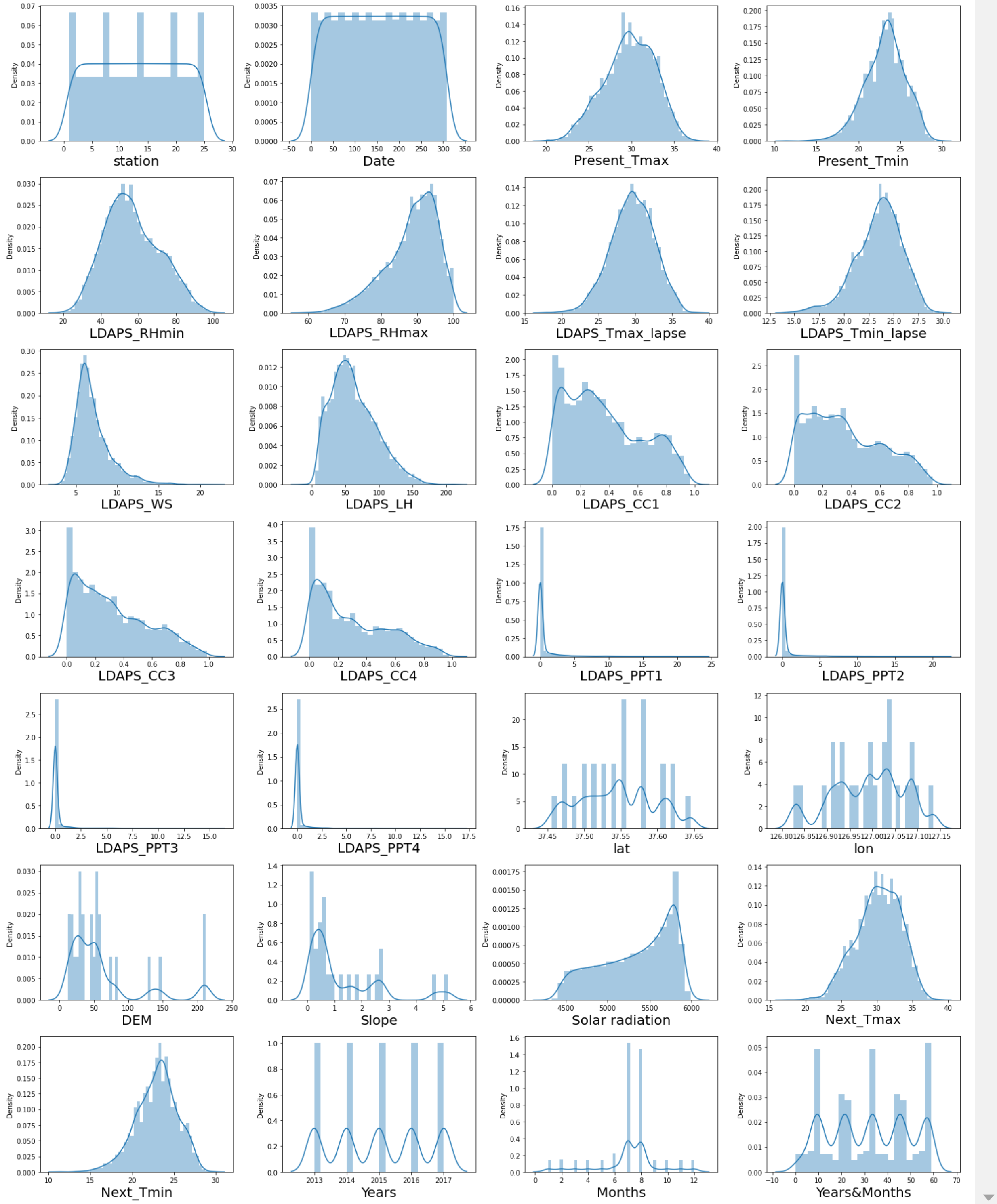
In [36]:

```
print('---------------------')
print('Distribution Plot :- ')
print('---------------------')

plt.figure(figsize = (20,25))
plotnumber = 1

for column in df:
    if plotnumber <=28:
        ax = plt.subplot(7,4, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column, fontsize = 20)
    plotnumber +=1
plt.tight_layout()
```

```
---------------------
Distribution Plot :-
---------------------
```



