1. INTRODUCTION

Objective of this Notebook:

This notebook aims to:

- Easy and ***Begginers guide***.
- Analyse Each and Every ***Attributes*** in the data set.
- Build Various ***ML Models*** with the view of ***increasing accuracy*** of the Model.

The ***Machine learning Models used*** are:

```
1.K-Nearest Neighbour(KNN)

2.Support Vector Machine(SVM)

3.Gradient Boost

4.Extreme Gradient Boosting(XGBC)

# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd# data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/weather-prediction/seattle-weather.csv
```

2.IMPORTING THE REQUIRED LIBRARIES

```
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
import re
import missingno as mso
from scipy import stats
from scipy.stats import ttest_ind
from scipy.stats import pearsonr
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
```

3.ANALYSING THE DATASET

There are **6 Variables** in this Dataset:

- **4 Continuous** Variables.
- **1 Variable** to accommodate the Date.
- **1 Variable** refers the Weather.

```
data=pd.read_csv("/kaggle/input/weather-prediction/seattle-weather.csv")
data.head()
```

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

```
data.shape
```

```
(1461, 6)
```

As of it has **6 Columns** with total of **1461 Rows** as our observations in the Data set.

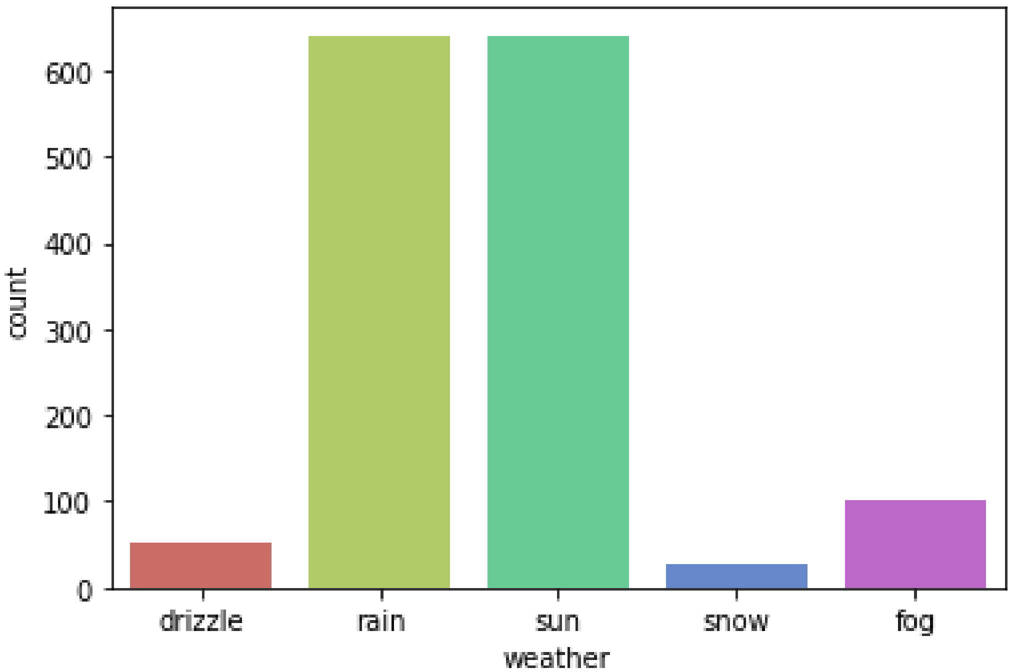
4.DATA EXPLORATION

It is the process of Exploring the data from the **"RAW"** data set tha we have taken or Imported.

First let us Deal with the Categorical variables

```
import warnings
warnings.filterwarnings('ignore')
sns.countplot("weather",data=data,palette="hls")
```

```
<AxesSubplot:xlabel='weather', ylabel='count'>
```




```
countrain=len(data[data.weather=="rain"])
countsun=len(data[data.weather=="sun"])
countdrizzle=len(data[data.weather=="drizzle"])
countsnow=len(data[data.weather=="snow"])
countfog=len(data[data.weather=="fog"])
print("Percent of Rain:{:2f}
%".format((countrain/(len(data.weather))*100)))
print("Percent of Sun:{:2f}
%".format((countsun/(len(data.weather))*100)))
print("Percent of Drizzle:{:2f}
%".format((countdrizzle/(len(data.weather))*100)))
print("Percent of Snow:{:2f}
%".format((countsnow/(len(data.weather))*100)))
print("Percent of Fog:{:2f}
%".format((countfog/(len(data.weather))*100)))
```

```
Percent of Rain:43.874059%
Percent of Sun:43.805613%
Percent of Drizzle:3.627652%
Percent of Snow:1.779603%
Percent of Fog:6.913073%
```

From the Above countplot the data set contains higher amount of data with the weather datail of ***Rain and Sun*** and it also have some additionals like ***drizzle,snow and fog*** .

