Big Data Visualization – Project Report

Exploring the patterns: Analysis of US weather events (2016-2022)

(Group 2)

Student	Name	Email
Team member 1	Shivani Battu	sbattu1@kent.edu
Team member 2	Chandini Karrothu	ckarroth@kent.edu
Team member 3	AnuRanjani Thota	athota3@kent.edu

FINAL REPORT

CONTENTS

Abstract:

Weather events have significant implications across numerous societal domains, including agriculture, transportation, and disaster management. Understanding the intricate spatiotemporal dynamics of weather phenomena is paramount for informed decision-making and proactive risk mitigation strategies. This project embarks on a thorough analysis of US weather events spanning the period from 2016 to 2022. Leveraging an extensive dataset comprising over 8.6 million recorded events across 49 states sourced from 2,071 airport-based weather stations, our objective is to derive insightful patterns and trends. Our methodology encompasses data preprocessing, exploratory data analysis, pattern recognition, visualization techniques, and Classification. We address inherent challenges such as handling large-scale spatiotemporal data, and crafting intuitive visualizations for effective communication of insights. By delving into this dataset, we aim to contribute to advancements in weather forecasting, disaster preparedness, and urban resilience planning. Our findings hold potential to inform and empower diverse stakeholders, including farmers, transportation authorities, urban planners, and emergency responders, thereby bolstering societal resilience to weather-related hazards.

I. Introduction

In an age where the impacts of climate change are increasingly felt, understanding and predicting weather events have become paramount for a multitude of applications across various sectors. Our project focuses on harnessing the extensive US Weather Events dataset spanning from 2016 to 2022 to delve into the dynamics of weather phenomena across the United States. This introduction delineates the driving motivations behind our exploration and underscores the practical applications that underscore the significance of our endeavor.

Weather events exert a profound influence on society, economy, and the environment, touching upon diverse aspects of human life. From optimizing agricultural practices and enhancing transportation safety to informing urban planning and disaster management strategies, the implications of weather data are far-reaching and multifaceted. By delving into the wealth of historical weather data available in the US Weather Events dataset, we aim to uncover valuable insights that can inform decision-making, drive innovation, and contribute to building a more resilient and sustainable future.

Motivated by the pressing needs across various sectors, our project seeks to address critical challenges and unlock opportunities for positive change. Farmers can optimize planting schedules and crop management practices, transportation authorities can plan routes and schedules more effectively, urban planners can develop resilient city designs, and emergency responders can prepare for and respond more effectively to natural disasters. Additionally, our project aims to aid in enhancement of weather forecasting models, develop early warning systems for extreme weather events, and integrate weather data into urban planning and disaster management systems to enable real-time decision-making.

By embarking on this data-driven exploration of weather events, we aspire to empower decision-makers with actionable insights that can drive positive change and facilitate informed decision-making in a world increasingly shaped by the forces of nature. Through rigorous analysis and interpretation of the US Weather Events dataset, our project endeavors to pave the way towards a more sustainable and resilient future, where the impacts of weather events are understood, anticipated, and mitigated effectively.

II. Project Description

Our project, "US Weather Insights: Analyzing 2016-2022 Weather Events," aims to conduct a comprehensive analysis of weather events in the United States spanning from 2016 to 2022. Leveraging the US Weather Events dataset comprising over 8.6 million recorded events from 2,071 airport-based weather stations, we seek to uncover insightful patterns and trends in weather phenomena. Our approach involves data preprocessing, exploratory analysis, pattern recognition, and visualization techniques to derive actionable insights for stakeholders across various sectors.

Challenges and Technical Contributions:

One of the primary challenges of our project is handling the large-scale spatiotemporal data and ensuring its accuracy and reliability. Additionally, we will face the task of developing robust algorithms for weather event classification and visualization to extract meaningful insights from the dataset. Our technical contributions include novel methodologies for weather data analysis, such as advanced machine learning techniques for predictive modeling, Classification, and innovative visualization approaches for effective communication of insights.

Workload Distribution:

- **Data Preprocessing**: AnuRanjani Thota

  Anu Ranjani will be responsible for Performing data cleaning to handle missing values, outliers, and inconsistencies. Formatting the data appropriately for further analysis. Ensuring data integrity by checking for duplicates and errors.

- **Exploratory Data Analysis and Visualization:** Shivani Battu

  Shivani Battu will focus on conducting exploratory data analysis to uncover patterns and trends in the weather data. She will also be responsible for creating visualizations and clusters to communicate key findings effectively.

- **Feature Engineering and Model Development:** Chandini Karrothu

  Chandini Karrothu will lead the effort in feature engineering and model development. She will work on extracting relevant features from the data and building classification models to segregate weather events.

III. Background

Related Papers

For our project, we drew inspiration and insights from multiple research papers in the field of weather data analysis and related domains. Some of the key papers we referenced include,

- Sathiaraj, D., Punkasem, T. O., Wang, F., & Seedah, D. P. (2018). Data-driven analysis on the effects of extreme weather elements on traffic volume in Atlanta, GA, USA. Computers, Environment and Urban Systems, 72, 212-220.
- Souza, C.V., Bonnet, S.M., Oliveira, D.D., Cataldi, M., Miranda, F., & Lage, M. (2023). PW: A Visual Approach for Building, Managing, and Analyzing Weather Simulation Ensembles at Runtime. IEEE Transactions on Visualization and Computer Graphics, 30, 738-747.
- Fathi, M., Haghi Kashani, M., Jameii, S.M. et al. Big Data Analytics in Weather Forecasting: A Systematic Review. Arch Computat Methods Eng 29, 1247–1275 (2022). https://doi.org/10.1007/s11831-021-09616-4

Software tools

- **Python:** Data preprocessing, Data Analysis, and Data visualization.
- **Pandas:** For data manipulation and analysis, especially for handling structured data such as CSV files containing weather event records.
- **NumPy:** For numerical computing, which may be useful for mathematical operations on the data, such as calculating statistics.
- **Matplotlib or Seaborn:** For data visualization, enabling you to create various types of plots and charts to visualize patterns and trends in the weather data.
- **Scikit-learn:** For machine learning tasks, such as predictive modeling and classification, providing a wide range of algorithms and tools for model development and evaluation.
- **Jupyter Notebook:** An integrated development environment (IDE) to code, visualize and document the computational outputs.
- **StatsModels:** For statistical modeling and hypothesis testing, which may be useful for analyzing relationships between weather variables or conducting time series analysis.
- **Geopandas or Folium:** If you're interested in spatial analysis or creating interactive maps to visualize the spatial distribution of weather events across geographical regions.

Required Hardware

- Processor: Intel Core i5 processor or higher.
- RAM: 8 GB or higher.
- Storage: 500 GB or higher.
- Operating System: Windows 10 or Linux

Related programming skills

- Proficient in utilizing Python for data analysis, visualization, and modeling purposes.
- Capable of cleaning, manipulating and analyzing data effectively using the Pandas library.
- Well-versed in employing statistical methods for hypothesis testing, correlation analysis, and establishing relationships between variables.
- Skilled in visualizing data using Matplotlib to discern patterns and trends effectively.
- Knowledge in applying time-series analysis techniques to detect patterns and trends within datasets.

IV. Problem Definition

Formal (mathematical) definitions of problem

1. **Classification:** Classify weather events into predefined categories (e.g., severe vs. non-severe, precipitation types) based on their characteristics and impacts.

• This problem focuses on categorizing observed weather events into distinct classes or categories based on their attributes, such as severity, duration, and impact. Examples of classification tasks include distinguishing between severe storms and ordinary rain showers, classifying precipitation types (e.g., rain, snow, sleet), and identifying hazardous weather conditions (e.g., tornadoes, hurricanes). The objective is to facilitate decision-making and risk assessment by providing clear labels for different types of weather phenomena.

2. **Pattern Recognition and Visualization:** Analyze historical weather event data to identify trends, patterns, and correlations between different variables.

• This problem involves exploring and understanding historical weather event data to uncover underlying trends, patterns, and relationships among various meteorological variables. It includes tasks such as visualizing temporal and spatial distributions of weather events, detecting recurring patterns or anomalies, and assessing correlations between different weather variables (e.g., temperature, humidity, wind speed). The insights gained from analysis and visualization can inform decision-making in areas such as agriculture, infrastructure planning, and disaster preparedness.

3. **Clustering**: Cluster weather events based on their intrinsic characteristics and patterns, grouping similar events together while maximizing the dissimilarity between clusters.

   • This problem involves partitioning observed weather events into cohesive groups, or clusters, where events within the same cluster exhibit high similarity in terms of their attributes such as spatial location, temporal occurrence, and meteorological properties. The objective is to uncover hidden structures and patterns in the data, enabling insights into weather variability, spatial distribution, and temporal trends.

Challenges of tackling these problems:

- **High-dimensional data**: Weather event datasets may contain a large number of variables (e.g., temperature, humidity, wind speed), making visualization and interpretation challenging.
- **Temporal dependencies:** Weather events exhibit temporal dependencies and seasonality, requiring specialized techniques for time series analysis.
- **Handling imbalanced classes:** Some weather events may be rare or infrequent compared to others, leading to imbalanced datasets.
- **Spatial and temporal variability**: Weather patterns vary significantly across different geographical regions and seasons, posing challenges in generalizing predictive models.