Data has skewed

Check skweness

In [37]:
```
df.skew()
```

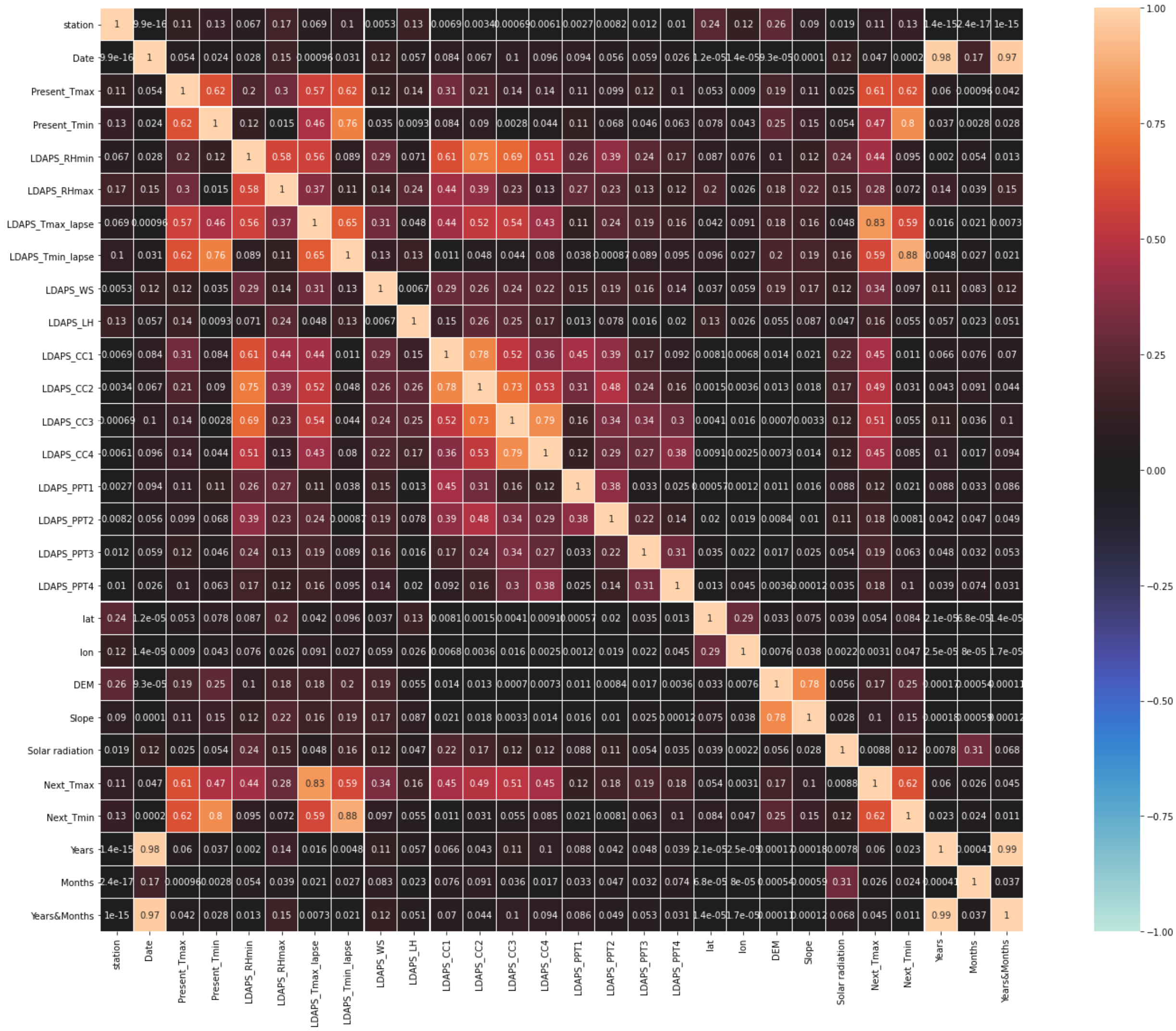
Out[37]:
```
station              0.000000
Date                -0.000284
Present_Tmax        -0.264137
Present_Tmin        -0.367538
LDAPS_RHmin          0.300220
LDAPS_RHmax         -0.855015
LDAPS_Tmax_lapse    -0.227880
LDAPS_Tmin_lapse    -0.581763
LDAPS_WS             1.579236
LDAPS_LH             0.673757
LDAPS_CC1            0.459458
LDAPS_CC2            0.472350
LDAPS_CC3            0.640735
LDAPS_CC4            0.666482
LDAPS_PPT1           5.393821
LDAPS_PPT2           5.775355
LDAPS_PPT3           6.457129
LDAPS_PPT4           6.825464
lat                  0.087062
lon                 -0.285213
DEM                  1.723257
Slope                1.563020
Solar radiation     -0.511210
Next_Tmax           -0.340200
Next_Tmin           -0.404447
Years               -0.000456
Months              -0.705185
Years&Months        -0.007382
dtype: float64
```