5.NUMERICAL OR CONTINUOUS VARIABLES

Next we will explore the ***Continuous variables***

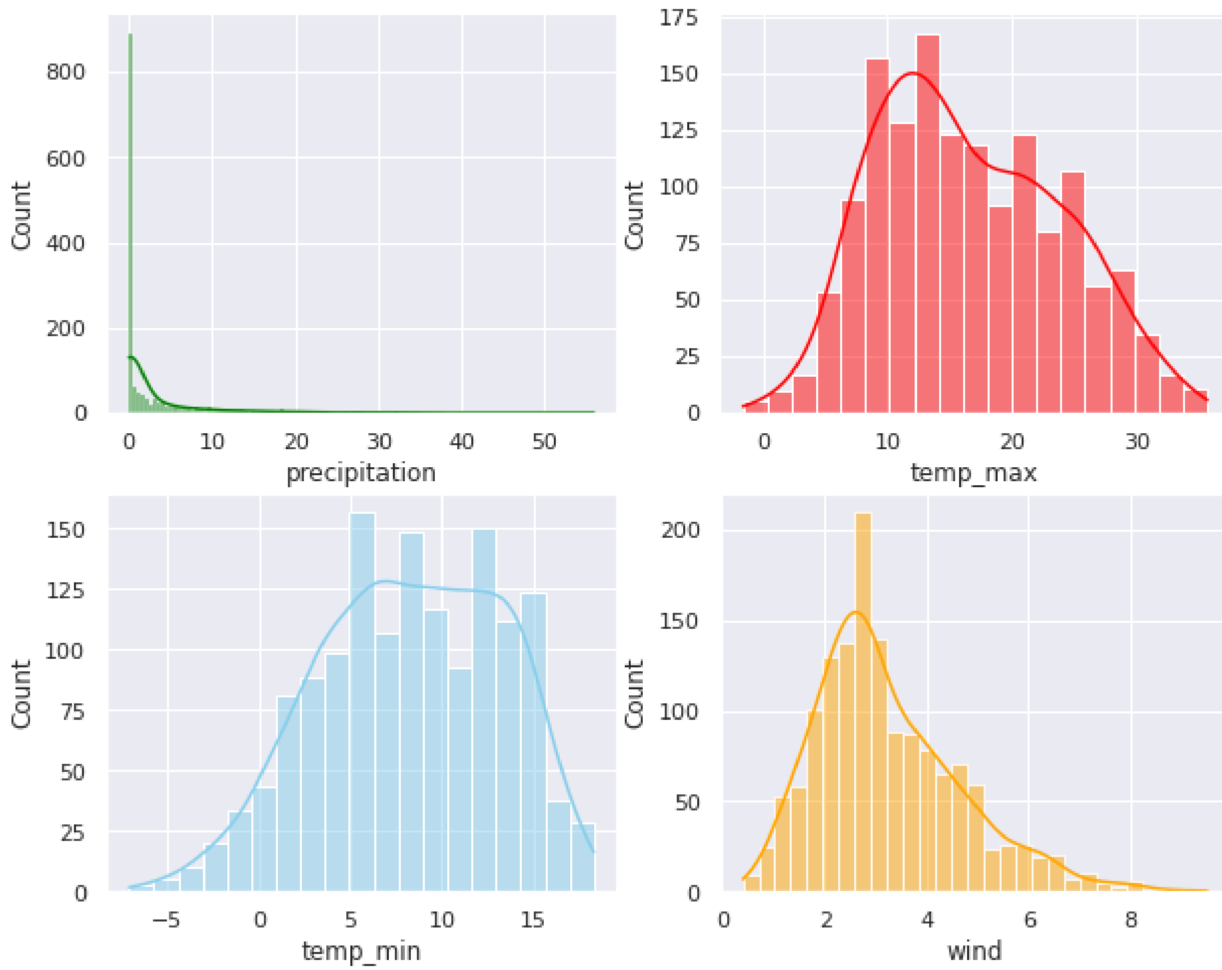
```
data[["precipitation","temp_max","temp_min","wind"]].describe()
```

	precipitation	temp_max	temp_min	wind
count	1461.000000	1461.000000	1461.000000	1461.000000
mean	3.029432	16.439083	8.234771	3.241136
std	6.680194	7.349758	5.023004	1.437825
min	0.000000	-1.600000	-7.100000	0.400000
25%	0.000000	10.600000	4.400000	2.200000
50%	0.000000	15.600000	8.300000	3.000000
75%	2.800000	22.200000	12.200000	4.000000
max	55.900000	35.600000	18.300000	9.500000

Distribution of numerical value using ***Histogram and Violin plot .***

```
sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=data,x="precipitation",kde=True,ax=axs[0,0],color='green')
sns.histplot(data=data,x="temp_max",kde=True,ax=axs[0,1],color='red')
sns.histplot(data=data,x="temp_min",kde=True,ax=axs[1,0],color='skyblue')
sns.histplot(data=data,x="wind",kde=True,ax=axs[1,1],color='orange')
```

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



From the above distribution it is clear that **precipitation and wind** are **Positively skewed.**

And **temp_min** is **Negatively skewed** and both has some ***outliers.***

6.**HOW TO FIND THE OUTILERS OR SKEW IN DATA SET?**

- ***We can find the outliers in the dataset by using following plots:***

  1.**Hist plot**

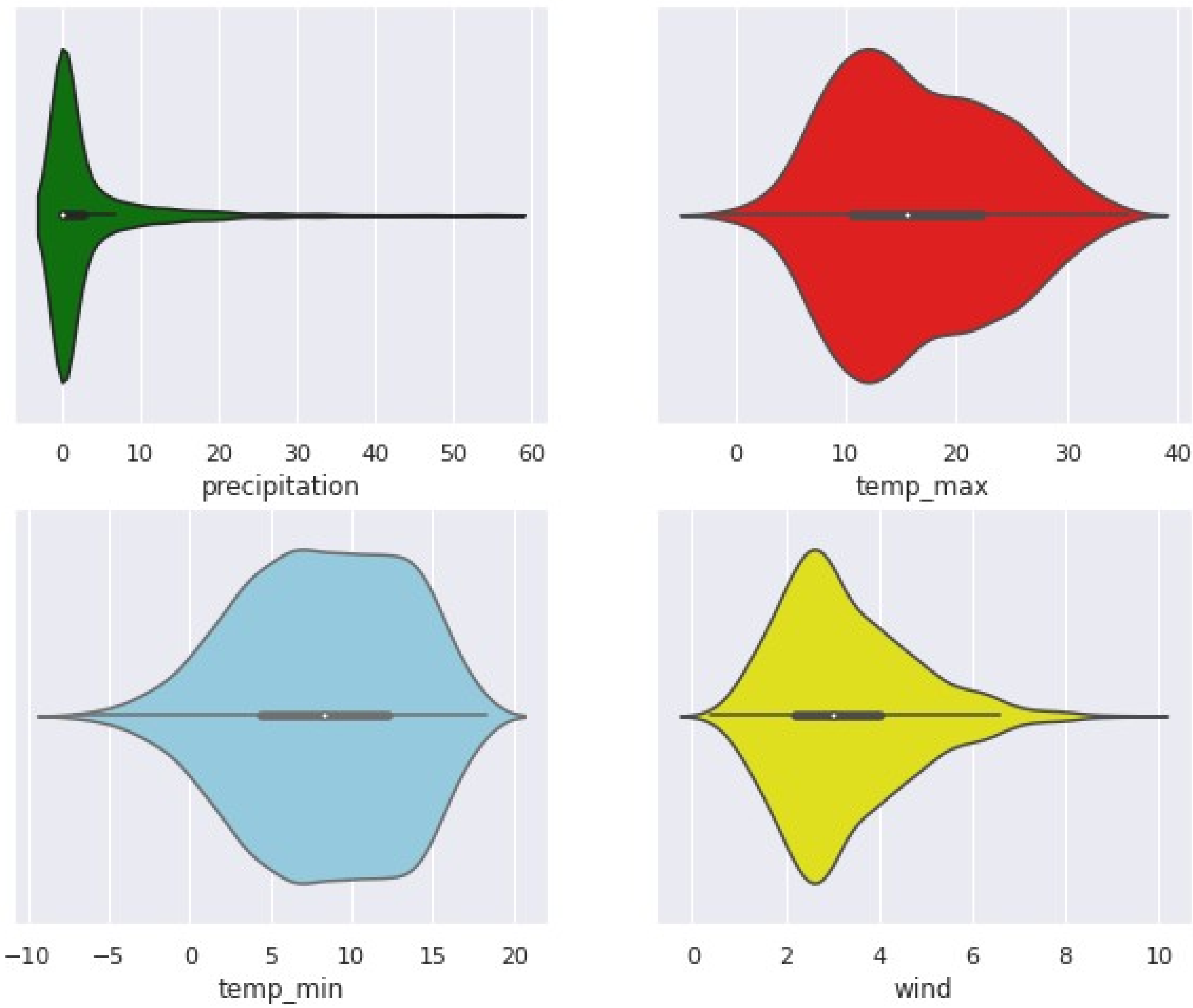
  2.**Box plot**

  3.**Violin plot**

  4.**Dist plot** yet both ***box and violin plots*** are easier to handel with.