Solutions discussed:

- **Feature engineering**: Incorporate spatial and temporal features (e.g., geographical coordinates, time of year, local climate characteristics) to improve model performance and generalization.
  • K-means clustering for grouping similar events
  • Encoding categorical variables for model compatibility
- **Data Transformation:**
  • Feature creation (duration) and extraction (temporal features)
  • Scaling numerical features
- **Exploratory Data Analysis**
  - Distribution analysis of event types over time and severity
  - Visualization of weather type and severity distribution
- **Machine Learning Model Building:**
  - Training and evaluating various classification models.
  - Identifying relationships between variables
- **Data Segmentation with K-means Clustering:**
  - Explanation: K-means clustering divides weather data into distinct groups.
  - Purpose: Facilitate structured analysis of the data.
- **Identifying Similar Weather Patterns:**
  - Description: Clustering helps pinpoint locations with similar weather patterns.
  - Applications: Useful for resource allocation, and risk assessment.
- **Pattern Recognition and Visualization:**
  - Analyze historical weather data for trends and patterns.
  - Identify correlations between variables.
  - Tasks: Visualize distributions and detect anomalies.
  - Insights inform decision-making in various sectors.

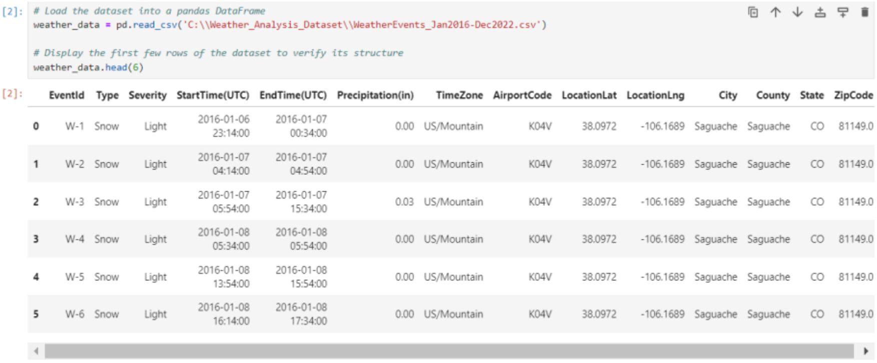
V. Visual Applications and Experimental Results

I. DATASET DESCRIPTION:

The weather dataset contains comprehensive information about weather events across the contiguous United States from January 2016 to December 2022 which offers a rich source of information for analyzing and understanding weather patterns, trends, and phenomena across the contiguous United States over a seven-year period. It can be utilized for various applications, including weather forecasting, climate research, and risk assessment.

Size: The dataset comprises approximately 8.6 million weather events.
Source: The data is collected from 2,071 airport-based weather stations nationwide.

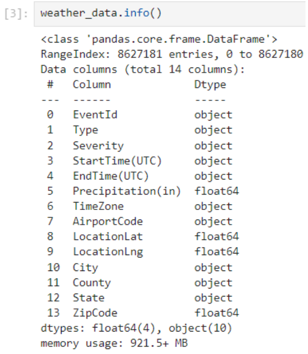
Attributes:
Type: Describes the type of weather event (e.g., rain, snow, storm, fog, hail, cold, precipitation, etc.).
Severity: Indicates the severity level of the weather event (e.g., heavy, light, moderate, severe, unknown, etc.).
Timezone: Represents the timezone of the weather observation.
Airport Code: Unique identifier for the airport-based weather station.
Location: Provides the geographic location of the weather station (latitude and longitude).
City: Name of the city where the weather station is located.
County: Name of the county where the weather station is located.
State: Name of the state where the weather station is located.
Zipcode: Zip code associated with the weather station's location.
Temporal Resolution: The dataset records weather events with timestamps, allowing for analysis at various temporal resolutions (e.g., hourly, daily, monthly).
Spatial Resolution: Weather events are recorded at specific airport-based weather stations, providing a spatial resolution that corresponds to the coverage area of these stations.



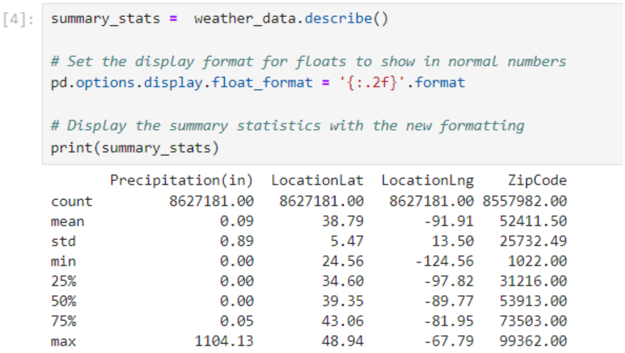
```
[2]: # Load the dataset into a pandas DataFrame
     weather_data = pd.read_csv('C:\\Weather_Analysis_Dataset\\WeatherEvents_Jan2016-Dec2022.csv')

     # Display the first few rows of the dataset to verify its structure
     weather_data.head(6)
```

[2]:

	EventId	Type	Severity	StartTime(UTC)	EndTime(UTC)	Precipitation(in)	TimeZone	AirportCode	LocationLat	LocationLng	City	County	State	ZipCode
0	W-1	Snow	Light	2016-01-06 23:14:00	2016-01-07 00:34:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
1	W-2	Snow	Light	2016-01-07 04:14:00	2016-01-07 04:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
2	W-3	Snow	Light	2016-01-07 05:54:00	2016-01-07 15:34:00	0.03	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
3	W-4	Snow	Light	2016-01-08 05:34:00	2016-01-08 05:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
4	W-5	Snow	Light	2016-01-08 13:54:00	2016-01-08 15:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
5	W-6	Snow	Light	2016-01-08 16:14:00	2016-01-08 17:34:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0



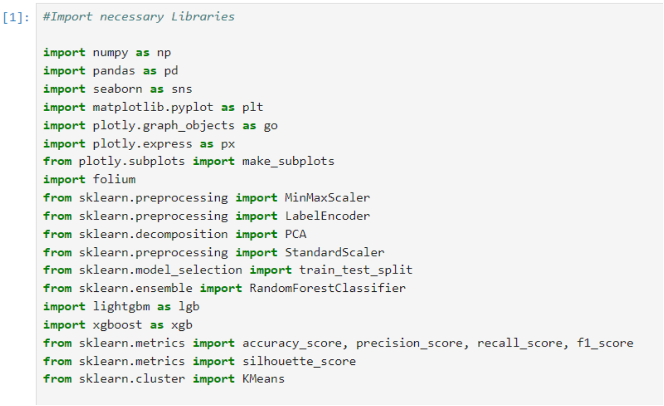
```
[3]: weather_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8627181 entries, 0 to 8627180
Data columns (total 14 columns):
 #   Column             Dtype
---  ------             -----
 0   EventId            object
 1   Type               object
 2   Severity           object
 3   StartTime(UTC)     object
 4   EndTime(UTC)       object
 5   Precipitation(in)  float64
 6   TimeZone           object
 7   AirportCode        object
 8   LocationLat        float64
 9   LocationLng        float64
 10  City               object
 11  County             object
 12  State              object
 13  ZipCode            float64
dtypes: float64(4), object(10)
memory usage: 921.5+ MB
```



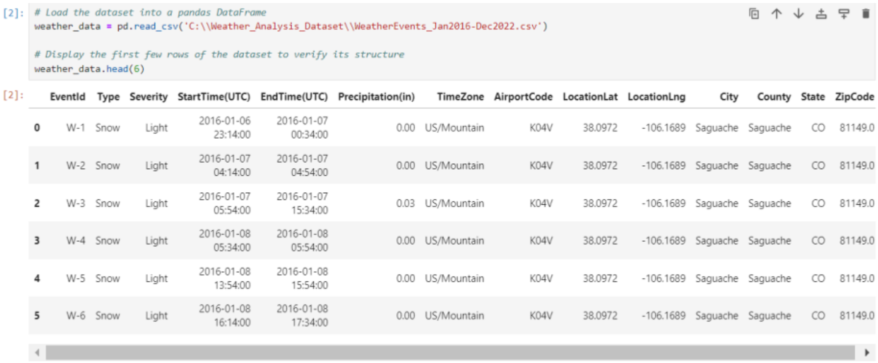
```
[4]: summary_stats =  weather_data.describe()

     # Set the display format for floats to show in normal numbers
     pd.options.display.float_format = '{:.2f}'.format

     # Display the summary statistics with the new formatting
     print(summary_stats)

       Precipitation(in)  LocationLat  LocationLng    ZipCode
count         8627181.00   8627181.00   8627181.00 8557982.00
mean                0.09        38.79       -91.91   52411.50
std                 0.89         5.47        13.50   25732.49
min                 0.00        24.56      -124.56    1022.00
25%                 0.00        34.60       -97.82   31216.00
50%                 0.00        39.35       -89.77   53913.00
75%                 0.05        43.06       -81.95   73503.00
max              1104.13        48.94       -67.79   99362.00
```

II. PRE-PROCESSING:

Loading the Dataset: The first step is to load the dataset into the environment. This involves reading the data from the provided file (e.g., CSV format) and storing it in a data structure that can be manipulated and analyzed, such as a pandas DataFrame in Python.



```
[1]: #Import necessary Libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import folium
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
```

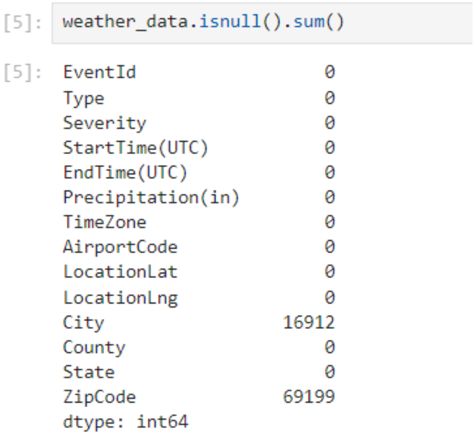


```
[2]: # Load the dataset into a pandas DataFrame
weather_data = pd.read_csv('C:\\Weather_Analysis_Dataset\\WeatherEvents_Jan2016-Dec2022.csv')

# Display the first few rows of the dataset to verify its structure
weather_data.head(6)
```

