











WEATHER PREDICTION

90% accurate					80% accurate		50% accurate		
Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed
									
76°	74°	70°	70°	71°	76°	75°			

Project Overview

Our project is about *weather prediction*. Here we will predict whether it will rain or not using some weather conditions. We have collected our dataset from [Kaggle \(https://www.kaggle.com\)](https://www.kaggle.com). There are six attributes in our dataset. They are—*date*, *precipitation*, *temp_max*, *temp_min*, *wind*, and *weather*. Where the first five are independent variables and the last one is the dependent variable. The weather attribute contains 5 different classes—drizzle, rain, sun, snow, and fog. We have visualized the dataset with different plotting for checking the outliers or skew.

We have done some preprocessing and standardization using **DataFrame.Quantile()** and **StandardScale()**. As it is a classification problem, we have trained different ML classification and regression models with the dataset. Namely, [Support Vector Machine \(SVM\) \(https://scikit-learn.org/stable/modules/svm.html#:~:text=Support%20vector%20machines%20\(SVMs\)%20are,that](https://scikit-learn.org/stable/modules/svm.html#:~:text=Support%20vector%20machines%20(SVMs)%20are,that) [Decision Tree \(DT\) \(https://scikit-learn.org/stable/modules/tree.html#:~:text=Decision%20Trees%20\(DTs\)%20are%20a,as%20a%20p](https://scikit-learn.org/stable/modules/tree.html#:~:text=Decision%20Trees%20(DTs)%20are%20a,as%20a%20p) [Logistic Regression \(LR\) \(https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html), [K-Nearest Neighbors \(KNN\) \(https://scikit-learn.org/stable/modules/neighbors.html\)](https://scikit-learn.org/stable/modules/neighbors.html), and [Naive Bayes \(NB\) \(https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html\)](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html).

After comparing all the models, SVM & KNN achieved the best score, which is 82.26% . In contrast, DT provides the lowest score, 70.16%.



Importing The Required Libraries

```
In [1]: 1 import pandas as pd
2 import copy
3 import numpy as np
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 import missingno as mso
7 from sklearn.metrics import classification_report
```

Dataset Overview

About Dataset

There are six attributes in our dataset. They are—*date*, *precipitation*, *temp_max*, *temp_min*, *wind*, and *weather*. Where the first five are independent variables and the last one is the dependent variable.

Let's describe how this data set work to predict the weather-

- Date -- We use the date for check how much rain occur in one month and which month it occurs more. So that every year we be aware that this month rain may occur or not.
- Precipitation -- It means product of the condensation of atmospheric water. All the forms in which water falls on the land surface and open water bodies as rain, sleet, snow, hail or drizzle.
- Temp_max, Temp_min -- Predict the maximum temperature and minimum temperature. As we know rainfall and temperature are important climatic inputs for agricultural production, especially in the context of climate change. However, accurate analysis and simulation of the joint distribution of rainfall and temperature are difficult due to possible interdependence between them. And the physical rationale behind the relationship between rainfall and temperature is that rainfall may affect soil moisture which may in turn affect surface temperature by controlling the partitioning between the sensible and heat.
- Wind -- wind speed
- Weather -- As we are going to predict the weather condition. Those are- drizzle, rain, sun, snow, fog.

Dataset link

The dataset available in <https://www.kaggle.com/datasets/ananthr1/weather-prediction>
(<https://www.kaggle.com/datasets/ananthr1/weather-prediction>)

Loading the dataset

```
In [2]: 1 dataset = pd.read_csv('./Dataset/seattle-weather.csv')
```

In [3]: 1 dataset.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   date            1461 non-null   object
1   precipitation    1461 non-null   float64
2   temp_max        1461 non-null   float64
3   temp_min        1461 non-null   float64
4   wind            1461 non-null   float64
5   weather         1461 non-null   object
dtypes: float64(4), object(2)
memory usage: 68.6+ KB
```

In [4]: 1 dataset.head(10)

Out[4]:

	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
5	2012-01-06	2.5	4.4	2.2	2.2	rain
6	2012-01-07	0.0	7.2	2.8	2.3	rain
7	2012-01-08	0.0	10.0	2.8	2.0	sun
8	2012-01-09	4.3	9.4	5.0	3.4	rain
9	2012-01-10	1.0	6.1	0.6	3.4	rain

Data Preprocessing and Exploratory data analysis

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

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

Data Exploration

In [5]: 1 dataset.shape

Out[5]: (1461, 6)

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

```
In [6]: 1 #convert the data type into datetime
        2 dataset['date'] = pd.to_datetime(dataset['date'])
```

```
In [7]: 1 dataset.groupby('weather').size()
```

```
Out[7]: weather
drizzle      53
fog          101
rain         641
snow         26
sun          640
dtype: int64
```

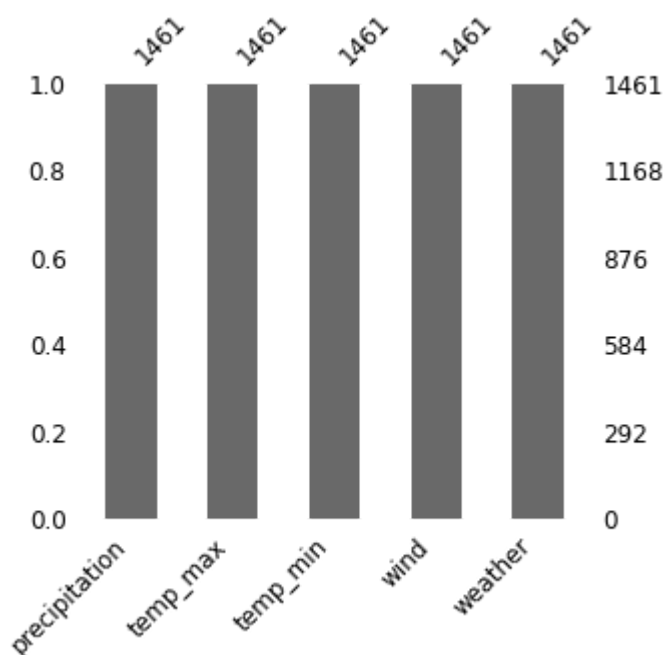
```
In [8]: 1 dataset.duplicated().sum()
```

```
Out[8]: 0
```

```
In [9]: 1 dataset.isnull().sum()
```

```
Out[9]: date          0
precipitation        0
temp_max             0
temp_min             0
wind                 0
weather              0
dtype: int64
```

```
In [10]: 1 plt.figure(figsize=(10,4))
        2 axz=plt.subplot(1,2,2)
        3 mso.bar(dataset.drop(["date"],axis=1),ax=axz,fontsize=12);
```



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.

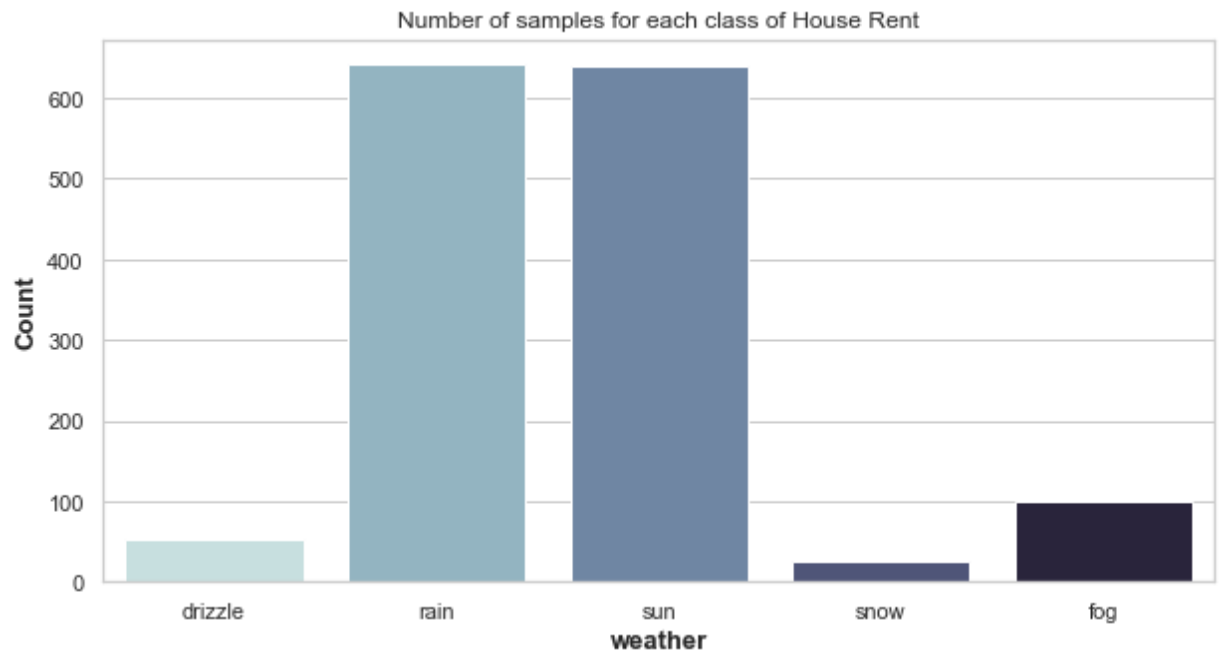
```
In [11]: 1 cols = list(dataset.columns)
          2 print(cols)