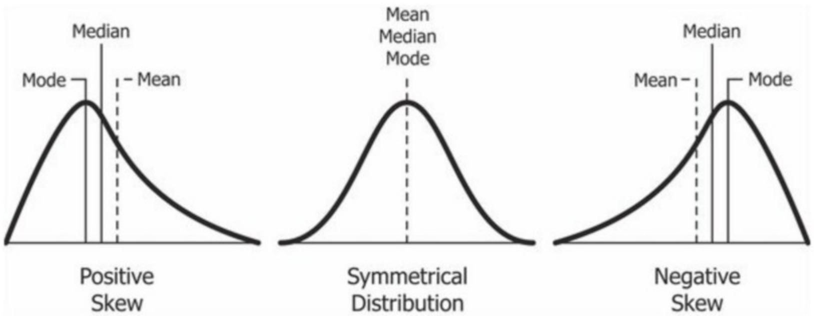
6.1. ***VIOLIN PLOT***

```
sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.violinplot(data=data,x="precipitation",kde=True,ax=axs[0,0],color=
'green')
sns.violinplot(data=data,x="temp_max",kde=True,ax=axs[0,1],color='red'
)
sns.violinplot(data=data,x="temp_min",kde=True,ax=axs[1,0],color='skyb
lue')
sns.violinplot(data=data,x="wind",kde=True,ax=axs[1,1],color='yellow')

<AxesSubplot:xlabel='wind'>
```



From the above ***Violin plot*** we can clearly understand the Skewness of the Data as the **TAIL** indicates the skewness.

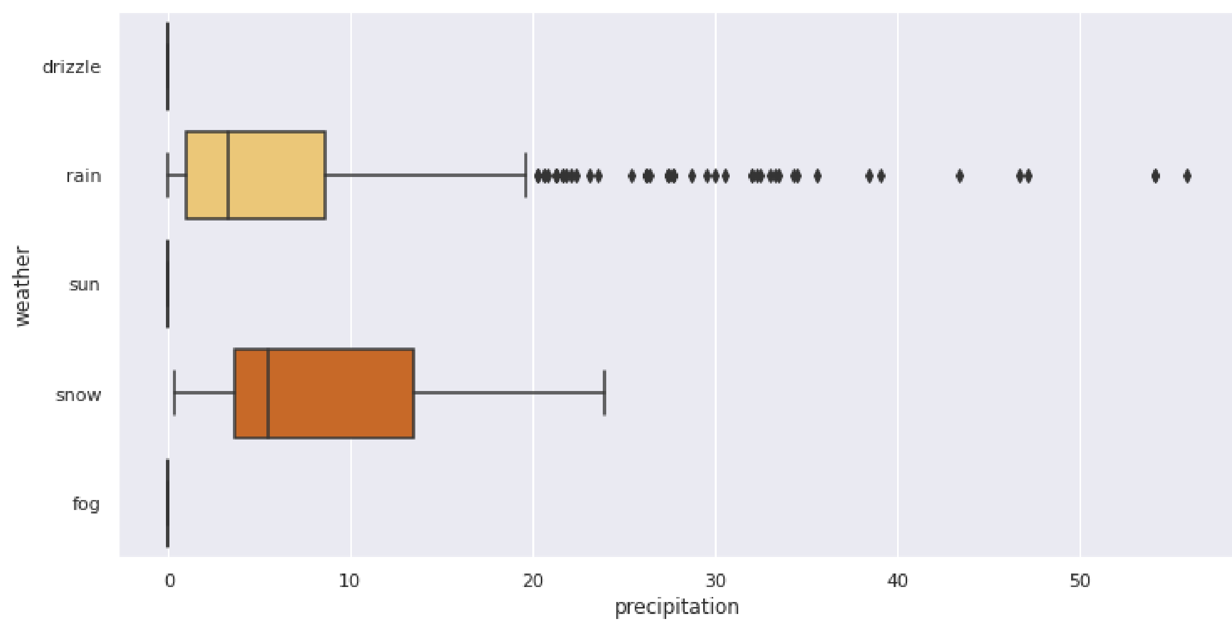
6.2. *BELOW DIAGRAM SHOWS THE EXACT OF HOW THE* **SKEWNESS LOOKS:**



6.3. ***SKEWNESS USING BOXPLOT***

OTHER EXPLORATION

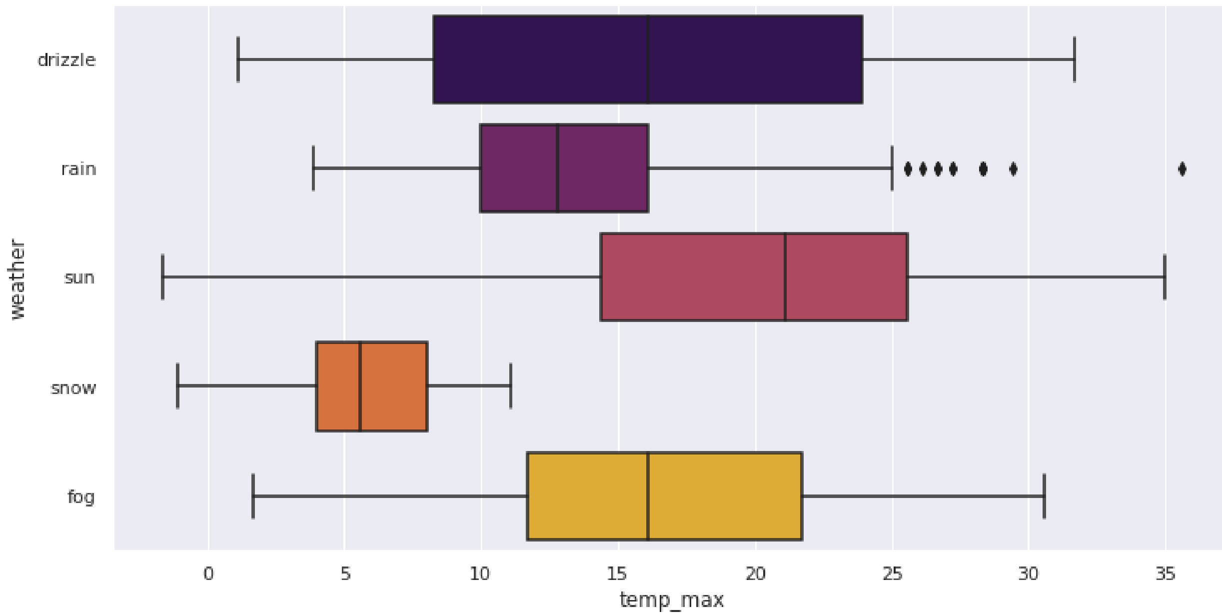
```
plt.figure(figsize=(12,6))
sns.boxplot("precipitation","weather",data=data,palette="YlOrBr")

<AxesSubplot:xlabel='precipitation', ylabel='weather'>
```



From the above box plot between the ***Weather and Precipitation*** the value ***Rain*** has many **positive outliers** and both **Rain and Snow** were ***positively skewed/has positive skewness***.