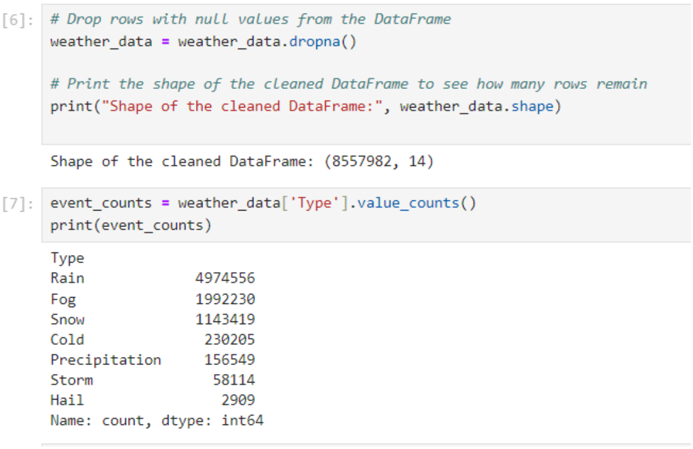
[2]:

	EventId	Type	Severity	StartTime(UTC)	EndTime(UTC)	Precipitation(in)	TimeZone	AirportCode	LocationLat	LocationLng	City	County	State	ZipCode
0	W-1	Snow	Light	2016-01-06 23:14:00	2016-01-07 00:34:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
1	W-2	Snow	Light	2016-01-07 04:14:00	2016-01-07 04:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
2	W-3	Snow	Light	2016-01-07 05:54:00	2016-01-07 15:34:00	0.03	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
3	W-4	Snow	Light	2016-01-08 05:34:00	2016-01-08 05:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
4	W-5	Snow	Light	2016-01-08 13:54:00	2016-01-08 15:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0
5	W-6	Snow	Light	2016-01-08 16:14:00	2016-01-08 17:34:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	81149.0

Cleaning the Data and Handling Missing Values: Missing values in the dataset are addressed to prevent errors during analysis. In this preprocessing step, missing values in the 'City' and 'ZipCode' column were handled by removing them from our dataset.

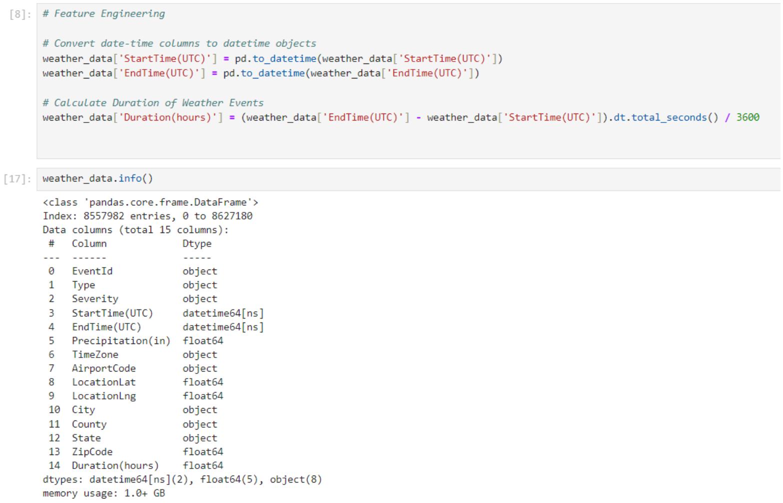


```
[5]: weather_data.isnull().sum()

[5]: EventId                  0
     Type                     0
     Severity                 0
     StartTime(UTC)           0
     EndTime(UTC)             0
     Precipitation(in)        0
     TimeZone                 0
     AirportCode              0
     LocationLat              0
     LocationLng              0
     City                 16912
     County                   0
     State                    0
     ZipCode              69199
     dtype: int64
```



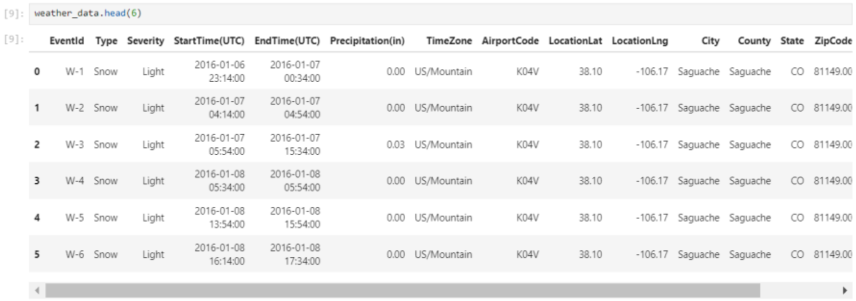
```
[6]: # Drop rows with null values from the DataFrame
     weather_data = weather_data.dropna()

     # Print the shape of the cleaned DataFrame to see how many rows remain
     print("Shape of the cleaned DataFrame:", weather_data.shape)

     Shape of the cleaned DataFrame: (8557982, 14)

[7]: event_counts = weather_data['Type'].value_counts()
     print(event_counts)

     Type
     Rain             4974556
     Fog              1992230
     Snow             1143419
     Cold              230205
     Precipitation     156549
     Storm              58114
     Hail                2909
     Name: count, dtype: int64
```

Data Type Conversion: Columns containing dates were converted to datetime objects, while categorical variables may need to be converted to categorical data types.



```
[8]: # Feature Engineering

     # Convert date-time columns to datetime objects
     weather_data['StartTime(UTC)'] = pd.to_datetime(weather_data['StartTime(UTC)'])
     weather_data['EndTime(UTC)'] = pd.to_datetime(weather_data['EndTime(UTC)'])

     # Calculate Duration of Weather Events
     weather_data['Duration(hours)'] = (weather_data['EndTime(UTC)'] - weather_data['StartTime(UTC)']).dt.total_seconds() / 3600
```

```
[17]: weather_data.info()

<class 'pandas.core.frame.DataFrame'>
Index: 8557982 entries, 0 to 8627180
Data columns (total 15 columns):
 #   Column            Dtype
---  ------            -----
 0   EventId           object
 1   Type              object
 2   Severity          object
 3   StartTime(UTC)    datetime64[ns]
 4   EndTime(UTC)      datetime64[ns]
 5   Precipitation(in) float64
 6   TimeZone          object
 7   AirportCode       object
 8   LocationLat       float64
 9   LocationLng       float64
 10  City              object
 11  County            object
 12  State             object
 13  ZipCode           float64
 14  Duration(hours)   float64
dtypes: datetime64[ns](2), float64(5), object(8)
memory usage: 1.0+ GB
```

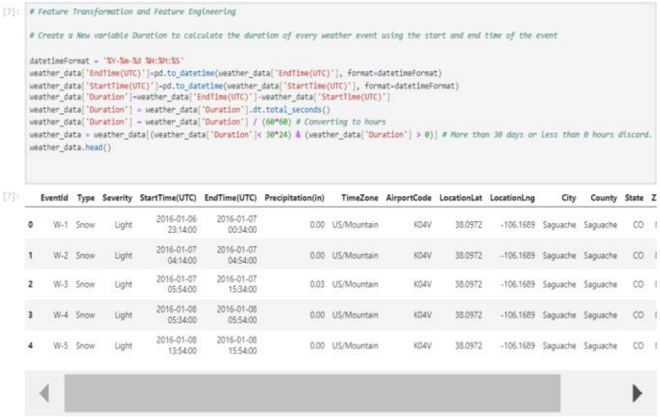


```
[9]: weather_data.head(6)
```

[9]:

	EventId	Type	Severity	StartTime(UTC)	EndTime(UTC)	Precipitation(in)	TimeZone	AirportCode	LocationLat	LocationLng	City	County	State	ZipCode
0	W-1	Snow	Light	2016-01-06 23:14:00	2016-01-07 00:34:00	0.00	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00
1	W-2	Snow	Light	2016-01-07 04:14:00	2016-01-07 04:54:00	0.00	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00
2	W-3	Snow	Light	2016-01-07 05:54:00	2016-01-07 15:34:00	0.03	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00
3	W-4	Snow	Light	2016-01-08 05:34:00	2016-01-08 05:54:00	0.00	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00
4	W-5	Snow	Light	2016-01-08 13:54:00	2016-01-08 15:54:00	0.00	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00
5	W-6	Snow	Light	2016-01-08 16:14:00	2016-01-08 17:34:00	0.00	US/Mountain	K04V	38.10	-106.17	Saguache	Saguache	CO	81149.00