['date', 'precipitation', 'temp_max', 'temp_min', 'wind', 'weather']
```

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

```
In [12]: 1 plt.figure(figsize=(10,5))
          2 sns.set(style="whitegrid")
          3 sns.countplot(x = 'weather',data = dataset,palette="ch:start=.2,rot=-.3")
          4 plt.xlabel("weather",fontweight='bold',size=13)
          5 plt.ylabel("Count",fontweight='bold',size=13)
          6 plt.title("Number of samples for each class of House Rent")
          7 plt.show()
```



```
In [13]: 1 countrain=len(dataset[dataset.weather=="rain"])
2 countsun=len(dataset[dataset.weather=="sun"])
3 countdrizzle=len(dataset[dataset.weather=="drizzle"])
4 countsnow=len(dataset[dataset.weather=="snow"])
5 countfog=len(dataset[dataset.weather=="fog"])
6 print("Percent of Rain:{:2f}%".format((countrain/(len(dataset.weather))*100))
7 print("Percent of Sun:{:2f}%".format((countsun/(len(dataset.weather))*100))
8 print("Percent of Drizzle:{:2f}%".format((countdrizzle/(len(dataset.weather))
9 print("Percent of Snow:{:2f}%".format((countsnow/(len(dataset.weather))*100)
10 print("Percent of Fog:{:2f}%".format((countfog/(len(dataset.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 detail of **Rain and Sun** and it also have some additional like **drizzle,snow and fog** .

NUMERICAL OR CONTINUOUS VARIABLES

Next we will explore the **Continuous variables**

```
In [14]: 1 dataset.describe()
```

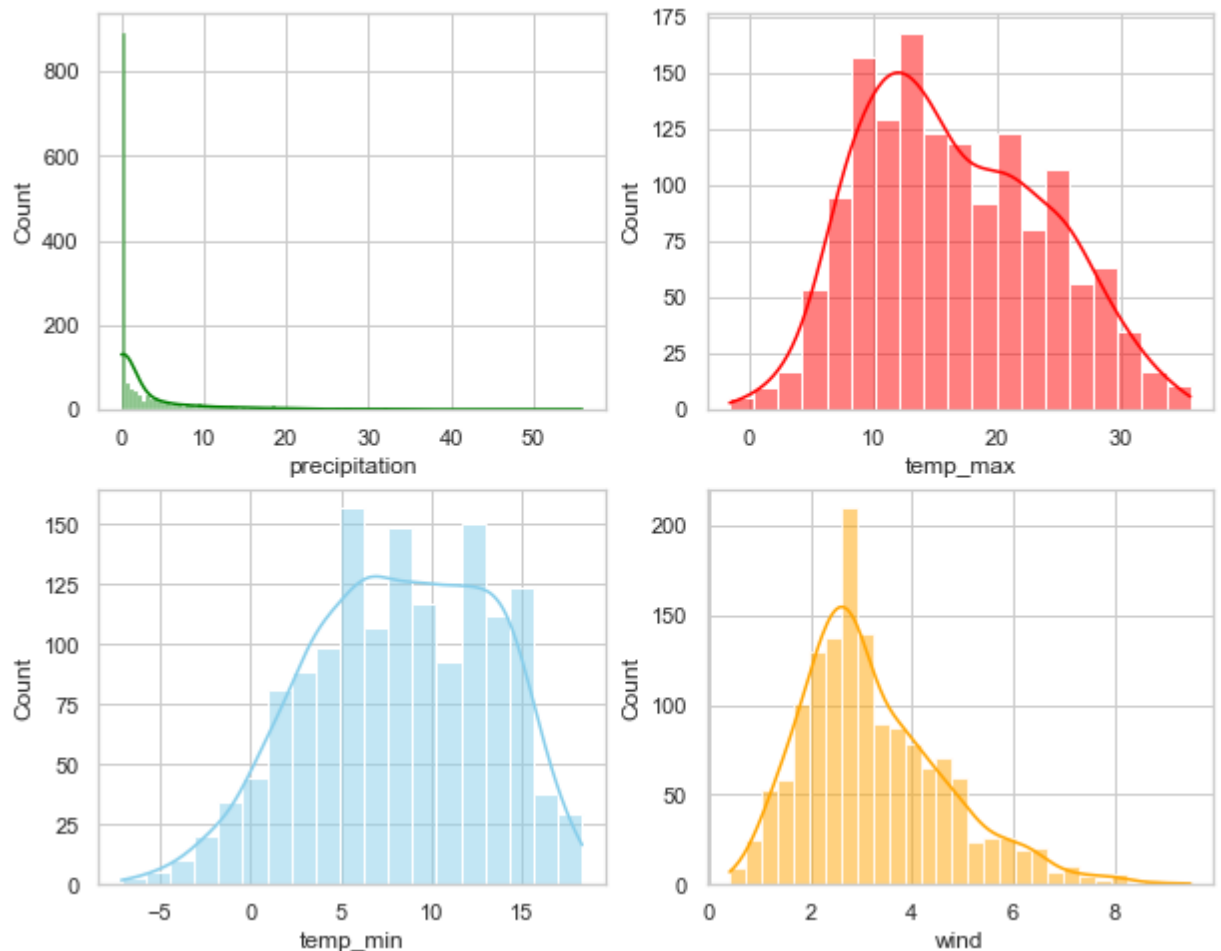
```
Out[14]:
```

	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** .

```
In [15]: 1 sns.set(style="whitegrid")
2 fig,axs=plt.subplots(2,2,figsize=(10,8))
3 sns.histplot(data=dataset,x="precipitation",kde=True,ax=axs[0,0],color='green')
4 sns.histplot(data=dataset,x="temp_max",kde=True,ax=axs[0,1],color='red')
5 sns.histplot(data=dataset,x="temp_min",kde=True,ax=axs[1,0],color='skyblue')
6 sns.histplot(data=dataset,x="wind",kde=True,ax=axs[1,1],color='orange')
```

Out[15]: <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**.

