# Red Planet Farming

```
# Importing Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as pt
import seaborn as sn
from sklearn.preprocessing import MinMaxScaler

# loading dataset (Mars-weather, Mars Climate)
```

# Loading Mars Soil Data

```
data_soil = pd.read_csv('/home/ali/Downloads/soil_unclean.csv')
```

# Loading Mars Weather Data

```
data_weather = pd.read_csv('/home/ali/Downloads/archive (1) (1)/mars-weather.csv')

df_soil = pd.DataFrame(data_soil)

df_weather = pd.DataFrame(data_weather)
```

## Printing Soil Data

```
df_soil.head()
```

```
    Soil_pH  Iron_Content (%)  SiO2_Content (%)  Magnesium (%)  Calcium (%)  \
0  8.186779         12.452108         42.822046       4.391012     0.641588
1  7.665686         11.191043         47.201392       1.694737     1.040821
2  7.669166         12.249248         42.755846       2.294371     0.909053
3  8.037380         17.024623         47.761369       4.673164     0.400824
4  8.070532         11.824930         46.467487       2.335770     0.711514

   Sulfur (%)  Temperature (°C)  Moisture_Content (%)  Radiation_Level (mSv)  \
0    0.400589        -74.606272              0.963659             381.828311
```

```
1    0.443367        -72.768567              0.659950
270.308820
2    0.332224        -38.202303              0.604375
719.346008
3    0.195027        -58.271329              0.926341
758.049931
4    0.495119        -77.836886              0.503156
361.748616

   Organic_Matter (%)  Soil_Nutrient_Index  Soil_pH_copy
0            0.175318             2.424220      8.186779
1            0.590627             2.553363      7.665686
2            1.273567             2.763104      7.669166
3            1.027846             2.739775      8.037380
4            1.835618             2.788502      8.070532
```

# Printing Weather Data

```
df_weather.head()

     id terrestrial_date   sol   ls    month  min_temp  max_temp
pressure  \
0  1895       2018-02-27  1977  135  Month 5     -77.0     -10.0
727.0
1  1893       2018-02-26  1976  135  Month 5     -77.0     -10.0
728.0
2  1894       2018-02-25  1975  134  Month 5     -76.0     -16.0
729.0
3  1892       2018-02-24  1974  134  Month 5     -77.0     -13.0
729.0
4  1889       2018-02-23  1973  133  Month 5     -78.0     -18.0
730.0

   wind_speed atmo_opacity
0         NaN        Sunny
1         NaN        Sunny
2         NaN        Sunny
3         NaN        Sunny
4         NaN        Sunny
```

Preprocessing Individual Dataset before Integaration

## 1- Dataset for soil

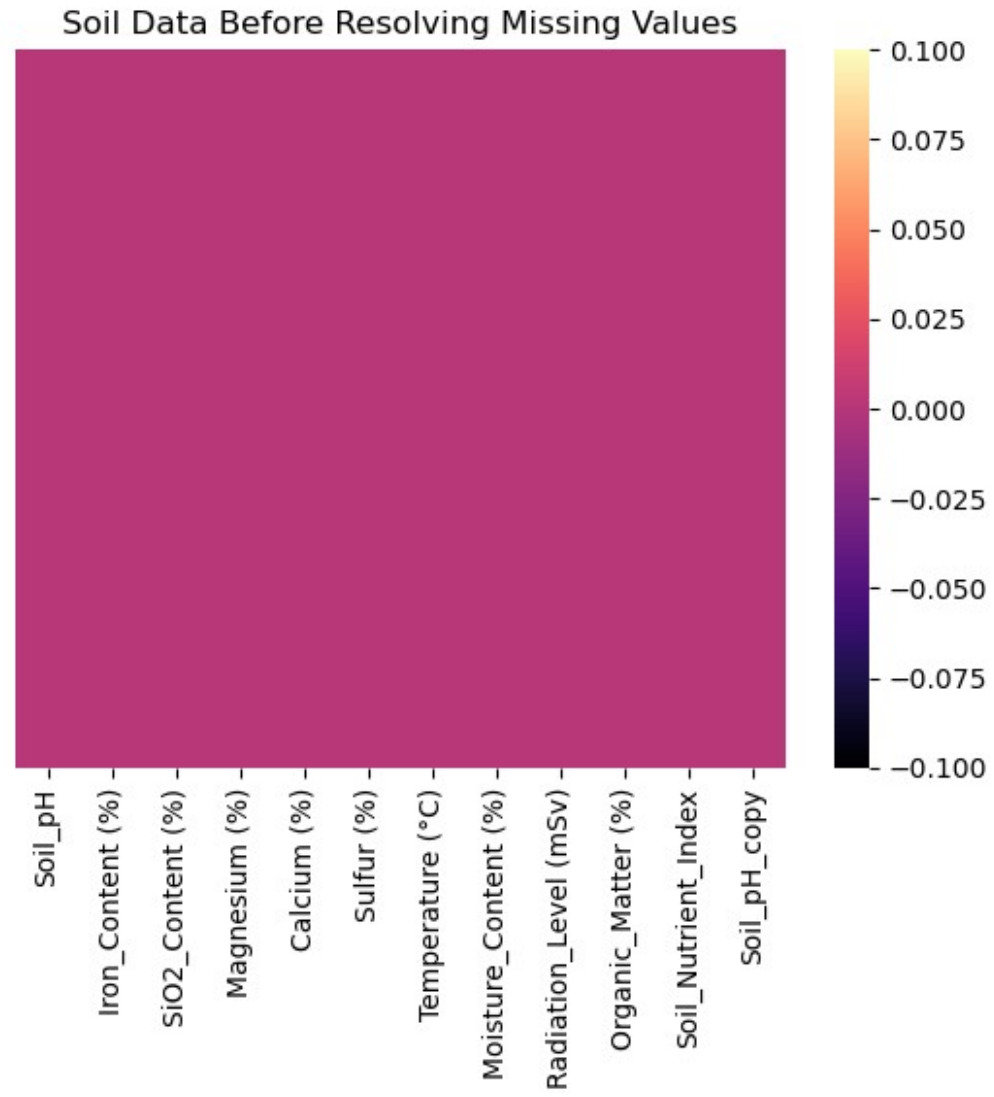
```
df_soil.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2274 entries, 0 to 2273
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
```

```
---  ------                  --------------  -----
 0   Soil_pH                 1942 non-null   float64
 1   Iron_Content (%)        1942 non-null   float64
 2   SiO2_Content (%)        1942 non-null   float64
 3   Magnesium (%)           1942 non-null   float64
 4   Calcium (%)             1942 non-null   float64
 5   Sulfur (%)              1942 non-null   float64
 6   Temperature (°C)        1942 non-null   float64
 7   Moisture_Content (%)    1942 non-null   float64
 8   Radiation_Level (mSv)   1942 non-null   float64
 9   Organic_Matter (%)      1942 non-null   float64
 10  Soil_Nutrient_Index     1942 non-null   float64
 11  Soil_pH_copy            1942 non-null   float64
dtypes: float64(12)
memory usage: 213.3 KB
```

```
sn.heatmap(df_soil.isnull(),yticklabels=False,cbar=True,cmap='magma')
plt.title("Soil Data Before Resolving Missing Values")
```

```
<IPython.core.display.Javascript object>
```

```
Text(0.5, 1.0, 'Soil Data Before Resolving Missing Values')
```