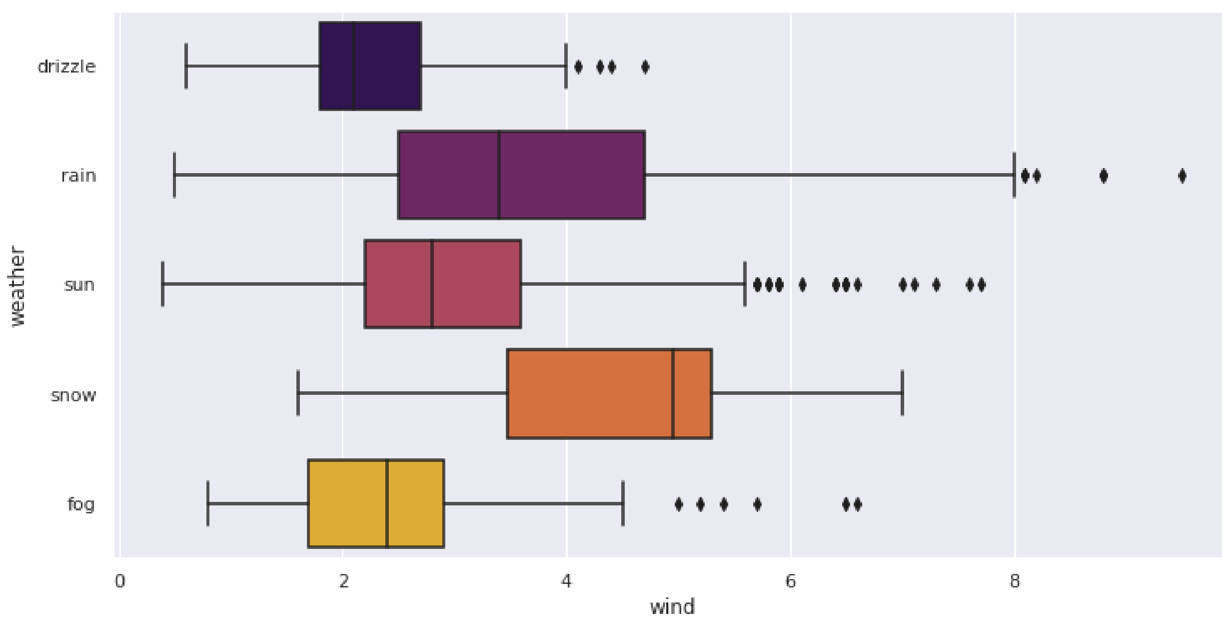
```
plt.figure(figsize=(12,6))
sns.boxplot("temp_max","weather",data=data,palette="inferno")
```

```
<AxesSubplot:xlabel='temp_max', ylabel='weather'>
```




```
plt.figure(figsize=(12,6))
sns.boxplot("wind","weather",data=data,palette="inferno")
```

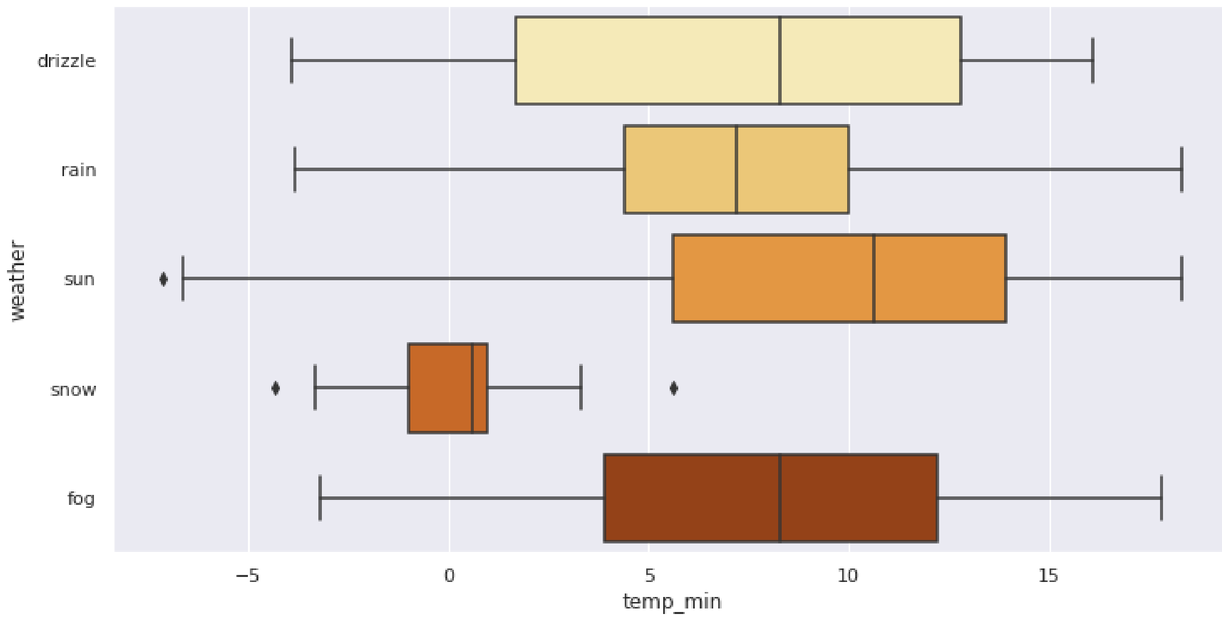
```
<AxesSubplot:xlabel='wind', ylabel='weather'>
```