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

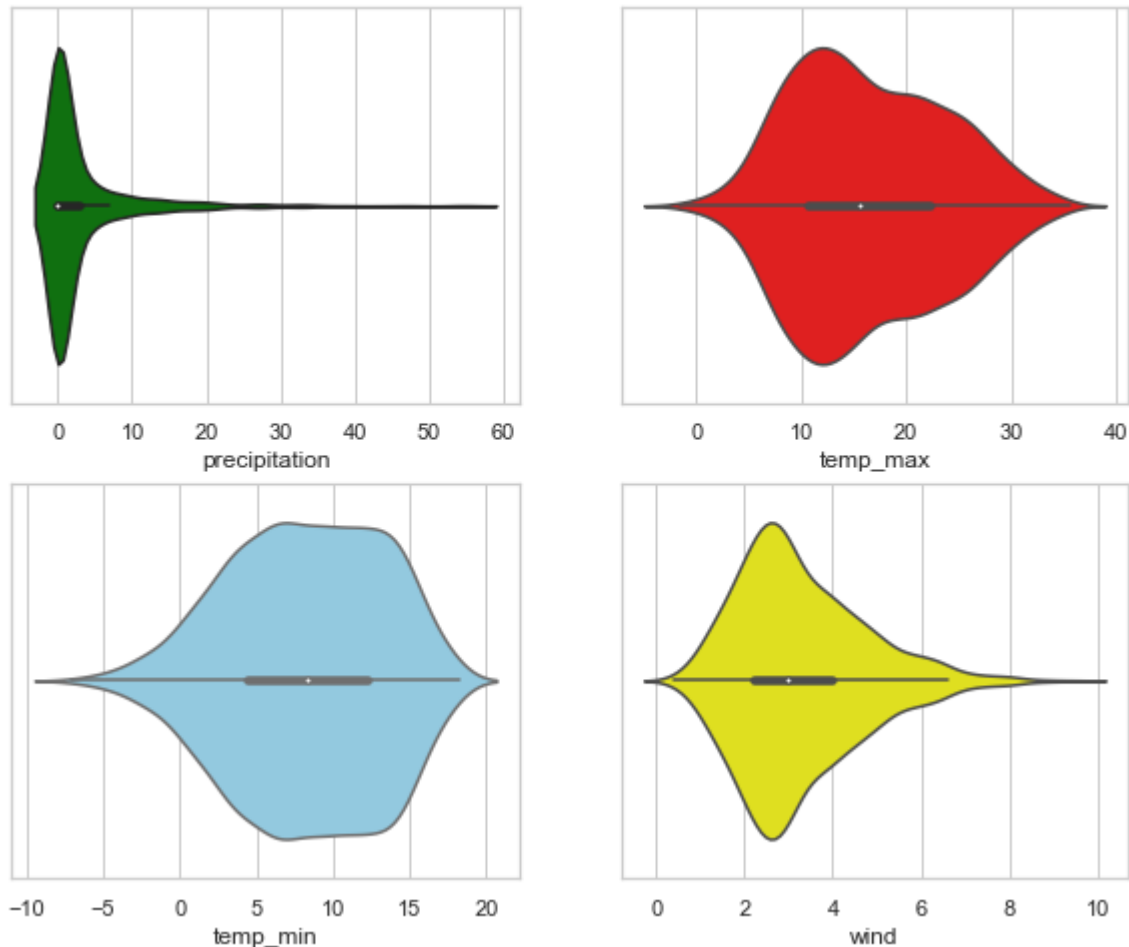
2.Box plot

3.Violin plot

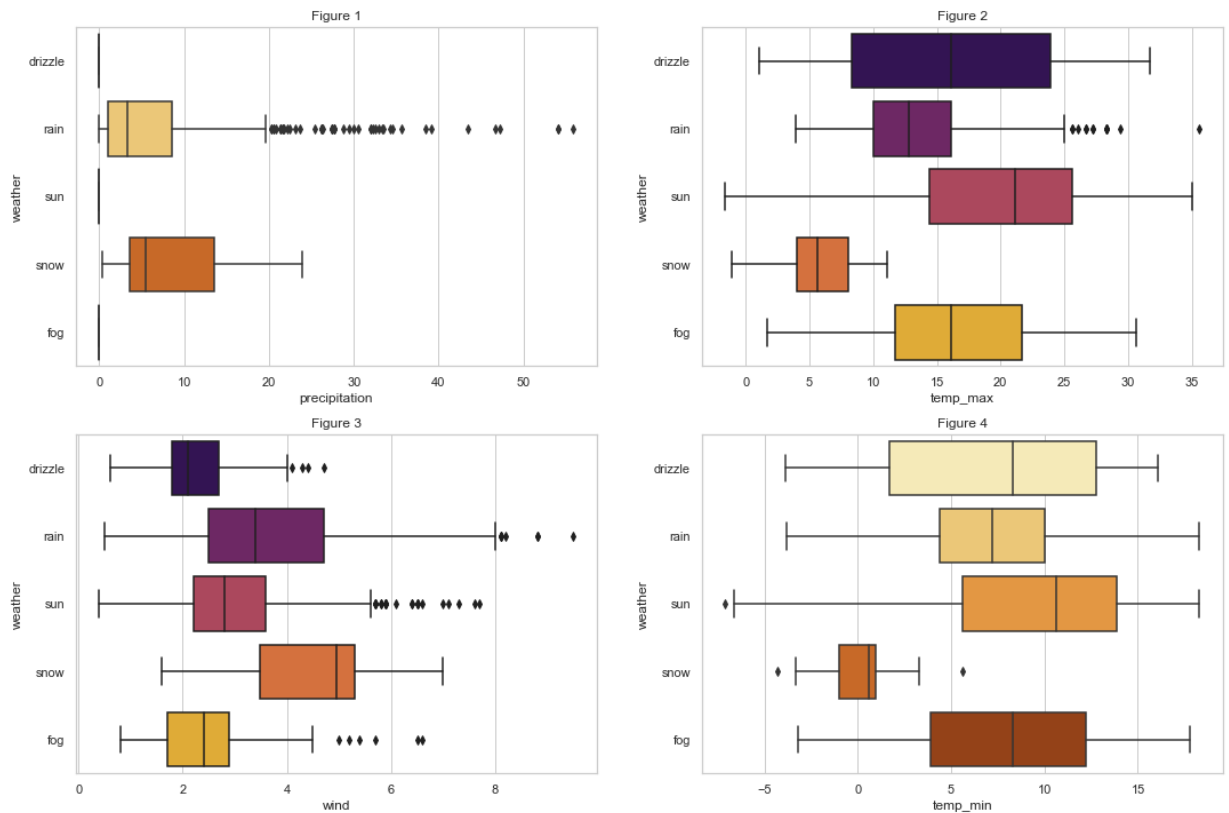
4.Dist plot yet both **box and violin plots** are easier to handel with.

```
In [16]: 1 fig,axs=plt.subplots(2,2,figsize=(10,8))
2         sns.violinplot(data=dataset,x="precipitation",kde=True,ax=axs[0,0],color='gr
3         sns.violinplot(data=dataset,x="temp_max",kde=True,ax=axs[0,1],color='red')
4         sns.violinplot(data=dataset,x="temp_min",kde=True,ax=axs[1,0],color='skyblue
5         sns.violinplot(data=dataset,x="wind",kde=True,ax=axs[1,1],color='yellow')
```

Out[16]: <AxesSubplot:xlabel='wind'>




```
In [17]: 1 fig,axs=plt.subplots(2,2,figsize=(18,12))
2 sns.boxplot(x="precipitation",y="weather",data=dataset,ax=axs[0,0],palette="
3 sns.boxplot(x="temp_max",y="weather",data=dataset,ax=axs[0,1],palette="infer
4 sns.boxplot(x="wind",y="weather",data=dataset,ax=axs[1,0],palette="inferno")
5 sns.boxplot(x="temp_min",y="weather",data=dataset,ax=axs[1,1],palette="YlOrB
6 plt.show()
```



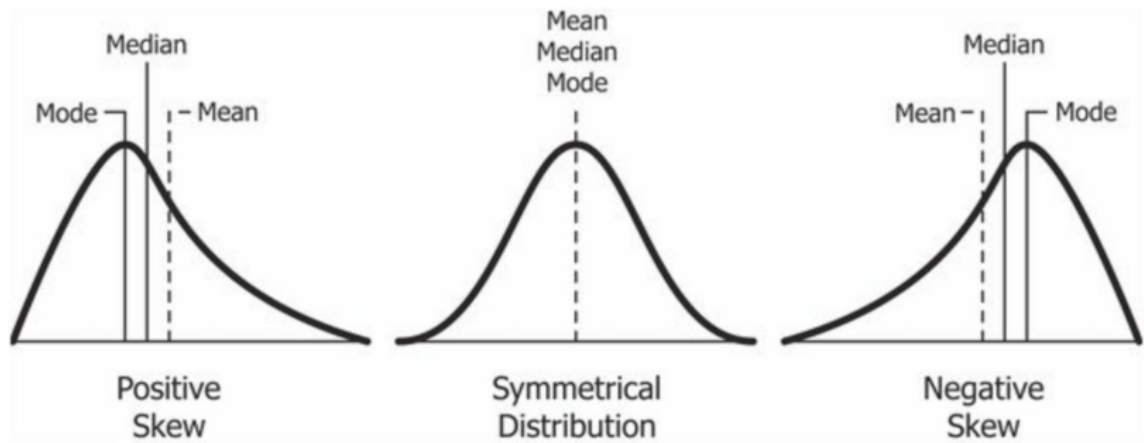
For figure-1: From the above box plot between the **Weather and Precipitation** the value **Rain** has many **positive outliers** and both **Rain** and **Snow** were **positively skewed/has positive skewness**.

For figure-3: 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**.

For figure-4: here some data has **negative** and some have both **positive and negative** outliers and ***snow is negatively skewed**. ***SKEWNESS AND ITS CORRECTIONS:*****

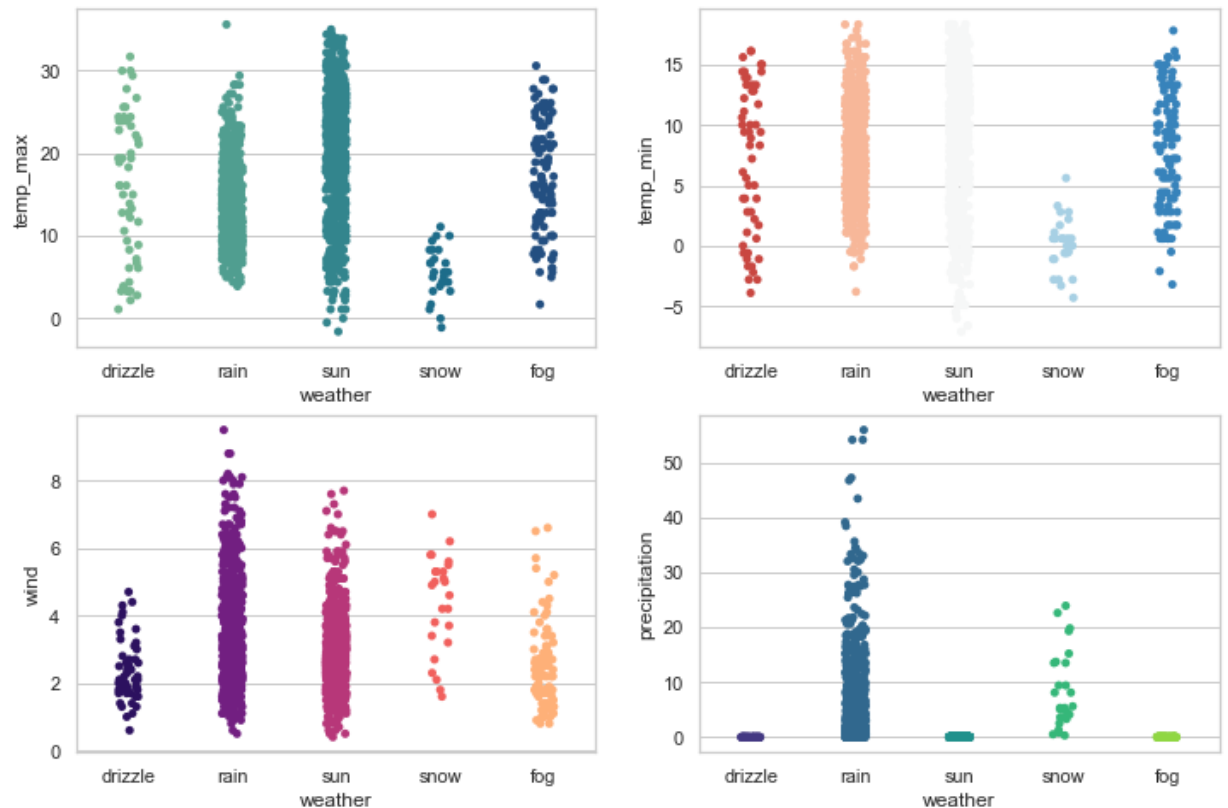
From the above **Violin plot** we can clearly understand the Skewness of the Data as the **TAIL** indicates the skewness.