## Resolving NaN values

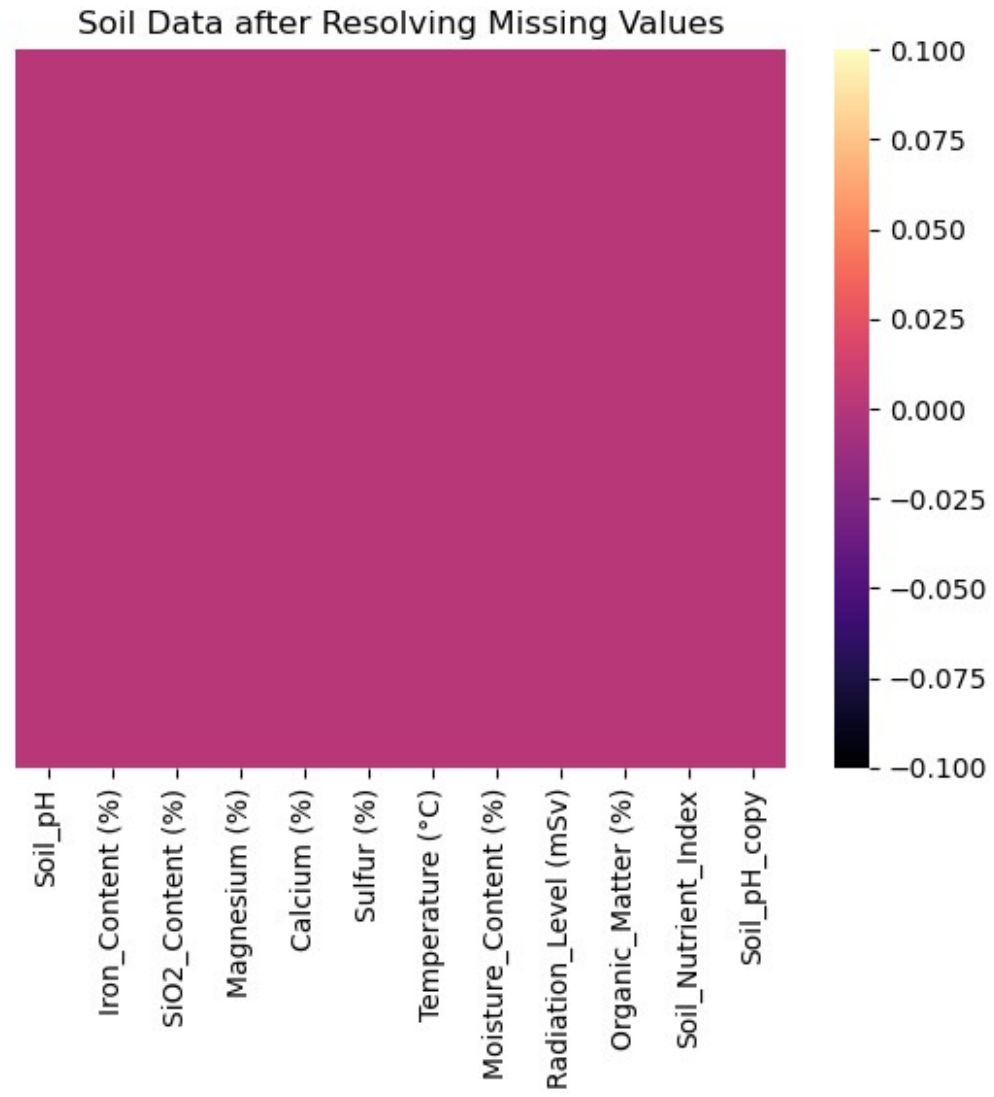
```
def refilling(df_soil):
    columns = ['Soil_pH','Iron_Content (%)','SiO2_Content
(%)','Magnesium (%)','Calcium (%)','Sulfur (%)','Moisture_Content
(%)','Radiation_Level (mSv)','Organic_Matter
(%)','Soil_Nutrient_Index','Temperature (°C)','Soil_pH_copy']
    try:
        for col in columns:
            mean_value = df_soil[col].mean()
            df_soil[col].fillna(mean_value,inplace=True)
        print("Status: {}".format(1))
    except Exception as ex:
        print(f"The Issue is: {ex}")
    return df_soil

df_soil = refilling(df_soil)
```

```
Status: 1

/tmp/ipykernel_6179/3071527732.py:6: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df_soil[col].fillna(mean_value,inplace=True)
```

```
sn.heatmap(df_soil.isnull(),yticklabels=False,cbar=True,cmap='magma')
plt.title("Soil Data after Resolving Missing Values")
```

```
<IPython.core.display.Javascript object>
```

```
Text(0.5, 1.0, 'Soil Data after Resolving Missing Values')
```




```
# All the Missing values are being resolved
```

## Resolving Duplicacy

```
df_soil.duplicated().sum()
```

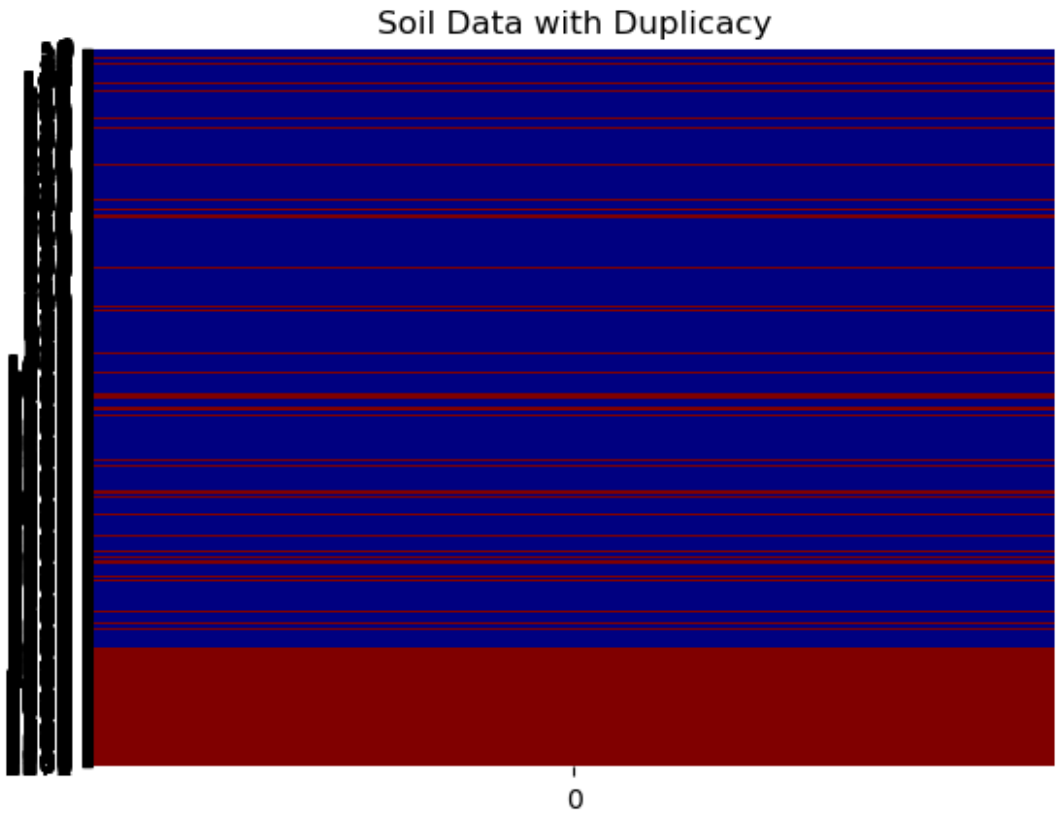
```
600
```

```
sns.heatmap(df_soil.duplicated().astype(int).to_frame(),yticklabels=Tr
ue,cbar=False,cmap='jet')
plt.title('Soil Data with Duplicacy')
```