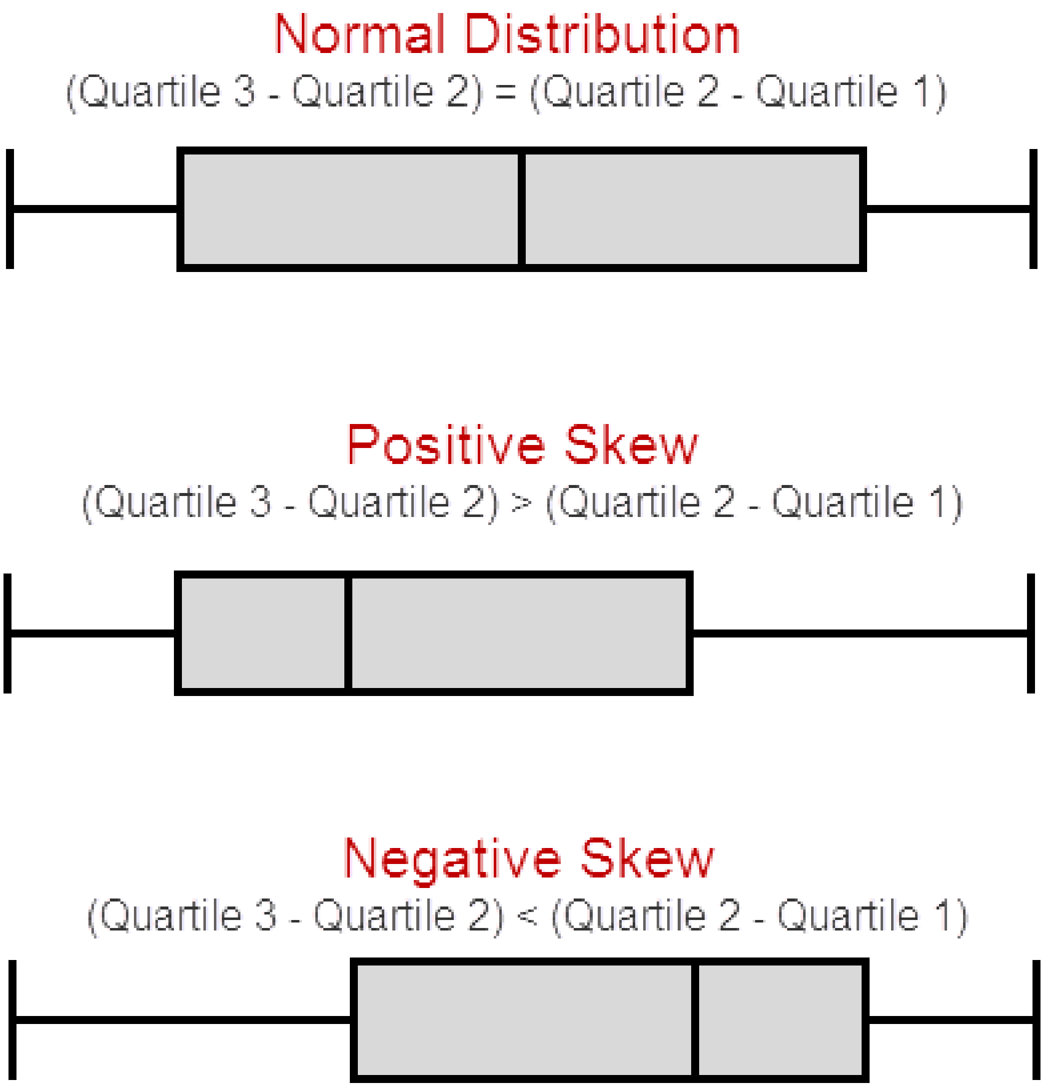
From the above box plots ,we came to know that Every ***attribute of weather*** has some ***positive outliers*** *and it is* **both types of skewness***.

```
plt.figure(figsize=(12,6))
sns.boxplot("temp_min","weather",data=data,palette="YlOrBr")
```

```
<AxesSubplot:xlabel='temp_min', ylabel='weather'>
```

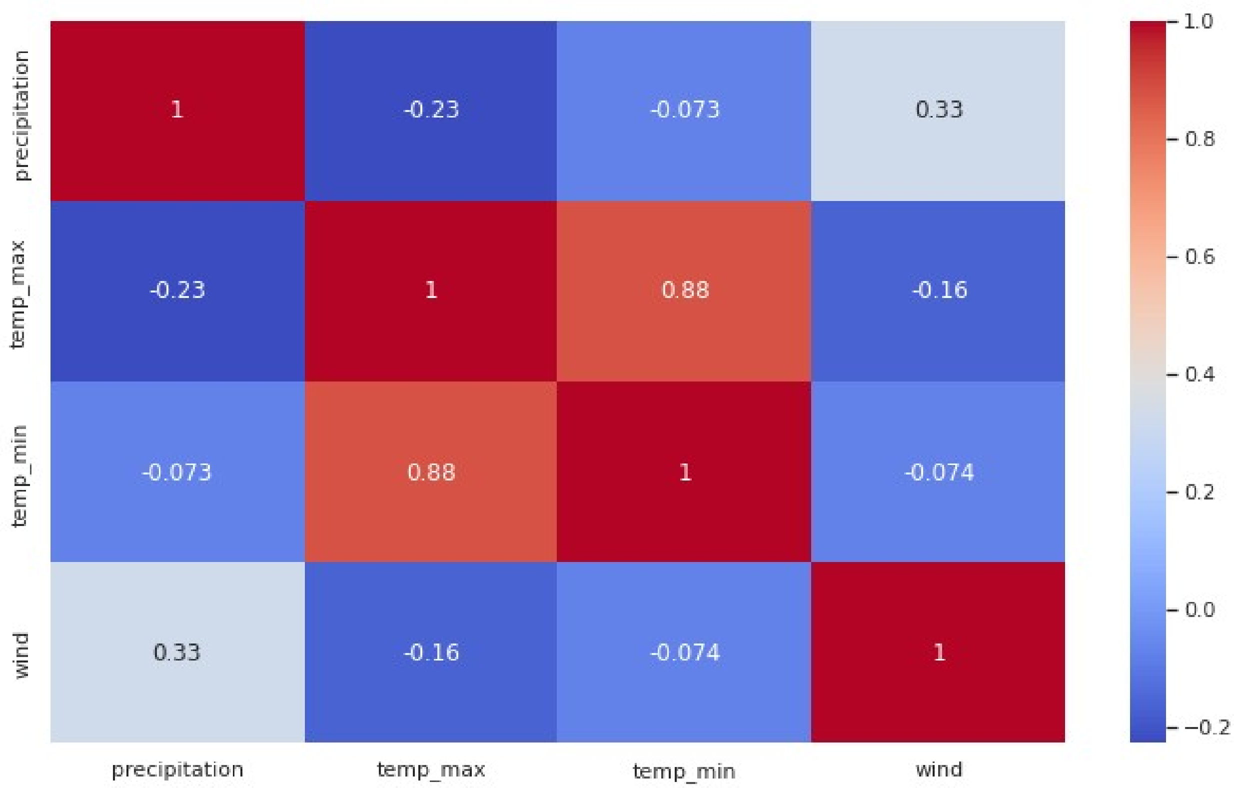


here some data has ***negative*** and some have both ***positive and negative*** outliers and ***snow is negatively skewed.*** SKEWNESS AND ITS CORRECTIONS:***