BELOW DIAGRAM SHOWS THE EXACT OF HOW THE SKEWNESS LOOKS:



```
In [18]: 1 fig, axes = plt.subplots(2, 2, figsize=(12, 8))
2
3 fig.suptitle('Weather vs all numerical factor')
4 sns.stripplot(ax=axes[0, 0], x='weather', y='temp_max', data=dataset, palette=
5 sns.stripplot(ax=axes[0, 1], x='weather', y='temp_min', data=dataset, palette
6 sns.stripplot(ax=axes[1, 0], x='weather', y='wind', data=dataset, palette = "m
7 sns.stripplot(ax=axes[1, 1], x='weather', y='precipitation', data=dataset, pal
8 plt.show()
```

Weather vs all numerical factor

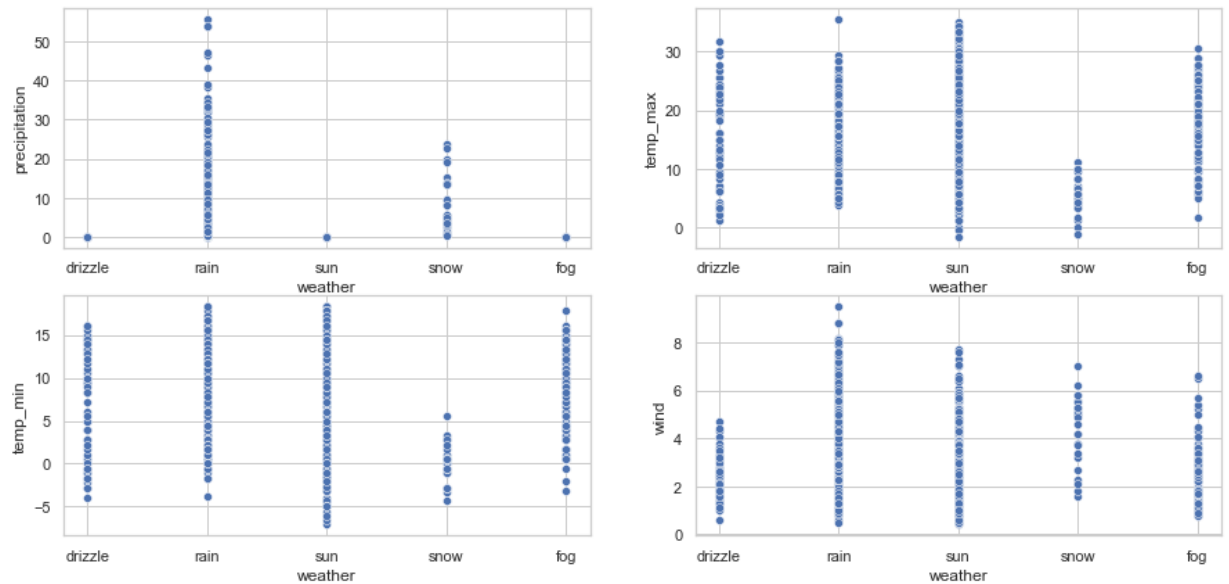


```

In [19]: 1 fig, axes = plt.subplots(2, 2, figsize=(15, 7))
          2
          3 fig.suptitle('Weather vs all numerical factor')
          4
          5 sns.scatterplot(ax=axes[0, 0], data=dataset, x='weather', y='precipitation')
          6 sns.scatterplot(ax=axes[0, 1], data=dataset, x='weather', y='temp_max')
          7 sns.scatterplot(ax=axes[1, 0], data=dataset, x='weather', y='temp_min')
          8 sns.scatterplot(ax=axes[1, 1], data=dataset, x='weather', y='wind')
          9 plt.show()

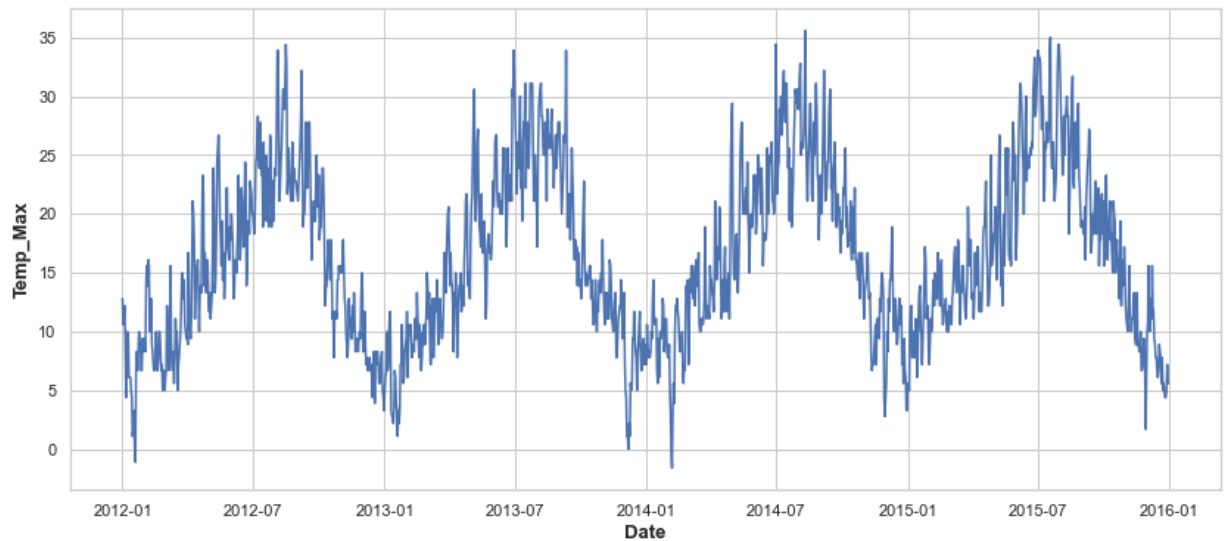
```

Weather vs all numerical factor



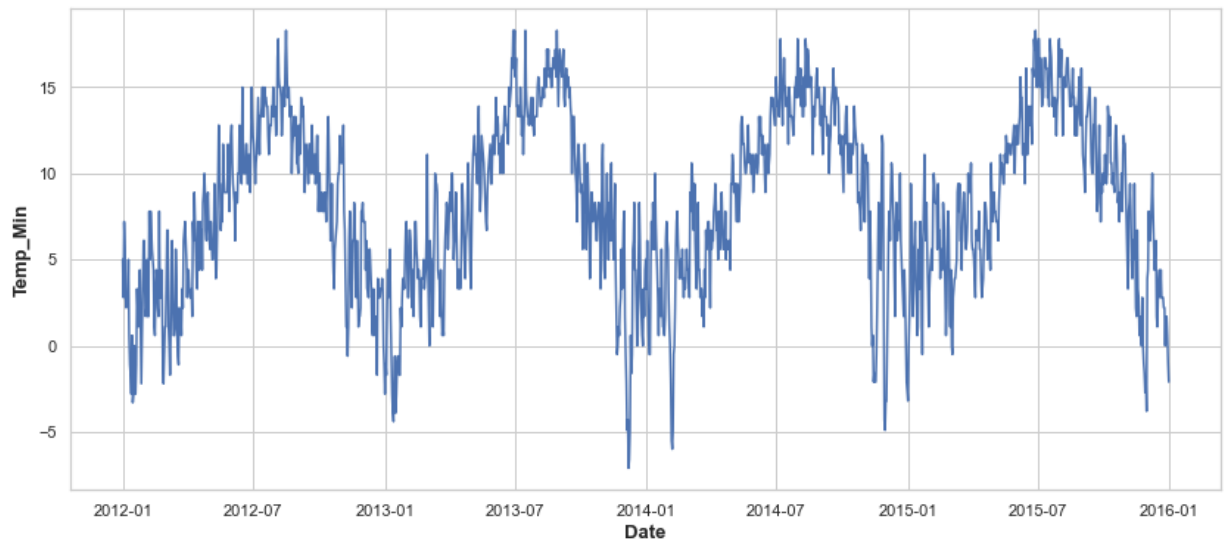
In [20]:

```
1 plt.figure(figsize=(14,6))
2 sns.lineplot(x = 'date',y='temp_max',data=dataset)
3 plt.xlabel("Date",fontweight='bold',size=13)
4 plt.ylabel("Temp_Max",fontweight='bold',size=13)
5 plt.show()
```