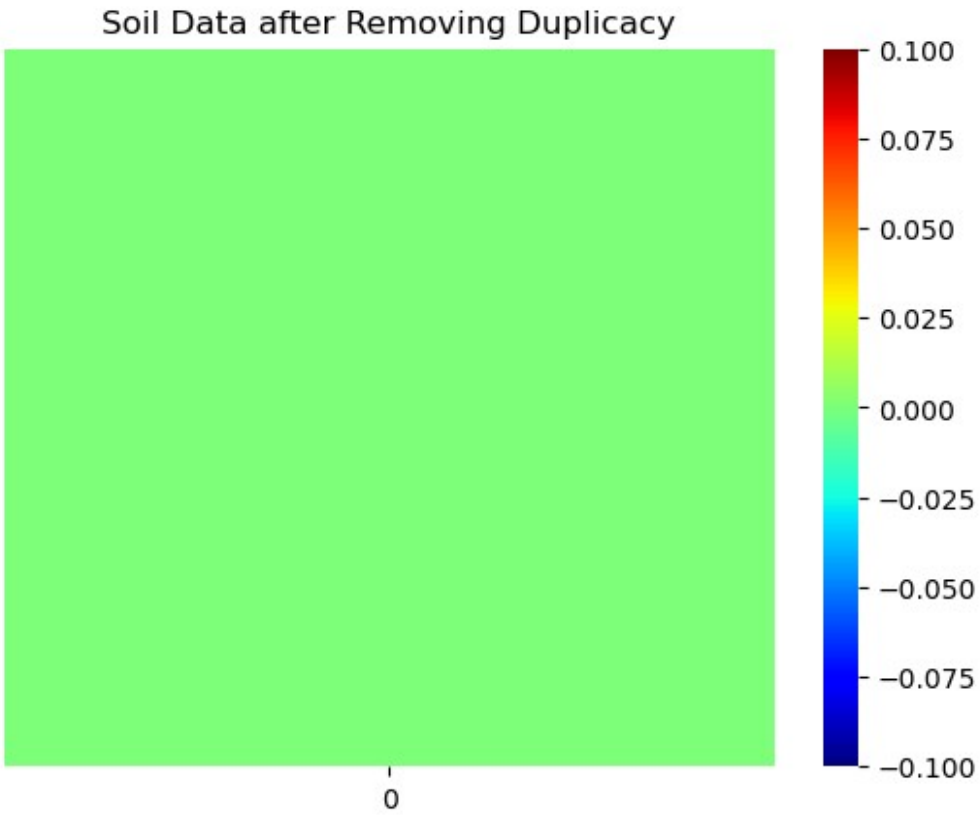
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
Text(0.5, 1.0, 'Soil Data with Duplicacy')
```




```
def removingDups(df_soil):
    try:
        return df_soil.drop_duplicates()
    except Exception as ex:
        print("The issue is:".format(ex))

df_soil = removingDups(df_soil)

sn.heatmap(df_soil.duplicated().astype(int).to_frame(),yticklabels=Fal
se,cbar=True,cmap='jet')
pt.title('Soil Data after Removing Duplicacy')

Text(0.5, 1.0, 'Soil Data after Removing Duplicacy')
```




```
df_soil.describe()
```

| | Soil_pH | Iron_Content (%) | SiO2_Content (%) | Magnesium (%) |
|---|---|---|---|---|
| count | 1674.000000 | 1674.000000 | 1674.000000 | 1674.000000 |
| mean | 7.995279 | 14.995683 | 45.059296 | 2.997471 |
| std | 0.287168 | 2.876331 | 2.882797 | 1.138867 |
| min | 7.500352 | 10.004374 | 40.022281 | 1.001432 |
| 25% | 7.747627 | 12.493214 | 42.568539 | 2.021282 |
| 50% | 8.001685 | 15.061470 | 45.022217 | 3.004048 |
| 75% | 8.247097 | 17.509920 | 47.546873 | 3.957083 |
| max | 8.499454 | 19.999237 | 49.982491 | 4.998670 |

| | Calcium (%) | Sulfur (%) | Temperature (°C) | Moisture_Content (%) |
|---|---|---|---|---|
| count | 1674.000000 | 1674.000000 | 1674.000000 | 1674.000000 |

```
mean      1.070976     0.251488        -30.471611
0.545007
std       0.540142     0.141087         28.949915
0.261068
min       0.100551     0.010146        -79.884044
0.100682
25%       0.618244     0.128495        -56.188225
0.317186
50%       1.086482     0.248375        -30.367376
0.544573
75%       1.534774     0.372433         -5.922783
0.762500
max       1.999486     0.499468         19.966298
0.999831

       Radiation_Level (mSv)  Organic_Matter (%)  Soil_Nutrient_Index
\
count            1674.000000         1674.000000          1674.000000

mean               501.886134            1.012848             2.618003

std                176.663643            0.570007             0.204840

min                200.069820            0.012021             2.103027

25%                345.826263            0.504444             2.458555

50%                497.224951            1.012772             2.620392

75%                663.116755            1.514166             2.774868

max                799.917914            1.999154             3.171595


       Soil_pH_copy
count   1674.000000
mean       7.995279
std        0.287168
min        7.500352
25%        7.747627
50%        8.001685
75%        8.247097
max        8.499454

# removing the Soil_pH_copy column
df_soil.drop('Soil_pH_copy',axis=1)

       Soil_pH  Iron_Content (%)  SiO2_Content (%)  Magnesium (%)  \
0     8.186779         12.452108         42.822046       4.391012
1     7.665686         11.191043         47.201392       1.694737
2     7.669166         12.249248         42.755846       2.294371
```

```
3     8.037380         17.024623         47.761369       4.673164
4     8.070532         11.824930         46.467487       2.335770
...        ...               ...               ...            ...
1890  8.011381         18.371312         43.463997       4.262066
1891  8.349989         12.865125         44.158167       3.412221
1892  8.287879         11.149358         47.997427       3.997462
1893  8.310323         19.471555         47.524722       4.155494
1894  8.249164         12.805329         42.061938       2.564028

      Calcium (%)  Sulfur (%)  Temperature (°C)  Moisture_Content (%)
\
0        0.641588    0.400589        -74.606272              0.963659

1        1.040821    0.443367        -72.768567              0.659950

2        0.909053    0.332224        -38.202303              0.604375

3        0.400824    0.195027        -58.271329              0.926341

4        0.711514    0.495119        -77.836886              0.503156

...           ...         ...               ...                   ...

1890     0.879589    0.067332        -74.497890              0.581762

1891     0.595169    0.127893        -37.384248              0.242791

1892     0.955257    0.186524         -8.056451              0.461055

1893     0.901316    0.195600        -53.475468              0.197721

1894     1.203516    0.416324         -3.320512              0.807525


      Radiation_Level (mSv)  Organic_Matter (%)  Soil_Nutrient_Index
0                381.828311            0.175318             2.424220
1                270.308820            0.590627             2.553363
2                719.346008            1.273567             2.763104
3                758.049931            1.027846             2.739775
4                361.748616            1.835618             2.788502
...                     ...                 ...                  ...
1890             239.357829            0.695691             2.567493
1891             724.896609            1.253325             2.553258
1892             313.593943            0.202610             2.304219
1893             680.063704            0.827146             2.406276
1894             770.007233            0.316181             2.404014

[1674 rows x 11 columns]
```

## Detecting Outliers

```
def detectOutliers(df_soil):
    outliers = {}
    for cols in df_soil.select_dtypes(include=['float64',
'int64']).columns:
        Q1 = df_soil[cols].quantile(0.25)
        Q3 = df_soil[cols].quantile(0.75)

        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        outliers[cols] = df_soil[(df_soil[cols] < lower_bound) |
(df_soil[cols] > upper_bound)].shape[0]

    return outliers

outliers = detectOutliers(df_soil)
print(outliers)
```

```
{'Soil_pH': 0, 'Iron_Content (%)': 0, 'SiO2_Content (%)': 0,
'Magnesium (%)': 0, 'Calcium (%)': 0, 'Sulfur (%)': 0, 'Temperature
(°C)': 0, 'Moisture_Content (%)': 0, 'Radiation_Level (mSv)': 0,
'Organic_Matter (%)': 0, 'Soil_Nutrient_Index': 0, 'Soil_pH_copy': 0}
```

```
# There are no such outliers
```

## Checking Out-of-the-Range Values

```
print(df_soil[df_soil['Temperature (°C)'] < -50])  # Values below -
50°C
print(df_soil[df_soil['Temperature (°C)'] > 60])   # Values above 60°C
```

```
       Soil_pH  Iron_Content (%)  SiO2_Content (%)  Magnesium (%)  \
0     8.186779         12.452108         42.822046       4.391012
1     7.665686         11.191043         47.201392       1.694737
3     8.037380         17.024623         47.761369       4.673164
4     8.070532         11.824930         46.467487       2.335770
5     8.398018         15.759500         43.879286       2.657505
...        ...               ...               ...            ...
1881  8.073880         11.623662         40.702433       4.738478
1883  8.320147         12.943969         40.789945       4.310861
1887  7.993858         12.026110         49.289399       2.570878
1890  8.011381         18.371312         43.463997       4.262066
1893  8.310323         19.471555         47.524722       4.155494