Skewness present in our dataset

In [38]:
```python
print('-----------')
print('Heat Map :-')
print('-----------')
df_corr = df.corr().abs()

plt.figure(figsize = (22,16))
sns.heatmap(df_corr, vmin = -1, annot = True, square = True, center = 0, fmt = '.
plt.tight_layout()
```

```
-----------
Heat Map :-
-----------
```



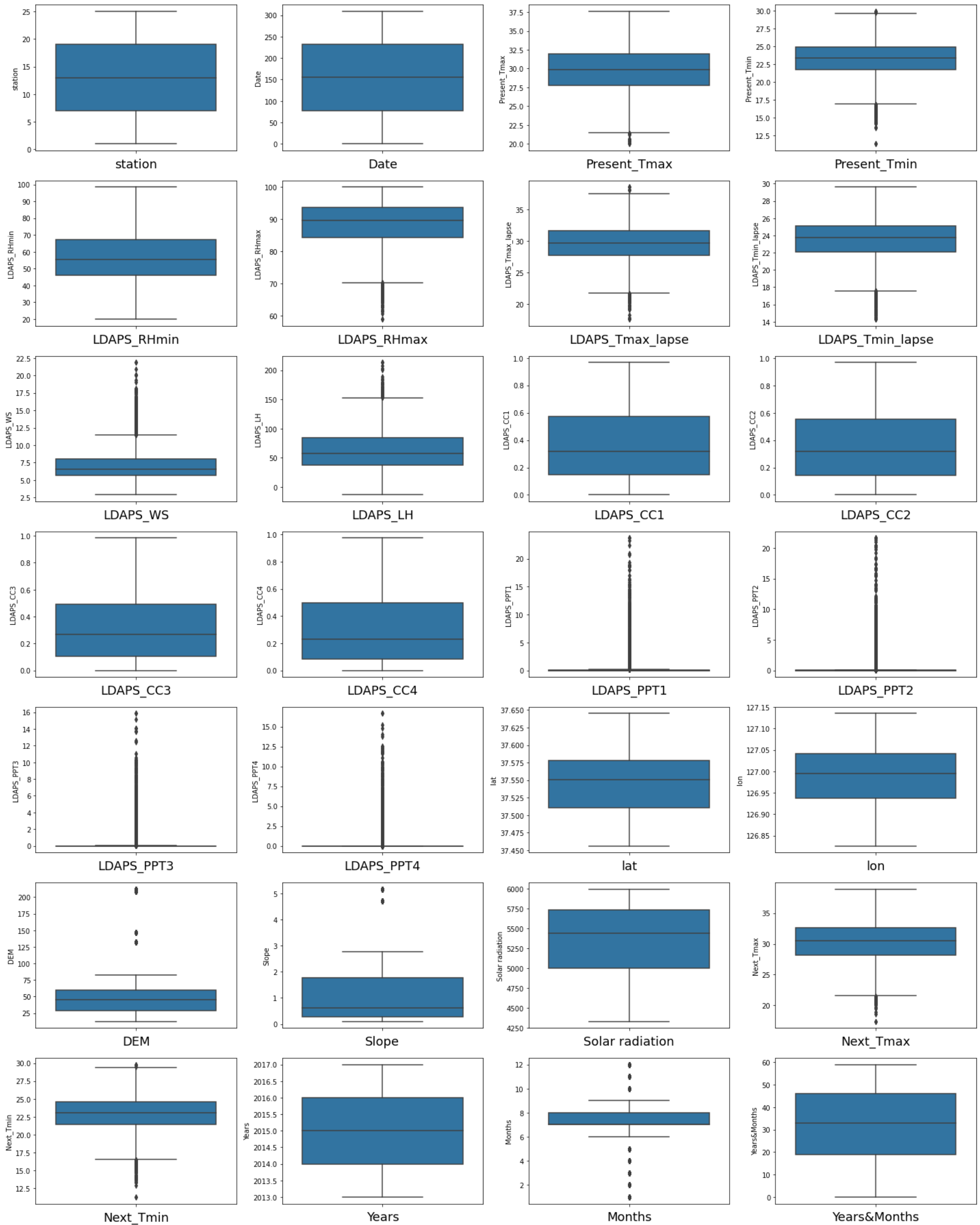
Date and Years has highest corelation with label