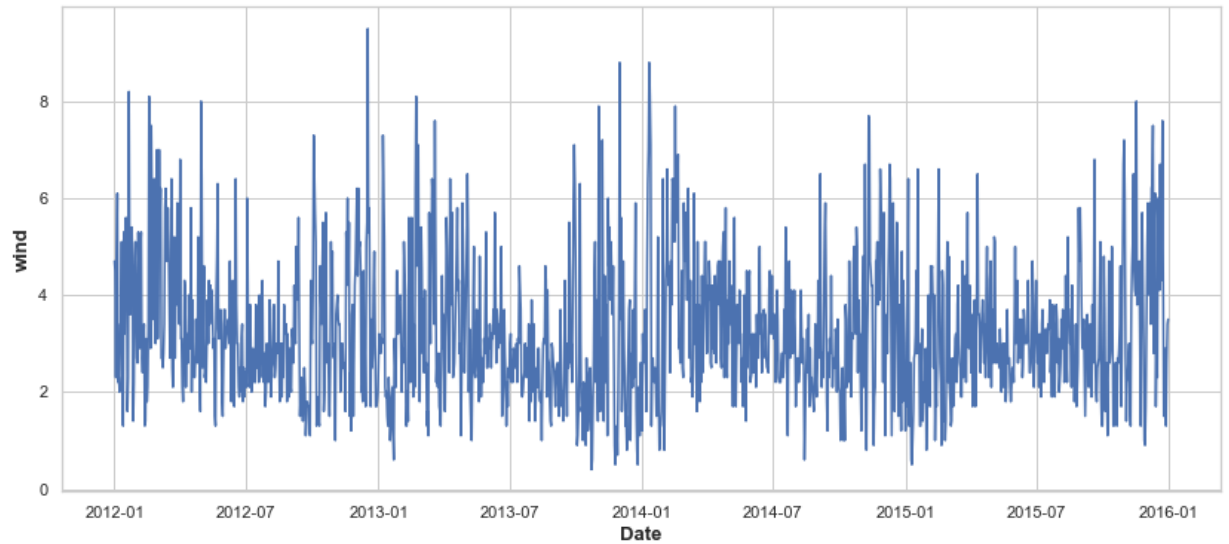


In [21]:

```
1 plt.figure(figsize=(14,6))
2 sns.lineplot(x = 'date',y='temp_min',data=dataset)
3 plt.xlabel("Date",fontweight='bold',size=13)
4 plt.ylabel("Temp_Min",fontweight='bold',size=13)
5 plt.show()
```

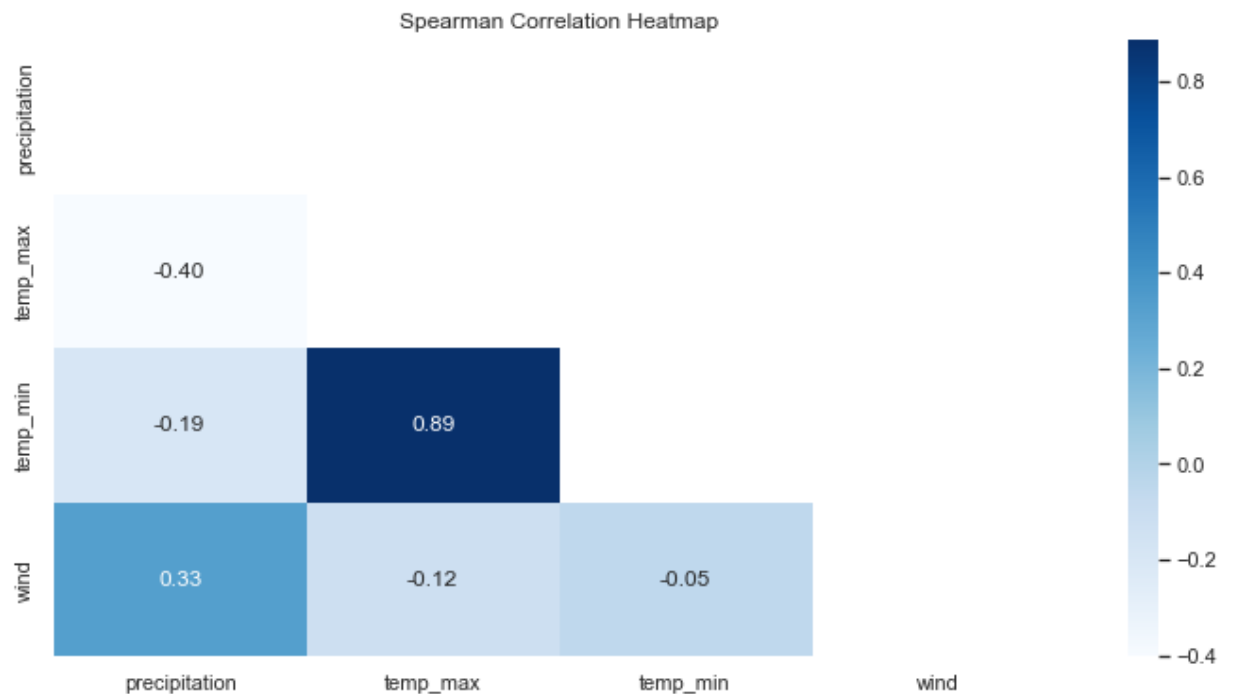


```
In [22]: 1 plt.figure(figsize=(14,6))
2         sns.lineplot(x = 'date',y='wind',data=dataset)
3         plt.xlabel("Date",fontweight='bold',size=13)
4         plt.ylabel("wind",fontweight='bold',size=13)
5         plt.show()
```



Heatmap

```
In [23]: 1 corr = dataset.corr(method = 'spearman')
2 # Steps to remove redundant values
3 # Return a array filled with zeros
4 mask = np.zeros_like(corr)
5 # Return the indices for the upper-triangle of array
6 mask[np.triu_indices_from(mask)] = True
7 # changing the figure size
8 plt.figure(figsize = (12, 6))
9 # "annot = True" to print the values inside the square
10 sns.heatmap(corr, annot=True, fmt='.2f', cmap='Blues', mask=mask)
11 plt.title('Spearman Correlation Heatmap');
```




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

```
In [24]: 1 print(dataset.apply(lambda x: x.nunique()))
2 dataset.describe().T.style.background_gradient(
3     vmin=-1, vmax=1, cmap=sns.color_palette("vlag", as_cmap=True))
```

```
date          1461
precipitation   111
temp_max        67
temp_min        55
wind            79
weather         5
dtype: int64
```

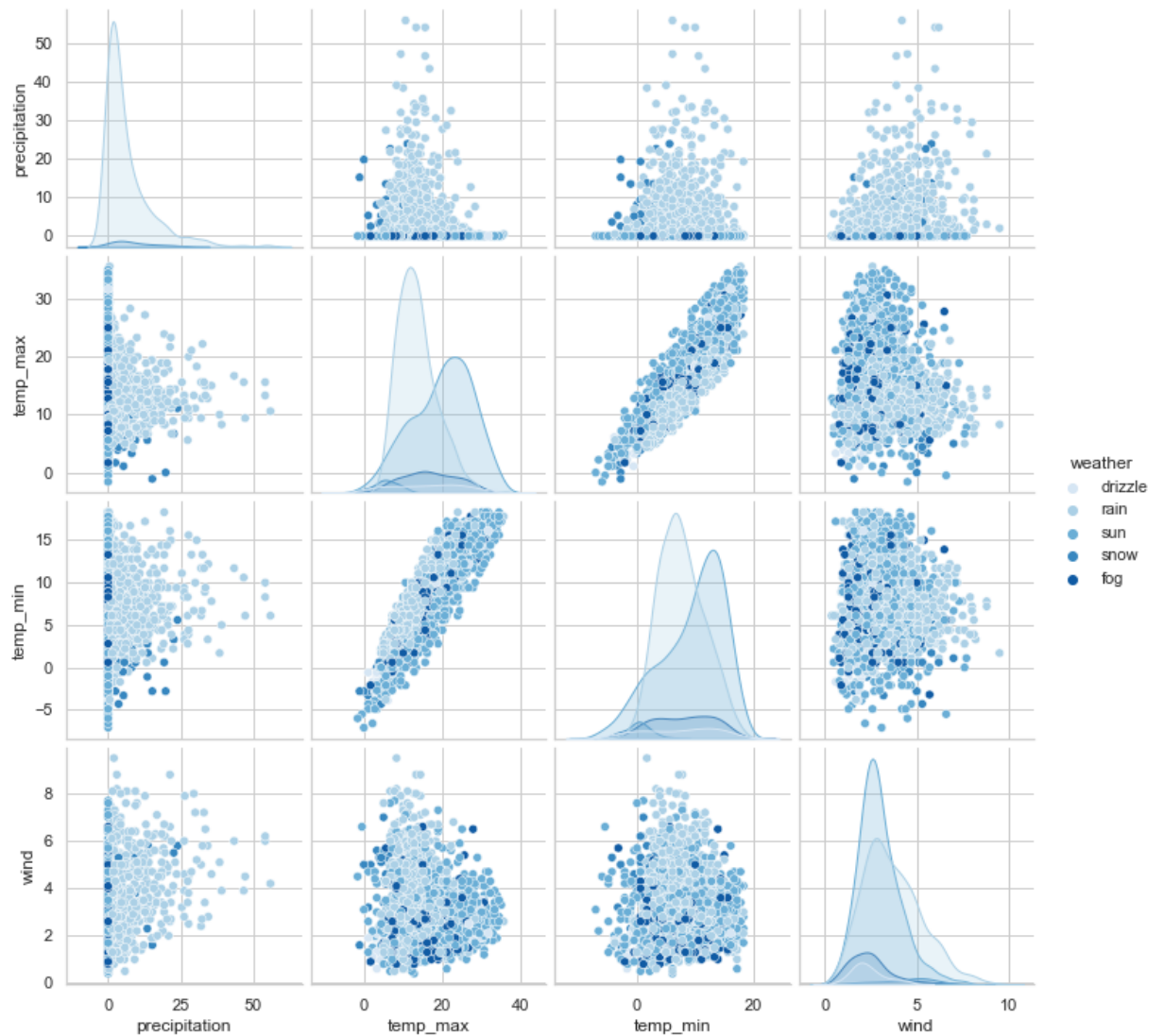
Out[24]:

	count	mean	std	min	25%	50%	75%	n
precipitation	1461.000000	3.029432	6.680194	0.000000	0.000000	0.000000	2.800000	55.9000
temp_max	1461.000000	16.439083	7.349758	-1.600000	10.600000	15.600000	22.200000	35.6000
temp_min	1461.000000	8.234771	5.023004	-7.100000	4.400000	8.300000	12.200000	18.3000
wind	1461.000000	3.241136	1.437825	0.400000	2.200000	3.000000	4.000000	9.5000



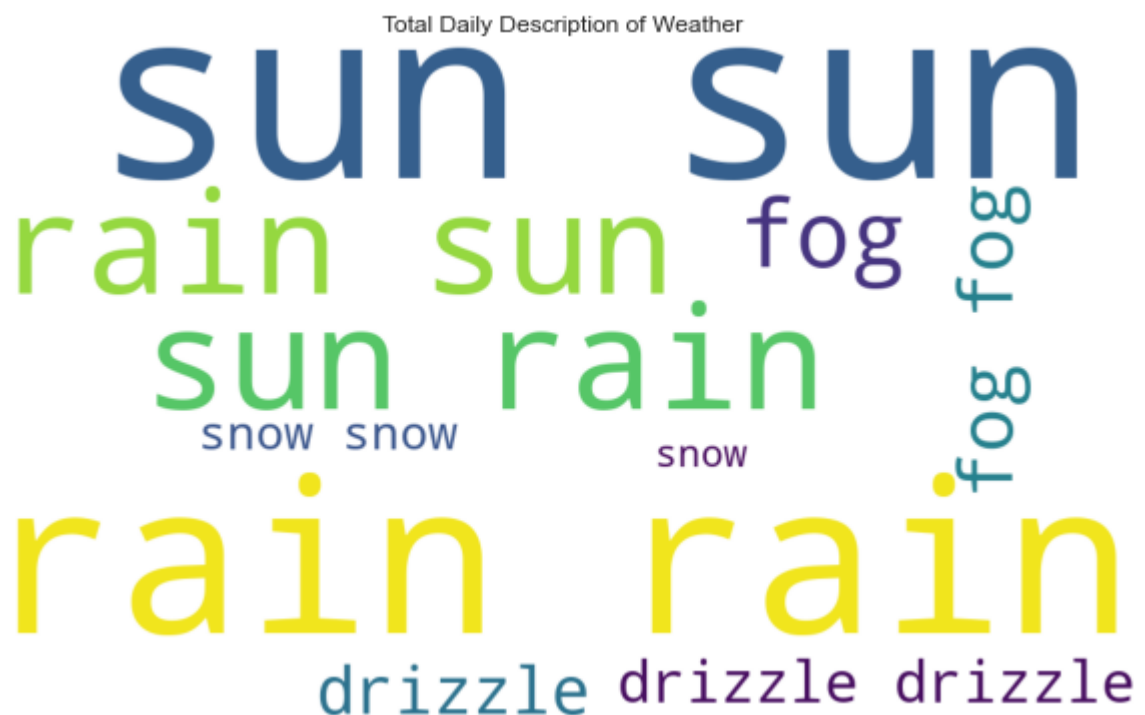
```
In [25]: 1 plt.figure(figsize=(16,8))
2         sns.pairplot(dataset.drop('date',axis=1),hue='weather',palette="Blues")
3         plt.show()
```

<Figure size 1152x576 with 0 Axes>



In [26]:

```
1 from wordcloud import WordCloud, STOPWORDS
2 comment_words = ''
3 stopwords = set(STOPWORDS)
4
5 # iterate through the csv file
6 for val in dataset["weather"]:
7
8     # typecaste each val to string
9     val = str(val)
10
11     # split the value
12     tokens = val.split()
13
14     # Converts each token into lowercase
15     for i in range(len(tokens)):
16         tokens[i] = tokens[i].lower()
17
18     comment_words += " ".join(tokens)+" "
19     pass
20
21 wordcloud = WordCloud(width = 1000, height = 600,
22                       background_color = 'white',
23                       stopwords = stopwords,
24                       min_font_size = 10).generate(comment_words)
25
26 # plot the WordCloud image
27 plt.figure(figsize = (8, 8), facecolor = None)
28 plt.imshow(wordcloud)
29 plt.axis("off")
30 plt.tight_layout(pad = 0)
31 plt.gca().set_title('Total Daily Description of Weather')
32 plt.show()
```



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.

```
In [27]: 1 df = dataset.drop(['date'],axis=1)
```

Scaling the weather variables using label Encoder:

```
In [28]: 1 def LABEL_ENCODING(c1):
2         from sklearn import preprocessing
3         label_encoder = preprocessing.LabelEncoder()
4         dataset[c1]= label_encoder.fit_transform(dataset[c1])
5         dataset[c1].unique()
6 LABEL_ENCODING("weather")
7 dataset
```

```
Out[28]:
```

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	0
1	2012-01-02	10.9	10.6	2.8	4.5	2
2	2012-01-03	0.8	11.7	7.2	2.3	2
3	2012-01-04	20.3	12.2	5.6	4.7	2
4	2012-01-05	1.3	8.9	2.8	6.1	2
...
1456	2015-12-27	8.6	4.4	1.7	2.9	2
1457	2015-12-28	1.5	5.0	1.7	1.3	2
1458	2015-12-29	0.0	7.2	0.6	2.6	1
1459	2015-12-30	0.0	5.6	-1.0	3.4	4
1460	2015-12-31	0.0	5.6	-2.1	3.5	4

1461 rows × 6 columns

```
In [29]: 1 Q1=df.quantile(0.25)
2 Q3=df.quantile(0.75)
3 IQR=Q3-Q1
4 df=df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
```

C:\Users\ASUS\AppData\Local\Temp\ipykernel_4812\2492161268.py:4: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is deprecated and will raise ValueError in a future version. Do `left, right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
df=df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]

```
In [30]: 1 df.precipitation=np.sqrt(df.precipitation)
         2 df.wind=np.sqrt(df.wind)
```

C:\Users\ASUS\AppData\Local\Temp\ipykernel_4812\4268603128.py:1: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

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

C:\Users\ASUS\AppData\Local\Temp\ipykernel_4812\4268603128.py:2: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

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

SPLITTING THE DATASET INTO DEPENDANT AND INDEPENDANT VARIABLES

```
In [31]: 1 X=df.drop(['weather'], axis=1)
         2 y=df["weather"].values
         3 X
```

```
Out[31]:
```

	precipitation	temp_max	temp_min	wind
0	0.000000	12.8	5.0	2.167948
2	0.894427	11.7	7.2	1.516575
4	1.140175	8.9	2.8	2.469818
5	1.581139	4.4	2.2	1.483240
6	0.000000	7.2	2.8	1.516575
...
1455	0.000000	4.4	0.0	1.581139
1457	1.224745	5.0	1.7	1.140175
1458	0.000000	7.2	0.6	1.612452
1459	0.000000	5.6	-1.0	1.843909
1460	0.000000	5.6	-2.1	1.870829

1233 rows × 4 columns

Standardization is a scaling technique wherein it makes the data scale-free by converting the statistical distribution of the data into the below format:

- mean - 0 (zero)
- standard deviation - 1

Here's the formula for Standardization:

$$z = \frac{x - \mu}{\sigma}$$

```
In [32]: 1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
3 X = sc.fit_transform(X)
4 X
```

```
Out[32]: array([[ -0.60933199, -0.5794075 , -0.66479802,  1.34668256],
 [  0.57361988, -0.72521075, -0.23932352, -0.49985515],
 [  0.89864167, -1.09634631, -1.09027252,  2.20243364],
 ...,
 [ -0.60933199, -1.32167861, -1.51574702, -0.2280609 ],
 [ -0.60933199, -1.53375608, -1.82518302,  0.42808319],
 [ -0.60933199, -1.53375608, -2.03792027,  0.50439647]])
```