      Calcium (%)  Sulfur (%)  Temperature (°C)  Moisture_Content (%)
\
0        0.641588    0.400589        -74.606272              0.963659
```

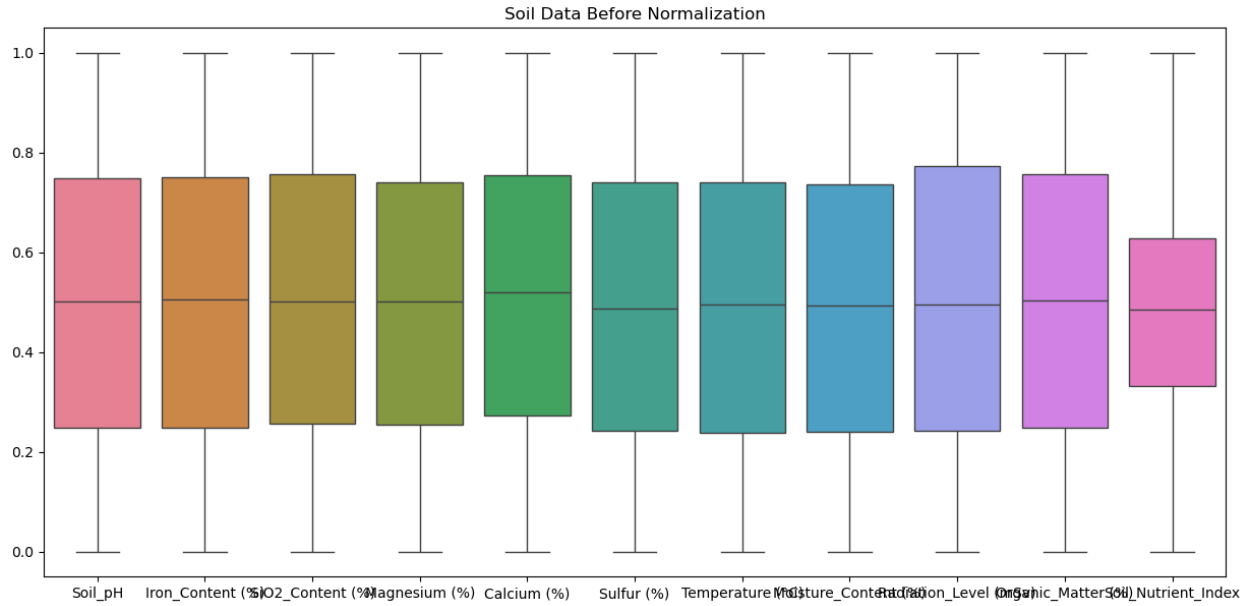
```
1        1.040821    0.443367        -72.768567                0.659950

3        0.400824    0.195027        -58.271329                0.926341

4        0.711514    0.495119        -77.836886                0.503156

5        1.543286    0.157139        -58.055992                0.501180

...           ...         ...                ...                     ...

1881     1.273061    0.174730        -51.732246                0.665719

1883     0.208155    0.452266        -50.483927                0.513829

1887     1.674179    0.445035        -62.112147                0.990417

1890     0.879589    0.067332        -74.497890                0.581762

1893     0.901316    0.195600        -53.475468                0.197721


      Radiation_Level (mSv)  Organic_Matter (%)
Soil_Nutrient_Index  \
0                381.828311            0.175318             2.424220

1                270.308820            0.590627             2.553363

3                758.049931            1.027846             2.739775

4                361.748616            1.835618             2.788502

5                423.418589            1.906458             2.811260

...                     ...                 ...                  ...

1881             447.182688            1.371723             2.761511

1883             269.446846            0.569144             2.370409

1887             688.302147            1.530950             2.868631

1890             239.357829            0.695691             2.567493

1893             680.063704            0.827146             2.406276


      Soil_pH_copy
0         8.186779
1         7.665686
3         8.037380
4         8.070532
```

```
5          8.398018
...             ...
1881       8.073880
1883       8.320147
1887       7.993858
1890       8.011381
1893       8.310323

[513 rows x 12 columns]
Empty DataFrame
Columns: [Soil_pH, Iron_Content (%), SiO2_Content (%), Magnesium (%),
Calcium (%), Sulfur (%), Temperature (°C), Moisture_Content (%),
Radiation_Level (mSv), Organic_Matter (%), Soil_Nutrient_Index,
Soil_pH_copy]
Index: []
```

## Normalization

To take all the columns to a consistent scale. So they have equal influence during the training

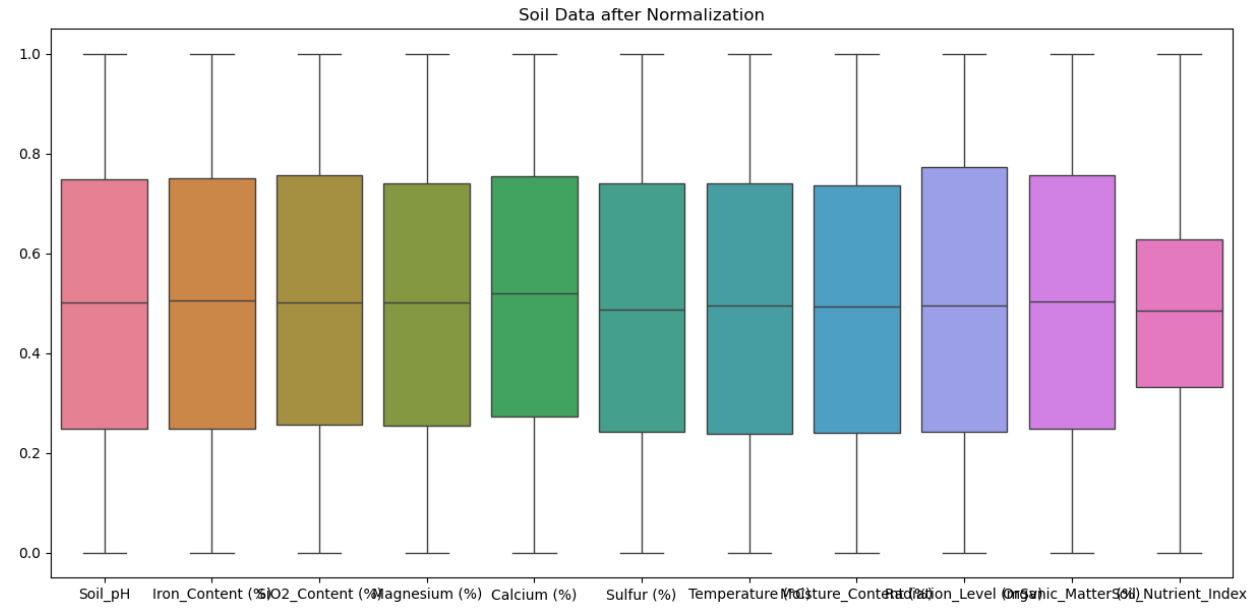
```
columns_to_normalize = ['Soil_pH', 'Iron_Content (%)', 'SiO2_Content
(%)', 'Magnesium (%)',
                        'Calcium (%)', 'Sulfur (%)', 'Temperature
(°C)', 'Moisture_Content (%)',
                        'Radiation_Level (mSv)', 'Organic_Matter (%)',
'Soil_Nutrient_Index']

pt.figure(figsize=(15, 7))
sn.boxplot(data=df_soil[columns_to_normalize])
pt.title('Soil Data Before Normalization')

Text(0.5, 1.0, 'Soil Data Before Normalization')
```




```
print("Before Normalization:")
print(df_soil[columns_to_normalize].min())
print(df_soil[columns_to_normalize].max())
```