Checking Outliers

In [39]:
```
print('===========')
print('Box Plot :-')
print('===========')

plt.figure(figsize = (20,25), facecolor = 'white')
plotnumber = 1
for column in df:
    if plotnumber <=28:
        ax = plt.subplot(7,4, plotnumber)
        sns.boxplot(y=df[column]) # It is the axis for vertical set as y
        plt.xlabel(column, fontsize = 18)
    plotnumber += 1
plt.tight_layout()
```

```
===========
Box Plot :-
===========
```



There are outliers presents in dataset

Removing Outliers

In [40]:

```
# with std 3 Lets see the stats

z_score = zscore(df[['LDAPS_Tmin_lapse', 'LDAPS_Tmax_lapse', 'Present_Tmin', 'Pre
abs_z_score = np.abs(z_score)

filtering_entry = (abs_z_score < 3).all(axis = 1)

df = df[filtering_entry]
df.describe()
```

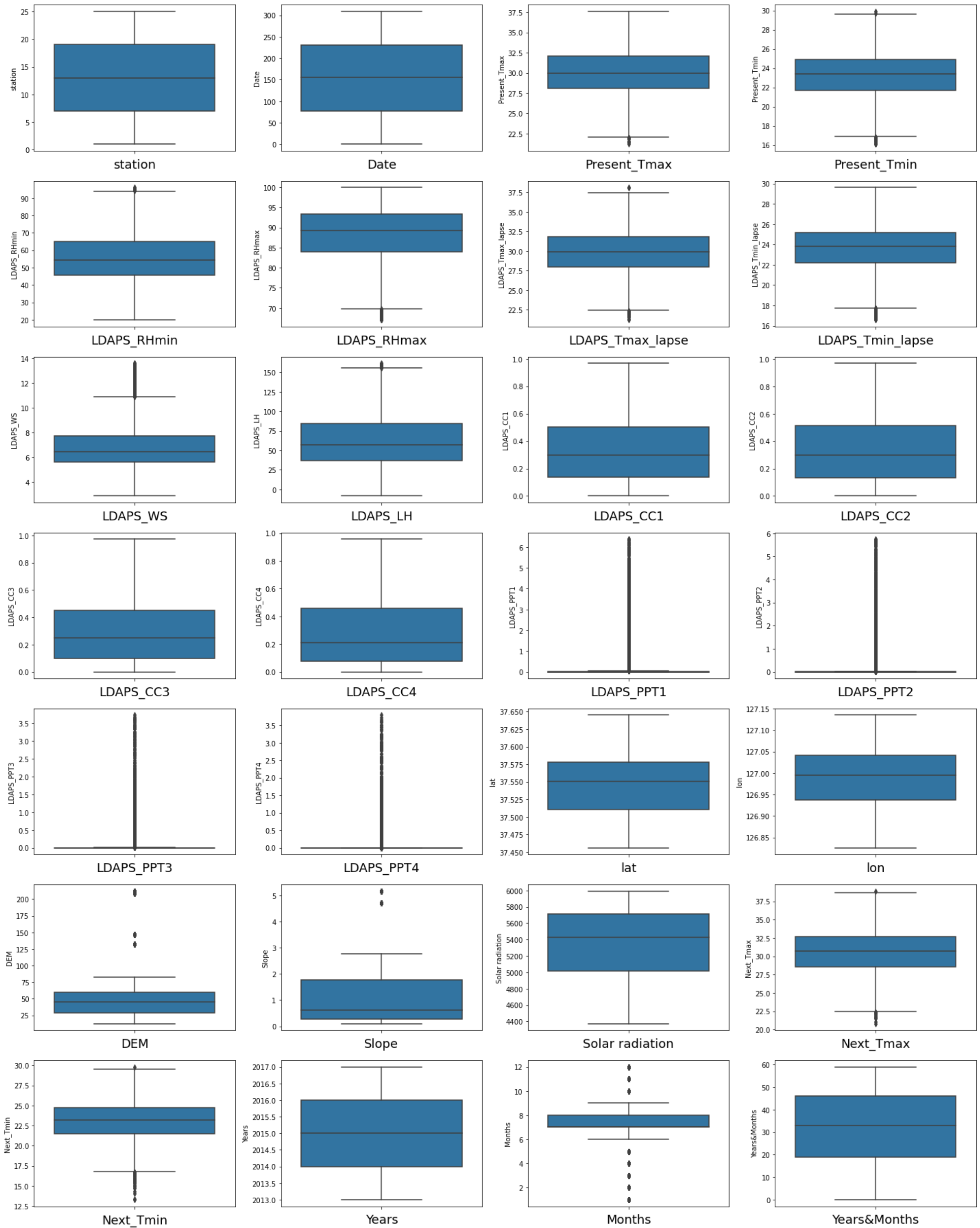
Out[40]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LD
count	6907.000000	6907.000000	6907.000000	6907.000000	6907.000000	6907.000000	
mean	13.033300	154.235558	29.931282	23.302013	55.583421	88.088971	
std	7.197559	88.742729	2.858598	2.314743	13.831826	6.914275	
min	1.000000	0.000000	21.200000	16.100000	19.794666	66.989464	
25%	7.000000	77.000000	28.100000	21.700000	45.672770	83.927235	
50%	13.000000	155.000000	30.000000	23.400000	54.305267	89.179153	
75%	19.000000	230.000000	32.100000	24.900000	65.023258	93.405521	
max	25.000000	309.000000	37.600000	29.900000	96.169815	99.999008	

