Weather Prediction

Machine Learning project

Afifah Hadi Lestari

Portfolio

Table of Contents

1. Project Overview

Weather data is a big deal in many fields, from farming and transportation to city planning. Having accurate weather info can make a huge difference—whether it's preparing for storms or simply planning outdoor events. Thanks to modern technology, I can now dive into weather data using exploration and visualization techniques to uncover meaningful patterns and trends.

In this project, I'll analyze weather data to get a better understanding of weather trends based on a real dataset. By identifying patterns and correlations, we can make more accurate predictions and provide valuable insights that help different industries make smarter, data-driven decisions.

Project Goals

The primary objectives of this project:

1. Performing Exploratory Data Analysis (EDA) to understand patterns and trends in weather conditions.
2. Building a predictive model to forecast future weather conditions based on historical data.
3. Evaluating the model's performance using suitable metrics to ensure accuracy and reliability.
4. Derive actionable insights from the model to understand how different factors influence weather conditions.

Technologies Used

- **Programming Language**: Python
- **Libraries**: Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, Plotly
- **Machine Learning Techniques**: Decision Tree, Random Forest, Gradient Boosting

2. Dataset

For this project, I'm using a dataset from Kaggle, one of the top platforms for sharing datasets. It contains a variety of weather-related data points, such as temperature, humidity, air pressure, and wind speed—factors that influence weather conditions. This dataset was chosen because of its rich and diverse information, making it ideal for deep analysis and insightful exploration.

The dataset used in this project consists of historical weather data collected over several years. The key features in the dataset include:

- **Date**: The recorded date of weather conditions
- **Precipitation**: The amount of rainfall measured in millimeters
- **Temperature (Max & Min)**: The highest and lowest temperatures recorded in degrees Celsius
- **Wind Speed**: The velocity of wind in meters per second
- **Weather Condition**: The categorical label describing the weather (e.g., sunny, rainy, foggy)

Link the dataset: https://www.kaggle.com/datasets/ananthr1/weather-prediction

3. Exploratory Data Analysis – EDA

Exploratory Data Analysis (EDA) is the process of exploring data to understand patterns, distributions, and relationships between variables before building Machine Learning models. This stage includes examining the data structure, identifying missing values, and basic statistical analysis. The main goal of EDA is to find valuable insights in the data, identify trends or anomalies, and determine the most relevant features for the prediction model. By performing EDA, we can ensure that the data used is clean, meaningful, and ready for further processing in the development of Machine Learning models.

The dataset used in this project contains weather data consisting of 1,461 rows and 6 columns, this data can be used to understand weather change trends and build Machine Learning-based prediction models.

```
#Load the dataset
df = pd.read_csv("weather.csv")
```

```
df
```

	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
...	...	...	...	...	...	...
1456	2015-12-27	8.6	4.4	1.7	2.9	rain
1457	2015-12-28	1.5	5.0	1.7	1.3	rain
1458	2015-12-29	0.0	7.2	0.6	2.6	fog
1459	2015-12-30	0.0	5.6	-1.0	3.4	sun
1460	2015-12-31	0.0	5.6	-2.1	3.5	sun

1461 rows × 6 columns

Based on the output, there are no missing values from all columns, 1,461. So overall this dataset is clean without missing values, this can be directly used for further exploration and construction of weather prediction models.

```
df.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
```

Based on the output there are no null values in this dataset, this indicates that the data is clean and does not require imputation techniques or handling of missing values before being used for further analysis or Machine Learning model building.

```
# Check null values
df.isnull().sum()
```

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

Convert the 'date' column from object (string) data type to datetime data type. This conversion is important because it enables time-based analysis, such as extraction of date information (year, month, day), seasonal trends, as well as more efficient processing of time series data in Machine Learning and data visualization.

```
#convert the data type into datetime
df['date'] = pd.to_datetime(df['date'])
```

This output shows the number of unique values in each column of this weather dataset. The date column has 1,461 unique values, which means each row represents a different day in the dataset. The precipitation column has 111 unique values, which shows the variation in rainfall during the observed period. For temperature, temp_max has 67 unique values, while temp_min has 55 unique values, which illustrates the wide range of daily temperature changes. The wind column records 79 unique values, reflecting the variation in wind speed in this dataset. Finally, the weather column has only 5 unique values, indicating that the weather conditions in this dataset fall into five main categories. This information is important in exploratory analysis to understand the distribution and patterns of the data before further modeling.

```
df.nunique()

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

shows the frequency of each weather condition in the dataset. Rain is the most common weather condition, appearing 641 times, followed closely by sunny weather, which occurs 640 times. Foggy conditions are recorded 101 times, while drizzle appears 53 times. Snow is the least frequent weather condition, occurring only 26 times. This analysis helps in understanding the distribution of weather types and can be useful for further predictive modeling or trend analysis.

```
df['weather'].value_counts()

weather
rain       641
sun        640
fog        101
drizzle     53
snow        26
Name: count, dtype: int64
```

4. Data Visualization

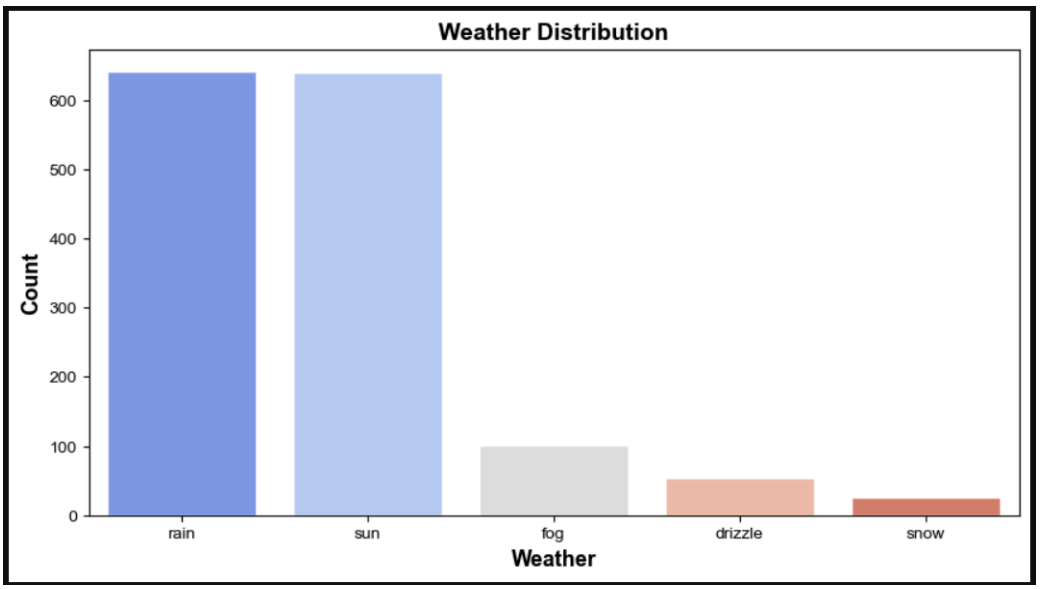
Data Visualization is the process of representing data graphically to identify patterns, trends, and insights more easily. It helps transform raw data into visual formats like charts, graphs, and plots, making complex information more understandable. The main goal of data visualization is to simplify data interpretation, detect correlations, and support decision-making. In the context of weather data, visualizing temperature trends, precipitation levels, and weather occurrences over time can provide valuable insights into climate patterns and anomalies.

This code creates a visualization of the weather distribution in the dataset using seaborn and matplotlib. The generated chart is a bar plot, where the x-axis represents weather categories, and the y-axis shows their frequency. The whitegrid theme is applied for a cleaner look, and the coolwarm color palette is used to enhance the distinction between weather categories.

```
# Set ukuran figure
fig, ax = plt.subplots(figsize=(10, 5))

# Tema
sns.set_style("whitegrid")

# Membuat countplot dengan pendekatan
sns.barplot(
    x=df["weather"].value_counts().index,
    y=df["weather"].value_counts().values,
    palette="coolwarm", ax=ax
)

# Memberikan label
ax.set_xlabel("Weather", fontsize=13, fontweight="bold")
ax.set_ylabel("Count", fontsize=13, fontweight="bold")
ax.set_title("Weather Distribution", fontsize=14, fontweight="bold")
plt.show()
```

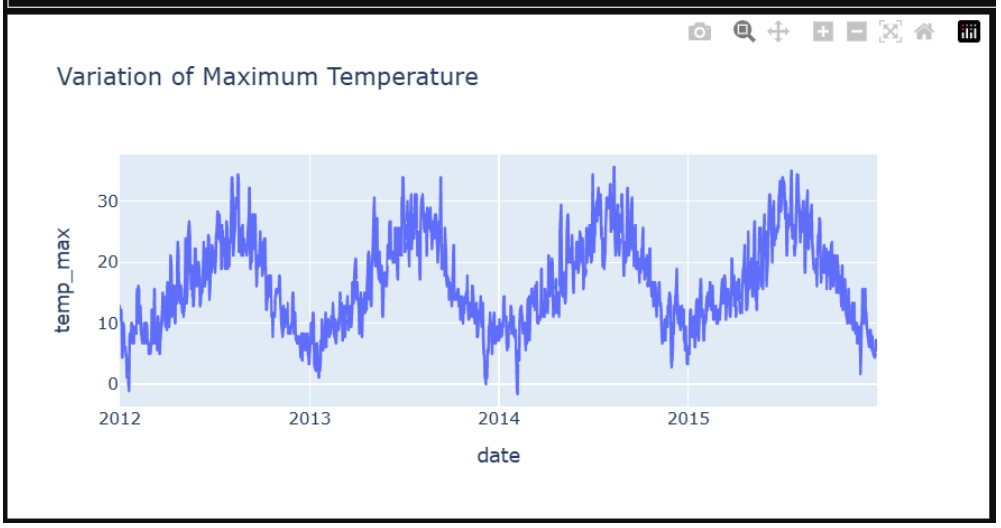
This is the output, using bar chart illustrates the distribution of different weather conditions, with rain and sun being the most frequent, each occurring over 600 times.

The fog, drizzle, and snow categories appear significantly less often, with snow being the least common.



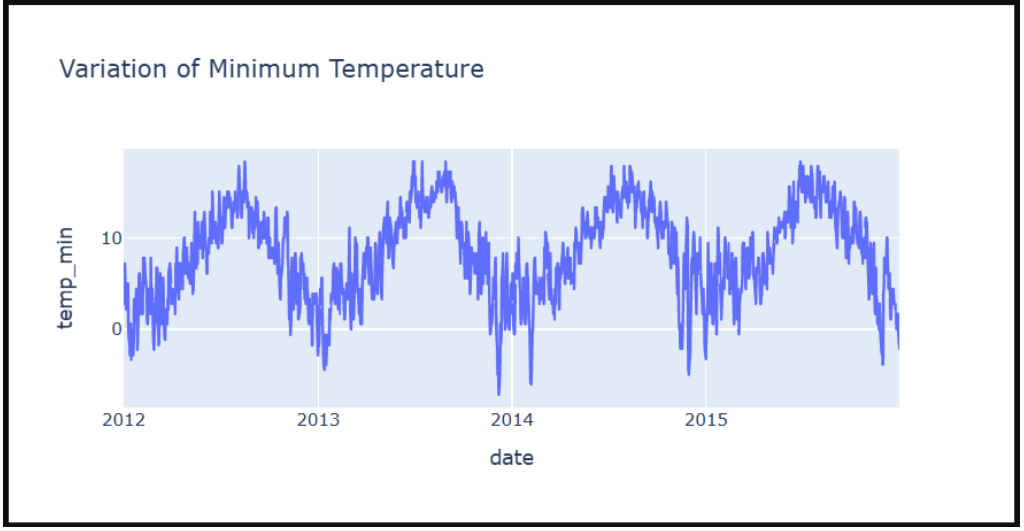
This line chart illustrates the variation in maximum temperature based on the analyzed dataset. The graph is created using Plotly Express, with the x-axis representing the date and the y-axis representing the maximum temperature (temp_max). The observed pattern shows fluctuations in temperature throughout the year, with an increasing trend during summer and a decline during winter, repeating annually. This visualization helps in understanding the pattern of maximum temperature changes over time and provides insights into seasonal trends in the observed weather data.

```
px.line(data_frame = df,
        x = 'date',
        y = 'temp_max',
        title = 'Variation of Maximum Temperature')
```

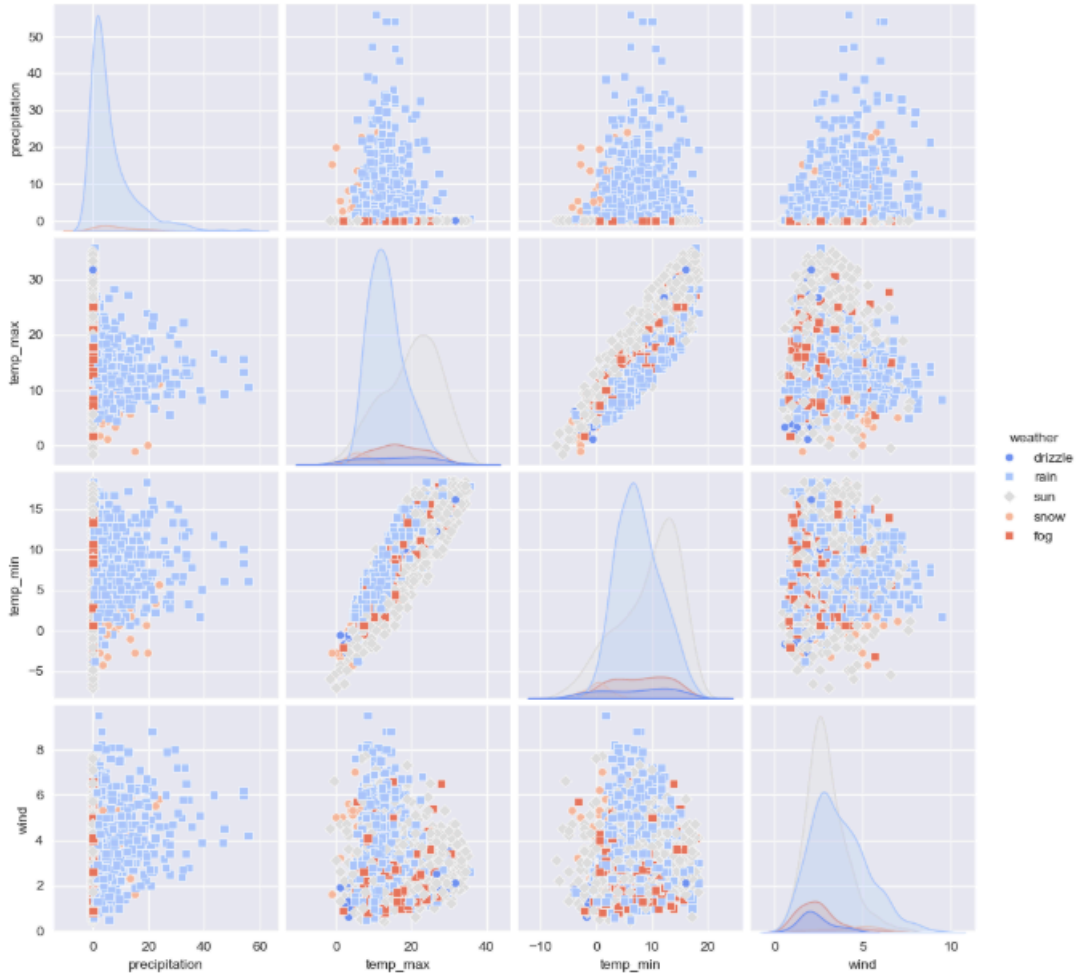


This line chart represents the variation in minimum temperature using Plotly Express. The x-axis indicates the date, while the y-axis displays the minimum temperature (temp_min). The graph exhibits periodic fluctuations, with an increase in minimum temperatures during summer months and a decline in winter, forming a cyclical pattern over the years. The data reveals consistent seasonal trends, showing that temperatures generally rise and fall in an expected pattern each year. This visualization helps in analyzing long-term changes in minimum temperature and identifying potential climate patterns.

```
px.line(data_frame = df,
        x = 'date',
        y = 'temp_min',
        title = 'Variation of Minimum Temperature')
```



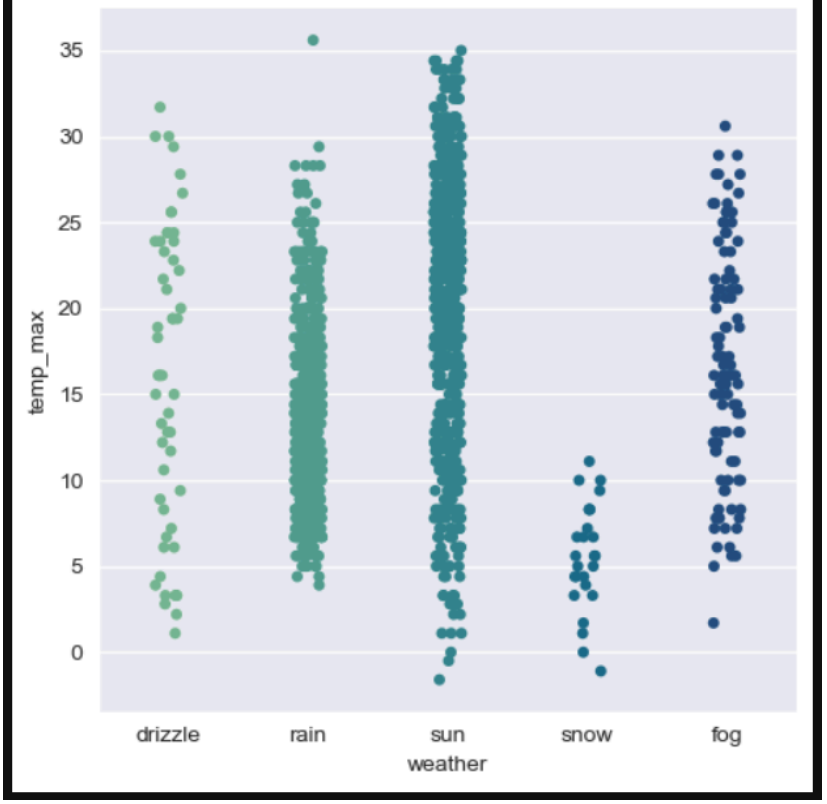
This output shows a pair plot that displays the relationship between several weather variables, including precipitation, maximum temperature (temp_max), minimum temperature (temp_min), and wind speed. Data points are categorized by weather type such as drizzle, rain, sun, snow, and fog, with different colors for each category. The diagonal graph shows the probability distribution for each variable, while the rest of the scatter plot graph illustrates the correlation between the variables. For example, there is a strong positive correlation between maximum temperature and minimum temperature. Also, the precipitation distribution shows that most of the precipitation is of low value. This visualization helps in understanding weather patterns and the relationships between environmental factors.



The output displays a categorical scatter plot (*catplot*) that visualizes the distribution of maximum temperature (*temp_max*) across different weather conditions, including drizzle, rain, sun, snow, and fog. Each dot represents an individual data point, and colors are applied using the *crest* palette. The plot shows that sunny weather has the highest temperature variation, reaching above 35°C, while snow has the lowest maximum temperatures, staying below 10°C. Rain and drizzle exhibit moderate temperature ranges, while fog has a wider spread, with some cases overlapping with both low and moderate temperatures. This visualization helps in understanding how temperature varies under different weather conditions.

```
plt.figure(figsize=(10,5))
sns.catplot(x='weather',y ='temp_max',data=df,palette="crest")
plt.show()
```

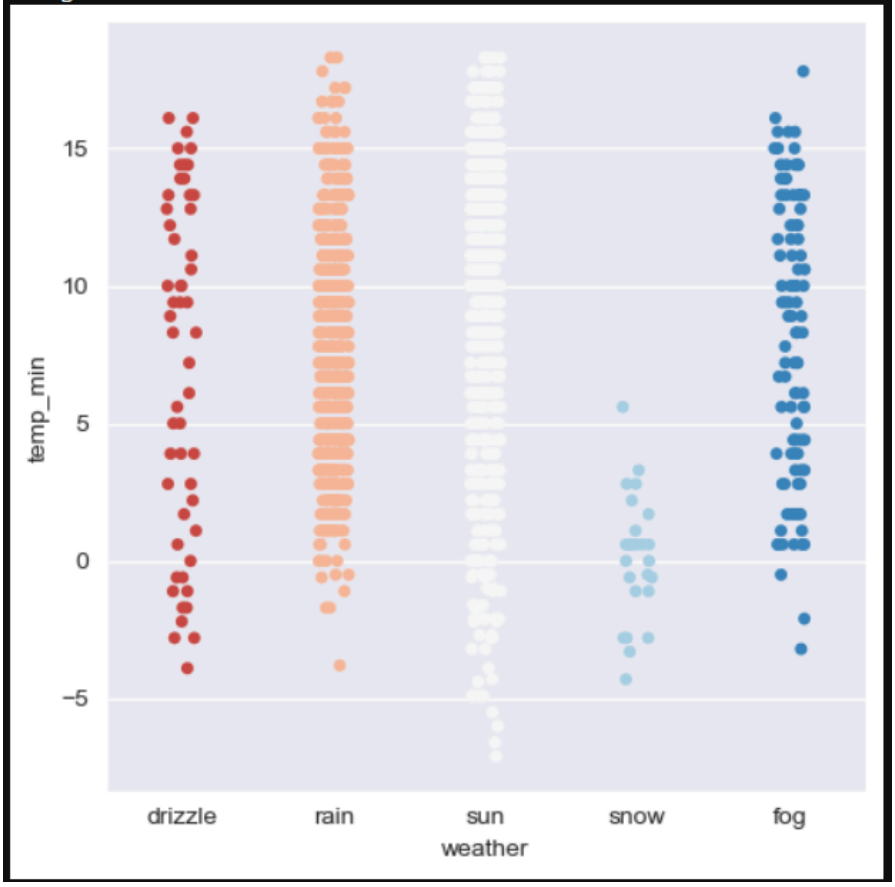
```
<Figure size 1000x500 with 0 Axes>
```



The output is a categorical scatter plot (*catplot*) that visualizes the distribution of minimum temperature (*temp_min*) across different weather conditions, including drizzle, rain, sun, snow, and fog. The colors are mapped using the *RdBu* palette, where red represents higher temperatures, blue represents lower temperatures, and white represents mid-range values. The plot indicates that drizzle and rain generally have higher minimum temperatures, while snow and fog have lower minimum temperatures, with snow showing values below 0°C. Sunny weather has a balanced distribution of minimum temperatures. This visualization helps in analyzing temperature variations under different weather conditions.

```
plt.figure(figsize=(10,5))
sns.catplot(x='weather',y ='temp_min',data=df,palette = "RdBu")
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```



5. Data Preprocessing

The purpose of data preprocessing is to clean and transform data to make it ready for use in machine learning models. This process includes handling missing values, encoding categorical data into numeric, normalization, and removal of irrelevant features. With proper preprocessing, the data becomes more structured, allowing the model to work more efficiently and produce more accurate predictions.

This code uses LabelEncoder from sklearn.preprocessing to convert categorical values in a specific column of the dataframe into numerical values. The function LABEL_ENCODING(c1) first initializes a LabelEncoder, then applies the fit_transform() method to transform categorical data into numbers, making it suitable for machine learning models. After the transformation, the code displays the unique encoded values, and the function is then called to convert the "weather" column.

```
def LABEL_ENCODING(c1):
    from sklearn import preprocessing
    label_encoder = preprocessing.LabelEncoder()
    df[c1]= label_encoder.fit_transform(df[c1])
    df[c1].unique()
LABEL_ENCODING("weather")
```

This code processes the dataframe by removing the 'date' column using .drop('date', axis=1), ensuring that this column is no longer used in the analysis. Next, the dataframe x is created by dropping the 'weather' column, meaning that x contains the independent features of the dataset. Meanwhile, the 'weather' column is stored in the variable y, which is likely the target variable in a machine learning model. With this separation, x can be used as input for the model, while y serves as the label to be predicted.

```
df = df.drop('date',axis=1)
```

```
x = df.drop('weather',axis=1)
y = df['weather']
```

6. Machine Learning Modelling

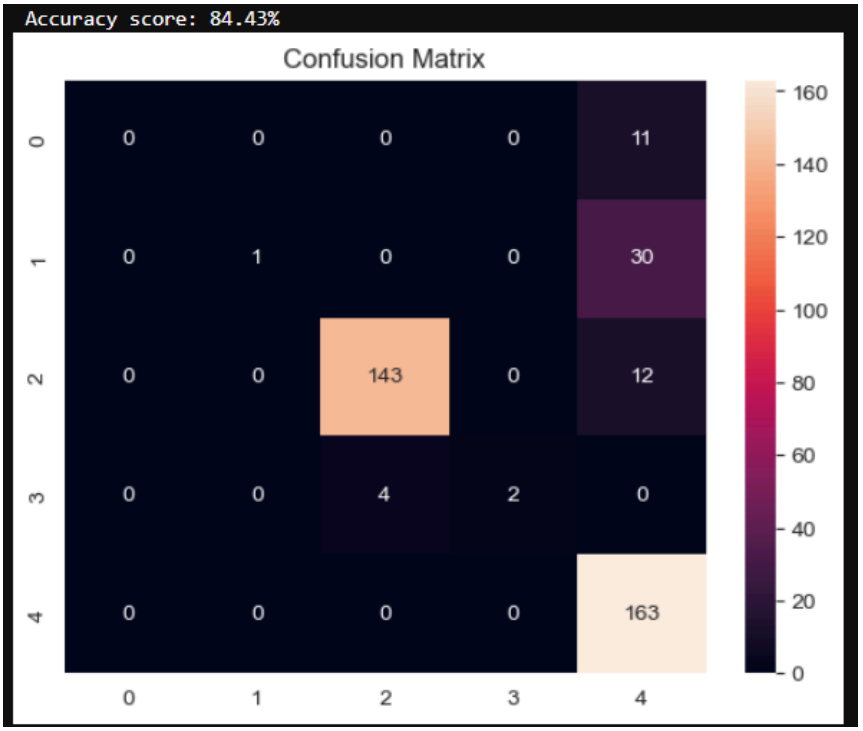
At the Machine Learning Modeling stage, this case uses Decision Tree, Random Forest, Gradient Boosting. Models are evaluated using metrics such as accuracy and precision, and can be improved through hyperparameter tuning before being applied to new data.

This code divides the dataset into 75% training data and 25% test data to keep the division results consistent. After that, feature normalization is performed to ensure that each feature has a mean of 0 and a standard deviation of 1.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

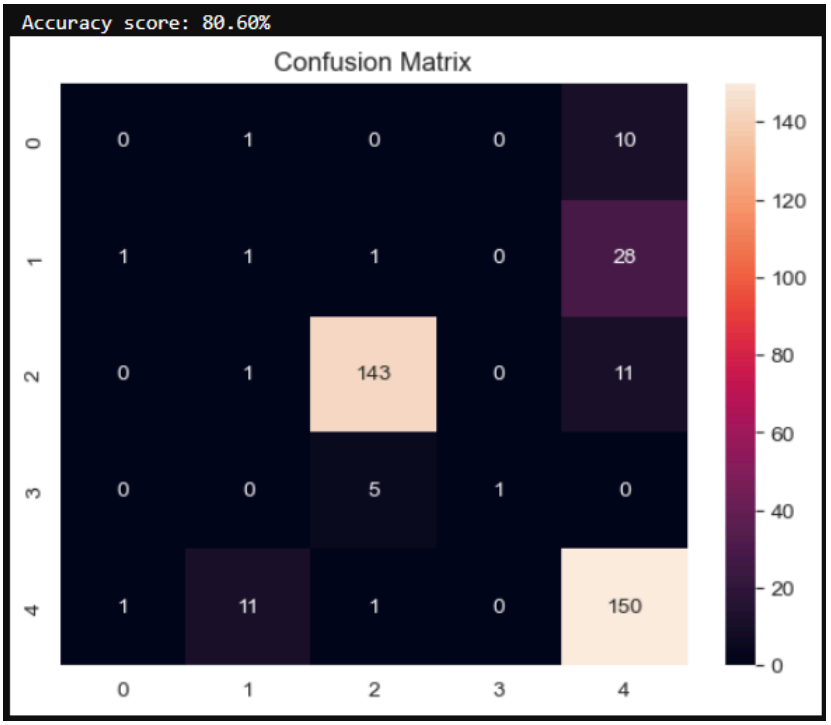
Random Forest



```
print(classification_report(y_test, y_pred_rf))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	1.00	0.03	0.06	31
2	0.97	0.92	0.95	155
3	1.00	0.33	0.50	6
4	0.75	1.00	0.86	163
accuracy			0.84	366
macro avg	0.75	0.46	0.47	366
weighted avg	0.85	0.84	0.80	366

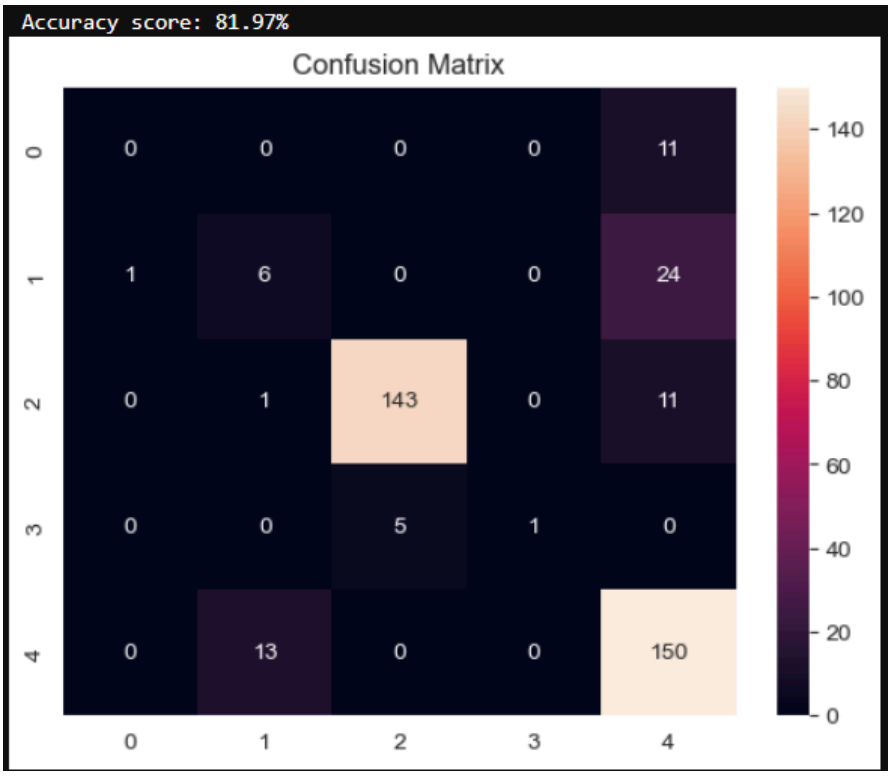
Gradient Boosting




```
print(classification_report(y_test, y_pred_gb))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.07	0.03	0.04	31
2	0.95	0.92	0.94	155
3	1.00	0.17	0.29	6
4	0.75	0.92	0.83	163
accuracy			0.81	366
macro avg	0.56	0.41	0.42	366
weighted avg	0.76	0.81	0.77	366

Decision Tree




```
print(classification_report(y_test, y_pred_dt))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.30	0.19	0.24	31
2	0.97	0.92	0.94	155
3	1.00	0.17	0.29	6
4	0.77	0.92	0.84	163
accuracy			0.82	366
macro avg	0.61	0.44	0.46	366
weighted avg	0.79	0.82	0.80	366

The evaluation results show that Random Forest performs the best with an accuracy of 0.844262, precision of 0.849145, recall of 0.844262, and an F1-score of 0.797627. The Decision Tree has a slightly lower accuracy of 0.819672, with a precision of 0.791826, recall of 0.819672, and an F1-score of 0.796513, indicating that this model is simpler but still competitive. Meanwhile, Gradient Boosting has an accuracy of 0.806011, precision of 0.761872, recall of 0.806011, and an F1-score of 0.774643, showing that while it captures complex patterns, its performance is slightly lower than that of Random Forest. Overall, Random Forest provides the best balance between accuracy and other evaluation metrics, making it the most optimal choice for this model.

	Accuracy	Precision	Recall	F1-score
Decision Tree	0.819672	0.791826	0.819672	0.796513
Random Forest	0.844262	0.849145	0.844262	0.797627
Gradient Boosting	0.806011	0.761872	0.806011	0.774643

7. Conclusion

This Weather Prediction project shows how machine learning techniques can be used to predict weather conditions more accurately. By applying various algorithms like a Decision Tree, Random Forest, and Gradient Boosting, the model can analyze weather patterns based on historical data and produce better predictions.

The evaluation results show that Random Forest has the best performance in terms of accuracy and precision, while other algorithms also provide competitive results. Data preprocessing, including normalization and proper feature selection, plays an important role in improving model performance.

This project proved that machine learning can be an effective tool in weather analysis and prediction. For further development, the model can be improved by using more data, fine-tuning hyperparameters, and using deep learning techniques for more precise results.