```
Before Normalization:
Soil_pH                    7.500352
Iron_Content (%)          10.004374
SiO2_Content (%)          40.022281
Magnesium (%)              1.001432
Calcium (%)                0.100551
Sulfur (%)                 0.010146
Temperature (°C)         -79.884044
Moisture_Content (%)       0.100682
Radiation_Level (mSv)    200.069820
Organic_Matter (%)         0.012021
Soil_Nutrient_Index        2.103027
dtype: float64
Soil_pH                    8.499454
Iron_Content (%)          19.999237
SiO2_Content (%)          49.982491
Magnesium (%)              4.998670
Calcium (%)                1.999486
Sulfur (%)                 0.499468
Temperature (°C)          19.966298
Moisture_Content (%)       0.999831
Radiation_Level (mSv)    799.917914
Organic_Matter (%)         1.999154
Soil_Nutrient_Index        3.171595
dtype: float64
```

```
scaler = MinMaxScaler()
df_soil[columns_to_normalize] =
scaler.fit_transform(df_soil[columns_to_normalize])

pt.figure(figsize=(15,7))
sn.boxplot(data=df_soil[columns_to_normalize])
pt.title('Soil Data after Normalization')

Text(0.5, 1.0, 'Soil Data after Normalization')
```

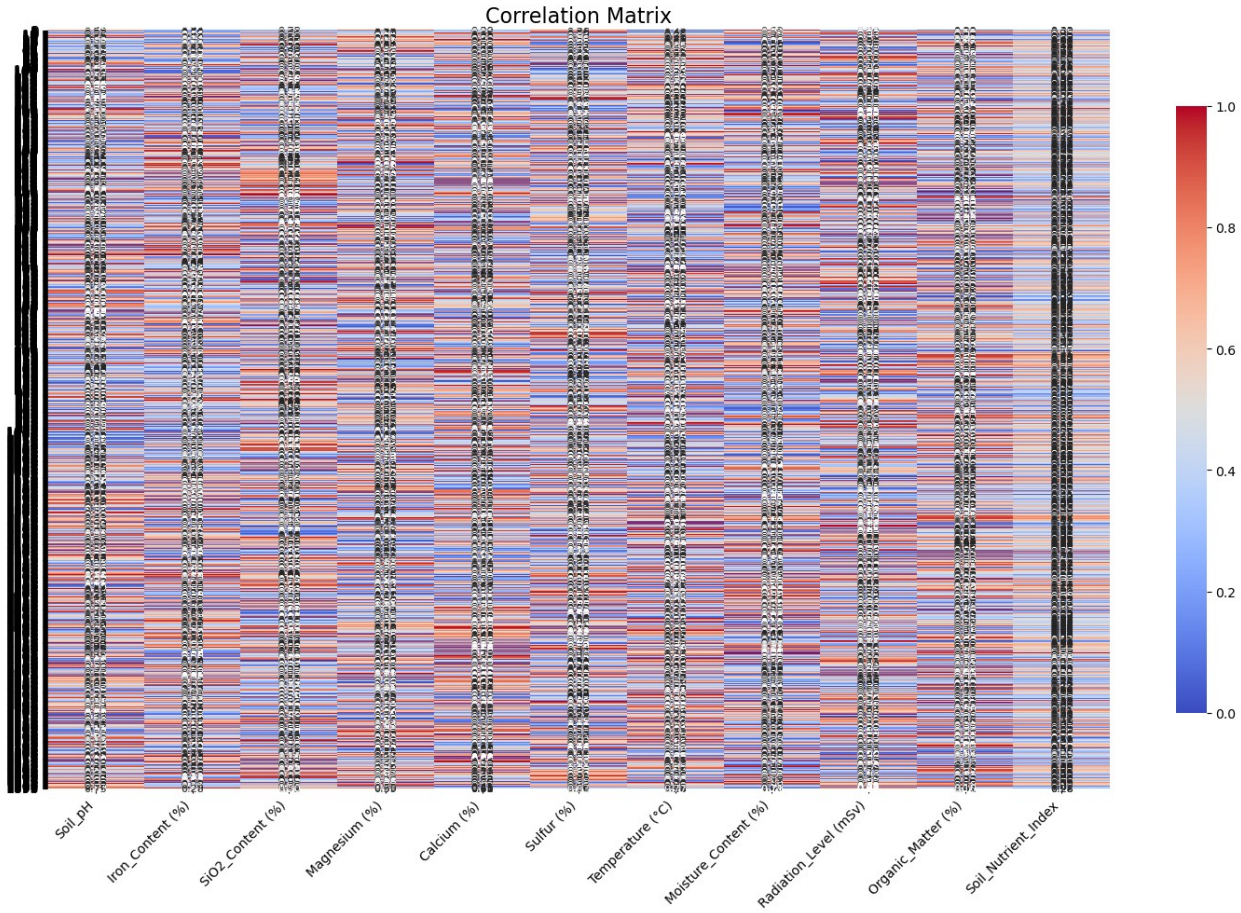



```
print("After Normalization:")
print(df_soil[columns_to_normalize].min())
print(df_soil[columns_to_normalize].max())

After Normalization:
Soil_pH                  0.0
Iron_Content (%)         0.0
SiO2_Content (%)         0.0
Magnesium (%)            0.0
Calcium (%)              0.0
Sulfur (%)               0.0
Temperature (°C)         0.0
Moisture_Content (%)     0.0
Radiation_Level (mSv)    0.0
Organic_Matter (%)       0.0
Soil_Nutrient_Index      0.0
dtype: float64
Soil_pH                  1.0
Iron_Content (%)         1.0
SiO2_Content (%)         1.0
Magnesium (%)            1.0
```

```
Calcium (%)                1.0
Sulfur (%)                 1.0
Temperature (°C)           1.0
Moisture_Content (%)       1.0
Radiation_Level (mSv)      1.0
Organic_Matter (%)         1.0
Soil_Nutrient_Index        1.0
dtype: float64
```

## Feature Selection

```
# I will check the correlation between certain columns so we can
decrease the number of columns

columns_to_correlate = ['Soil_pH', 'Iron_Content (%)', 'SiO2_Content
(%)', 'Magnesium (%)',
                        'Calcium (%)', 'Sulfur (%)', 'Temperature
(°C)', 'Moisture_Content (%)',
                        'Radiation_Level (mSv)', 'Organic_Matter (%)',
'Soil_Nutrient_Index']
correlation_list = df_soil[columns_to_correlate]

pt.figure(figsize=(15, 10))
sn.heatmap(correlation_list, annot=True, cmap='coolwarm', fmt=".2f",
annot_kws={'size': 8}, cbar_kws={'shrink': 0.8}, xticklabels=True,
yticklabels=True)

pt.xticks(rotation=45, ha='right', fontsize=10)
pt.yticks(rotation=0, fontsize=10)


pt.title('Correlation Matrix', fontsize=16)
pt.tight_layout()
pt.show()
```




## Adding a New Column for Data Integaration

```
df_soil['id'] = range(1,len(df_soil)+1)

df_soil.head()
```

```
    Soil_pH  Iron_Content (%)  SiO2_Content (%)  Magnesium (%)
Calcium (%)  \
0  0.687044          0.244899          0.281095       0.847981
0.284916
1  0.165483          0.118728          0.720779       0.173446
0.495157
2  0.168966          0.224603          0.274449       0.323458
0.425766
3  0.537510          0.702386          0.777000       0.918567
0.158127
4  0.570692          0.182149          0.647095       0.333815
0.321740

   Sulfur (%)  Temperature (°C)  Moisture_Content (%)  Radiation_Level
(mSv)  \
0    0.797927          0.052857              0.959771
0.303008
```

```
1    0.885350          0.071261              0.621997
0.117095
2    0.658213          0.417442              0.560188
0.865679
3    0.377830          0.216451              0.918267
0.930202
4    0.991111          0.020502              0.447617
0.269533

   Organic_Matter (%)  Soil_Nutrient_Index  Soil_pH_copy  id
0            0.082177             0.300582      8.186779   1
1            0.291176             0.421439      7.665686   2
2            0.634857             0.617722      7.669166   3
3            0.511201             0.595889      8.037380   4
4            0.917703             0.641490      8.070532   5
```

## 2- Dataset for Weather