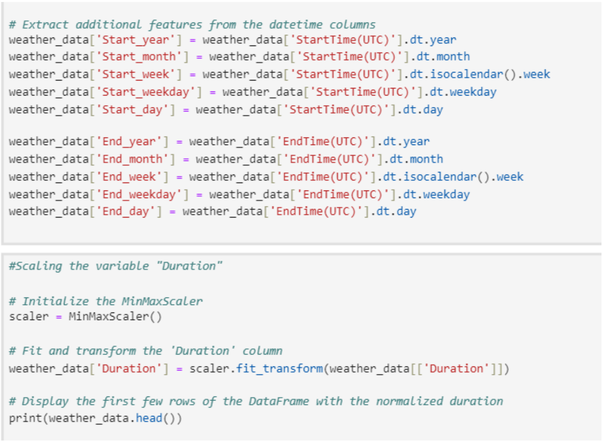
Deriving New Features: Additional features such as 'Duration', 'Year', 'Month', 'Day' were be derived from timestamps - StartTime(UTC) and EndTime(UTC) to provide more useful information for analysis.



```
[7]: # Feature Transformation and Feature Engineering

     # Create a New variable Duration to calculate the duration of every weather event using the start and end time of the event

     datetimeFormat = '%Y-%m-%d %H:%M:%S'
     weather_data['EndTime(UTC)']=pd.to_datetime(weather_data['EndTime(UTC)'], format=datetimeFormat)
     weather_data['StartTime(UTC)']=pd.to_datetime(weather_data['StartTime(UTC)'], format=datetimeFormat)
     weather_data['Duration']=weather_data['EndTime(UTC)']-weather_data['StartTime(UTC)']
     weather_data['Duration'] = weather_data['Duration'].dt.total_seconds()
     weather_data['Duration'] = weather_data['Duration'] / (60*60) # Converting to hours
     weather_data = weather_data[(weather_data['Duration']< 30*24) & (weather_data['Duration'] > 0)] # More than 30 days or less than 0 hours discard.
     weather_data.head()
```

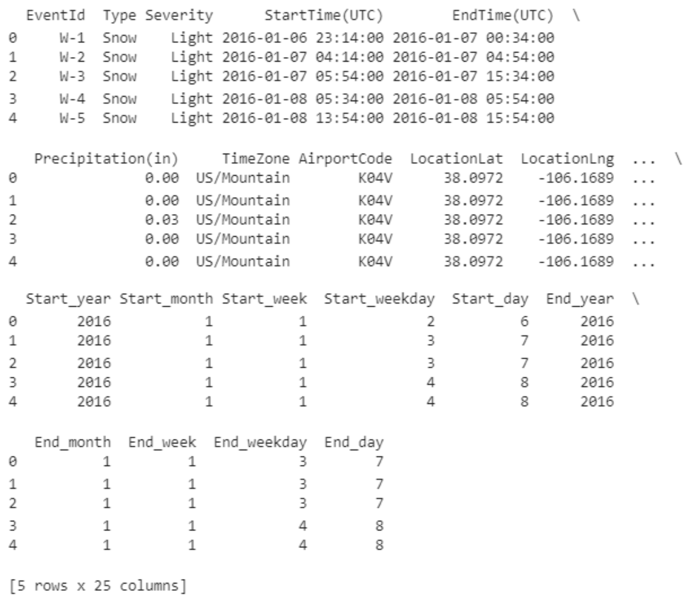
[7]:

	EventId	Type	Severity	StartTime(UTC)	EndTime(UTC)	Precipitation(in)	TimeZone	AirportCode	LocationLat	LocationLng	City	County	State	Z
0	W-1	Snow	Light	2016-01-06 23:14:00	2016-01-07 00:34:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	
1	W-2	Snow	Light	2016-01-07 04:14:00	2016-01-07 04:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	
2	W-3	Snow	Light	2016-01-07 05:54:00	2016-01-07 15:34:00	0.03	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	
3	W-4	Snow	Light	2016-01-08 05:34:00	2016-01-08 05:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	
4	W-5	Snow	Light	2016-01-08 13:54:00	2016-01-08 15:54:00	0.00	US/Mountain	K04V	38.0972	-106.1689	Saguache	Saguache	CO	

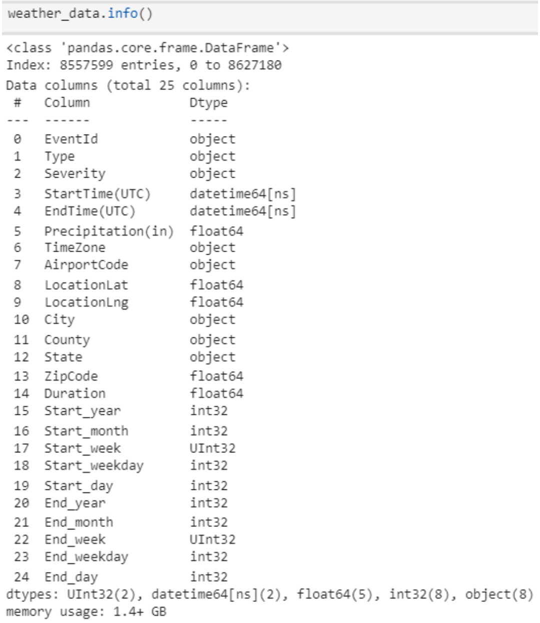


```
# Extract additional features from the datetime columns
weather_data['Start_year'] = weather_data['StartTime(UTC)'].dt.year
weather_data['Start_month'] = weather_data['StartTime(UTC)'].dt.month
weather_data['Start_week'] = weather_data['StartTime(UTC)'].dt.isocalendar().week
weather_data['Start_weekday'] = weather_data['StartTime(UTC)'].dt.weekday
weather_data['Start_day'] = weather_data['StartTime(UTC)'].dt.day

weather_data['End_year'] = weather_data['EndTime(UTC)'].dt.year
weather_data['End_month'] = weather_data['EndTime(UTC)'].dt.month
weather_data['End_week'] = weather_data['EndTime(UTC)'].dt.isocalendar().week
weather_data['End_weekday'] = weather_data['EndTime(UTC)'].dt.weekday
weather_data['End_day'] = weather_data['EndTime(UTC)'].dt.day
```

```
#Scaling the variable "Duration"

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the 'Duration' column
weather_data['Duration'] = scaler.fit_transform(weather_data[['Duration']])

# Display the first few rows of the DataFrame with the normalized duration
print(weather_data.head())
```



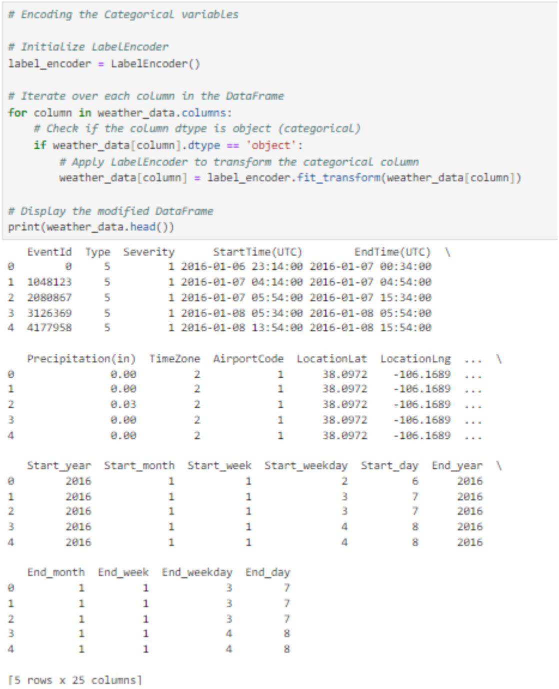
```
  EventId  Type Severity      StartTime(UTC)        EndTime(UTC)  \
0     W-1  Snow    Light 2016-01-06 23:14:00 2016-01-07 00:34:00
1     W-2  Snow    Light 2016-01-07 04:14:00 2016-01-07 04:54:00
2     W-3  Snow    Light 2016-01-07 05:54:00 2016-01-07 15:34:00
3     W-4  Snow    Light 2016-01-08 05:34:00 2016-01-08 05:54:00
4     W-5  Snow    Light 2016-01-08 13:54:00 2016-01-08 15:54:00

   Precipitation(in)     TimeZone AirportCode  LocationLat  LocationLng  ...  \
0               0.00  US/Mountain        K04V      38.0972    -106.1689  ...
1               0.00  US/Mountain        K04V      38.0972    -106.1689  ...
2               0.03  US/Mountain        K04V      38.0972    -106.1689  ...
3               0.00  US/Mountain        K04V      38.0972    -106.1689  ...
4               0.00  US/Mountain        K04V      38.0972    -106.1689  ...

  Start_year Start_month Start_week  Start_weekday  Start_day  End_year  \
0       2016           1          1              2          6      2016
1       2016           1          1              3          7      2016
2       2016           1          1              3          7      2016
3       2016           1          1              4          8      2016
4       2016           1          1              4          8      2016

   End_month  End_week  End_weekday  End_day
0          1         1            3        7
1          1         1            3        7
2          1         1            3        7
3          1         1            4        8
4          1         1            4        8

[5 rows x 25 columns]
```



```
weather_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8557599 entries, 0 to 8627180
Data columns (total 25 columns):
 #   Column             Dtype
---  ------             -----
 0   EventId            object
 1   Type               object
 2   Severity           object
 3   StartTime(UTC)     datetime64[ns]
 4   EndTime(UTC)       datetime64[ns]
 5   Precipitation(in)  float64
 6   TimeZone           object
 7   AirportCode        object
 8   LocationLat        float64
 9   LocationLng        float64
 10  City               object
 11  County             object
 12  State              object
 13  ZipCode            float64
 14  Duration           float64
 15  Start_year         int32
 16  Start_month        int32
 17  Start_week         UInt32
 18  Start_weekday      int32
 19  Start_day          int32
 20  End_year           int32
 21  End_month          int32
 22  End_week           UInt32
 23  End_weekday        int32
 24  End_day            int32
dtypes: UInt32(2), datetime64[ns](2), float64(5), int32(8), object(8)
memory usage: 1.4+ GB
```