Splitting the Dataset into train & test

```
In [33]: 1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, ran
4
5 print("X_train shape: ", X_train.shape)
6 print("X_test shape: ", X_test.shape)
7 print("y_train shape: ", y_train.shape)
8 print("y_test shape: ", y_test.shape)
```

```
X_train shape: (1109, 4)
X_test shape: (124, 4)
y_train shape: (1109,)
y_test shape: (124,)
```

Model Development

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

- 1.K-Nearest Neighbour(KNN)
- 2.Support Vector Machine(SVM)
- 3.Decision Tree (DT)
- 4.Logistic Regression (LR)
- 5.Naive Bayes (NB)

- Support Vector Machine (SVM)

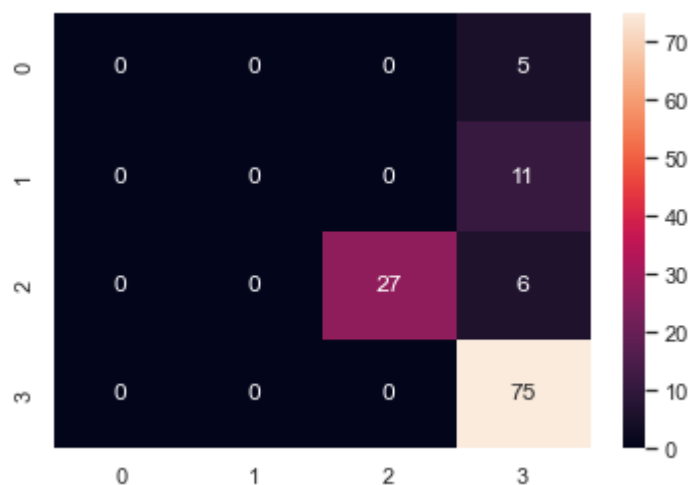
```
In [34]: 1 from sklearn import svm
2 from sklearn import metrics
3 model_svm = svm.SVC(kernel = 'linear', random_state= 1) #select the algorithm
4 model_svm.fit(X_train, y_train) #train the model with the training dataset
5 y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the
6 # checking the accuracy of the algorithm.
7 # by comparing predicted output by the model and the actual output
8 score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
9 print("-----")
10 print('The accuracy of the SVM is: {}'.format(score_svm))
11 print("-----")
12 # save the accuracy score
13 score = set()
14 score.add(('SVM', score_svm))
```

```
-----
The accuracy of the SVM is: 0.8226
-----
```

```
In [35]: 1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_prediction_svm)
3 print(cm)
```

```
[[ 0  0  0  5]
 [ 0  0  0 11]
 [ 0  0 27  6]
 [ 0  0  0 75]]
```

```
In [36]: 1 sns.heatmap(cm,annot=True)
2 plt.show()
```



```
In [37]: 1 print(classification_report(y_test, y_prediction_svm, zero_division=0))
```

	precision	recall	f1-score	support
drizzle	0.00	0.00	0.00	5
fog	0.00	0.00	0.00	11
rain	1.00	0.82	0.90	33
sun	0.77	1.00	0.87	75
accuracy			0.82	124
macro avg	0.44	0.45	0.44	124
weighted avg	0.73	0.82	0.77	124

- **Decision Tree (DT)**

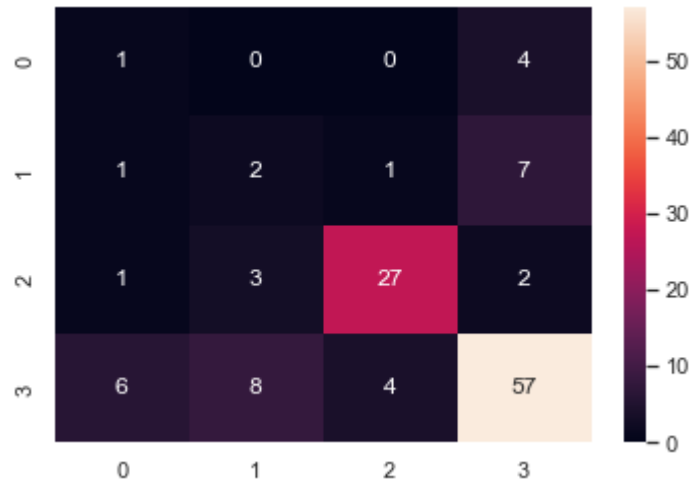
```
In [38]: 1 # importing the necessary package to use the classification algorithm
2 from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Alg
3 model_dt = DecisionTreeClassifier(random_state=5)
4 model_dt.fit(X_train, y_train) #train the model with the training dataset
5 y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the tra
6 # checking the accuracy of the algorithm.
7 # by comparing predicted output by the model and the actual output
8 score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
9 print("-----")
10 print('The accuracy of the DT is: {}'.format(score_dt))
11 print("-----")
12 # save the accuracy score
13 score.add(('DT', score_dt))
```

```
-----
The accuracy of the DT is: 0.7016
-----
```

```
In [39]: 1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_prediction_dt)
3 print(cm)
```

```
[[ 1  0  0  4]
 [ 1  2  1  7]
 [ 1  3 27  2]
 [ 6  8  4 57]]
```

```
In [40]: 1 sns.heatmap(cm,annot=True)
2 plt.show()
```



```
In [41]: 1 print(classification_report(y_test, y_prediction_dt, zero_division=0))
```

	precision	recall	f1-score	support
drizzle	0.11	0.20	0.14	5
fog	0.15	0.18	0.17	11
rain	0.84	0.82	0.83	33
sun	0.81	0.76	0.79	75
accuracy			0.70	124
macro avg	0.48	0.49	0.48	124
weighted avg	0.74	0.70	0.72	124

- **K-Nearest Neighbors (KNN) Classifier**

In [42]:

```
1  # importing the necessary package to use the classification algorithm
2  from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbour
3  #from sklearn.linear_model import LogisticRegression # for Logistic Regression
4  k_score=[]
5  for k in range(1,50):
6      model_knn = KNeighborsClassifier(n_neighbors=k) # 12 neighbours for putting
7      model_knn.fit(X_train, y_train) #train the model with the training dataset
8      y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the t
9      # checking the accuracy of the algorithm.
10     # by comparing predicted output by the model and the actual output
11     score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
12     k_score.append(score_knn)
13
14     k = k_score.index(max(k_score))+1
15
16     #Train the model with the best k value
17     model_knn = KNeighborsClassifier(n_neighbors=k) # 12 neighbours for putting
18     model_knn.fit(X_train, y_train) #train the model with the training dataset
19     y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the t
20     # checking the accuracy of the algorithm.
21     # by comparing predicted output by the model and the actual output
22     score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
23     k_score.append(score_knn)
24     print("-----")
25     print('The accuracy of the KNN is: {}'.format(score_knn))
26     print("-----")
27     # save the accuracy score
28     score.add(('KNN', score_knn))
```

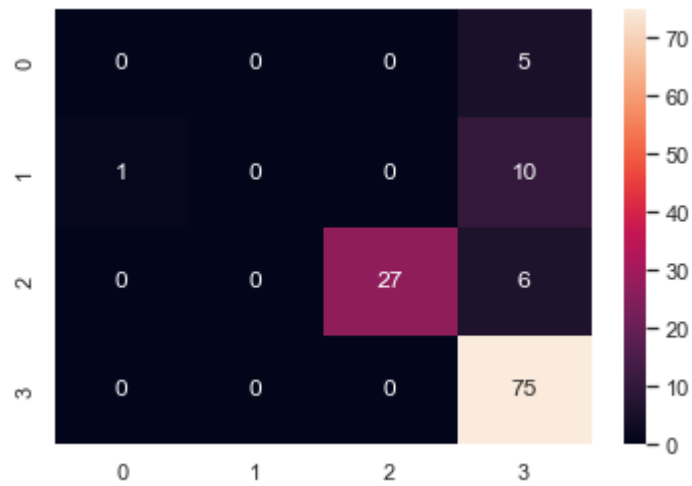
The accuracy of the KNN is: 0.8226

In [43]:

```
1  from sklearn.metrics import confusion_matrix, accuracy_score
2  cm = confusion_matrix(y_test, y_prediction_knn)
3  print(cm)
```

```
[[ 0  0  0  5]
 [ 1  0  0 10]
 [ 0  0 27  6]
 [ 0  0  0 75]]
```

```
In [44]: 1 sns.heatmap(cm,annot=True)
          2 plt.show()
```



```
In [45]: 1 print(classification_report(y_test, y_prediction_knn, zero_division=0))
```

	precision	recall	f1-score	support
drizzle	0.00	0.00	0.00	5
fog	0.00	0.00	0.00	11
rain	1.00	0.82	0.90	33
sun	0.78	1.00	0.88	75
accuracy			0.82	124
macro avg	0.45	0.45	0.44	124
weighted avg	0.74	0.82	0.77	124