In [41]:
```python
# Let' see outliers are removed in columns or not.
print('===========')
print('Box Plot :-')
print('===========')

plt.figure(figsize = (20,25), facecolor = 'white')
plotnumber = 1
for column in df:
    if plotnumber <=28:
        ax = plt.subplot(7,4, plotnumber)
        sns.boxplot(y=df[column]) # It is the axis for vertical set as y
        plt.xlabel(column, fontsize = 18)
    plotnumber += 1
plt.tight_layout()
```

```
===========
Box Plot :-
===========
```



In [42]:
```
df.shape # Here we check shape of remaining data after removal of outliers.
```

Out[42]:
```
(6907, 28)
```

Outliers are removed

Spliting Dataset into features and label

In [43]:

```
x = df.drop('Next_Tmin', axis = 1)
y = df. Next_Tmin
print('Data has been splited')
```

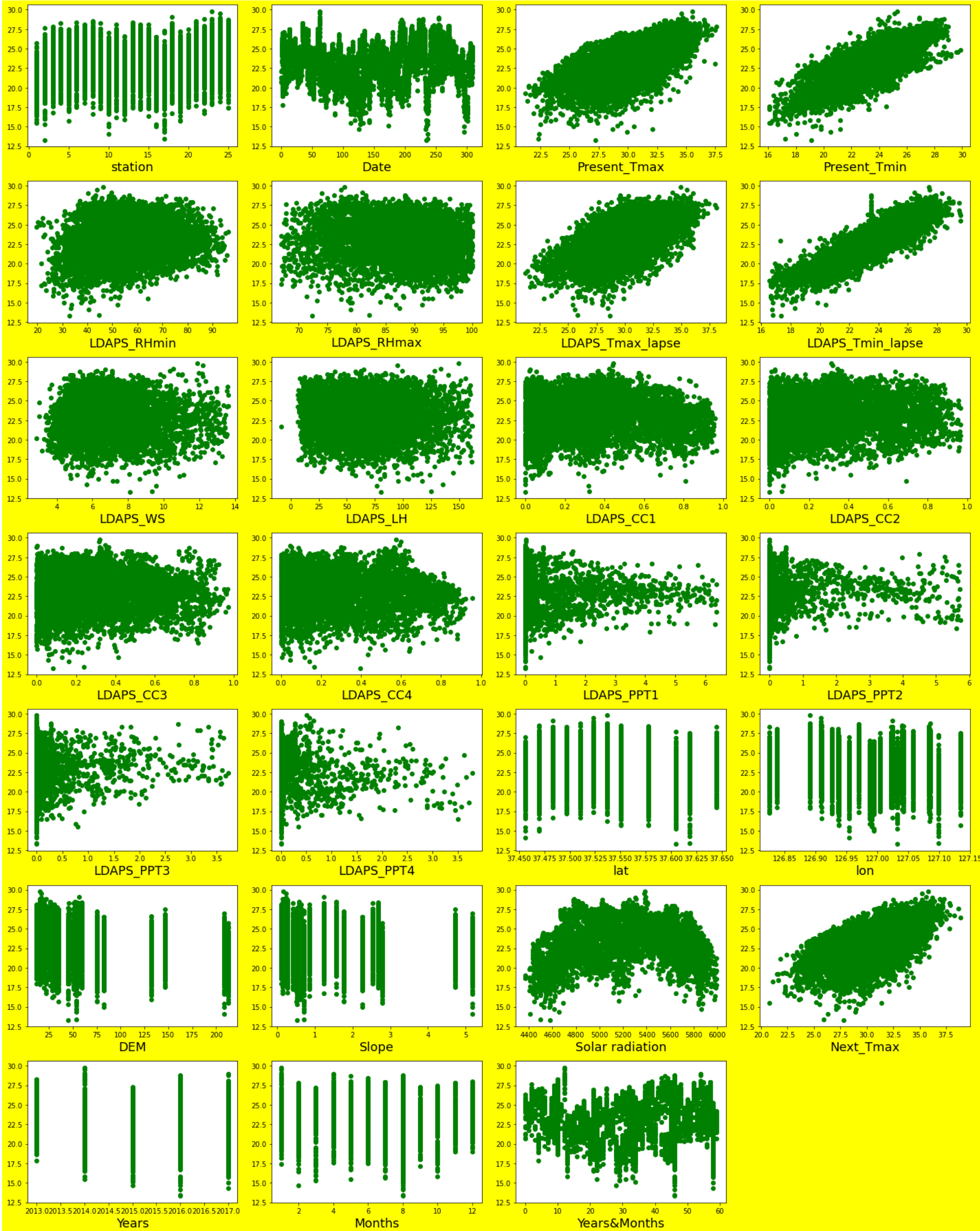
```
Data has been splited
```

In [44]:

```python
# Let' see relation between features and label.
print('-----------------')
print('Scatter Plot :-')
print('-----------------')

plt.figure(figsize = (20,25), facecolor = 'yellow')
plotnumber = 1
for column in x:
    if plotnumber <=28:
        ax = plt.subplot(7,4, plotnumber)
        plt.scatter(x[column],y, c = 'g')
        plt.xlabel(column, fontsize = 18)
    plotnumber += 1
plt.tight_layout()
```

```
-----------------
Scatter Plot :-
-----------------
```



Positive relation in feature and label

Data Scaling

In [45]:
```
df.head()
```

Out[45]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse
0	1.0	12.0	28.7	21.4	58.255688	91.116364	28.074101