HEATMAP:

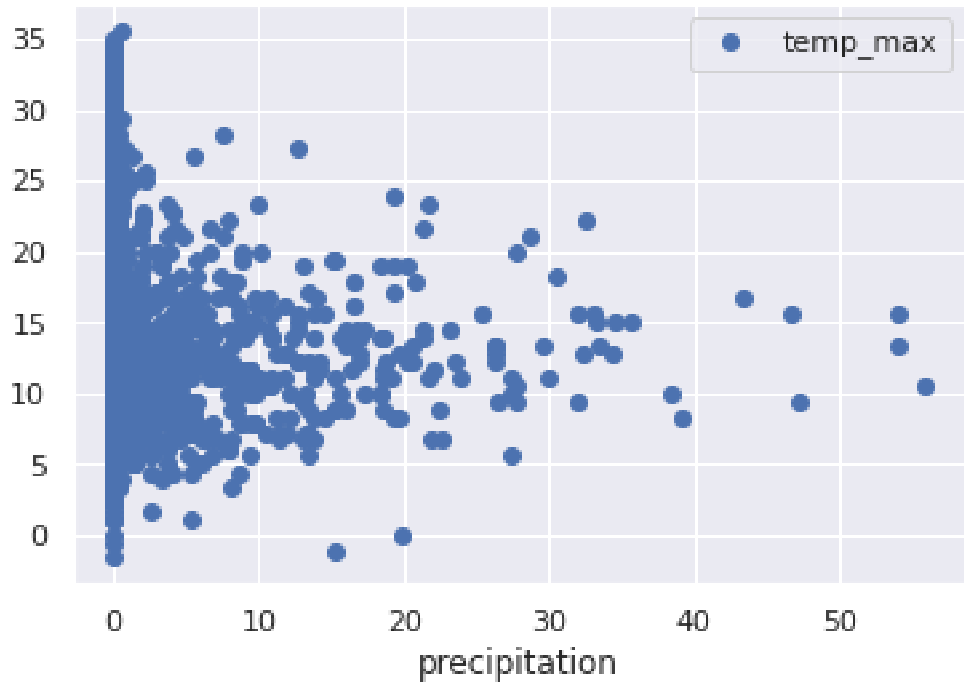
```
plt.figure(figsize=(12,7))
sns.heatmap(data.corr(),annot=True,cmap='coolwarm')

<AxesSubplot:>
```



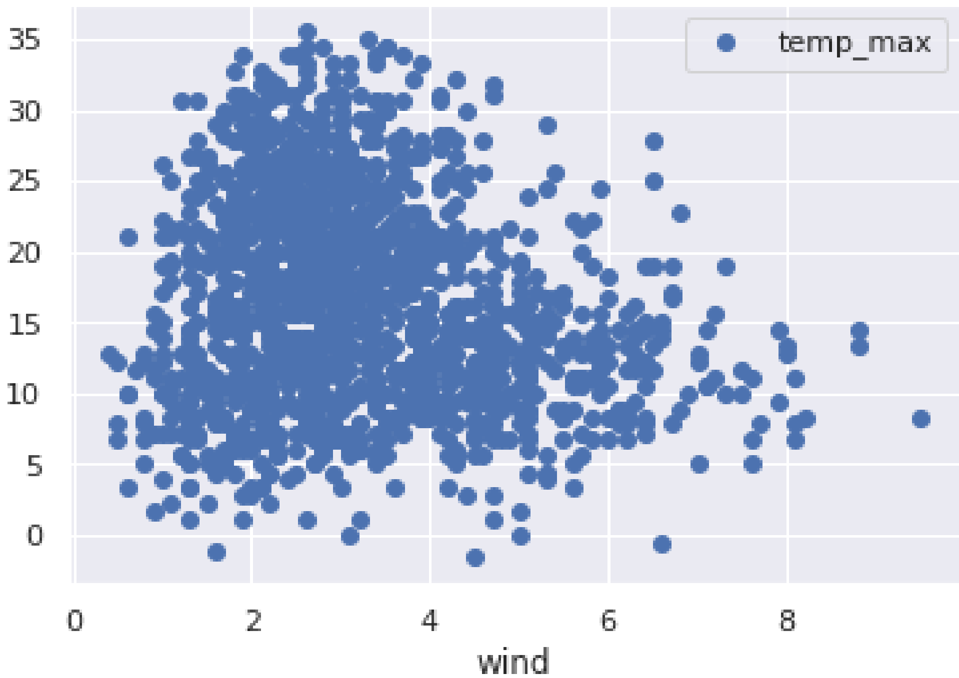
There is a ***positive correlation*** between ***temp_max and temp_min***.

Numerical - Numerical

```
data.plot("precipitation","temp_max",style='o')
print("Pearson
correlation:",data["precipitation"].corr(data["temp_max"]))
print("T Test and P
value:",stats.ttest_ind(data["precipitation"],data["temp_max"]))

Pearson correlation: -0.22855481643297046
T Test and P value: Ttest_indResult(statistic=-51.60685279531918,
pvalue=0.0)
```



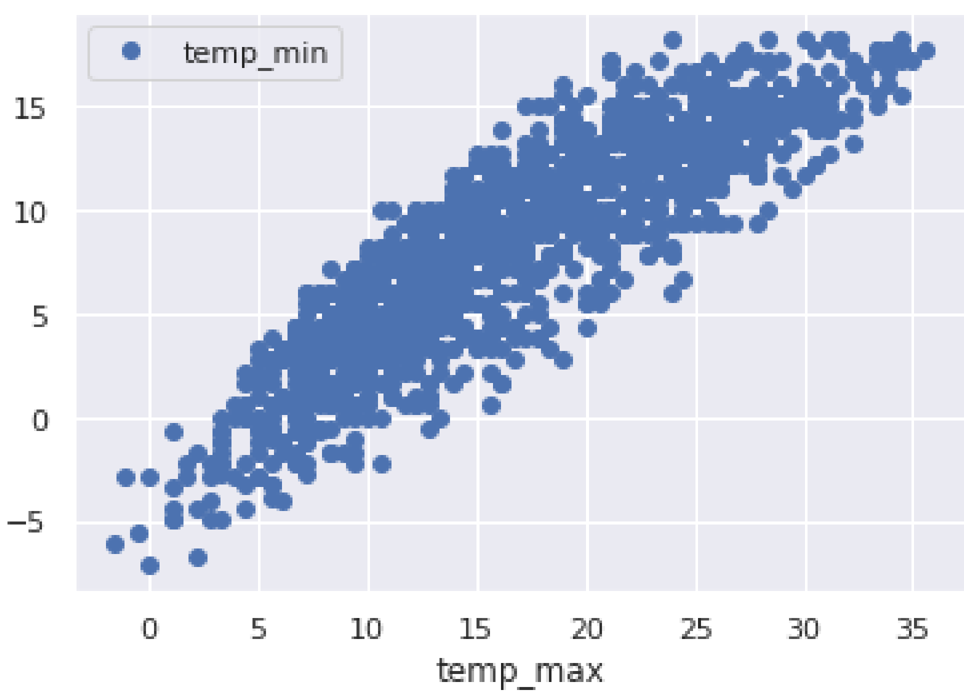

```
data.plot("wind","temp_max",style='o')
print("Pearson correlation:",data["wind"].corr(data["temp_max"]))
print("T Test and P
value:",stats.ttest_ind(data["wind"],data["temp_max"]))

Pearson correlation: -0.16485663487495486
T Test and P value: Ttest_indResult(statistic=-67.3601643301846,
pvalue=0.0)
```