```
weather = pd.read_csv('/home/ali/Downloads/archive (1) (1)/mars-
weather.csv')

weather.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1894 entries, 0 to 1893
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                1894 non-null   int64
 1   terrestrial_date  1894 non-null   object
 2   sol               1894 non-null   int64
 3   ls                1894 non-null   int64
 4   month             1894 non-null   object
 5   min_temp          1867 non-null   float64
 6   max_temp          1867 non-null   float64
 7   pressure          1867 non-null   float64
 8   wind_speed        0 non-null      float64
 9   atmo_opacity      1894 non-null   object
dtypes: float64(4), int64(3), object(3)
memory usage: 148.1+ KB

weather.describe()

                id          sol           ls     min_temp     max_temp
\
count  1894.000000  1894.000000  1894.000000  1867.000000  1867.000000

mean    948.372228  1007.930306   169.180570   -76.121050   -12.510445

std     547.088173   567.879561   105.738532     5.504098    10.699454
```

```
min       1.000000     1.000000     0.000000   -90.000000   -35.000000

25%     475.250000   532.250000    78.000000   -80.000000   -23.000000

50%     948.500000  1016.500000   160.000000   -76.000000   -11.000000

75%    1421.750000  1501.750000   259.000000   -72.000000    -3.000000

max    1895.000000  1977.000000   359.000000   -62.000000    11.000000
```

```
          pressure  wind_speed
count  1867.000000         0.0
mean    841.066417         NaN
std      54.253226         NaN
min     727.000000         NaN
25%     800.000000         NaN
50%     853.000000         NaN
75%     883.000000         NaN
max     925.000000         NaN
```

Checking the Null values in DF_Weather

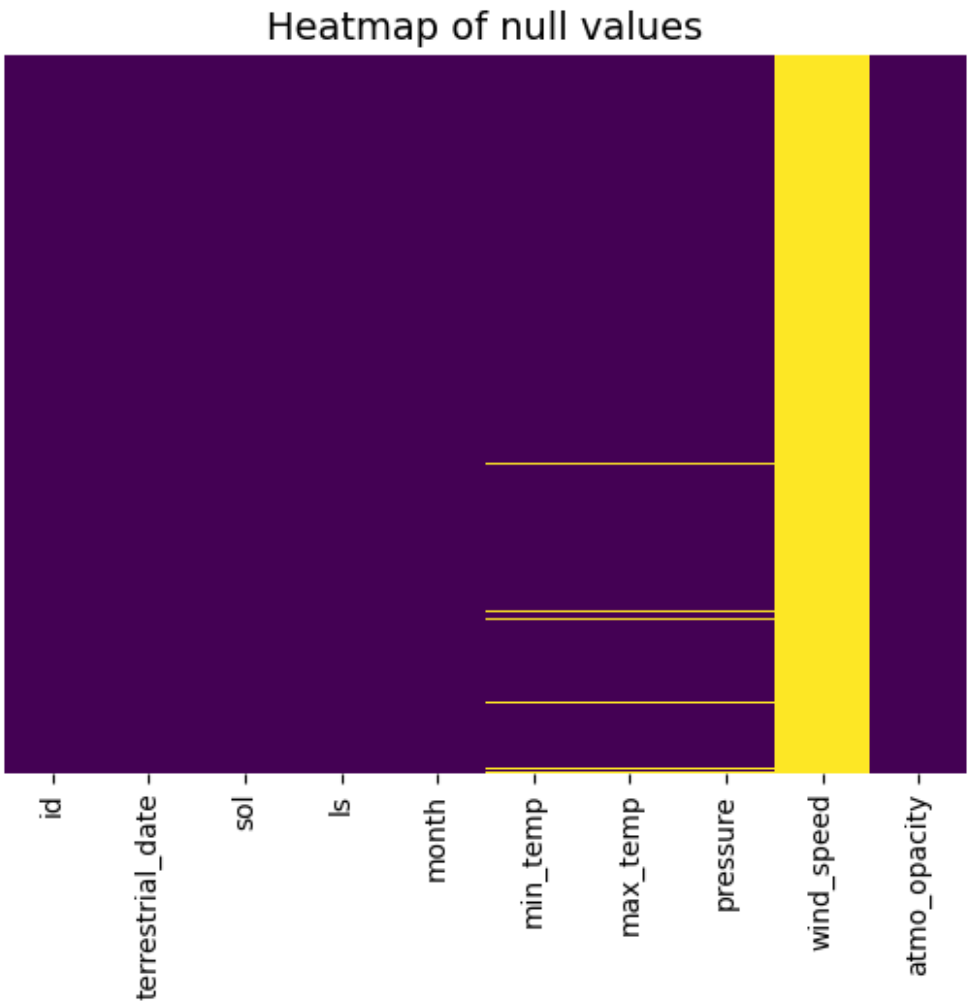
```
df_weather.isnull().sum()
```

```
id                    0
terrestrial_date      0
sol                   0
ls                    0
month                 0
min_temp             27
max_temp             27
pressure             27
wind_speed         1894
atmo_opacity          0
dtype: int64
```

```
# using a heatmap to show the null values

weather = pd.read_csv('/home/ali/Downloads/archive (1) (1)/mars-
weather.csv')

sn.heatmap(weather.isnull(),yticklabels=False,cbar=False,cmap='viridis
')
plt.title('Heatmap of null values',fontsize=14)
```

```
<IPython.core.display.Javascript object>
```

```
Text(0.5, 1.0, 'Heatmap of null values')
```




Droping Wind Speed from Weather

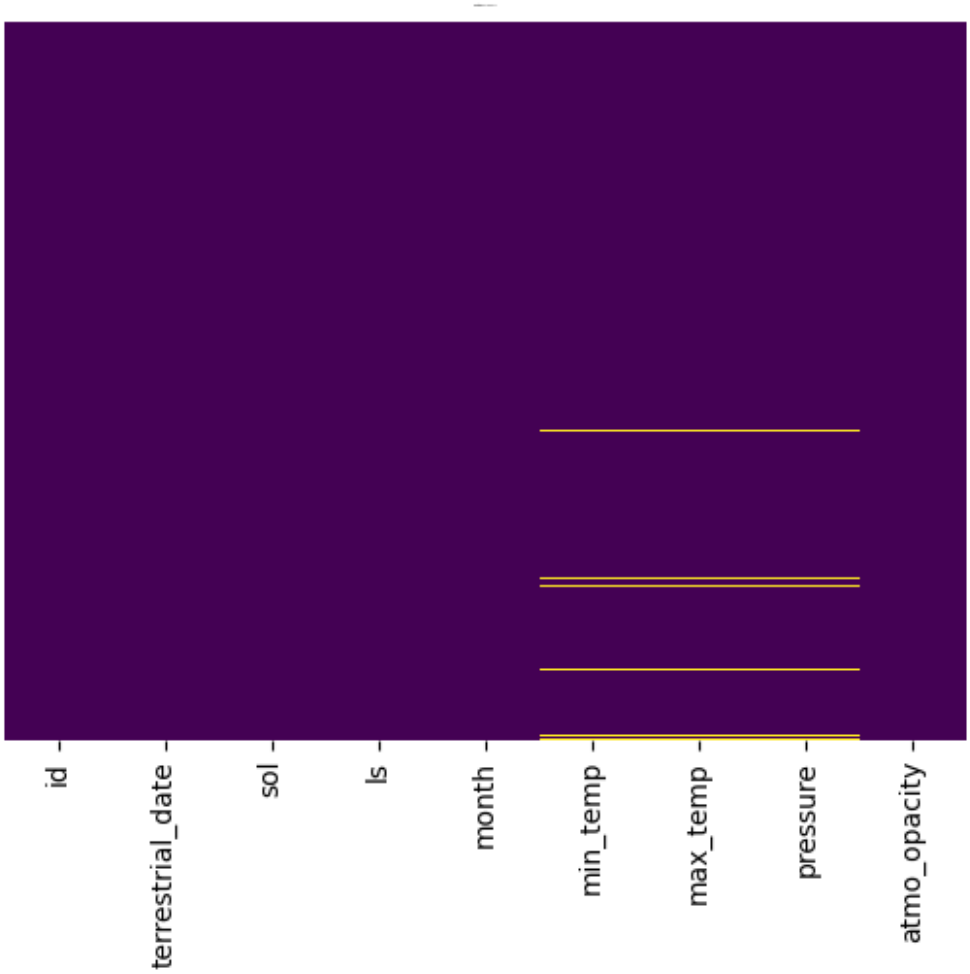
```
# As we can see the column = wind speed is completly empty so we
should drop this column.