1	2.0	12.0	31.9	21.6	52.263397	90.604721	29.850689
2	3.0	12.0	31.6	23.3	48.690479	83.973587	30.091292
3	4.0	12.0	32.0	23.4	58.239788	96.483688	29.704629
4	5.0	12.0	31.4	21.9	56.174095	90.155128	29.113934

In [46]:
```
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

Out[46]:
```
array([[0.        , 0.03883495, 0.45731707, ..., 0.        , 0.45454545,
        0.13559322],
       [0.04166667, 0.03883495, 0.65243902, ..., 0.        , 0.45454545,
        0.13559322],
       [0.08333333, 0.03883495, 0.63414634, ..., 0.        , 0.45454545,
        0.13559322],
       ...,
       [0.91666667, 0.97411003, 0.12804878, ..., 1.        , 0.63636364,
        0.98305085],
       [0.95833333, 0.97411003, 0.12804878, ..., 1.        , 0.63636364,
        0.98305085],
       [1.        , 0.97411003, 0.12195122, ..., 1.        , 0.63636364,
        0.98305085]])
```

Split data into train and test. Model will be bulit on training data and tested on test data

In [47]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, rando
print('Data has been splited.')
```

```
Data has been splited.
```

Model Building

In [48]:
```
bag_dt = BaggingRegressor(DecisionTreeRegressor(), n_estimators = 15, max_samples
                          random_state= 3, oob_score = True)
```

In [49]:
```
bag_dt.oob_score
```
Out[49]:
```
True
```

In [50]:
```
bag_dt.fit(x_train, y_train)
print('Bagging DT score ------->', bag_dt.score(x_test, y_test))
```
```
Bagging DT score -------> 0.881936042407601
```