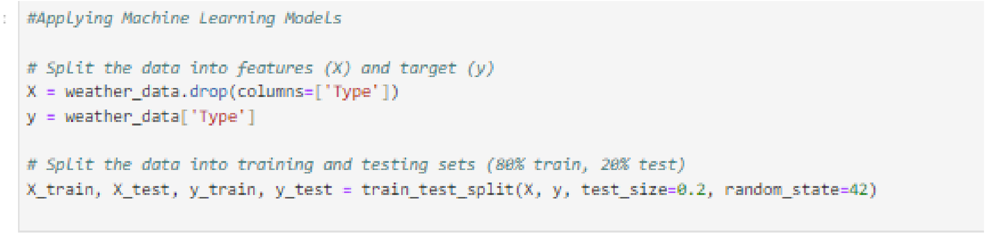
Encoding Categorical Variables: Categorical variables are typically encoded into numerical representations for analysis. We have used label encoding to convert categorical variables into a format suitable for machine learning algorithms. In this preprocessing step, categorical variables like 'Type' and 'Severity' were encoded using label encoding.



```
# Encoding the Categorical variables

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each column in the DataFrame
for column in weather_data.columns:
    # Check if the column dtype is object (categorical)
    if weather_data[column].dtype == 'object':
        # Apply LabelEncoder to transform the categorical column
        weather_data[column] = label_encoder.fit_transform(weather_data[column])

# Display the modified DataFrame
print(weather_data.head())
```

```
   EventId  Type  Severity      StartTime(UTC)        EndTime(UTC)  \
0        0     5         1 2016-01-06 23:14:00 2016-01-07 00:34:00
1  1048123     5         1 2016-01-07 04:14:00 2016-01-07 04:54:00
2  2080867     5         1 2016-01-07 05:54:00 2016-01-07 15:34:00
3  3126369     5         1 2016-01-08 05:34:00 2016-01-08 05:54:00
4  4177958     5         1 2016-01-08 13:54:00 2016-01-08 15:54:00

   Precipitation(in)  TimeZone  AirportCode  LocationLat  LocationLng  ...  \
0               0.00         2            1      38.0972    -106.1689  ...
1               0.00         2            1      38.0972    -106.1689  ...
2               0.03         2            1      38.0972    -106.1689  ...
3               0.00         2            1      38.0972    -106.1689  ...
4               0.00         2            1      38.0972    -106.1689  ...

   Start_year  Start_month  Start_week  Start_weekday  Start_day  End_year  \
0        2016            1           1              2          6      2016
1        2016            1           1              3          7      2016
2        2016            1           1              3          7      2016
3        2016            1           1              4          8      2016
4        2016            1           1              4          8      2016

   End_month  End_week  End_weekday  End_day
0          1         1            3        7
1          1         1            3        7
2          1         1            3        7
3          1         1            4        8
4          1         1            4        8

[5 rows x 25 columns]
```

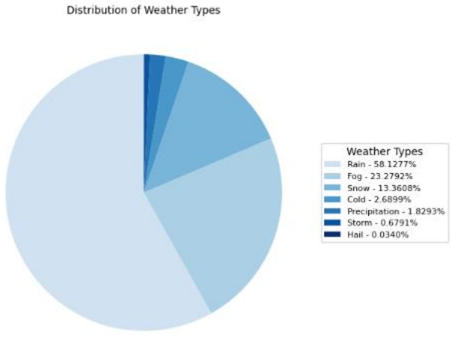
Splitting the Dataset (Train-Test Split): The dataset is split into separate training and testing sets to evaluate the performance of machine learning models. The training set is used to train the model, while the testing set is used to evaluate its performance on unseen data.

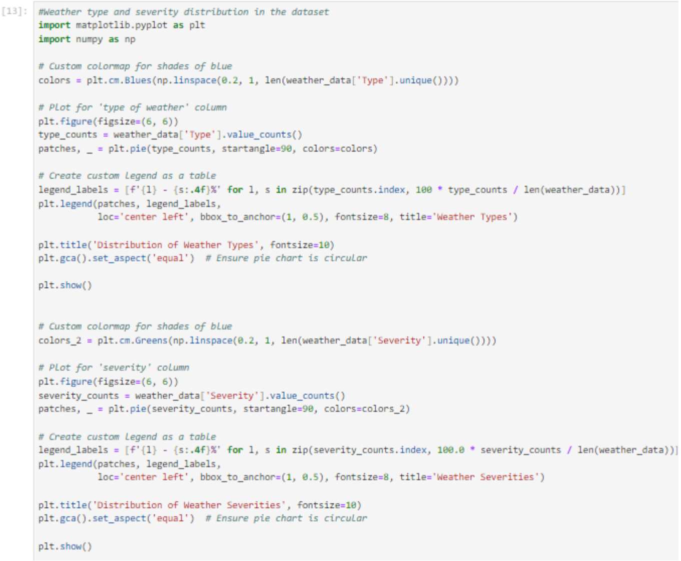


```
#Applying Machine Learning Models

# Split the data into features (X) and target (y)
X = weather_data.drop(columns=['Type'])
y = weather_data['Type']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

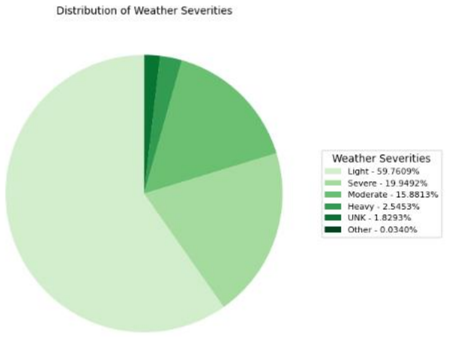
III. EXPLORATORY DATA ANALYSIS:

Distribution of Weather Types and Weather Severities:



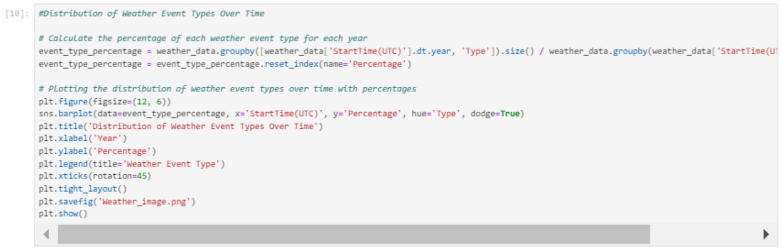


```
[13]: #Weather type and severity distribution in the dataset
import matplotlib.pyplot as plt
import numpy as np

# Custom colormap for shades of blue
colors = plt.cm.Blues(np.linspace(0.2, 1, len(weather_data['Type'].unique())))

# Plot for 'type of weather' column
plt.figure(figsize=(6, 6))
type_counts = weather_data['Type'].value_counts()
patches, _ = plt.pie(type_counts, startangle=90, colors=colors)

# Create custom legend as a table
legend_labels = [f'{l} - {s:.4f}%' for l, s in zip(type_counts.index, 100 * type_counts / len(weather_data))]
plt.legend(patches, legend_labels,
           loc='center left', bbox_to_anchor=(1, 0.5), fontsize=8, title='Weather Types')

plt.title('Distribution of Weather Types', fontsize=10)
plt.gca().set_aspect('equal')  # Ensure pie chart is circular

plt.show()


# Custom colormap for shades of blue
colors_2 = plt.cm.Greens(np.linspace(0.2, 1, len(weather_data['Severity'].unique())))

# Plot for 'severity' column
plt.figure(figsize=(6, 6))
severity_counts = weather_data['Severity'].value_counts()
patches, _ = plt.pie(severity_counts, startangle=90, colors=colors_2)

# Create custom legend as a table
legend_labels = [f'{l} - {s:.4f}%' for l, s in zip(severity_counts.index, 100.0 * severity_counts / len(weather_data))]
plt.legend(patches, legend_labels,
           loc='center left', bbox_to_anchor=(1, 0.5), fontsize=8, title='Weather Severities')

plt.title('Distribution of Weather Severities', fontsize=10)
plt.gca().set_aspect('equal')  # Ensure pie chart is circular

plt.show()
```

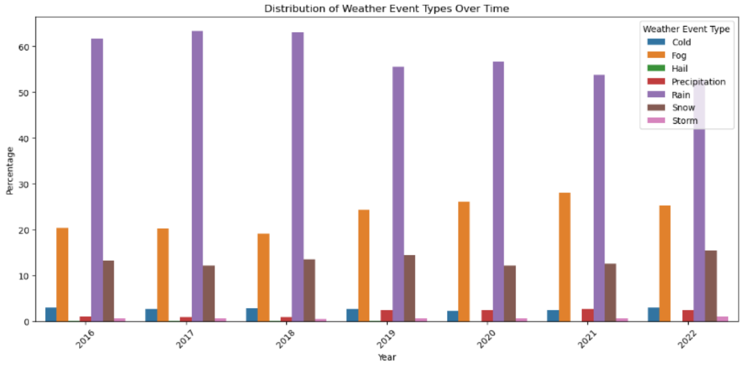


Distribution of Weather Events over time:

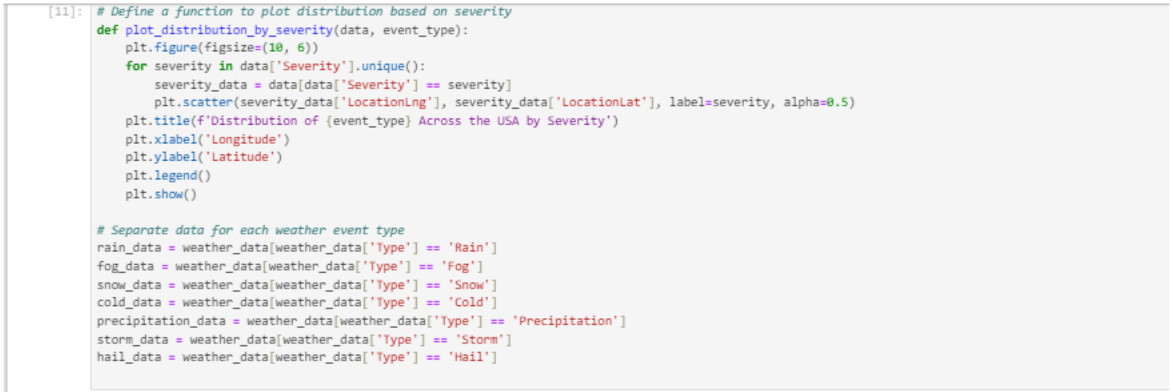
The graph may use lines to visualize the distribution of different weather events over time. Each bar represents a specific type of weather event, and its height or position on the graph indicates the frequency or percentage of occurrence during each time period. From this graph we can say that Rain is the most occurrent weather event.



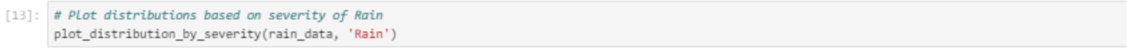
```
#Distribution of Weather Event Types Over Time