def droping_wind_speed(weather):
    if 'wind_speed' in weather.columns:
        weather.drop(columns=['wind_speed'], inplace=True)
        print("Yes the wind speed column has been droped
successfully.")
    else:
        print("No there's an issue while droping the column.")

    return weather

weather = droping_wind_speed(weather)
```

```
Yes the wind speed column has been droped successfully.
```

```
sn.heatmap(weather.isnull(),yticklabels=False,cbar=False,cmap='viridis
')
plt.title('Heatmap for Null',fontsize=True)
```

```
<IPython.core.display.Javascript object>
```

```
Text(0.5, 1.0, 'Heatmap for Null')
```




Refilling the Min_Temp, Max_Temp and Pressure Columns with mode values

```
def refill(weather):
    columns = ['min_temp','max_temp','pressure']
    try:
        for col in columns:
            mean_value = weather[col].mean()
            weather[col].fillna(mean_value, inplace=True)
        print("status {}".format(1))
    except Exception as ex:
        print(f"Issue is: {ex}")
```

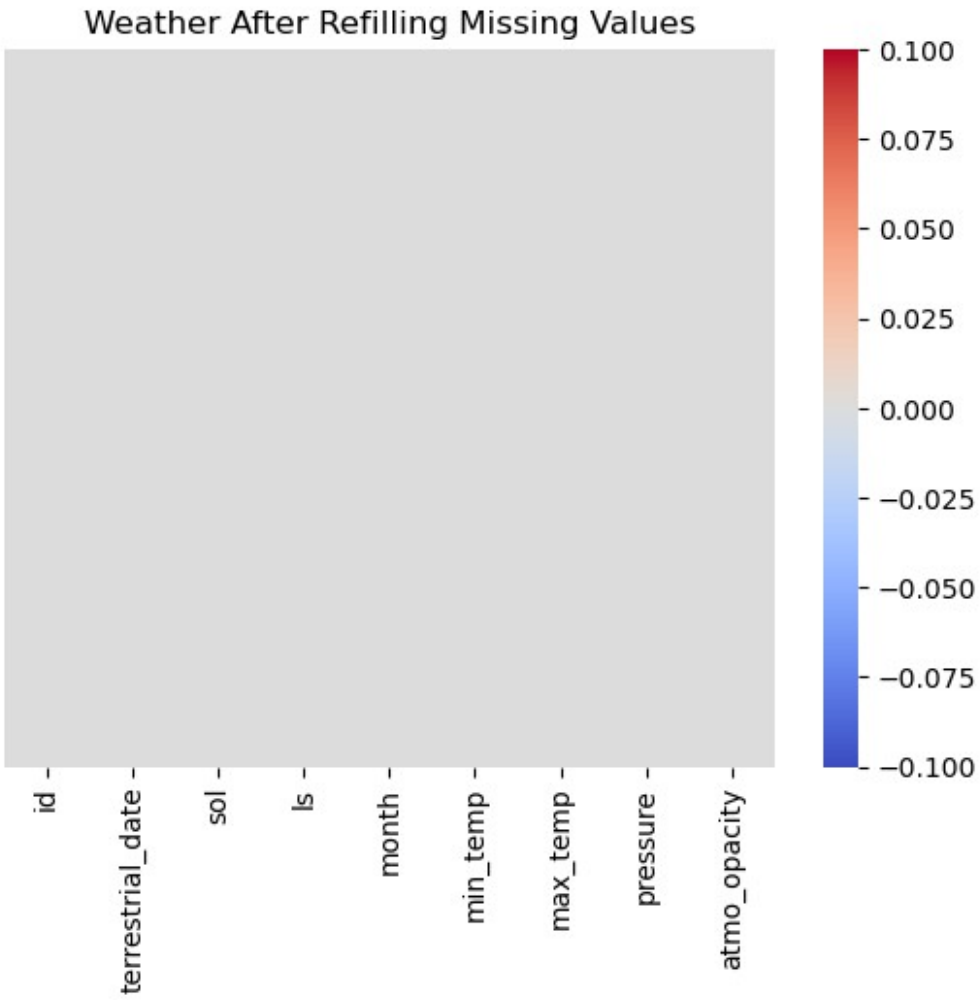
```
    return weather

weather = refill(weather)
```

```
status 1
```

```
/tmp/ipykernel_6179/1801965515.py:6: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  weather[col].fillna(mean_value, inplace=True)
```

```
# After Refilling the NaN values

sn.heatmap(weather.isnull(),yticklabels=False,cbar=True,cmap='coolwarm
')
pt.title('Weather After Refilling Missing Values')
```

```
Text(0.5, 1.0, 'Weather After Refilling Missing Values')
```




# Checking the Duplicate values in DF_Weather

```
weather.duplicated().sum()
```

```
0
```

```
# we can see there are no duplicate values in weather dataset
```

```
weather.head()
```

```
     id terrestrial_date   sol   ls     month  min_temp  max_temp
pressure  \
0  1895       2018-02-27  1977  135  Month 5     -77.0     -10.0
727.0
1  1893       2018-02-26  1976  135  Month 5     -77.0     -10.0
728.0
2  1894       2018-02-25  1975  134  Month 5     -76.0     -16.0
729.0
```

```
3  1892       2018-02-24  1974  134  Month 5     -77.0     -13.0
729.0
4  1889       2018-02-23  1973  133  Month 5     -78.0     -18.0
730.0

  atmo_opacity
0        Sunny
1        Sunny
2        Sunny
3        Sunny
4        Sunny
```

## Fixing Inconsistencies

```
def removing_inconsistencies(weather):
    # fixing date
    weather['terrestrial_date'] =
pd.to_datetime(weather['terrestrial_date'])
    # removing "month" from month column
    weather['month'] =
weather['month'].str.replace('Month','',regex=False)
    # chaning atmo_opacity to numerical value
    weather_map = {'Sunny':1}
    weather['atmo_opacity'] = weather['atmo_opacity'].map(weather_map)
    return weather


weather = removing_inconsistencies(weather)

weather.head()
```

```
     id terrestrial_date   sol   ls month  min_temp  max_temp
pressure  \
0  1895       2018-02-27  1977  135     5     -77.0     -10.0
727.0
1  1893       2018-02-26  1976  135     5     -77.0     -10.0
728.0
2  1894       2018-02-25  1975  134     5     -76.0     -16.0
729.0
3  1892       2018-02-24  1974  134     5     -77.0     -13.0
729.0
4  1889       2018-02-23  1973  133     5     -78.0     -18.0
730.0

   atmo_opacity
0           1.0
1           1.0
2           1.0
3           1.0
4           1.0
```

## Normalization

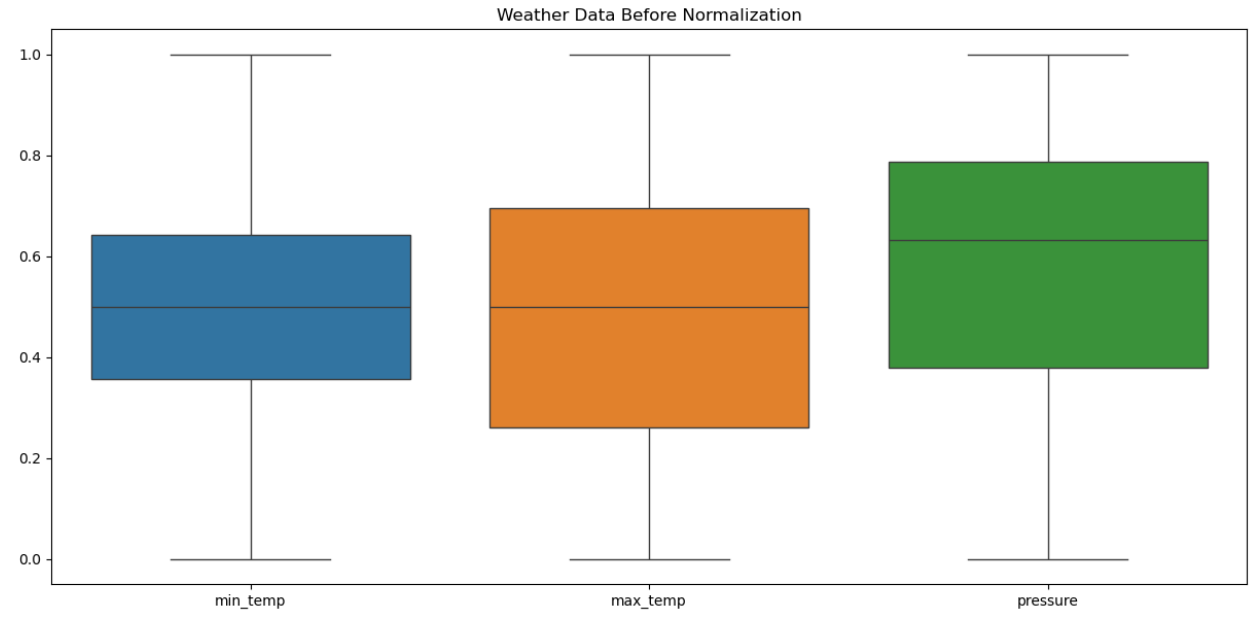
Scaling all the features on a consistent Scale

```
weather_columns_to_normalize = ['min_temp', 'max_temp', 'pressure']

print("Before Normalization:")
print(weather[weather_columns_to_normalize].min())
print(weather[weather_columns_to_normalize].max())
```

```
Before Normalization:
min_temp    -90.0
max_temp    -35.0
pressure    727.0
dtype: float64
min_temp    -62.0
max_temp     11.0
pressure    925.0
dtype: float64
```

```
pt.figure(figsize=(15,7))
sn.boxplot(data=weather[weather_columns_to_normalize])
pt.title('Weather Data Before Normalization')
```

```
Text(0.5, 1.0, 'Weather Data Before Normalization')
```



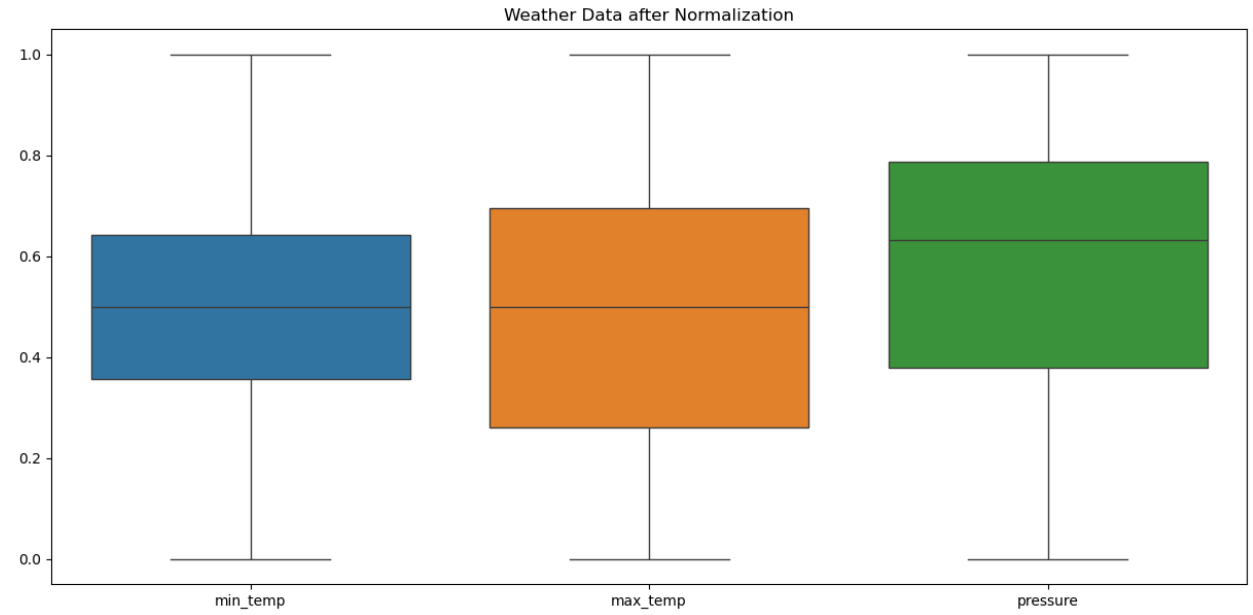

```
scaler = MinMaxScaler()

weather[weather_columns_to_normalize] =
scaler.fit_transform(weather[weather_columns_to_normalize])
```

```
print("After Normalization:")
print(weather[weather_columns_to_normalize].min())
print(weather[weather_columns_to_normalize].max())
```

```
After Normalization:
min_temp    0.0
max_temp    0.0
pressure    0.0
dtype: float64
min_temp    1.0
max_temp    1.0
pressure    1.0
dtype: float64
```

```
pt.figure(figsize=(15,7))
sn.boxplot(data=weather[weather_columns_to_normalize])
pt.title('Weather Data after Normalization')
```

```
Text(0.5, 1.0, 'Weather Data after Normalization')
```




## Feature Selection

Removing all the un-neccassary Features

id -> no need for that

terrestrial_date -> we are not performing time based analysis so there is no need for this column

sol -> this is representing the martial days no need for this

ls (lansing solar) -> for monitoring season as on the Earth so no need for this also

```
print(weather.columns)
```

```
Index(['month', 'min_temp', 'max_temp', 'pressure'], dtype='object')
```

```
def droping_columns(weather):
    cols_to_drop = ['ls','sol','atmo_opacity','terrestrial_date']
    weather.drop(columns=cols_to_drop,axis=1,inplace=True)
    return weather

weather.head()
```

```
  month  min_temp  max_temp  pressure
0     5  0.464286  0.543478  0.000000
1     5  0.464286  0.543478  0.005051
2     5  0.500000  0.413043  0.010101
3     5  0.464286  0.478261  0.010101
4     5  0.428571  0.369565  0.015152
```

```
weather['id'] = range(1,len(weather)+1)

print(weather.head())
```

```
  month  min_temp  max_temp  pressure  id
0     5  0.464286  0.543478  0.000000   1
1     5  0.464286  0.543478  0.005051   2
2     5  0.500000  0.413043  0.010101   3
3     5  0.464286  0.478261  0.010101   4
4     5  0.428571  0.369565  0.015152   5
```

## Data Integaration

```
data = pd.merge(weather,df_soil, on='id',how='inner')

data.head()
```

```
  month  min_temp  max_temp  pressure  id   Soil_pH  Iron_Content (%)
\
0     5  0.464286  0.543478  0.000000   1  0.687044          0.244899

1     5  0.464286  0.543478  0.005051   2  0.165483          0.118728

2     5  0.500000  0.413043  0.010101   3  0.168966          0.224603

3     5  0.464286  0.478261  0.010101   4  0.537510          0.702386

4     5  0.428571  0.369565  0.015152   5  0.570692          0.182149


   SiO2_Content (%)  Magnesium (%)  Calcium (%)  Sulfur (%)
Temperature (°C)  \
0          0.281095       0.847981     0.284916    0.797927
0.052857
```

```
1          0.720779       0.173446     0.495157    0.885350
0.071261
2          0.274449       0.323458     0.425766    0.658213
0.417442
3          0.777000       0.918567     0.158127    0.377830
0.216451
4          0.647095       0.333815     0.321740    0.991111
0.020502

   Moisture_Content (%)  Radiation_Level (mSv)  Organic_Matter (%)  \
0              0.959771               0.303008            0.082177
1              0.621997               0.117095            0.291176
2              0.560188               0.865679            0.634857
3              0.918267               0.930202            0.511201
4              0.447617               0.269533            0.917703

   Soil_Nutrient_Index  Soil_pH_copy
0             0.300582      8.186779
1             0.421439      7.665686
2             0.617722      7.669166
3             0.595889      8.037380
4             0.641490      8.070532
```