In [51]:
```
y_pred = bag_dt.predict(x_test)
```

In [52]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_dt.score(x_test, y_test))
print('==========================================================')
```
```
==========================================================
R2 Score -----> 0.881936042407601
==========================================================
RMSE of Model -------> 0.8229614948083417
==========================================================
MSE of Model -------> 0.6772656219371802
==========================================================
Score of test data ----> 0.881936042407601
==========================================================
```

Conclusion : Decision Tree model has 88% score

Xgboost model instantiaing, training and evaluating

In [53]:
```
bag_xgb = BaggingRegressor(xgb.XGBRegressor(eval_metric = 'mlogloss'), n_estimato
                          random_state= 3, oob_score = True)
```

In [54]:
```
bag_xgb.oob_score
```
Out[54]:
```
True
```

In [55]:
```
bag_xgb.fit(x_train, y_train)
print('Bagging XGBoost score ------->', bag_xgb.score(x_test, y_test))
```
```
Bagging XGBoost score -------> 0.9304298762527333
```

In [56]:
```
y_pred = bag_xgb.predict(x_test)
```

In [57]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_xgb.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.9304298762527333
==========================================================
RMSE of Model -------> 0.6317310671849325
==========================================================
MSE of Model -------> 0.39908414124661373
==========================================================
Score of test data ----> 0.9304298762527333
==========================================================
```

Conclusion : XGBoost model has 93% score

Knn model instantiaing, training and evaluating

In [58]:
```
bag_Knn = BaggingRegressor(KNeighborsRegressor(n_neighbors = 5), n_estimators = 3
                           random_state= 3, oob_score = True)
```

In [59]:
```
bag_Knn.oob_score
```

Out[59]:
```
True
```

In [60]:
```
bag_Knn.fit(x_train, y_train)
print('Bagging KNN score ------->', bag_Knn.score(x_test, y_test))
```

```
Bagging KNN score -------> 0.7599901059442604
```

In [61]:
```
y_pred = bag_Knn.predict(x_test)
```

In [62]:
```python
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_Knn.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.7599901059442604
==========================================================
RMSE of Model -------> 1.173371187528214
==========================================================
MSE of Model -------> 1.3767999437213714
==========================================================
Score of test data ----> 0.7599901059442604
==========================================================
```

Conclusion : Knn model has 75% score

Random Forest model instantiaing, training and evaluating

In [63]:
```python
bag_Rn = BaggingRegressor(RandomForestRegressor(), n_estimators = 30, max_samples
                          random_state= 3, oob_score = True)
```

In [64]:
```python
bag_Rn.oob_score
```

Out[64]:
```
True
```

In [65]:
```python
bag_Rn.fit(x_train, y_train)
print('Bagging Random Forest score ------->', bag_Rn.score(x_test, y_test))
```

```
Bagging Random Forest score -------> 0.8839390752176481
```

In [66]:
```python
y_pred = bag_Rn.predict(x_test)
```

In [67]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_Rn.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8839390752176481
==========================================================
RMSE of Model -------> 0.8159505895222902
==========================================================
MSE of Model -------> 0.6657753645417729
==========================================================
Score of test data ----> 0.8839390752176481
==========================================================
```

Conclusion : Random Forest model has 88% score

Linear Regression model instantiaing, training and evaluating

In [68]:
```
bag_Lr = BaggingRegressor(LinearRegression(), n_estimators = 30, max_samples = 0.
                          random_state= 3, oob_score = True)
```

In [69]:
```
bag_Lr.oob_score
```

Out[69]:
```
True
```

In [70]:
```
bag_Lr.fit(x_train, y_train)
print('Bagging Linear Regression score ------->', bag_Lr.score(x_test, y_test))
```

```
Bagging Linear Regression score -------> 0.8298079188960044
```

In [71]:
```
y_pred = bag_Lr.predict(x_test)
```

In [72]:
```
print('==========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('==========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('==========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('==========================================================')
print('Score of test data ---->', bag_Lr.score(x_test, y_test))
print('==========================================================')
```