# Calculate the percentage of each weather event type for each year
event_type_percentage = weather_data.groupby([weather_data['StartTime(UTC)'].dt.year, 'Type']).size() / weather_data.groupby(weather_data['StartTime(U
event_type_percentage = event_type_percentage.reset_index(name='Percentage')

# Plotting the distribution of weather event types over time with percentages
plt.figure(figsize=(12, 6))
sns.barplot(data=event_type_percentage, x='StartTime(UTC)', y='Percentage', hue='Type', dodge=True)
plt.title('Distribution of Weather Event Types Over Time')
plt.xlabel('Year')
plt.ylabel('Percentage')
plt.legend(title='Weather Event Type')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('Weather_image.png')
plt.show()
```



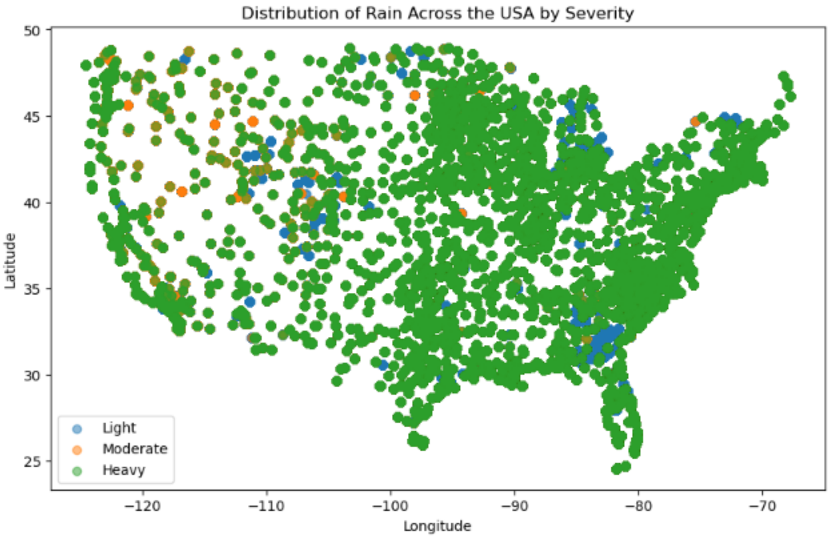
Plotting map graphs based on Severity types:

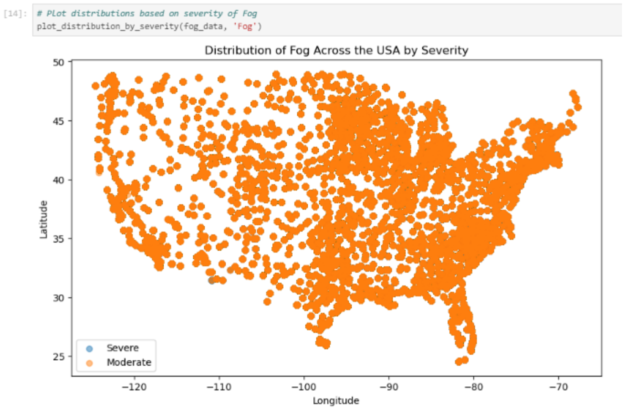


```
# Define a function to plot distribution based on severity
def plot_distribution_by_severity(data, event_type):
    plt.figure(figsize=(10, 6))
    for severity in data['Severity'].unique():
        severity_data = data[data['Severity'] == severity]
        plt.scatter(severity_data['LocationLng'], severity_data['LocationLat'], label=severity, alpha=0.5)
    plt.title(f'Distribution of {event_type} Across the USA by Severity')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.show()

# Separate data for each weather event type
rain_data = weather_data[weather_data['Type'] == 'Rain']
fog_data = weather_data[weather_data['Type'] == 'Fog']
snow_data = weather_data[weather_data['Type'] == 'Snow']
cold_data = weather_data[weather_data['Type'] == 'Cold']
precipitation_data = weather_data[weather_data['Type'] == 'Precipitation']
storm_data = weather_data[weather_data['Type'] == 'Storm']
hail_data = weather_data[weather_data['Type'] == 'Hail']
```

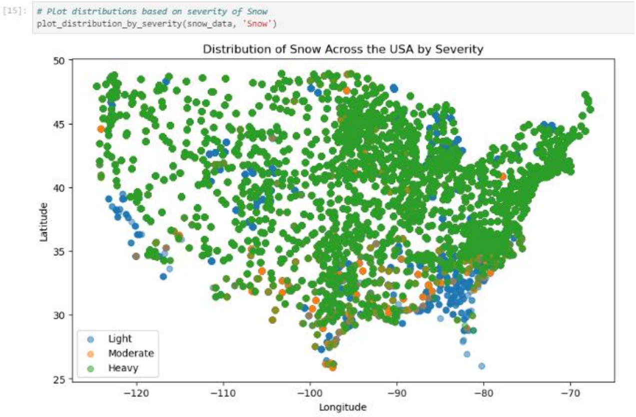


```
# Plot distributions based on severity of Rain
plot_distribution_by_severity(rain_data, 'Rain')
```

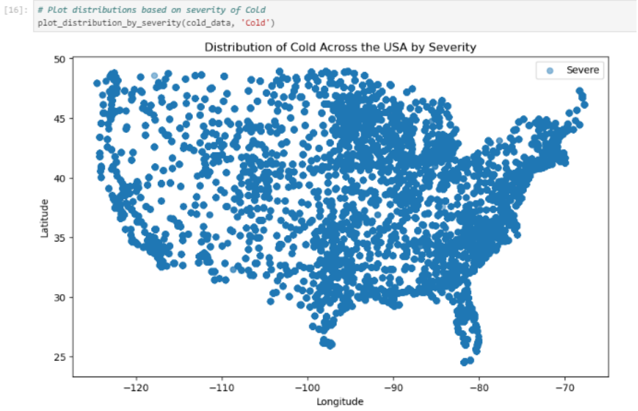




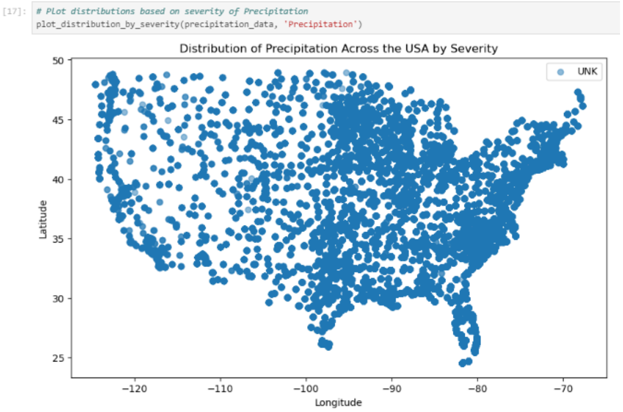
```
[14]: # Plot distributions based on severity of Fog
      plot_distribution_by_severity(fog_data, 'Fog')
```



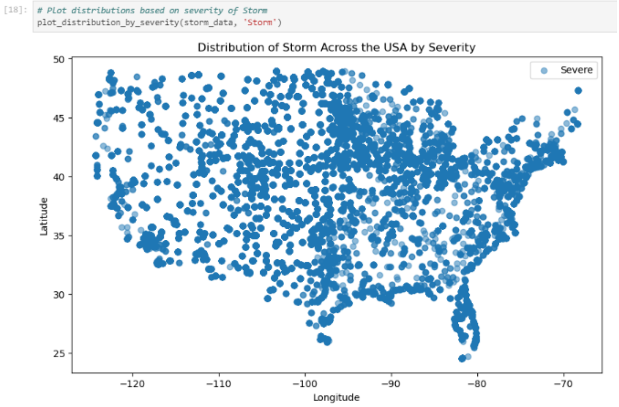
```
[15]: # Plot distributions based on severity of Snow
      plot_distribution_by_severity(snow_data, 'Snow')
```



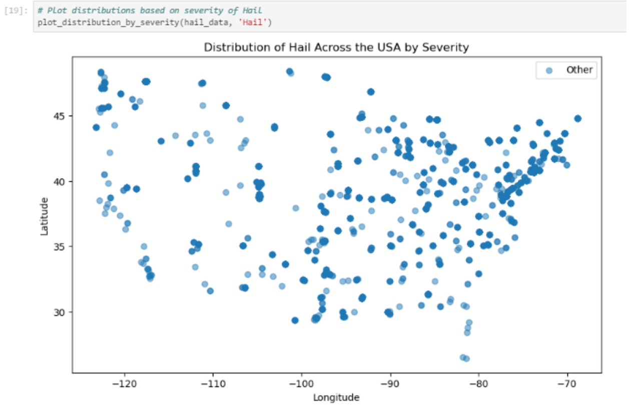
```
[16]: # Plot distributions based on severity of Cold
      plot_distribution_by_severity(cold_data, 'Cold')
```



```
[17]: # Plot distributions based on severity of Precipitation
      plot_distribution_by_severity(precipitation_data, 'Precipitation')
```



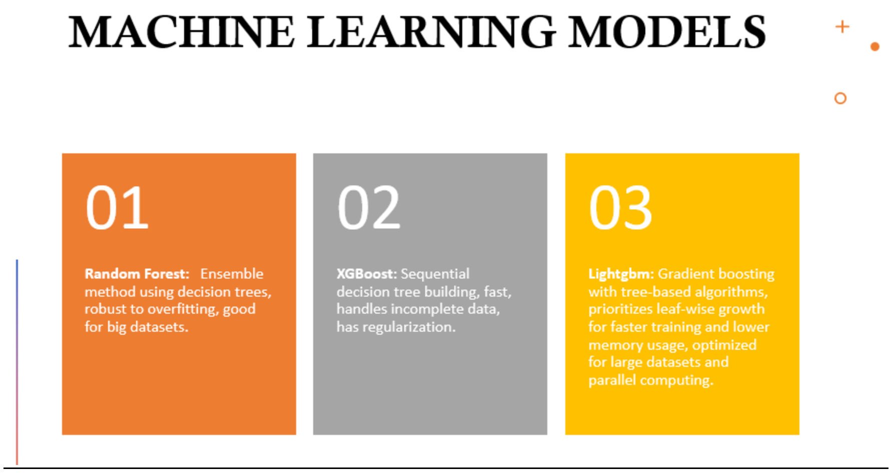
```
[18]: # Plot distributions based on severity of Storm
      plot_distribution_by_severity(storm_data, 'Storm')
```



```
[19]: # Plot distributions based on severity of Hail
      plot_distribution_by_severity(hail_data, 'Hail')
```

IV. MODEL DEVELOPMENT:

We utilized Random Forest, XGBoost, and LightGBM for classification tasks. These ensemble learning techniques were selected for their robustness, efficiency, and ability to handle complex datasets. By leveraging these models, we aimed to achieve high predictive accuracy and effectively classify the diverse range of data instances in our dataset.



MACHINE LEARNING MODELS

01

Random Forest: Ensemble method using decision trees, robust to overfitting, good for big datasets.

02

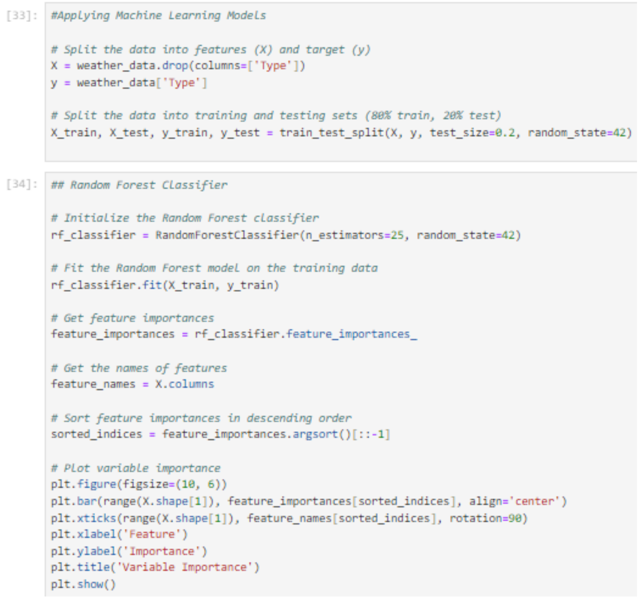
XGBoost: Sequential decision tree building, fast, handles incomplete data, has regularization.

03

Lightgbm: Gradient boosting with tree-based algorithms, prioritizes leaf-wise growth for faster training and lower memory usage, optimized for large datasets and parallel computing.

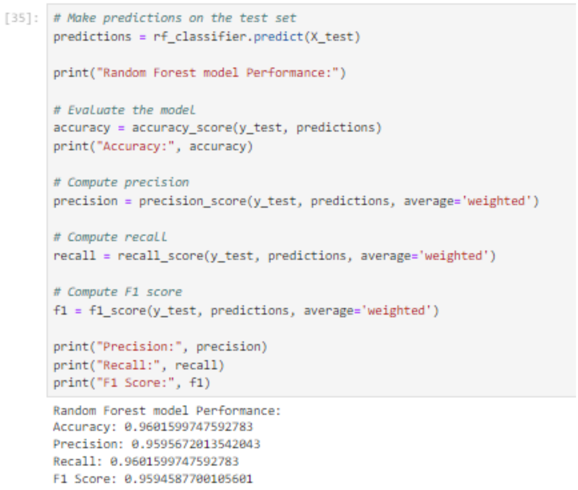
Random Forest, operates by constructing a multitude of decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It is robust to overfitting and works well with high-dimensional data.

*Interpretation***:** Based on the variable importance graph we can draw an interpretation that Severity, Precipitation and Location Latitude are the most important features for classification.

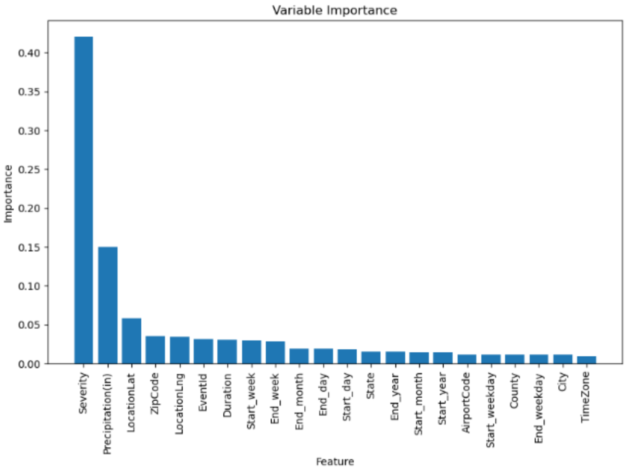


```
[33]: #Applying Machine Learning Models

      # Split the data into features (X) and target (y)
      X = weather_data.drop(columns=['Type'])
      y = weather_data['Type']

      # Split the data into training and testing sets (80% train, 20% test)
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

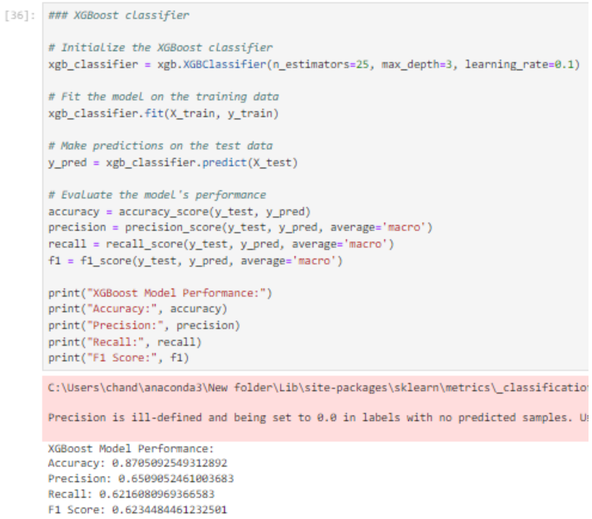
```
[34]: ## Random Forest Classifier

      # Initialize the Random Forest classifier
      rf_classifier = RandomForestClassifier(n_estimators=25, random_state=42)

      # Fit the Random Forest model on the training data
      rf_classifier.fit(X_train, y_train)

      # Get feature importances
      feature_importances = rf_classifier.feature_importances_

      # Get the names of features
      feature_names = X.columns

      # Sort feature importances in descending order
      sorted_indices = feature_importances.argsort()[::-1]

      # Plot variable importance
      plt.figure(figsize=(10, 6))
      plt.bar(range(X.shape[1]), feature_importances[sorted_indices], align='center')
      plt.xticks(range(X.shape[1]), feature_names[sorted_indices], rotation=90)
      plt.xlabel('Feature')
      plt.ylabel('Importance')
      plt.title('Variable Importance')
      plt.show()
```



```
[35]: # Make predictions on the test set
      predictions = rf_classifier.predict(X_test)

      print("Random Forest model Performance:")

      # Evaluate the model
      accuracy = accuracy_score(y_test, predictions)
      print("Accuracy:", accuracy)

      # Compute precision
      precision = precision_score(y_test, predictions, average='weighted')

      # Compute recall
      recall = recall_score(y_test, predictions, average='weighted')

      # Compute F1 score
      f1 = f1_score(y_test, predictions, average='weighted')

      print("Precision:", precision)
      print("Recall:", recall)
      print("F1 Score:", f1)

      Random Forest model Performance:
      Accuracy: 0.9601599747592783
      Precision: 0.9595672013542043
      Recall: 0.9601599747592783
      F1 Score: 0.9594587700105601
```



XGBoost, an implementation of gradient boosting, sequentially builds trees, correcting errors of the previous models, and combines weak learners to form a strong learner. It is highly efficient, parallelizable, and often yields state-of-the-art results.