As from the above result of ***T test and P value of 0*** indicates that the ***Null hypothesis*** in the corresponding columns is **rejected** and the columns are ***Statistically significant***

```
data.plot("temp_max","temp_min",style='o')
```

```
<AxesSubplot:xlabel='temp_max'>
```



7.NULL VALUES:

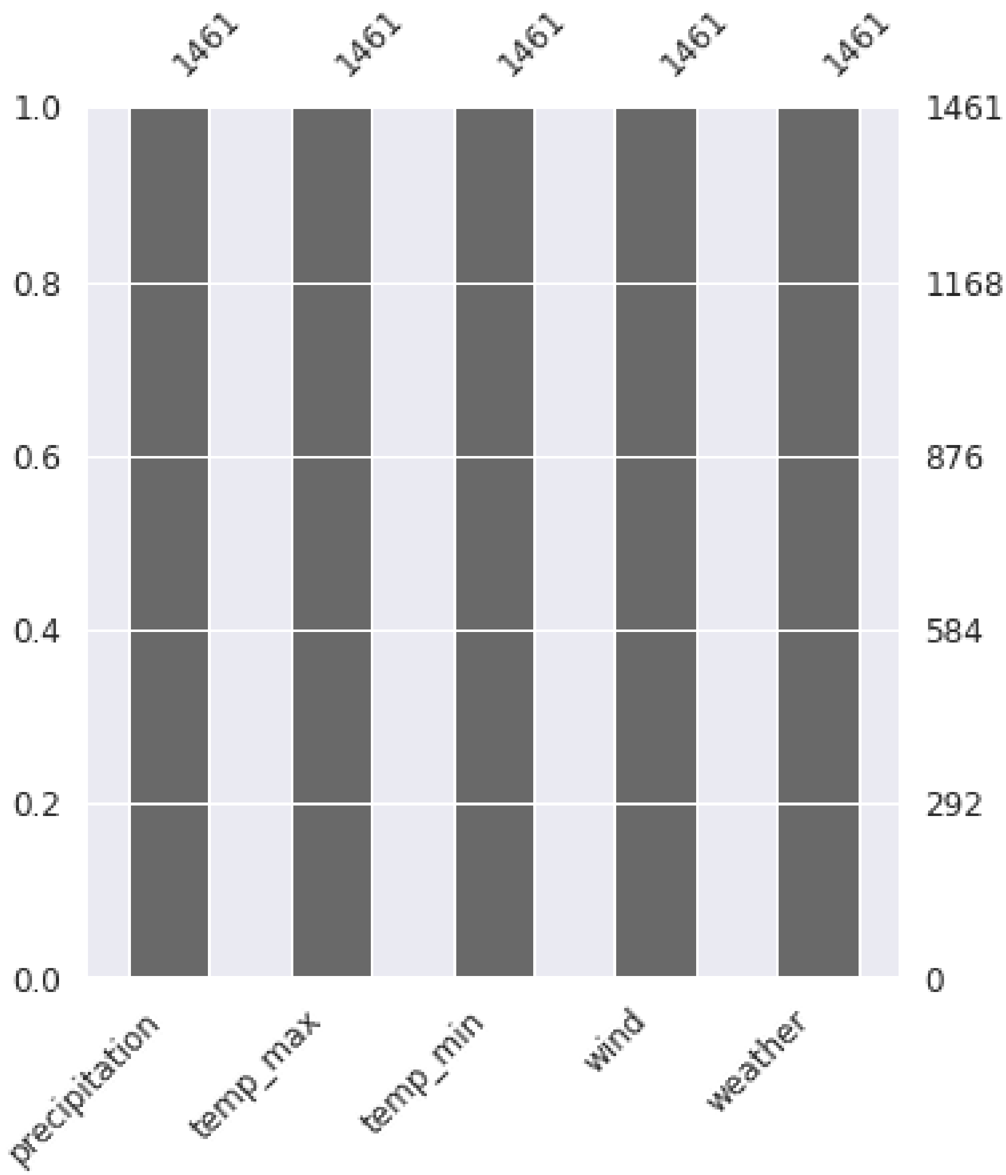
```
data.isna().sum()
```

```
date             0
precipitation    0
temp_max         0
temp_min         0
wind             0
weather          0
dtype: int64
```

Checking for Null values in the data set

The below plot shows that all the columns in the data set ***doesn't contains Null values*** as each columns contains a ***total of 1461*** observations.

```
plt.figure(figsize=(12,6))
axz=plt.subplot(1,2,2)
mso.bar(data.drop(["date"],axis=1),ax=axz,fontsize=12);
```



8.DATA PREPROCESSING:

Drop Unnecessary Variables

In this data set Date is a unnecessary variable as it does not affect the data so it can be dropped.

```
df=data.drop(["date"],axis=1)
```

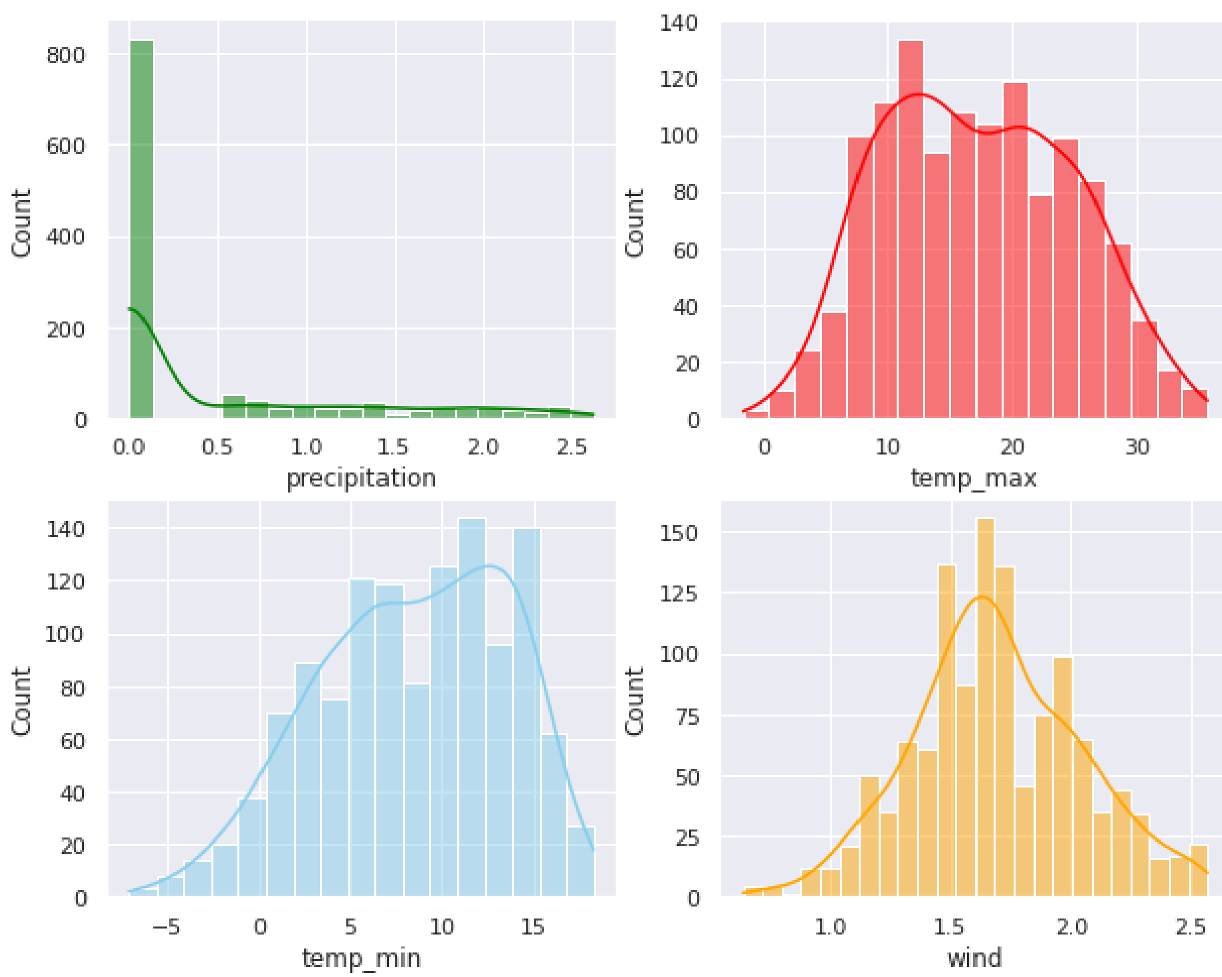
Remove Outliers & Infinite Values

Since this dataset contains ***Outliers ,it will be removed***,to make data set more even.

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

Skewed Distribution Treatment

```
df.precipitation=np.sqrt(df.precipitation)
df.wind=np.sqrt(df.wind)

sns.set(style="darkgrid")
fig,axs=plt.subplots(2,2,figsize=(10,8))
sns.histplot(data=df,x="precipitation",kde=True,ax=axs[0,0],color='gre
en')
sns.histplot(data=df,x="temp_max",kde=True,ax=axs[0,1],color='red')
sns.histplot(data=df,x="temp_min",kde=True,ax=axs[1,0],color='skyblue'
)
sns.histplot(data=df,x="wind",kde=True,ax=axs[1,1],color='orange')

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



```
df.head()
```

	precipitation	temp_max	temp_min	wind	weather
0	0.000000	12.8	5.0	2.167948	drizzle
2	0.894427	11.7	7.2	1.516575	rain
4	1.140175	8.9	2.8	2.469818	rain
5	1.581139	4.4	2.2	1.483240	rain
6	0.000000	7.2	2.8	1.516575	rain

Scaling the weather variables using label Encoder:

```
lc=LabelEncoder()
df["weather"]=lc.fit_transform(df["weather"])

df.head()
```

	precipitation	temp_max	temp_min	wind	weather
0	0.000000	12.8	5.0	2.167948	0
2	0.894427	11.7	7.2	1.516575	2
4	1.140175	8.9	2.8	2.469818	2
5	1.581139	4.4	2.2	1.483240	2
6	0.000000	7.2	2.8	1.516575	2

SPLITTING THE DATASET INTO DEPENDANT AND INDEPENDANT VARIABLES:

```
x=((df.loc[:,df.columns!="weather"]).astype(int)).values[:,0:]
y=df["weather"].values

df.weather.unique()

array([0, 2, 4, 3, 1])

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.1,rando
m_state=2)
```

9.ALGORITHMS AND MODEL TRAINING:

K-NEAREST NEIGHBOR CLASSIFIER:

```
knn=KNeighborsClassifier()
knn.fit(x_train,y_train)
print("KNN Accuracy:{:.2f}%".format(knn.score(x_test,y_test)*100))

KNN Accuracy:78.23%
```

SUPPORT VECTOR MACHINE - CLASSIFIER:

```
svm=SVC()
svm.fit(x_train,y_train)
print("SVM Accuracy:{:.2f}%".format(svm.score(x_test,y_test)*100))
```

```
SVM Accuracy:77.42%
```

GRADIENT BOOSTING CLASSIFIER:

```
gbc=GradientBoostingClassifier(subsample=0.5,n_estimators=450,max_dept
h=5,max_leaf_nodes=25)
gbc.fit(x_train,y_train)
print("Gradient Boosting Accuracy:{:.2f}
%".format(gbc.score(x_test,y_test)*100))
```

```
Gradient Boosting Accuracy:81.45%
```

EXTREME GRADIENT BOOSTING OR XGBCLASSIFIER:

```
import warnings
warnings.filterwarnings('ignore')
xgb=XGBClassifier()
xgb.fit(x_train,y_train)
print("XGB Accuracy:{:.2f}%".format(xgb.score(x_test,y_test)*100))
```

```
[02:34:55] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'multi:softprob'
was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
XGB Accuracy:83.06%
```

10.CHECKING FOR THE USER INPUT:

```
input=[[1.140175,8.9,2.8,2.469818]]
ot=xgb.predict(input)
print("The weather is:")
if(ot==0):
    print("Drizzle")
elif(ot==1):
    print("Fog")
elif(ot==2):
    print("Rain")
elif(ot==3):
    print("snow")
else:
    print("Sun")
```

```
The weather is:
Rain
```