```
==========================================================
R2 Score -----> 0.8298079188960044
==========================================================
RMSE of Model -------> 0.988076389103983
==========================================================
MSE of Model -------> 0.9762949507047656
==========================================================
Score of test data ----> 0.8298079188960044
==========================================================
```

Conclusion : Linear Regression model has 82% score

Looking RMSE we found XGBoost has best model so we do Hyperparameter Tuning on it

In [74]:
```
param = {'n_estimators': [50,100], 'max_samples': [1.0], 'bootstrap': [True]}
```

In [75]:
```
grid_search = GridSearchCV(estimator = bag_xgb, param_grid = param, cv = 5 , n_jo
```

```
In [76]: grid_search.fit(x_train, y_train)
```

```
Out[76]: GridSearchCV(cv=5,
             estimator=BaggingRegressor(base_estimator=XGBRegressor(base_scor
e=None,
                                                                    booster=N
one,
                                                                    colsample
_bylevel=None,
                                                                    colsample
_bynode=None,
                                                                    colsample
_bytree=None,
                                                                    eval_metr
ic='mlogloss',
                                                                    gamma=Non
e,
                                                                    gpu_id=No
ne,
                                                                    importanc
e_type='gain',
                                                                    interacti
on_constraints=None,
                                                                    learning_
rate=None,
                                                                    max_delta
_step=None,
                                                                    max_depth
=None,
                                                                    min_child
_weight=None,
                                                                    missin
g...
                                                                    monotone_
constraints=None,
                                                                    n_estimat
ors=100,
                                                                    n_jobs=No
ne,
                                                                    num_paral
lel_tree=None,
                                                                    random_st
ate=None,
                                                                    reg_alpha
=None,
                                                                    reg_lambd
a=None,
                                                                    scale_pos
_weight=None,
                                                                    subsample
=None,
                                                                    tree_meth
od=None,
                                                                    validate_
parameters=None,
                                                                    verbosity
```

```
=None),
                                          max_samples=0.5, n_estimators=30,
                                          oob_score=True, random_state=3),
             n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_samples': [1.0],
                         'n_estimators': [50, 100]})
```

In [77]:
```
best_parameters = grid_search.best_params_
print(best_parameters)
```

```
{'bootstrap': True, 'max_samples': 1.0, 'n_estimators': 100}
```

In [78]:
```
hxgb = BaggingRegressor(base_estimator=xgb.XGBRegressor(eval_metric = 'mlogloss')
hxgb.fit(x_train, y_train)
hxgb.score(x_test, y_test)
```

Out[78]:
```
0.94111442947522
```

In [79]:
```
y_pred = hxgb.predict(x_test)
```

In [80]:
```
print('=========================================================')
print('R2 Score ----->', r2_score(y_test, y_pred))
print('=========================================================')
print('RMSE of Model ------->', np.sqrt(mean_squared_error(y_test, y_pred)))
print('=========================================================')
print('MSE of Model ------->', mean_squared_error(y_test, y_pred))
print('=========================================================')
print('Score of test data ---->', bag_Lr.score(x_test, y_test))
print('=========================================================')
```

```
=========================================================
R2 Score -----> 0.94111442947522
=========================================================
RMSE of Model -------> 0.5811995786096222
=========================================================
MSE of Model -------> 0.3377929501760024
=========================================================
Score of test data ----> 0.8298079188960044
=========================================================
```

After Hyperparameter Tuning model accuracy score 94%.

Saving The Model

In [81]:
```
# saving the model to the Local file system
filename = 'Temperature Forecast Project (Next  Day Minimum  Air Temperatures).pi
pickle.dump(hxgb, open(filename, 'wb'))
```

Predict Temperature Forecast Project (Next Day Maximum Air Temperatures)

In [82]:
```
model = pickle.load(open('Temperature Forecast Project (Next  Day Minimum  Air Te
result = model.score(x_test, y_test)
print('Predicted Score --------->', result)
```

```
Predicted Score ---------> 0.94111442947522
```

In [83]:
```
Prediction = pd.DataFrame([model.predict(x_test)[:], y_test[:]], index = ['Predic
Prediction
```

Out[83]:

	0	1	2	3	4	5	6	7
Predicted	21.342047	21.685257	22.286797	23.86718	23.600315	21.268257	23.350143	24.788773
Orginal	21.500000	22.100000	22.100000	24.40000	23.700000	21.400000	23.300000	24.700000

Saving the predicted result in CSV file

In [85]:
```
Prediction.to_csv('Temperature Forecast Project (Next  Day Minimum  Air Temperatu
```

Final Conclusion : XGBoost is our best model.

In [ ]: