Table of Contents

1.0 Introduction

1.1 Problem Statement

With climate change increasingly influencing our current weather patterns, the ability to identify and classify extreme weather conditions is significant to our survivability. Out of the 19 project problems, the selected problem focuses on classifying meteorological weather into “normal” and “extreme” weather conditions.

1.2 Proposed Solution

The approach to the solution involves ignoring other types of extreme weather (heatwaves, air pollution, cyclones, etc.), as the dataset used in this project does not explicitly label any weather conditions as "extreme," and it relies on generic terms such as "partly cloudy," "sunny," or "misty." As shown in the figure below, the problem is that there are approximately 40 different descriptions of each weather condition, some of which are redundant or repeat the same point, making it difficult to reliably classify extreme weather conditions based solely on these descriptions. To ensure simplicity and consistency, "extreme" cases will be classified based on a threshold value for important meteorological indicators (temperature, UV index, precipitation, and air quality). This approach allows a more objective and scalable classification of the chosen problem statement.

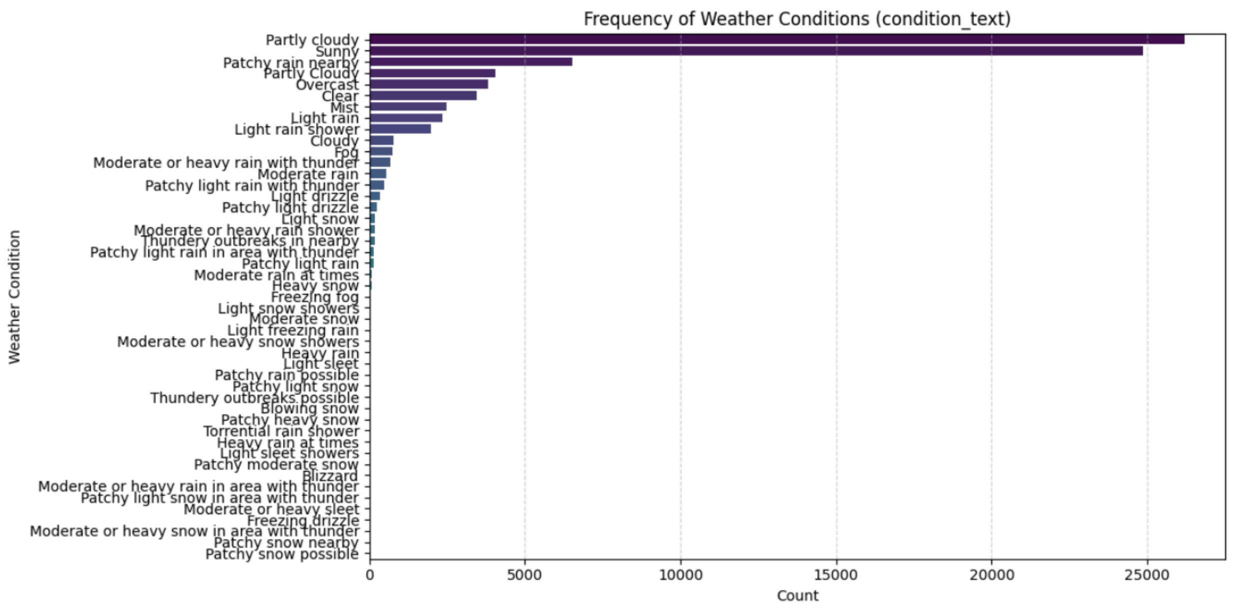


Figure 1. Count of Weather Conditions in the Dataset.

From there, key classes are narrowed down to those most relevant for classifying extreme weather data. After this selection, the data is processed, cleaned, and prepared for analysis and model training using Google Colab. To see if other machine learning (ML) models are applicable, two models were selected: logistic regression and random forest classifier. Using draw.io to create a flowchart, figure 2 describes this flow accordingly.

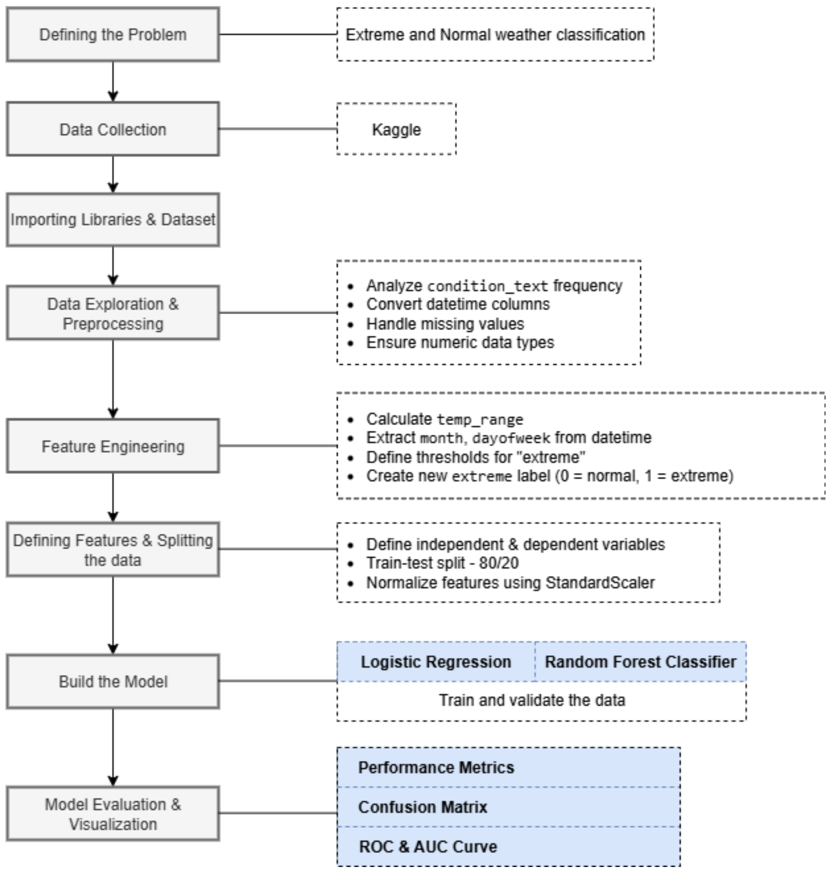

Defining the Problem
Extreme and Normal weather classification
Data Collection
Kaggle
Importing Libraries & Dataset
Data Exploration & Preprocessing
• Analyze condition_text frequency
• Convert datetime columns
• Handle missing values
• Ensure numeric data types
Feature Engineering
• Calculate temp_range
• Extract month, dayofweek from datetime
• Define thresholds for "extreme"
• Create new extreme label (0 = normal, 1 = extreme)
Defining Features & Splitting the data
• Define independent & dependent variables
• Train-test split - 80/20
• Normalize features using StandardScaler
Build the Model
Logistic Regression
Random Forest Classifier
Train and validate the data
Model Evaluation & Visualization
Performance Metrics
Confusion Matrix
ROC & AUC Curve

Figure 2. Flowchart of proposed solution.

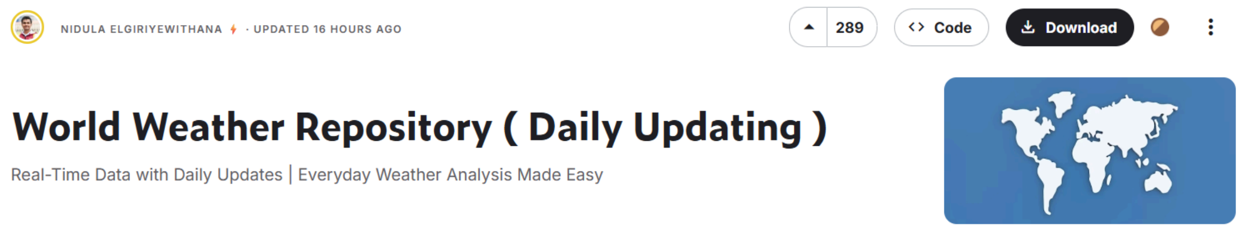
1.3 Objective

The goal of this assignment is to create a ML solution that is capable of accurately distinguishing between normal and extreme weather conditions using meteorological data. This assignment will describe the step-by-step processes and challenges encountered, followed by a discussion of improvements and suggestions. This enables us to apply our theoretical knowledge of machine learning to a practical problem.

2.0 Data Collection

2.1 Source of Data

Although a dataset was initially provided, an alternative was chosen from Kaggle, "World Weather Repository (Daily Updating)" by Nidula Elgiriyewithana. This was due to the initial dataset's lack of features and data (a lot of missing data). Therefore this new dataset was chosen for its large number of features and daily updates, which keep the data current and relevant to our problem statement.


NIDULA ELGIRIYEWITHANA · UPDATED 16 HOURS AGO

World Weather Repository (Daily Updating)

Real-Time Data with Daily Updates | Everyday Weather Analysis Made Easy

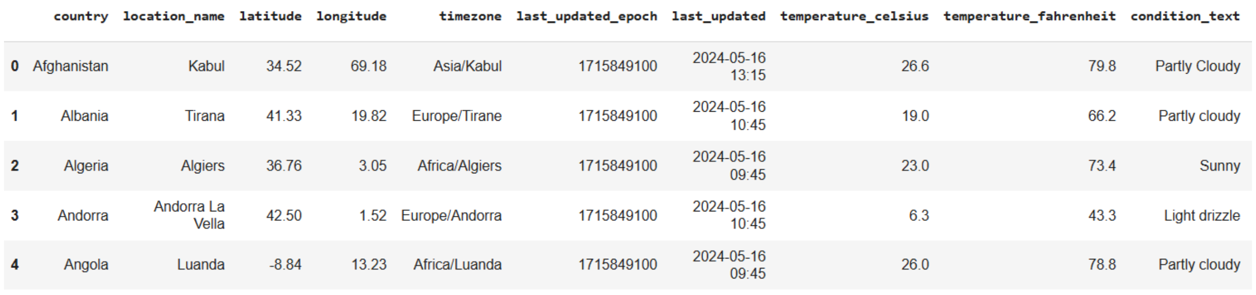
Figure 3. The Chosen Dataset from Kaggle.

2.2 Description of Dataset

This dataset, which started on August 29, 2023, provides daily weather information for capital cities around the world. Unlike forecast-based datasets, it includes actual observed conditions, making it ideal for training models on real-world patterns (Elgiriyewithana, 2025). It has more than 40 features, including:

- Temperature (actual and felt-like in Celsius and Fahrenheit)
- Wind (speed, direction, and gusts)
- Pressure, humidity, rainfall, and visibility
- Air quality measurements (e.g., PM2.5, PM10, ozone, and CO)
- Solar and lunar phases (moon phase, sunrise/sunset)

It currently has 80,866 rows of data, which is especially useful for studying global weather trends, detecting climate anomalies, and investigating the relationships between meteorological variables (Elgiriyewithana, 2025). The figure below displays a snapshot of the dataset, displaying the first 10 columns and 5 rows.



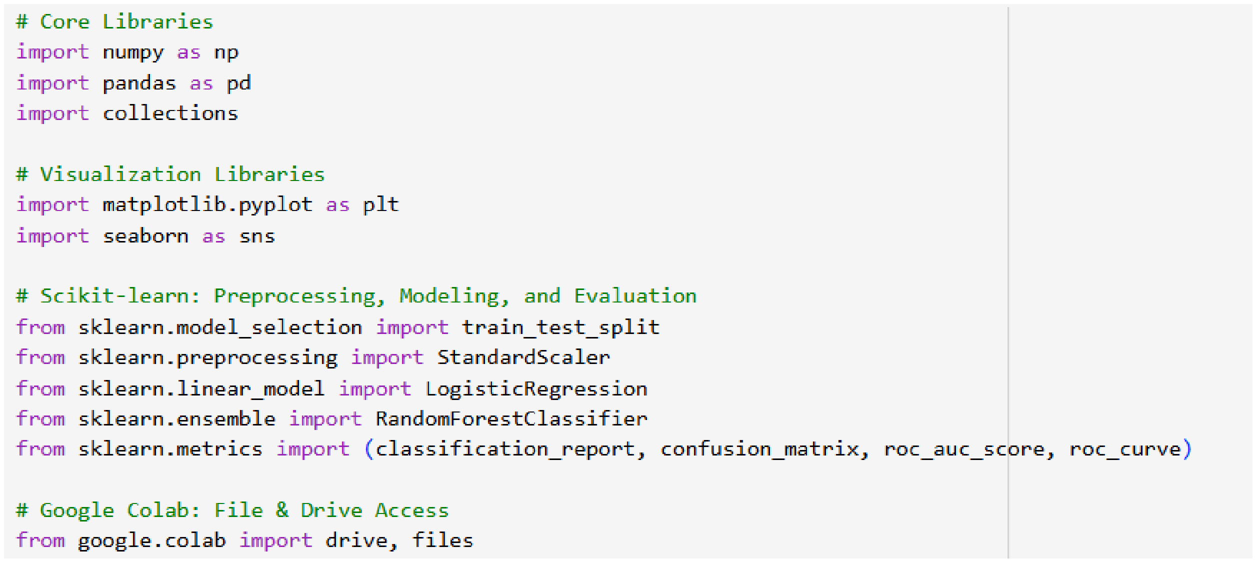
	country	location_name	latitude	longitude	timezone	last_updated_epoch	last_updated	temperature_celsius	temperature_fahrenheit	condition_text
0	Afghanistan	Kabul	34.52	69.18	Asia/Kabul	1715849100	2024-05-16 13:15	26.6	79.8	Partly Cloudy
1	Albania	Tirana	41.33	19.82	Europe/Tirane	1715849100	2024-05-16 10:45	19.0	66.2	Partly cloudy
2	Algeria	Algiers	36.76	3.05	Africa/Algiers	1715849100	2024-05-16 09:45	23.0	73.4	Sunny
3	Andorra	Andorra La Vella	42.50	1.52	Europe/Andorra	1715849100	2024-05-16 10:45	6.3	43.3	Light drizzle
4	Angola	Luanda	-8.84	13.23	Africa/Luanda	1715849100	2024-05-16 09:45	26.0	78.8	Partly cloudy

Figure 4. Snapshot of the dataset.

3.0 Data Preprocessing & Exploration

3.1 Importing Python Libraries

Without Python libraries like pandas, numpy, and matplotlib, exploring and visualizing data would be much more difficult and time-consuming. As a result, importing libraries is an essential step in the data preprocessing stage. This process allows access to functions, classes, and variables from external files or modules (Bar-Gil, 2023). The code snippet below displays the libraries used in this solution.

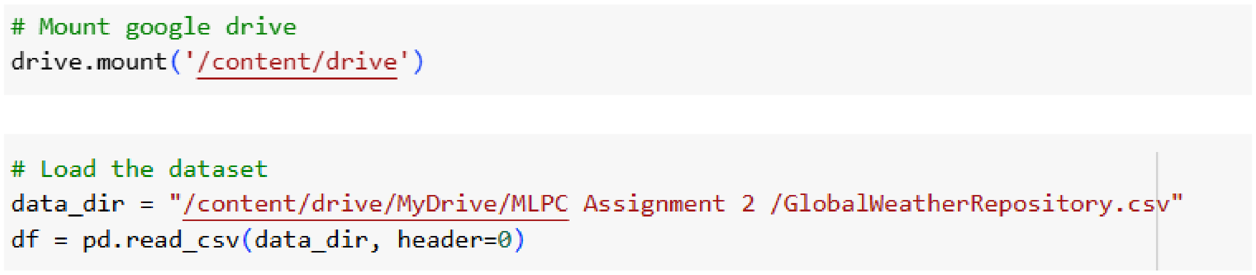


```python
# Core Libraries
import numpy as np
import pandas as pd
import collections

# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn: Preprocessing, Modeling, and Evaluation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix, roc_auc_score, roc_curve)

# Google Colab: File & Drive Access
from google.colab import drive, files
```

Figure 5. Code Snippet of Imported Libraries.

3.2 Import Dataset

Before data exploration can begin, the dataset must first be imported. In this solution, the dataset is stored in Google Drive and is loaded and read in Google Colab using the command as demonstrated in the figure below.



```
# Mount google drive
drive.mount('/content/drive')
```

```
# Load the dataset
data_dir = "/content/drive/MyDrive/MLPC Assignment 2 /GlobalWeatherRepository.csv"
df = pd.read_csv(data_dir, header=0)
```

Figure 6. Code Snippet of Importing and Loading Dataset.

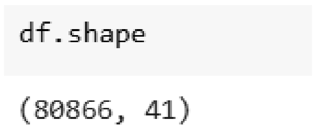
3.3 Data Exploration

Since the dataset contains over 80,000 rows, it is quite difficult to visualize the data by just looking at the .csv file. Therefore, in this section, basic data exploration techniques, which are shown in the figures below, are used to understand the structure, size, and format of the dataset.

This process increases familiarity with the dataset by providing an overview of its dimensions, data types, and source. Furthermore, it allows for a better understanding of the significance and meaning of each meteorological variable, (temperature, humidity, wind speed, and air quality indicators). These steps are essential for identifying key patterns, directing data cleaning efforts, and selecting the most impactful features for successful model development (GeeksforGeeks, 2024).

I. Check dataset dimensions.

Count the number of rows and columns in the dataset.



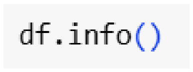
```
df.shape
```

```
(80866, 41)
```

Figure 7. Code Snippet & Output of Checking Dataset Dimensions.

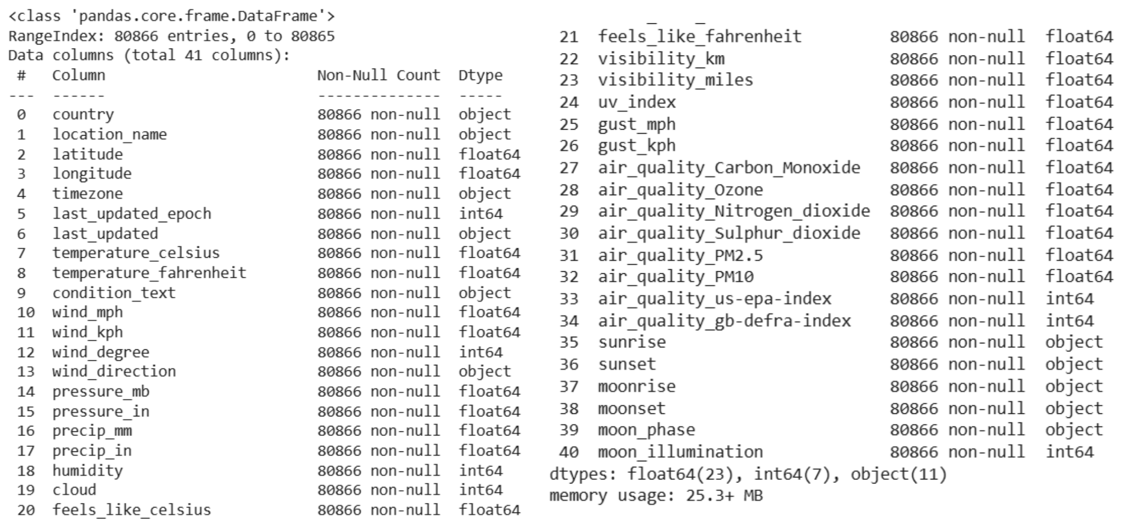
II. Dataset summary.

A summary of the dataset which includes number of entries, column names, number of non-null values in each column, data types of each column and memory usage.



```
df.info()
```

Figure 8. Code Snippet of Dataset Summary.

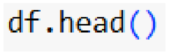


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80866 entries, 0 to 80865
Data columns (total 41 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   country                       80866 non-null  object
 1   location_name                 80866 non-null  object
 2   latitude                      80866 non-null  float64
 3   longitude                     80866 non-null  float64
 4   timezone                      80866 non-null  object
 5   last_updated_epoch            80866 non-null  int64
 6   last_updated                  80866 non-null  object
 7   temperature_celsius           80866 non-null  float64
 8   temperature_fahrenheit        80866 non-null  float64
 9   condition_text                80866 non-null  object
 10  wind_mph                      80866 non-null  float64
 11  wind_kph                      80866 non-null  float64
 12  wind_degree                   80866 non-null  int64
 13  wind_direction                80866 non-null  object
 14  pressure_mb                   80866 non-null  float64
 15  pressure_in                   80866 non-null  float64
 16  precip_mm                     80866 non-null  float64
 17  precip_in                     80866 non-null  float64
 18  humidity                      80866 non-null  int64
 19  cloud                         80866 non-null  int64
 20  feels_like_celsius            80866 non-null  float64
 21  feels_like_fahrenheit         80866 non-null  float64
 22  visibility_km                 80866 non-null  float64
 23  visibility_miles              80866 non-null  float64
 24  uv_index                      80866 non-null  float64
 25  gust_mph                      80866 non-null  float64
 26  gust_kph                      80866 non-null  float64
 27  air_quality_Carbon_Monoxide   80866 non-null  float64
 28  air_quality_Ozone             80866 non-null  float64
 29  air_quality_Nitrogen_dioxide  80866 non-null  float64
 30  air_quality_Sulphur_dioxide   80866 non-null  float64
 31  air_quality_PM2.5             80866 non-null  float64
 32  air_quality_PM10              80866 non-null  float64
 33  air_quality_us-epa-index      80866 non-null  int64
 34  air_quality_gb-defra-index    80866 non-null  int64
 35  sunrise                       80866 non-null  object
 36  sunset                        80866 non-null  object
 37  moonrise                      80866 non-null  object
 38  moonset                       80866 non-null  object
 39  moon_phase                    80866 non-null  object
 40  moon_illumination             80866 non-null  int64
dtypes: float64(23), int64(7), object(11)
memory usage: 25.3+ MB
```

Figure 9. Output of Dataset Summary.

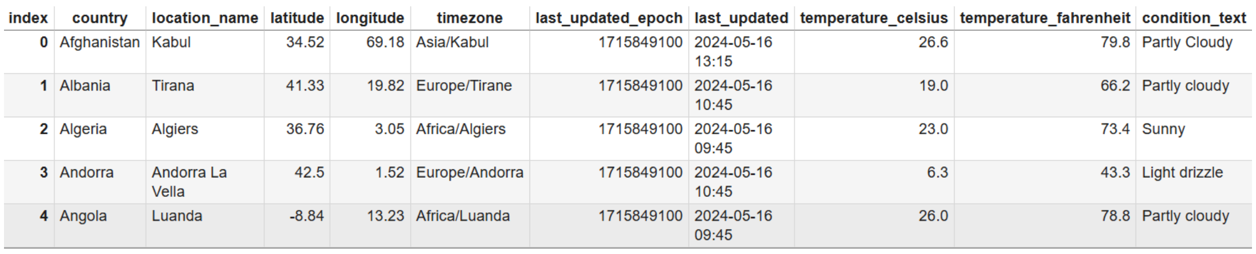
III. Preview of the first 5 rows of the dataset.

To display the first 5 rows to understand the structure and sample values.



```
df.head()
```

Figure 10. Code Snippet of Previewing First 5 Rows.

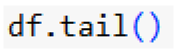


index	country	location_name	latitude	longitude	timezone	last_updated_epoch	last_updated	temperature_celsius	temperature_fahrenheit	condition_text
0	Afghanistan	Kabul	34.52	69.18	Asia/Kabul	1715849100	2024-05-16 13:15	26.6	79.8	Partly Cloudy
1	Albania	Tirana	41.33	19.82	Europe/Tirane	1715849100	2024-05-16 10:45	19.0	66.2	Partly cloudy
2	Algeria	Algiers	36.76	3.05	Africa/Algiers	1715849100	2024-05-16 09:45	23.0	73.4	Sunny
3	Andorra	Andorra La Vella	42.5	1.52	Europe/Andorra	1715849100	2024-05-16 10:45	6.3	43.3	Light drizzle
4	Angola	Luanda	-8.84	13.23	Africa/Luanda	1715849100	2024-05-16 09:45	26.0	78.8	Partly cloudy

Figure 11. Output of First 5 Rows and 10 Columns.

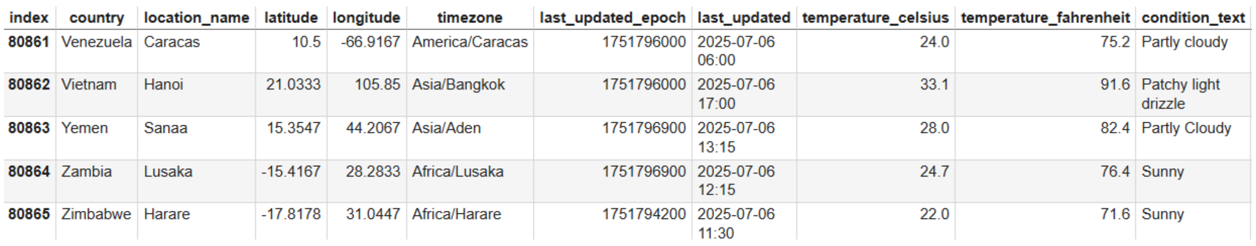
IV. Preview of the last 5 rows of the dataset.

To make sure that data is consistent throughout the whole dataset.



```
df.tail()
```

Figure 12. Code Snippet of Previewing Last 5 Rows.

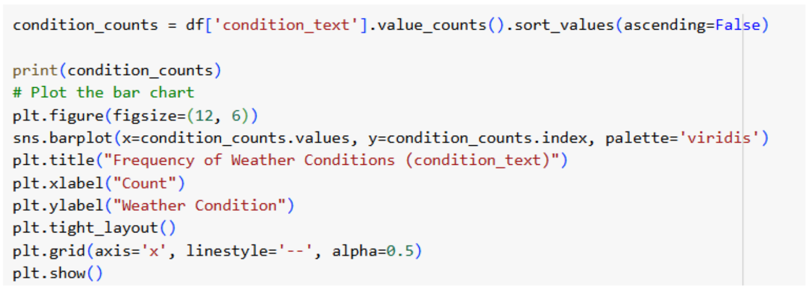


index	country	location_name	latitude	longitude	timezone	last_updated_epoch	last_updated	temperature_celsius	temperature_fahrenheit	condition_text
80861	Venezuela	Caracas	10.5	-66.9167	America/Caracas	1751796000	2025-07-06 06:00	24.0	75.2	Partly cloudy
80862	Vietnam	Hanoi	21.0333	105.85	Asia/Bangkok	1751796000	2025-07-06 17:00	33.1	91.6	Patchy light drizzle
80863	Yemen	Sanaa	15.3547	44.2067	Asia/Aden	1751796900	2025-07-06 13:15	28.0	82.4	Partly Cloudy
80864	Zambia	Lusaka	-15.4167	28.2833	Africa/Lusaka	1751796900	2025-07-06 12:15	24.7	76.4	Sunny
80865	Zimbabwe	Harare	-17.8178	31.0447	Africa/Harare	1751794200	2025-07-06 11:30	22.0	71.6	Sunny

Figure 13. Output of Last 5 Rows and 10 Columns.

V. Count unique weather conditions

Counting how many conditions are there under the column: condition_text, and displaying the results using a bar graph.



```
condition_counts = df['condition_text'].value_counts().sort_values(ascending=False)

print(condition_counts)
# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x=condition_counts.values, y=condition_counts.index, palette='viridis')
plt.title("Frequency of Weather Conditions (condition_text)")
plt.xlabel("Count")
plt.ylabel("Weather Condition")
plt.tight_layout()
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.show()
```

Figure 14. Code Snippet of Counting Unique Weather Conditions.

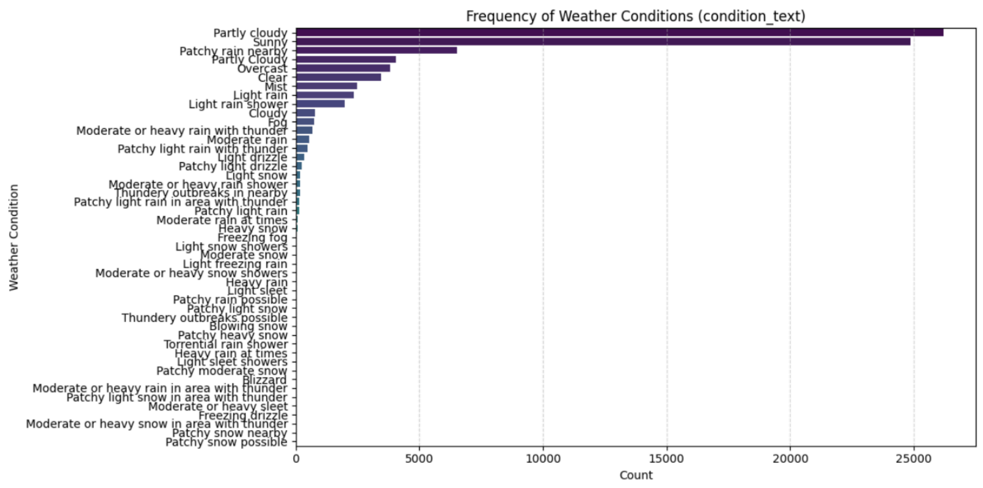


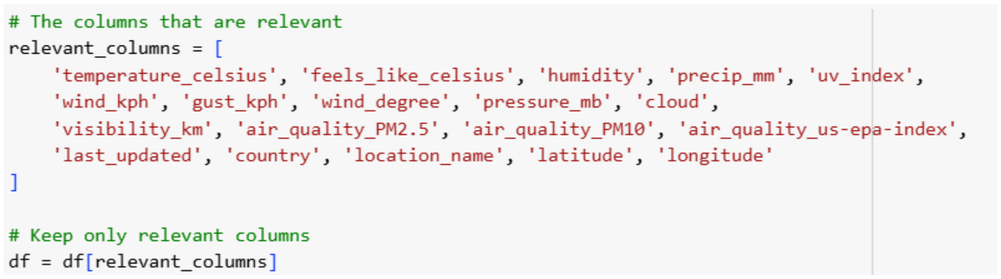
Figure 15. Output of Counting Unique Weather Conditions.

3.4 Data Cleaning

This section will cover data cleaning, which prepares the dataset for model training and tackles many problems in the dataset that might be caused by humans or machines (missing values, duplicated data, outliers, or irrelevant data) (Alotaibi et al., 2023). Therefore, this step is important because the selected dataset may contain some human/machine error. In this solution, this process involves identifying and handling repeated columns, incorrect data types, missing values, and duplicated data. By cleaning our dataset, it helps improve the model's performance while still being accurate and reliable.

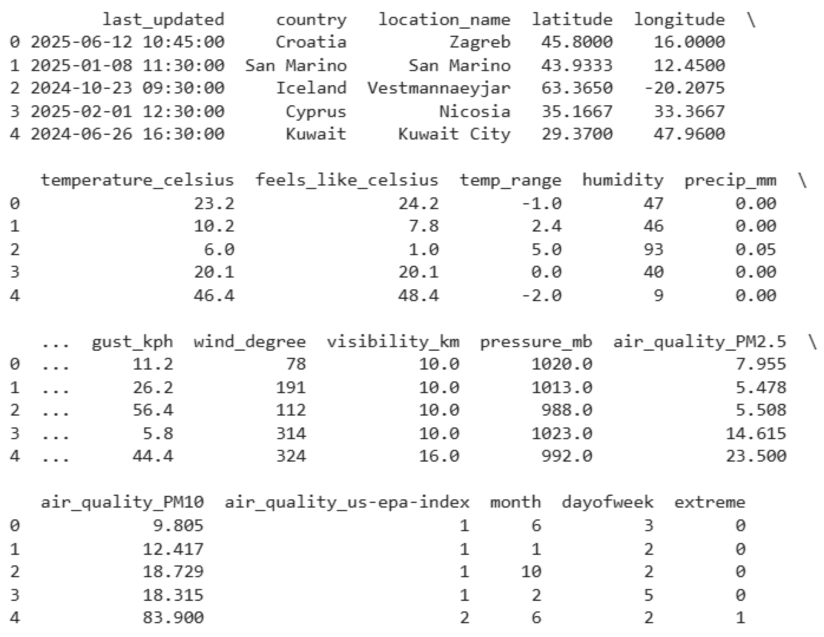
3.4.1 Dropping Repeated Columns

The first step to this solution was to identify and remove redundant or irrelevant columns in the dataset. As previously mentioned, there are more than 40 classes in this dataset, therefore redundancy is unavoidable. For example, the columns 'temperature_celsius' and 'temperature_fahrenheit' represent the same measurement but in different units. As a result, leaving both columns would make it difficult for the model to distinguish between the two temperatures. To avoid information duplication, this solution uses metric units rather than US customary systems. The figures below display the code snippet of this step.



```python
# The columns that are relevant
relevant_columns = [
    'temperature_celsius', 'feels_like_celsius', 'humidity', 'precip_mm', 'uv_index',
    'wind_kph', 'gust_kph', 'wind_degree', 'pressure_mb', 'cloud',
    'visibility_km', 'air_quality_PM2.5', 'air_quality_PM10', 'air_quality_us-epa-index',
    'last_updated', 'country', 'location_name', 'latitude', 'longitude'
]

# Keep only relevant columns
df = df[relevant_columns]
```

Figure 16. Code Snippet of Dropping Irrelevant Columns.



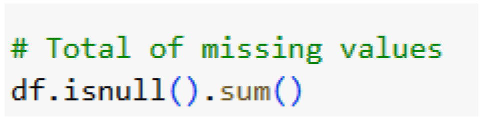
```
         last_updated     country    location_name  latitude  longitude  \
0 2025-06-12 10:45:00     Croatia           Zagreb   45.8000    16.0000
1 2025-01-08 11:30:00  San Marino       San Marino   43.9333    12.4500
2 2024-10-23 09:30:00     Iceland  Vestmannaeyjar   63.3650   -20.2075
3 2025-02-01 12:30:00      Cyprus          Nicosia   35.1667    33.3667
4 2024-06-26 16:30:00      Kuwait      Kuwait City   29.3700    47.9600

   temperature_celsius  feels_like_celsius  temp_range  humidity  precip_mm  \
0                 23.2                24.2        -1.0        47       0.00
1                 10.2                 7.8         2.4        46       0.00
2                  6.0                 1.0         5.0        93       0.05
3                 20.1                20.1         0.0        40       0.00
4                 46.4                48.4        -2.0         9       0.00

   ...  gust_kph  wind_degree  visibility_km  pressure_mb  air_quality_PM2.5  \
0  ...      11.2           78           10.0       1020.0              7.955
1  ...      26.2          191           10.0       1013.0              5.478
2  ...      56.4          112           10.0        988.0              5.508
3  ...       5.8          314           10.0       1023.0             14.615
4  ...      44.4          324           16.0        992.0             23.500

   air_quality_PM10  air_quality_us-epa-index  month  dayofweek  extreme
0             9.805                         1      6          3        0
1            12.417                         1      1          2        0
2            18.729                         1     10          2        0
3            18.315                         1      2          5        0
4            83.900                         2      6          2        1
```

Figure 17. Output of Dropping Irrelevant Columns.

Using the same code in data exploration (df.shape and df.head()), the new structure of the dataset can be identified. Now, instead of 41 columns, the data is reduced to 23 main features that our model will focus on.

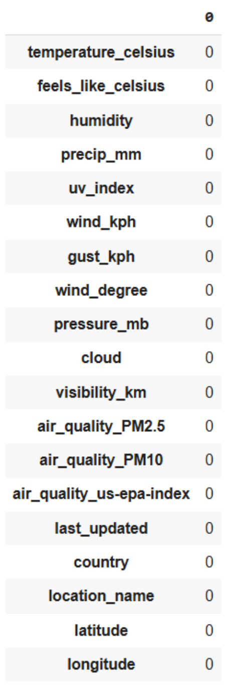
3.4.2 Checking for Missing Values

With missing data being unchecked, it can affect the accuracy of the ML models, therefore this is another important step to ensure the reliability of the model (Alotaibi et al., 2023). This step involves scanning the dataset for any null or missing entries. As shown in the figure below, the dataset used in this project does not contain any missing values. Hence, no imputation techniques (such as .fillna() or .dropna()) are required.



```
# Total of missing values
df.isnull().sum()
```

Figure 18. Code Snippet of Identifying Missing Values.

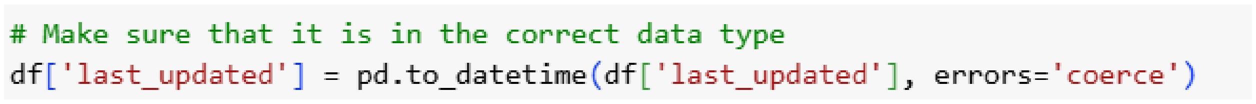


```
                            0
temperature_celsius         0
feels_like_celsius          0
humidity                    0
precip_mm                   0
uv_index                    0
wind_kph                    0
gust_kph                    0
wind_degree                 0
pressure_mb                 0
cloud                       0
visibility_km               0
air_quality_PM2.5           0
air_quality_PM10            0
air_quality_us-epa-index    0
last_updated                0
country                     0
location_name               0
latitude                    0
longitude                   0
```

Figure 19. Output of Identified Missing Values.

3.4.3 Ensuring Correct Data Types

In Section 3.3 (Concise Summary), the last_updated column was identified as being in string format. To ensure it is in the appropriate data type, it is converted to datetime using Pandas. Ensuring correct data types for each column is essential during the feature engineering stage. The figure below displays the code snippet for this step.

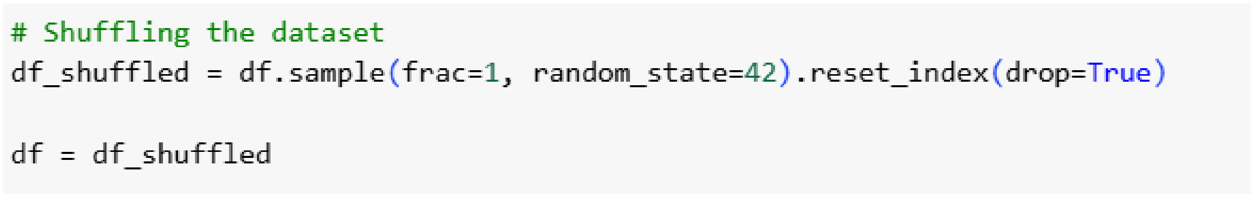


```python
# Make sure that it is in the correct data type
df['last_updated'] = pd.to_datetime(df['last_updated'], errors='coerce')
```

Figure 20. Code Snippet of Parsing last_updated into datetime.

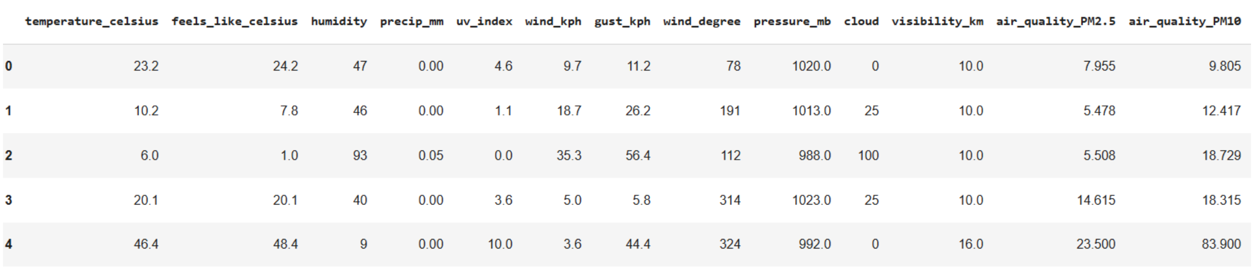
3.4.4 Shuffling dataset

Shuffling is performed to randomize the order of the dataset. This is especially important to prevent the model from learning any biases or unintended sequence patterns in the original data. Random shuffling ensures that both the training and test sets receive a fair and diverse representation of the data (Dutta, 2024). As shown in the code snippet below, this solution applies this technique to reduce the risk of overfitting.



```
# Shuffling the dataset
df_shuffled = df.sample(frac=1, random_state=42).reset_index(drop=True)

df = df_shuffled
```

Figure 21. Code Snippet for Shuffling Data.



	temperature_celsius	feels_like_celsius	humidity	precip_mm	uv_index	wind_kph	gust_kph	wind_degree	pressure_mb	cloud	visibility_km	air_quality_PM2.5	air_quality_PM10
0	23.2	24.2	47	0.00	4.6	9.7	11.2	78	1020.0	0	10.0	7.955	9.805
1	10.2	7.8	46	0.00	1.1	18.7	26.2	191	1013.0	25	10.0	5.478	12.417
2	6.0	1.0	93	0.05	0.0	35.3	56.4	112	988.0	100	10.0	5.508	18.729
3	20.1	20.1	40	0.00	3.6	5.0	5.8	314	1023.0	25	10.0	14.615	18.315
4	46.4	48.4	9	0.00	10.0	3.6	44.4	324	992.0	0	16.0	23.500	83.900

Figure 22. Output for Shuffling Data.

4.0 Feature Engineering

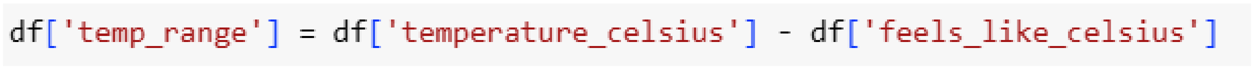
Just like humans, ML models require data to be in a specific format to be properly read and processed. Therefore, feature engineering plays a crucial role in transforming raw or unreadable data into a structured and meaningful format (Murel & Kavlakoglu, 2025). This section discusses the feature engineering techniques applied in this solution.

4.1 Derived Features

In this solution, two new features were derived to enhance the dataset:

I. **Temperature Range**

This feature was developed by calculating the difference between the measured temperature (temperature_celsius) and the perceived temperature (feels_like_celsius). This captures how different the weather feels from actual readings, which can be affected by wind or humidity. The code snippet for this step can be seen in the figure below.

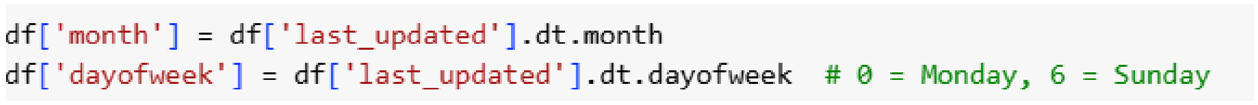


```
df['temp_range'] = df['temperature_celsius'] - df['feels_like_celsius']
```

Figure 23. Code Snippet for Temperature Range.

II. Datetime Extraction

The last_updated column, which has been converted in datetime format in the previous step, was used to derive new features, the month (1-12) and dayofweek (0 = Monday, 6 = Sunday). This extraction will help identify patterns in weather changes over weekdays vs weekends. Figure 24 shows the code snippet for this step.



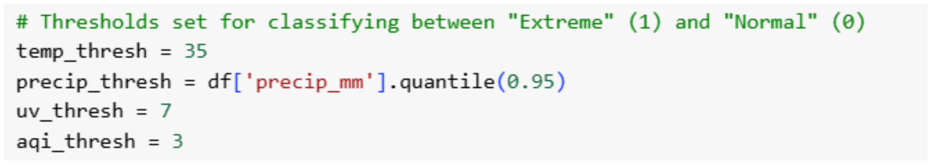
```
df['month'] = df['last_updated'].dt.month
df['dayofweek'] = df['last_updated'].dt.dayofweek  # 0 = Monday, 6 = Sunday
```

Figure 24. Code Snippet for Datetime Extraction.

4.2 Labeling Extreme Weather.

To train a model that classifies weather as normal or extreme, a binary target label (extreme) was introduced. Since the dataset does not explicitly define what constitutes "extreme" weather, domain knowledge and logical thresholds were used to label the data:

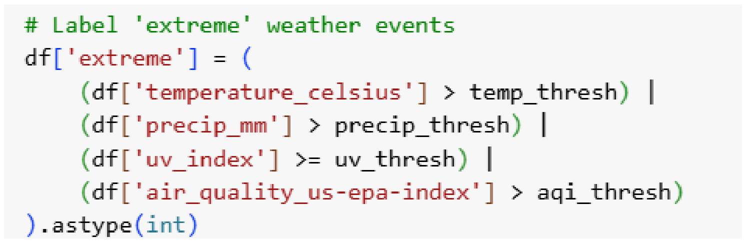
1. Temperature Threshold: Above 35°C
2. UV Index Threshold: Equal to or greater than 7
3. Precipitation Threshold: Top 5% of values (95th percentile)
4. Air Quality Index Threshold: Above level 3 on the U.S. EPA scale



```
# Thresholds set for classifying between "Extreme" (1) and "Normal" (0)
temp_thresh = 35
precip_thresh = df['precip_mm'].quantile(0.95)
uv_thresh = 7
aqi_thresh = 3
```

Figure 25. Code Snippet for Thresholds.

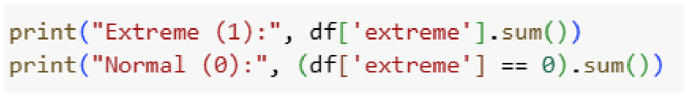
If any of these thresholds are met, the weather is classified as extreme (1). Otherwise, it is labeled normal (0). This step is also known as Label Encoding, which is a technique for converting categorical data into binary labels such as 1 and 0. This technique is demonstrated in the code snippet below.



```
# Label 'extreme' weather events
df['extreme'] = (
    (df['temperature_celsius'] > temp_thresh) |
    (df['precip_mm'] > precip_thresh) |
    (df['uv_index'] >= uv_thresh) |
    (df['air_quality_us-epa-index'] > aqi_thresh)
).astype(int)
```

Figure 26. Code Snippet for Label Encoding.

The code below counts the number of normal (0) and extreme (1) weather cases and then shows them in a bar chart to make it easier to understand.



```
print("Extreme (1):", df['extreme'].sum())
print("Normal (0):", (df['extreme'] == 0).sum())
```

Figure 27. Code Snippet for Printing out the count.

```
Extreme (1): 30065
Normal (0): 50801
```

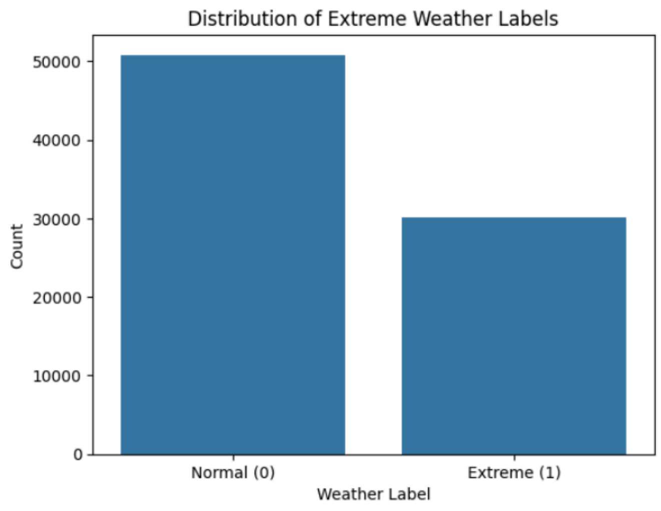
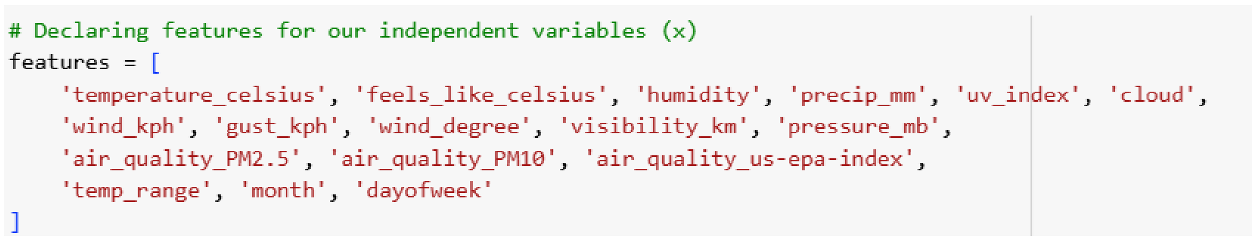


Figure 28. Output for Count of Normal and Extreme cases.

5.0 Model Preparation

5.1 Defining Features (X) & Target (y)

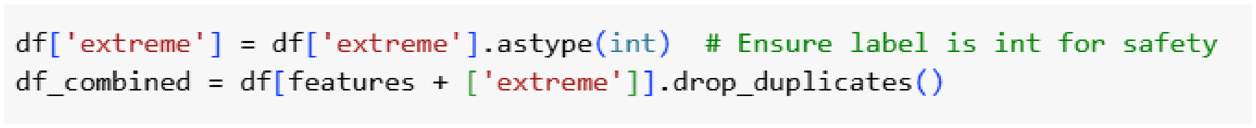
Before splitting the dataset into training and testing sets, it is important to define the independent (X) and dependent variables (y). The features (X) are the model's input variables for learning and making predictions (Mistry, 2025). In this case, they represent key weather indicators that influence whether a weather condition is considered normal or extreme. For the list of features it includes columns listed in the figure below.



```
# Declaring features for our independent variables (x)
features = [
    'temperature_celsius', 'feels_like_celsius', 'humidity', 'precip_mm', 'uv_index', 'cloud',
    'wind_kph', 'gust_kph', 'wind_degree', 'visibility_km', 'pressure_mb',
    'air_quality_PM2.5', 'air_quality_PM10', 'air_quality_us-epa-index',
    'temp_range', 'month', 'dayofweek'
]
```

Figure 29. List of Features for Independent Variables.

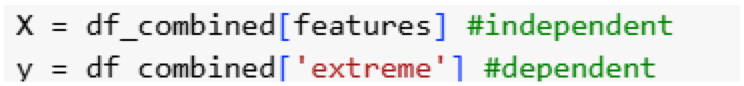
One optimization technique used in this solution was to improve model efficiency and reduce redundancy in the training data by dropping duplicated rows. These duplicated rows can cause bias in the model, leading to overfitting. Like in the figure below, it is shown that before dropping duplicates, the extreme label is explicitly converted to an integer type to ensure consistency during processing.



```
df['extreme'] = df['extreme'].astype(int)  # Ensure label is int for safety
df_combined = df[features + ['extreme']].drop_duplicates()
```

Figure 30. Code Snippet for Dropping Duplicated Rows.

As for our dependent variable (y), it is the extreme column as it includes the binary labels of extreme (1) and normal (0) that are assigned when the weather condition meets the defined thresholds.

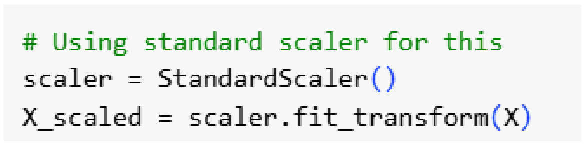


```
X = df_combined[features] #independent
y = df combined['extreme'] #dependent
```

Figure 31. Code Snippet for Defining X & y variables.

5.2 Feature Scaling

StandardScaler was used as another optimization technique in this solution because the ML model (Logistic Regression) is sensitive to the size of the input features. Therefore , without scaling, features with larger numerical ranges may have a greater impact on the model than those with smaller ranges. By standardizing the features to have a mean of 0 and a standard deviation of 1, the model can converge faster, perform more accurately, and train all features equally (Ahsan et al., 2021). This step also improves model stability and decreases the likelihood of overfitting due to dominant features, as illustrated in the figure below.

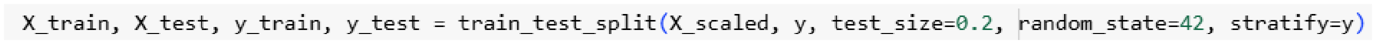


```
# Using standard scaler for this
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Figure 32. Code Snippet for Feature Scaling.

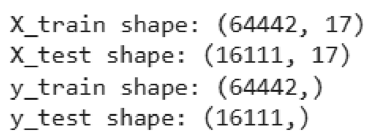
5.3 Train-Test Splitting

Using the Pareto Theory (80/20 rule) to split the dataset, it means that 80% of the data is used to train the ML model, while the remaining 20% is used for testing and validation, which is how well the model performs on unseen data. This split ensures the model has enough data to learn patterns while still being validated against a separate set, minimizing overfitting and improving generalizability (Mistry, 2025). The below code shows the code snippet, and the count for the training and testing sets. As shown below, the training set has 64,442 rows while the testing set has 16,111 rows.



```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

Figure 33. Code Snippet for Splitting the Dataset.



```
X_train shape: (64442, 17)
X_test shape: (16111, 17)
y_train shape: (64442,)
y_test shape: (16111,)
```

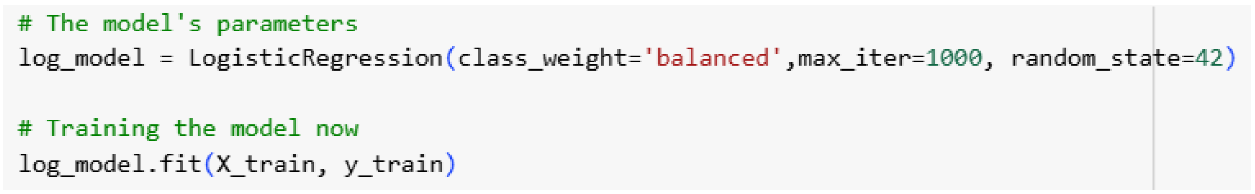
Figure 34. Output for Splitting the Dataset.

6.0 Model Building & Evaluation

This project implemented and evaluated two ML models: logistic regression (LR) and random forest classification (RF). These models are trained on meteorological features to determine whether weather conditions are extreme or normal, after that their performance is compared against a variety of evaluation metrics to determine the best model for the task.

6.1 Logistic Regression

LR is a linear model widely used in binary classification. It uses the sigmoid function to calculate the probability of a sample belonging to a class, making it ideal for interpreting feature importance and understanding how each variable influences predictions (Lee, 2025). The LR model in this solution is initialized with the following parameters shown in the figure below:



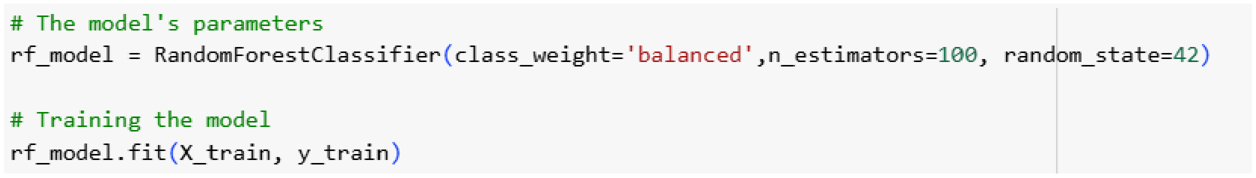
```
# The model's parameters
log_model = LogisticRegression(class_weight='balanced',max_iter=1000, random_state=42)

# Training the model now
log_model.fit(X_train, y_train)
```

Figure 35. Code Snippet of the Logistic Regression's Parameters.

Due to the dataset's imbalance of extreme and normal weather cases (shown in Section 4.2), there is a risk that the model will become biased toward predicting the majority class (normal). To address this, an optimization technique called class weighting is used in each model by setting class_weight='balanced'. This automatically adjusts the weights of the classes in inverse proportion to their frequencies, ensuring that both extreme and normal cases are adequately addressed during training. This approach reduces bias and the risk of overfitting to the majority class (Bakirarar & Elhan, 2023).

6.2 Random Forest Classification

RF is a ML model that constructs multiple decision trees and combines their outputs to improve accuracy and reduce overfitting. However it is prone to overfitting, as it is sensitive to outliers (Ibm, 2025). The RF model's parameters were initialized as the figure shown below.



```
# The model's parameters
rf_model = RandomForestClassifier(class_weight='balanced',n_estimators=100, random_state=42)

# Training the model
rf_model.fit(X_train, y_train)
```

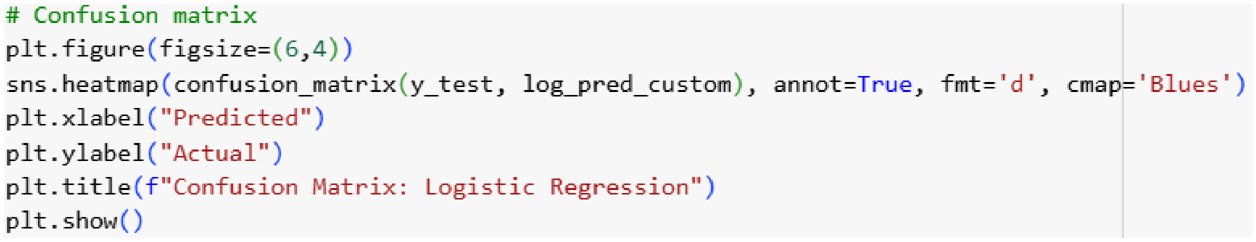
Figure 36. Code Snippet of the Random Forest Classification.

6.3 Evaluation Metrics & Comparison

To assess and compare the models' performance, the following metrics were used, these include confusion matrix, performance metrics (accuracy, precision, recall, F1-score), and lastly, ROC and AUC graph.

I. Confusion Matrix

First, the confusion matrix was plotted using the code shown in the figures below. The confusion matrix breaks down the model's predictions by displaying the number of true positives, false positives, true negatives, and false negatives during validation (testing). Understanding these values is important because they are directly used to calculate key performance metrics like precision, recall, and accuracy, which help determine how well the model is performing.



```
# Confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, log_pred_custom), annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix: Logistic Regression")
plt.show()
```

Figure 37. Code Snippet for Plotting the Confusion Matrix for Logistic Regression.

The same code can be used when plotting the confusion matrix for the RF model. As a result in the end, the confusion matrices for both models are shown in the figure below.

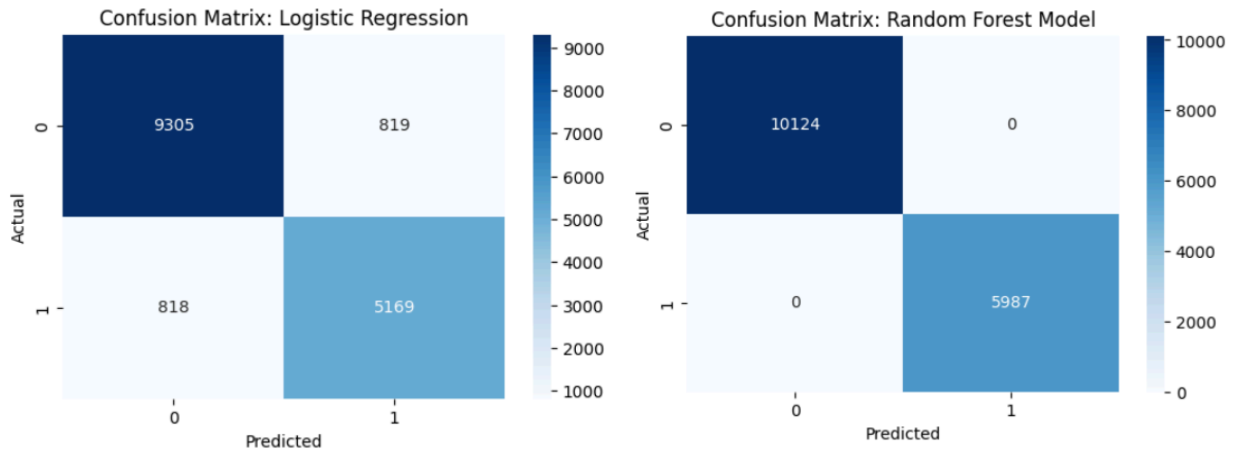


Figure 38. Confusion Matrix for both Models.

Comparison

When comparing the two models, the LR model has more false positives (819) and false negatives (818) than the RF model, which has zero of both. This suggests that the LR model has a harder time correctly identifying extreme weather cases, whereas the RF model was able to classify both normal and extreme conditions on the test set, however, such perfect performance may indicate a chance of overfitting.

II. Performance Metrics

Performance Metrics like accuracy, precision, recall and F1-Score all can be calculated using the values provided in the confusion matrix. In this solution True Positives stands for TP, True Negatives stands for TN, False Positives stands for FP and False Negatives stands for FN.

1. Accuracy

Using formula (1), the accuracy of the model can be calculated:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Logistic Regression

$$= \frac{9305 + 5169}{9305 + 5169 + 819 + 818}$$

$$= \frac{14{,}474}{16{,}111}$$

$$= 0.898 \; or \; 90\%$$

Random Forest Classification

$= \frac{10124 + 5987}{10124 + 5987 + 0 + 0}$

$= \frac{16,111}{16,111}$

$= 1 \; or \; 100\%$

Comparison

According to these calculations, the RF model has a perfect accuracy of 100%. However, such high accuracy may indicate overfitting, in which the model performs exceptionally well on training and test data but fails to generalize to unseen data.

2. **Precision**

Using formula (2) below, the Precision of the model can be calculated:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Logistic Regression

$= \frac{5169}{5169 + 819}$

$= \frac{5169}{5988}$

$= 0.863 \; or \; 86\%$

Random Forest Classifier

$= \frac{5987}{5987 + 0}$

$= \frac{5987}{5987}$

$= 1 \; or \; 100\%$

Comparison

Precision measures how many of the predicted "Extreme" cases were actually correct. It's useful when false positives are costly. The RF achieved a perfect precision of 100%, meaning all its extreme predictions were accurate. LR had a precision of 86.3%, with about 13.7% false positives. While RF seems better, a perfect score may suggest overfitting.

3. **Recall**

Using formula (3) below, the Recall of the model can be calculated:

$$Recall = \frac{TP}{TP + FN} \qquad (3)$$

Logistic Regression

$= \frac{5169}{5169 + 818}$

$= \frac{5169}{5987}$

$= 0.863 \; or \; 86\%$

Random Forest Classifier

$= \frac{5987}{5987 + 0}$

$= \frac{5987}{5987}$

$= 1 \; or \; 100\%$

Comparison

Recall measures how many actual "Extreme" cases were correctly identified. It's important when missing positive cases (false negatives) is risky. LR had a recall of 86.3%, meaning it missed about 13.7% of extreme cases. RF achieved 100% recall, but this perfect score could indicate overfitting.

4. **F1-Score**

Using formula (4) below, the F1-score of the model can be calculated:

$$FI - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \qquad (4)$$

Logistic Regression

$= \frac{2 \times 0.86 \times 0.86}{0.86 + 0.86}$

$= \frac{1.4792}{1.72}$

$= 0.86 \; or \; 86\%$

Random Forest Classifier

$$= \frac{2 \times 1 \times 1}{1 + 1}$$

$$= \frac{2}{2}$$

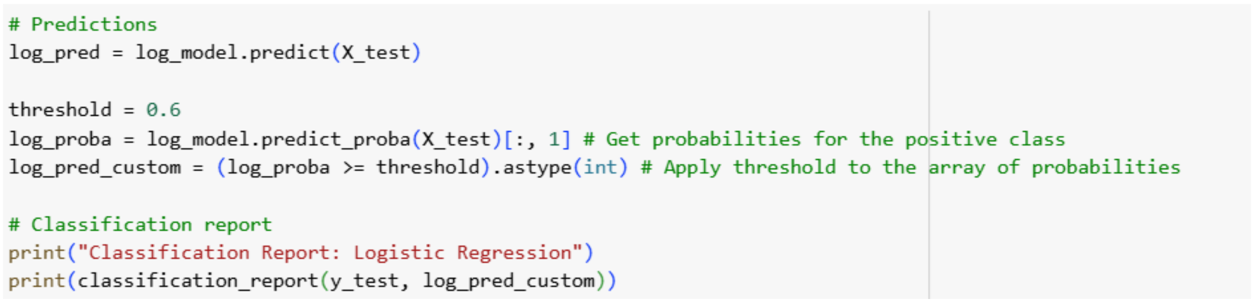
$$= \ 1 \ or \ 100\%$$

Comparison

The F1-score balances both precision and recall, making it useful when classes are imbalanced. LR had an F1-score of 86%, showing a good trade-off between catching extreme cases and avoiding false alarms. RF scored a perfect 100%, which again may point to overfitting rather than true performance.

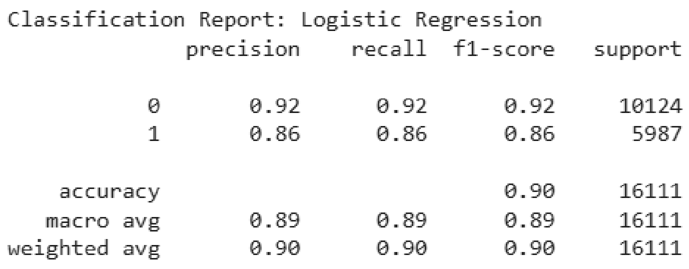
To further verify these calculations, the code snippet below shows how the model's performance metrics were calculated, as well as the resulting output. These snapshots confirm the performance metrics generated by the model.

Logistic Regression



```
# Predictions
log_pred = log_model.predict(X_test)

threshold = 0.6
log_proba = log_model.predict_proba(X_test)[:, 1] # Get probabilities for the positive class
log_pred_custom = (log_proba >= threshold).astype(int) # Apply threshold to the array of probabilities

# Classification report
print("Classification Report: Logistic Regression")
print(classification_report(y_test, log_pred_custom))
```

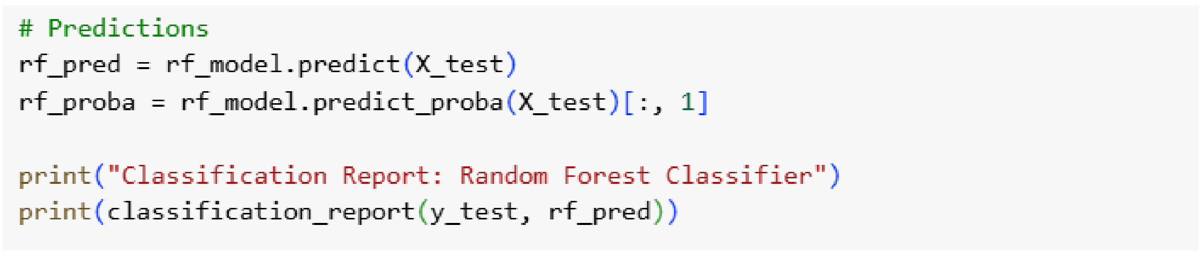
Figure 39. Code Snippet for Logistic Regression Performance Metrics.



```
Classification Report: Logistic Regression
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     10124
           1       0.86      0.86      0.86      5987

    accuracy                           0.90     16111
   macro avg       0.89      0.89      0.89     16111
weighted avg       0.90      0.90      0.90     16111
```

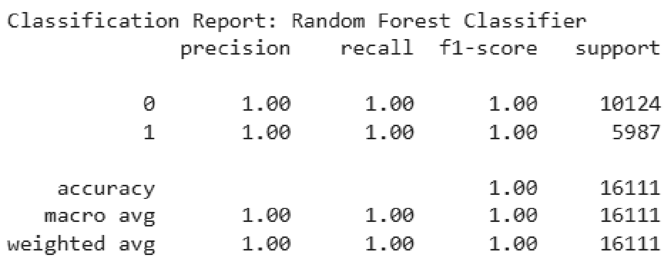
Figure 40. Output for Logistic Regression Performance Metrics.

Random Forest Classification



```
# Predictions
rf_pred = rf_model.predict(X_test)
rf_proba = rf_model.predict_proba(X_test)[:, 1]

print("Classification Report: Random Forest Classifier")
print(classification_report(y_test, rf_pred))
```

Figure 41. Code Snippet for Random Forest Classifier Performance Metrics.

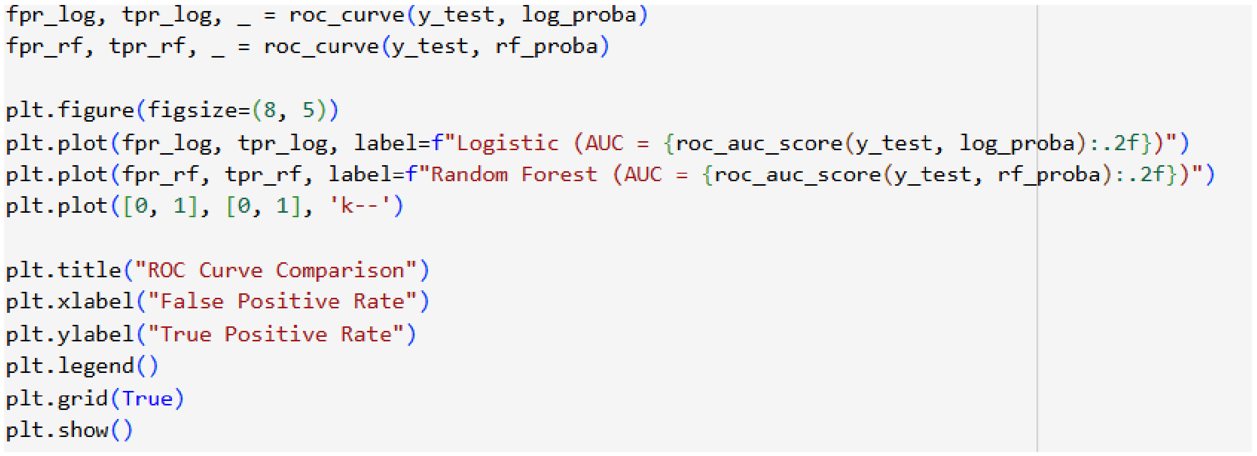


```
Classification Report: Random Forest Classifier
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10124
           1       1.00      1.00      1.00      5987

    accuracy                           1.00     16111
   macro avg       1.00      1.00      1.00     16111
weighted avg       1.00      1.00      1.00     16111
```

Figure 42. Output for Random Forest Classifier Performance Metrics.

III. ROC and AUC Graph

This last section describes the ROC and AUC graph of both models. The ROC curve shows how well a model can tell the difference between two classes, while the AUC score (Area Under the Curve) tells how good the model is overall. Usually, the closer it is to 1, the better the model. The code snippet below provides the code for the plotting of ROC and AUC graph.



```
fpr_log, tpr_log, _ = roc_curve(y_test, log_proba)
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_proba)

plt.figure(figsize=(8, 5))
plt.plot(fpr_log, tpr_log, label=f"Logistic (AUC = {roc_auc_score(y_test, log_proba):.2f})")
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {roc_auc_score(y_test, rf_proba):.2f})")
plt.plot([0, 1], [0, 1], 'k--')

plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```

Figure 43. Code Snippet for ROC and AUC graph plotting.

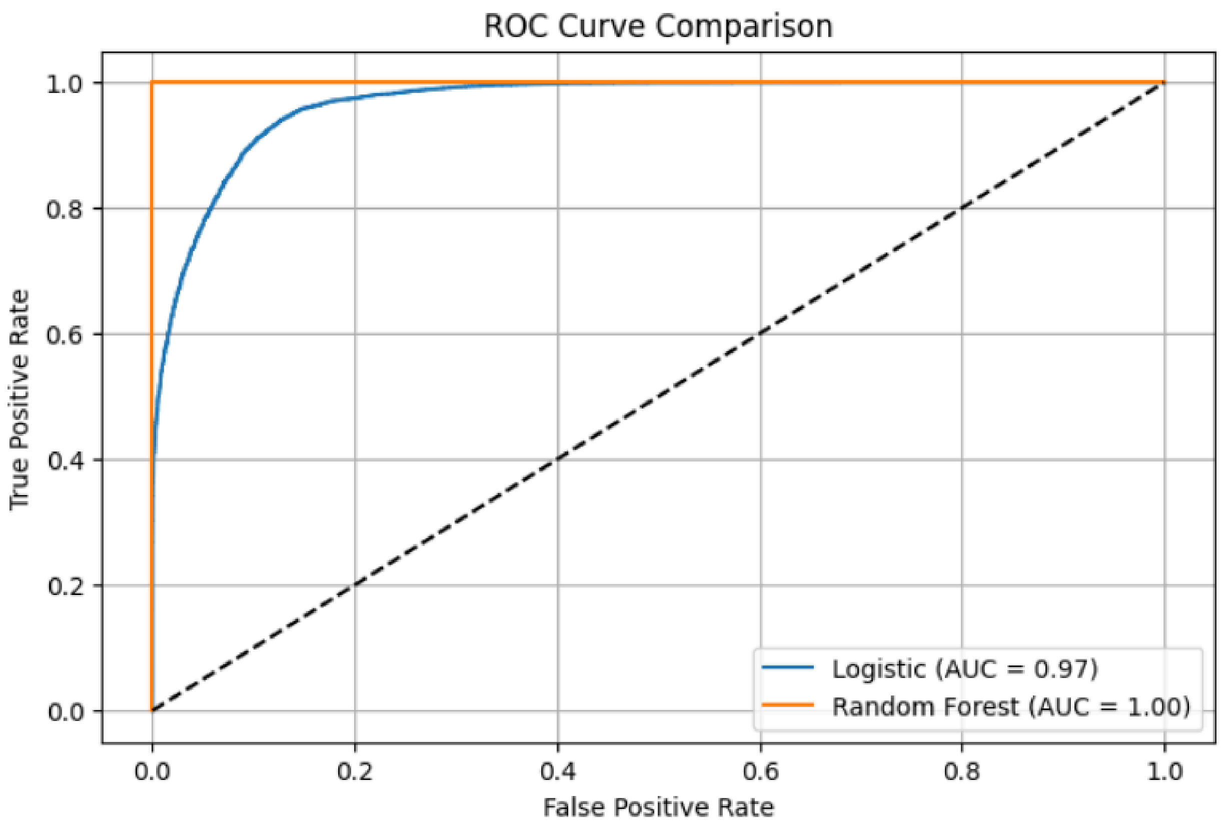


Figure 44. Output for ROC and AUC graph.

The blue line represents the LR model (0.97), which is very good at distinguishing between normal and extreme weather conditions. The orange line represents the RF model (1.00), which correctly predicts all cases. While this appears ideal, a perfect score could indicate that the model is overfitting by memorizing the training data rather than learning general patterns.

To summarize, based on the evaluation metrics and visualizations, the LR model appears to be more reliable in real-world applications. While the RF model achieved perfect scores, this could indicate overfitting, which occurs when a model performs exceptionally well on test data but fails to generalize to new, previously unseen data. As a result, the LR model, with its balanced performance, is a more practical and reliable option.

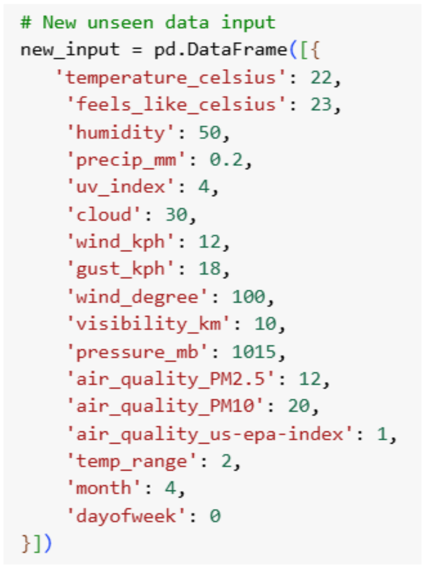
As a result, to test this hypothesis, the models are tested on new unseen data in the following section.

6.4 Testing on unseen data

Previously, it was mentioned that the RF model appears to be overfitted. This can be further proved by testing it with new unseen data under both “normal” and “extreme” weather conditions.

I. “Normal” Weather Condition

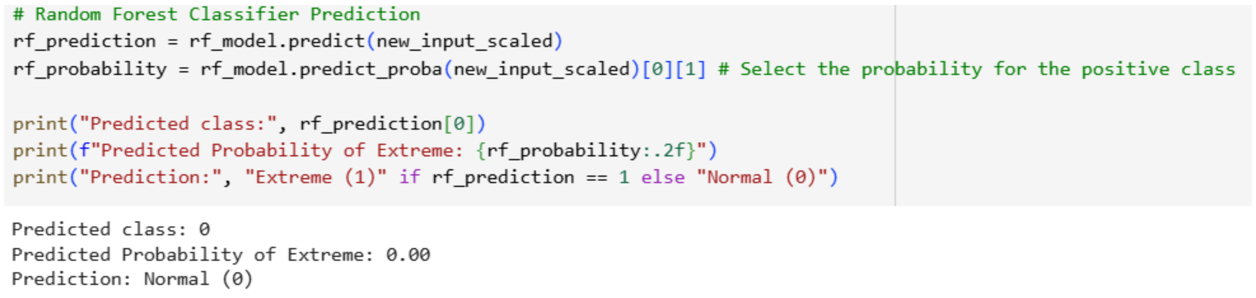
The following code snippet represents a sample of "normal" weather conditions.



```
# New unseen data input
new_input = pd.DataFrame([{
   'temperature_celsius': 22,
    'feels_like_celsius': 23,
    'humidity': 50,
    'precip_mm': 0.2,
    'uv_index': 4,
    'cloud': 30,
    'wind_kph': 12,
    'gust_kph': 18,
    'wind_degree': 100,
    'visibility_km': 10,
    'pressure_mb': 1015,
    'air_quality_PM2.5': 12,
    'air_quality_PM10': 20,
    'air_quality_us-epa-index': 1,
    'temp_range': 2,
    'month': 4,
    'dayofweek': 0
}])
```

Figure 45. Code Snippet for “Normal” Weather Conditions.

When this input is passed into the RFt model, the expected prediction should be 0, indicating a normal condition. As shown in the output, the model successfully identified the new unseen data as "normal," demonstrating its ability to generalize to real-world scenarios.



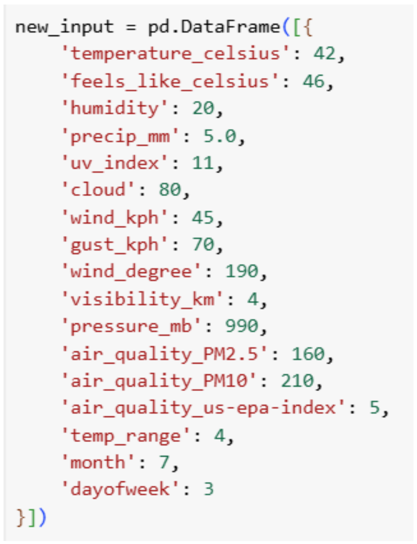
```
# Random Forest Classifier Prediction
rf_prediction = rf_model.predict(new_input_scaled)
rf_probability = rf_model.predict_proba(new_input_scaled)[0][1] # Select the probability for the positive class

print("Predicted class:", rf_prediction[0])
print(f"Predicted Probability of Extreme: {rf_probability:.2f}")
print("Prediction:", "Extreme (1)" if rf_prediction == 1 else "Normal (0)")
```

```
Predicted class: 0
Predicted Probability of Extreme: 0.00
Prediction: Normal (0)
```

Figure 46. Output for “Normal” Weather Conditions.

II. “Extreme” Weather Condition

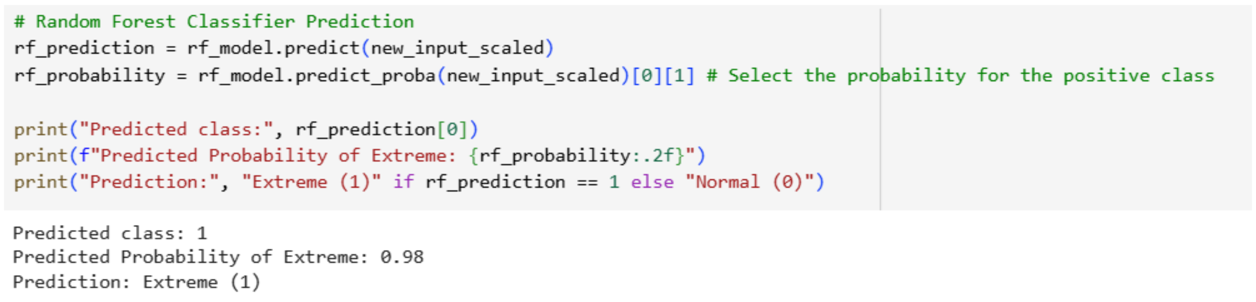
The following code snippet represents a sample of "extreme" weather conditions.



```
new_input = pd.DataFrame([{
    'temperature_celsius': 42,
    'feels_like_celsius': 46,
    'humidity': 20,
    'precip_mm': 5.0,
    'uv_index': 11,
    'cloud': 80,
    'wind_kph': 45,
    'gust_kph': 70,
    'wind_degree': 190,
    'visibility_km': 4,
    'pressure_mb': 990,
    'air_quality_PM2.5': 160,
    'air_quality_PM10': 210,
    'air_quality_us-epa-index': 5,
    'temp_range': 4,
    'month': 7,
    'dayofweek': 3
}])
```

Figure 47. Code Snippet for “Extreme” Weather Conditions.

When this input is passed into the RF model, the expected prediction should be 1, indicating an extreme condition. As shown in the output, the model successfully identified the new unseen data as "extreme," demonstrating its ability to generalize to real-world scenarios.



```
# Random Forest Classifier Prediction
rf_prediction = rf_model.predict(new_input_scaled)
rf_probability = rf_model.predict_proba(new_input_scaled)[0][1] # Select the probability for the positive class

print("Predicted class:", rf_prediction[0])
print(f"Predicted Probability of Extreme: {rf_probability:.2f}")
print("Prediction:", "Extreme (1)" if rf_prediction == 1 else "Normal (0)")
```

```
Predicted class: 1
Predicted Probability of Extreme: 0.98
Prediction: Extreme (1)
```

Figure 48. Output for “Extreme” Weather Conditions.

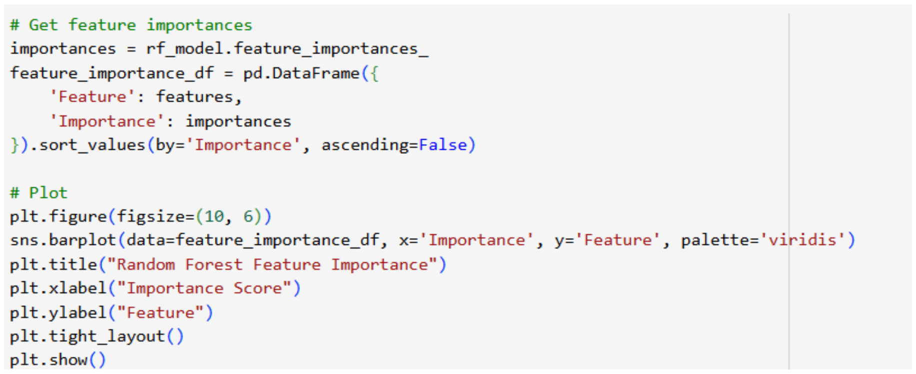
Based on these results, it can be confirmed that the RF model is not experiencing overfitting, as it consistently makes correct predictions on unseen data.

7.0 Data Visualization

Data visualization helps to better understand the behavior and performance of ML models. It turns complex results into visual formats such as graphs or plots, making it easier to identify patterns, feature contributions, and how confident a model is in its predictions.

7.1 Feature Importance

This visualization shows which features have the most influence on the model's decisions. In this case, the RF model provides feature importance scores for all the selected inputs. From the results, features like UV Index, precipitation, and temperature were the most important for predicting whether the weather is extreme or normal. Understanding this helps focus on the most useful weather parameters. The code snippet used for this part is provided below.



```
# Get feature importances
importances = rf_model.feature_importances_
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(10, 6))
sns.barplot(data=feature_importance_df, x='Importance', y='Feature', palette='viridis')
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Figure 49. Code Snippet of Feature Importance.

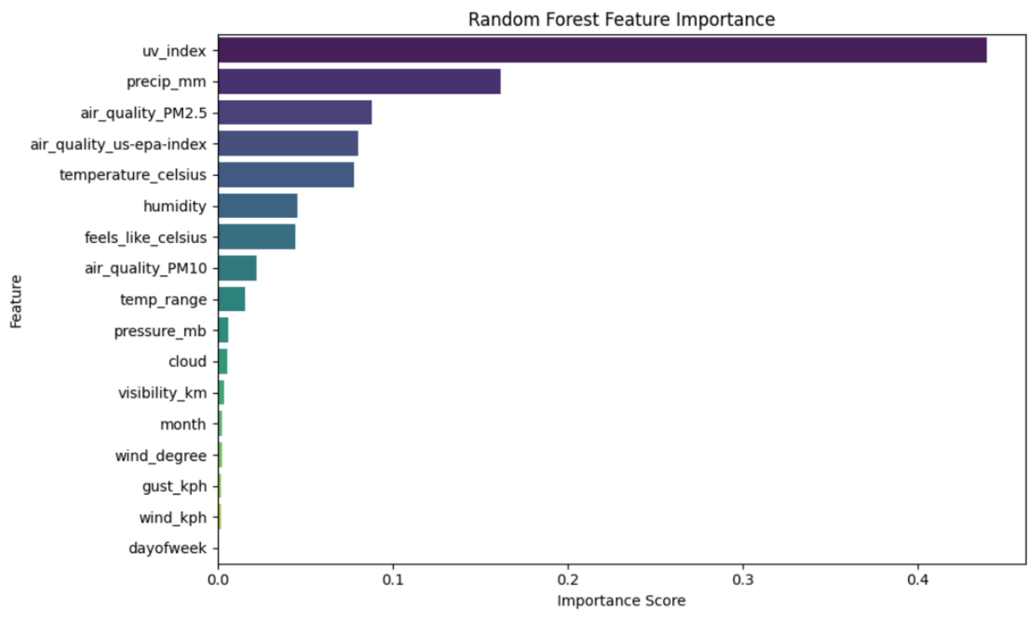
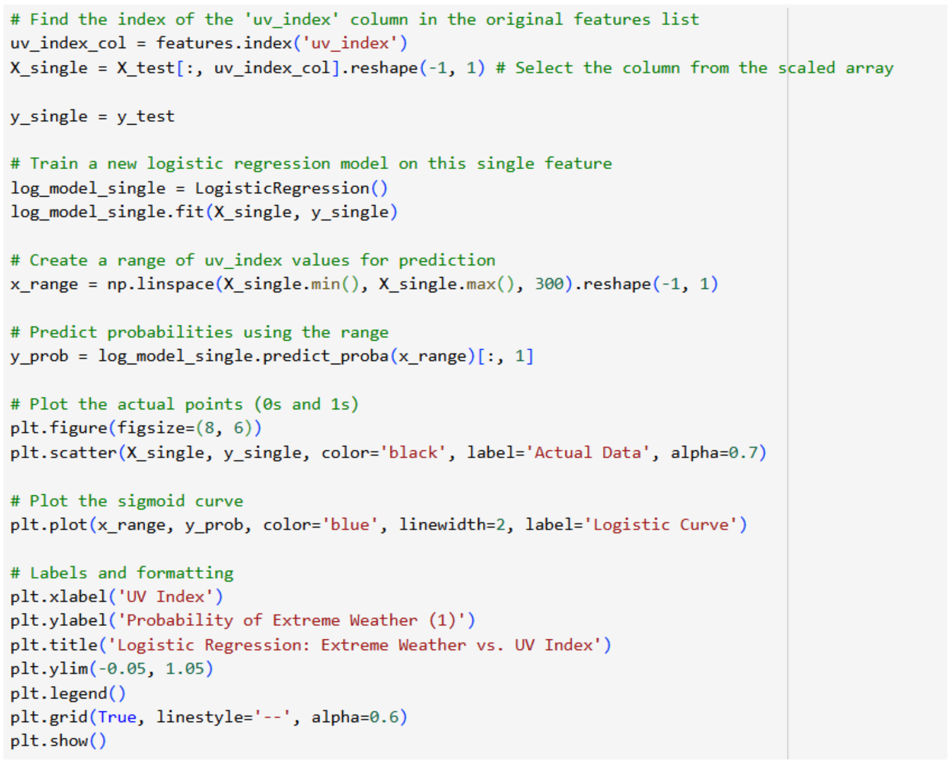


Figure 50. Code Snippet of Feature Importance.

7.2 Probability Curve for Logistic Regression Using UV Index

To visualize how the LR model predicts the probability of extreme weather, a graph was plotted using only the UV Index feature. As shown in the graph below, the black dots represent the actual data points labeled as either normal (0) or extreme (1), while the blue sigmoid curve shows the model's predicted probability of an event being extreme. When the UV Index is low (on the left side), the probability of extreme weather is close to 0. As the UV Index increases, the curve rises smoothly, indicating that the likelihood of extreme weather increases. Once the UV Index passes a certain threshold, the predicted probability sharply approaches 1, meaning the model becomes very confident that the weather is extreme.

This curve shows how the logistic model separates normal and extreme cases using just one feature. It also demonstrates that UV Index is a strong predictor for extreme weather conditions in this dataset.



```
# Find the index of the 'uv_index' column in the original features list
uv_index_col = features.index('uv_index')
X_single = X_test[:, uv_index_col].reshape(-1, 1) # Select the column from the scaled array

y_single = y_test

# Train a new logistic regression model on this single feature
log_model_single = LogisticRegression()
log_model_single.fit(X_single, y_single)

# Create a range of uv_index values for prediction
x_range = np.linspace(X_single.min(), X_single.max(), 300).reshape(-1, 1)

# Predict probabilities using the range
y_prob = log_model_single.predict_proba(x_range)[:, 1]

# Plot the actual points (0s and 1s)
plt.figure(figsize=(8, 6))
plt.scatter(X_single, y_single, color='black', label='Actual Data', alpha=0.7)

# Plot the sigmoid curve
plt.plot(x_range, y_prob, color='blue', linewidth=2, label='Logistic Curve')

# Labels and formatting
plt.xlabel('UV Index')
plt.ylabel('Probability of Extreme Weather (1)')
plt.title('Logistic Regression: Extreme Weather vs. UV Index')
plt.ylim(-0.05, 1.05)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

Figure 51. Code Snippet of Probability Curve using UV_Index.

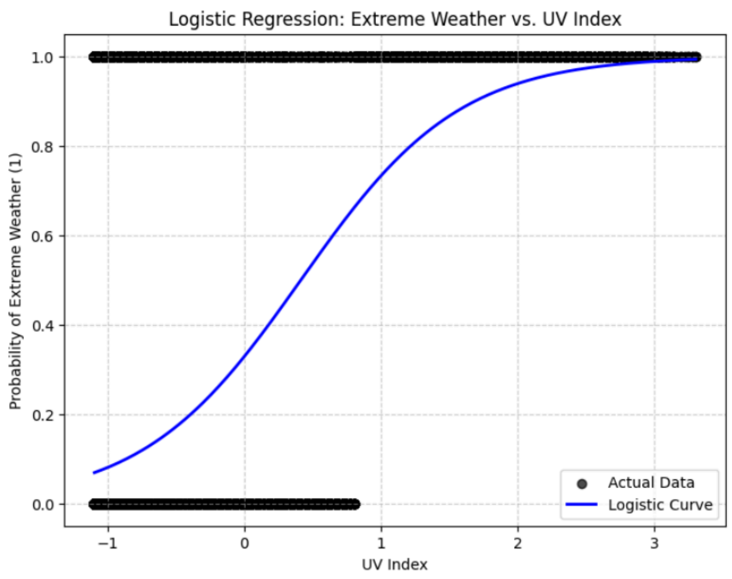


Figure 52. Output of Probability Curve using UV_Index.

8.0 Discussion

This section reflects on the model development process, highlights its strengths and weaknesses, and considers how the approach performs under different scenarios. The discussion also explores the practical implications of the model, its limitations, and areas for future development or deployment.

8.1 Scalability

The ML solution developed in this assignment is scalable and can be adapted to larger datasets with similar structures. Since the preprocessing steps (such as feature engineering, scaling, and class balancing) are modular and automated, they can easily be applied to new data from different locations or time periods. Additionally, the chosen models (Logistic Regression and Random Forest) can handle moderate to large datasets efficiently, making this approach suitable for real-world weather monitoring systems.

8.2 Challenges

Several challenges were encountered throughout the model-building process. One of the main difficulties was handling imbalanced data, where normal weather conditions greatly outnumbered extreme ones, risking biased predictions. This was mitigated using the class_weight='balanced' parameter. Another challenge was overfitting, especially with the Random Forest model, which showed near-perfect accuracy that may not generalize well to unseen data. However, this issue was solved by testing the model with new unseen data.

8.3 Improvement

There are several ways this model can be improved in future work. First, integrating more external data sources, such as satellite data or historical weather trends, could enhance feature richness. Second, experimenting with more advanced models like XGBoost or neural networks may lead to better generalization and predictive power. Additionally, implementing cross-validation and hyperparameter tuning can further optimize performance and reduce overfitting. Lastly, deploying the model in a real-time system would require performance optimization and regular updates to maintain accuracy with incoming data.

9.0 Conclusion

9.1 Summary of Findings

This project successfully developed and evaluated a ML solution to predict extreme weather events based on environmental and atmospheric features. Two models were implemented: Logistic Regression and Random Forest Classifier. The LR model provided clear interpretability and solid performance with an accuracy of 90 percent, while the RF achieved perfect accuracy and precision, although it showed signs of overfitting, which was debunked by testing the RF model with new unseen data.

9.2 Recommendations

To further enhance the reliability and applicability of this solution, future work should include regular retraining with updated weather data, the use of cross validation techniques to prevent overfitting, and exploration of more advanced models such as XGBoost or ensemble methods. Deploying the model into a real time prediction system can also provide timely alerts for extreme weather, which could benefit public safety and planning. Additionally, integrating geospatial data and long term climate trends may improve prediction accuracy and expand the model's use across different regions.

10.0 References

Ahsan, M., Mahmud, M., Saha, P., Gupta, K., & Siddique, Z. (2021). Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, *9*(3), 52. https://doi.org/10.3390/technologies9030052

Alotaibi, O., Pardede, E., & Tomy, S. (2023). Cleaning Big Data Streams: A systematic literature review. *Technologies*, *11*(4), 101. https://doi.org/10.3390/technologies11040101

Bakirarar, B., & Elhan, A. H. (2023). Class weighting technique to deal with imbalanced class problem in Machine Learning: Methodological Research. *Turkiye Klinikleri Journal of Biostatistics*, *15*(1), 19–29. https://doi.org/10.5336/biostatic.2022-93961

Bar-Gil, G. (2023, November 8). *Python import: Mastering the advanced features*. Mend.io. https://www.mend.io/blog/python-import-mastering-the-advanced-features/

Dutta, S. (2024, July 16). *What is shuffling the data? A guide for students*. Medium. https://medium.com/@sanjay_dutta/what-is-shuffling-the-data-a-guide-for-students-0f874572baf6

Elgiriyewithana, N. (2025, July 9). *World Weather Repository (daily updating)*. Kaggle. https://www.kaggle.com/datasets/nelgiriyewithana/global-weather-repository

GeeksforGeeks. (2024, May 31). *What is data exploration and its process?* https://www.geeksforgeeks.org/data-analysis/what-is-data-exploration-and-its-process/

Ibm. (2025, June 4). *What is Random Forest?*. IBM. https://www.ibm.com/think/topics/random-forest

Lee, F. (2025, June 18). *What is logistic regression?*. IBM. https://www.ibm.com/think/topics/logistic-regression

Mistry, R. (2025, March 14). *Data splitting (train-test-validation) in machine learning*. Medium.

https://medium.com/@rohanmistry231/data-splitting-train-test-validation-in-machine-learning-2d5d1927fa69

Murel, J., & Kavlakoglu, E. (2025, April 16). *What is a feature engineering?*. IBM. https://www.ibm.com/think/topics/feature-engineering