- **Logistic Regression (LR)**

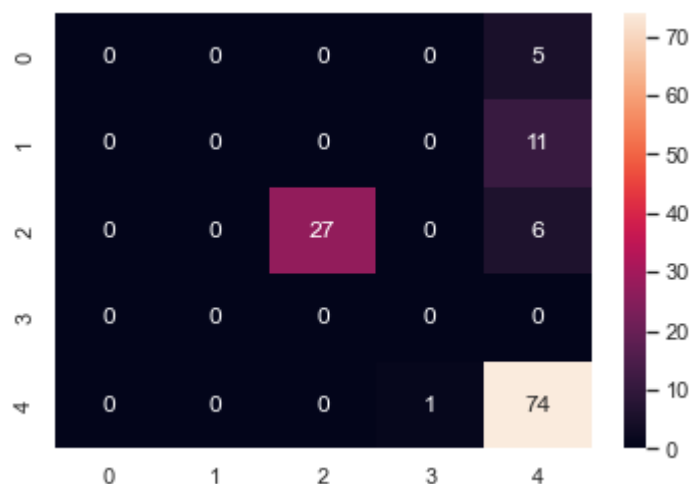
```
In [46]: 1 # importing the necessary package to use the classification algorithm
2 from sklearn.linear_model import LogisticRegression # for Logistic Regression
3 model_lr = LogisticRegression(solver='lbfgs', max_iter=1000)
4 model_lr.fit(X_train, y_train) #train the model with the training dataset
5 y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the tra
6 # checking the accuracy of the algorithm.
7 # by comparing predicted output by the model and the actual output
8 score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
9 print("-----")
10 print('The accuracy of the LR is: {}'.format(score_lr))
11 print("-----")
12 # save the accuracy score
13 score.add(('LR', score_lr))
```

```
-----
The accuracy of the LR is: 0.8145
-----
```

```
In [47]: 1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_prediction_lr)
3 print(cm)
```

```
[[ 0  0  0  0  5]
 [ 0  0  0  0 11]
 [ 0  0 27  0  6]
 [ 0  0  0  0  0]
 [ 0  0  0  1 74]]
```

```
In [48]: 1 sns.heatmap(cm,annot=True)
2 plt.show()
```



```
In [49]: 1 print(classification_report(y_test, y_prediction_lr, zero_division=0))
```

	precision	recall	f1-score	support
drizzle	0.00	0.00	0.00	5
fog	0.00	0.00	0.00	11
rain	1.00	0.82	0.90	33
snow	0.00	0.00	0.00	0
sun	0.77	0.99	0.87	75
accuracy			0.81	124
macro avg	0.35	0.36	0.35	124
weighted avg	0.73	0.81	0.76	124

- Naive Bayes (NB)

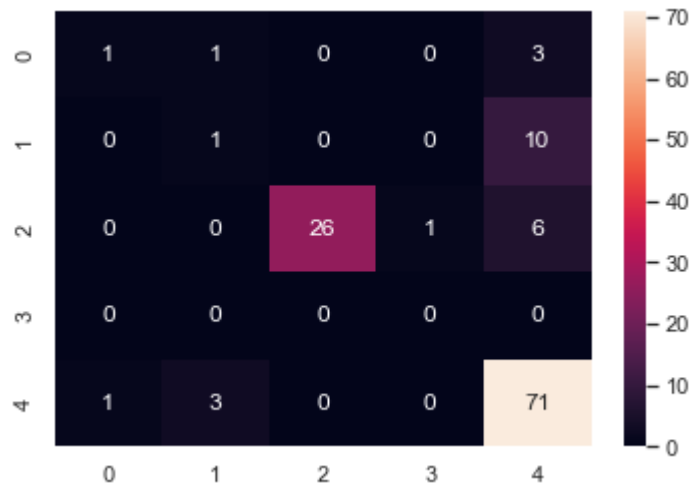
```
In [50]: 1 # importing the necessary package to use the classification algorithm
2 from sklearn.naive_bayes import GaussianNB
3 model_nb = GaussianNB()
4 model_nb.fit(X_train, y_train) #train the model with the training dataset
5 y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the tra
6 # checking the accuracy of the algorithm.
7 # by comparing predicted output by the model and the actual output
8 score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
9 print("-----")
10 print('The accuracy of the NB is: {}'.format(score_nb))
11 print("-----")
12 # save the accuracy score
13 score.add(('NB', score_nb))
```

```
-----
The accuracy of the NB is: 0.7984
-----
```

```
In [51]: 1 from sklearn.metrics import confusion_matrix, accuracy_score
2 cm = confusion_matrix(y_test, y_prediction_nb)
3 print(cm)
```

```
[[ 1  1  0  0  3]
 [ 0  1  0  0 10]
 [ 0  0 26  1  6]
 [ 0  0  0  0  0]
 [ 1  3  0  0 71]]
```

```
In [52]: 1 sns.heatmap(cm,annot=True)
2 plt.show()
```



```
In [53]: 1 print(classification_report(y_test, y_prediction_nb, zero_division=0))
```

	precision	recall	f1-score	support
drizzle	0.50	0.20	0.29	5
fog	0.20	0.09	0.13	11
rain	1.00	0.79	0.88	33
snow	0.00	0.00	0.00	0
sun	0.79	0.95	0.86	75
accuracy			0.80	124
macro avg	0.50	0.41	0.43	124
weighted avg	0.78	0.80	0.78	124

CHECKING FOR THE USER INPUT:

```
In [54]: 1 input=[[1.140175,8.9,2.8,2.469818]]
2 ot=model_svm.predict(input)
3 print("The weather is:")
4 if(ot==0):
5     print("Drizzle")
6 elif(ot==1):
7     print("Fog")
8 elif(ot==2):
9     print("Rain")
10 elif(ot==3):
11     print("snow")
12 else:
13     print("Sun")
```

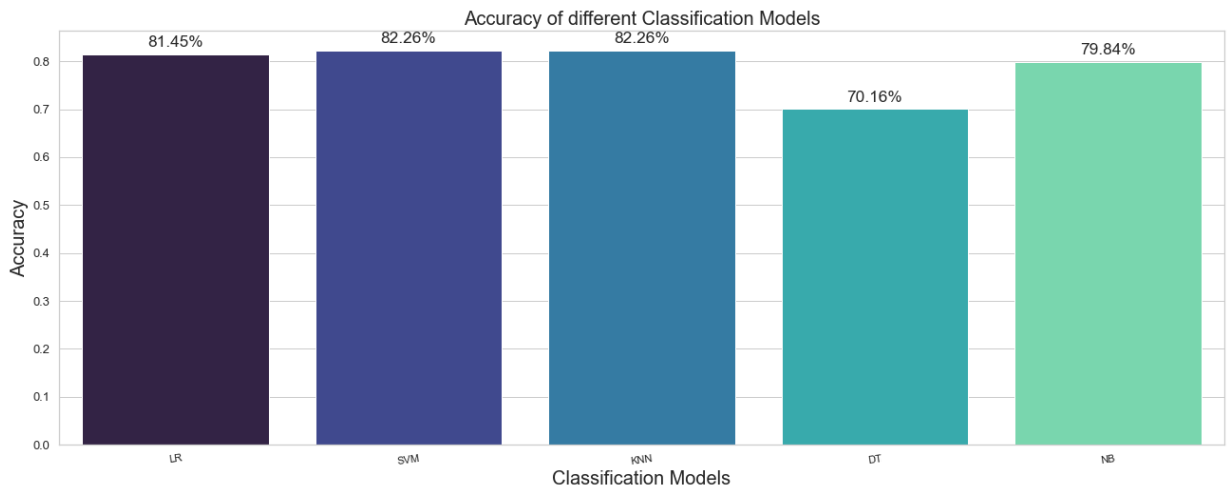
The weather is:
Sun

Create visualization for all model with their Accuracy

```
In [55]: 1 score = list(score)
2 model_list = []
3 acc = []
4 for i in score:
5     model_list.append(i[0])
6     acc.append(i[1])
7 model_list
```

Out[55]: ['LR', 'SVM', 'KNN', 'DT', 'NB']

```
In [56]: 1 plt.rcParams['figure.figsize']=8,6
2 plt.figure(figsize=(22,8))
3 ax = sns.barplot(x=model_list, y=acc, palette = "mako", saturation =1.5)
4 plt.xlabel("Classification Models", fontsize = 20 )
5 plt.ylabel("Accuracy", fontsize = 20)
6 plt.title("Accuracy of different Classification Models", fontsize = 20)
7 plt.xticks(fontsize = 11, horizontalalignment = 'center', rotation = 8)
8 plt.yticks(fontsize = 13)
9 for p in ax.patches:
10     width, height = p.get_width(), p.get_height()
11     x, y = p.get_xy()
12     ax.annotate(f'{height:.2%}', (x + width/2, y + height*1.02), ha='center')
13 plt.show()
```



Discussion and Conclusion

The purpose of this project was to find a suitable classifier for weather prediction which will classify weather as accurately as possible. After applying 5 different models we can see that for this particular dataset, **KNN and Kernel SVM** have **performed better than the rest**. So we can shortlist these 2 to work on this project. This is exactly the same conclusion we arrived at by training each of those algorithms individually.