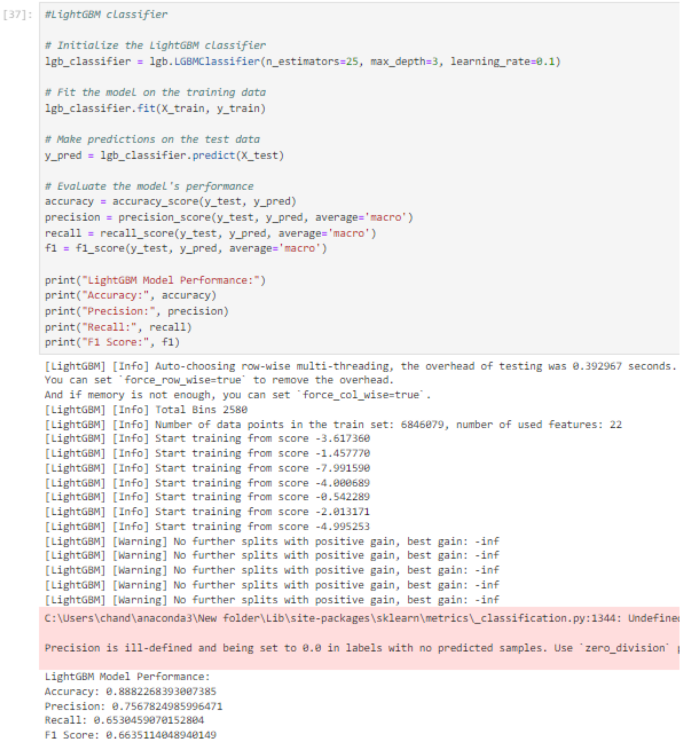


```
### XGBoost classifier

# Initialize the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(n_estimators=25, max_depth=3, learning_rate=0.1)

# Fit the model on the training data
xgb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = xgb_classifier.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print("XGBoost Model Performance:")
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
C:\Users\chand\anaconda3\New folder\Lib\site-packages\sklearn\metrics\_classificatio

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. U
```

```
XGBoost Model Performance:
Accuracy: 0.8705092549312892
Precision: 0.6509052461003683
Recall: 0.6216080969366583
F1 Score: 0.6234484461232501
```

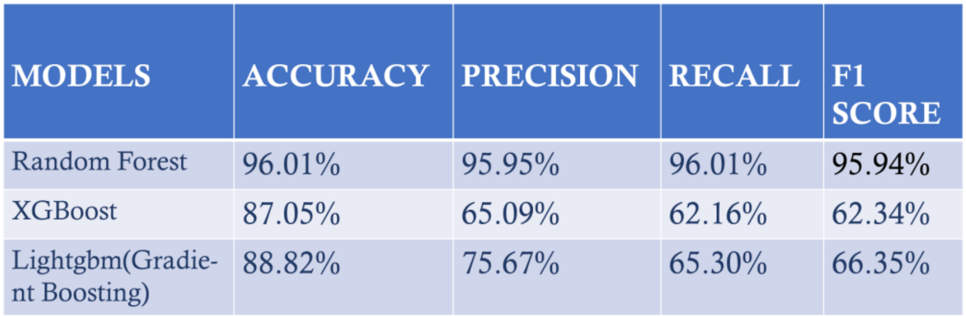
LightGBM, similar to XGBoost, also employs gradient boosting but uses a novel technique called Gradient-based One-Side Sampling (GOSS) to select only the data points that contribute most to the gradient during tree construction, resulting in faster training and higher efficiency with large-scale datasets. These models excel in handling complex datasets, capturing non-linear relationships, and achieving high predictive accuracy in various classification tasks.



```
[37]: #LightGBM classifier

      # Initialize the LightGBM classifier
      lgb_classifier = lgb.LGBMClassifier(n_estimators=25, max_depth=3, learning_rate=0.1)

      # Fit the model on the training data
      lgb_classifier.fit(X_train, y_train)

      # Make predictions on the test data
      y_pred = lgb_classifier.predict(X_test)

      # Evaluate the model's performance
      accuracy = accuracy_score(y_test, y_pred)
      precision = precision_score(y_test, y_pred, average='macro')
      recall = recall_score(y_test, y_pred, average='macro')
      f1 = f1_score(y_test, y_pred, average='macro')

      print("LightGBM Model Performance:")
      print("Accuracy:", accuracy)
      print("Precision:", precision)
      print("Recall:", recall)
      print("F1 Score:", f1)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.392967 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2580
[LightGBM] [Info] Number of data points in the train set: 6846079, number of used features: 22
[LightGBM] [Info] Start training from score -3.617360
[LightGBM] [Info] Start training from score -1.457770
[LightGBM] [Info] Start training from score -7.991590
[LightGBM] [Info] Start training from score -4.000689
[LightGBM] [Info] Start training from score -0.542289
[LightGBM] [Info] Start training from score -2.013171
[LightGBM] [Info] Start training from score -4.995253
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
C:\Users\chand\anaconda3\New folder\Lib\site-packages\sklearn\metrics\_classification.py:1344: Undefine

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division`

LightGBM Model Performance:
Accuracy: 0.8882268393007385
Precision: 0.7567824985996471
Recall: 0.6530459070152804
F1 Score: 0.6635114048940149
```

EXPERIMENTAL EVALUATION:



MODELS	ACCURACY	PRECISION	RECALL	F1 SCORE
Random Forest	96.01%	95.95%	96.01%	95.94%
XGBoost	87.05%	65.09%	62.16%	62.34%
Lightgbm(Gradie-nt Boosting)	88.82%	75.67%	65.30%	66.35%

Random Forest exhibits the best performance across all metrics, with high accuracy, precision, recall, and F1 score. XGBoost and LightGBM also show competitive performance, with XGBoost displaying a trade-off between precision and recall, while LightGBM shows improvement in precision and recall compared to XGBoost. These results provide valuable insights into the strengths and weaknesses of each model, guiding the selection of the most suitable model for the classification task at hand.

CLUSTERING

K-Means Clustering: K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into a predetermined number of clusters.

Objective: The main objective of K-means clustering is to partition a given training data into K clusters, where each data point belongs to the cluster with the nearest mean (centroid).

Initialization: The algorithm starts by randomly initializing K centroids, which represent the centers of the clusters. These centroids can be randomly chosen from the data points or placed at random locations within the feature space.

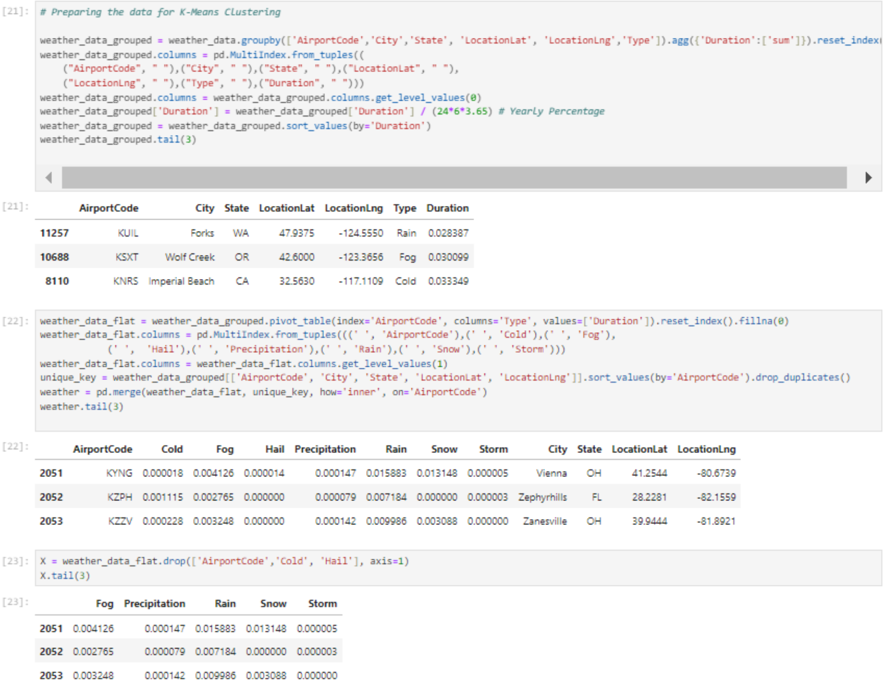
Assignment Step: In this step, each data point is assigned to the nearest centroid based on a distance metric, typically Euclidean distance. The data points are grouped into clusters based on their proximity to the centroids.

Update Step: After all data points have been assigned to clusters, the centroids are updated by computing the mean of all data points within each cluster. The centroids' positions are adjusted to the mean of the data points assigned to their respective clusters.

Iterations: The assignment and update steps are repeated iteratively until convergence criteria are met. Convergence occurs when the centroids no longer change significantly between iterations or when a maximum number of iterations is reached.

Cluster Evaluation: Once the algorithm converges, the resulting clusters can be evaluated using various metrics, such as the silhouette score or within-cluster sum of squares (WCSS), to assess the quality of the clustering solution.

Choosing the Number of Clusters (K): One of the key challenges in K-means clustering is determining the optimal number of clusters, K. This can be done using domain knowledge, the elbow method, silhouette analysis, or other techniques.



[21]:

```python
# Preparing the data for K-Means Clustering

weather_data_grouped = weather_data.groupby(['AirportCode','City','State', 'LocationLat', 'LocationLng','Type']).agg({'Duration':['sum']}).reset_index(
weather_data_grouped.columns = pd.MultiIndex.from_tuples((
    ("AirportCode", " "),("City", " "),("State", " "),("LocationLat", " "),
    ("LocationLng", " "),("Type", " "),("Duration", " ")))
weather_data_grouped.columns = weather_data_grouped.columns.get_level_values(0)
weather_data_grouped['Duration'] = weather_data_grouped['Duration'] / (24*6*3.65) # Yearly Percentage
weather_data_grouped = weather_data_grouped.sort_values(by='Duration')
weather_data_grouped.tail(3)
```

[21]:

	AirportCode	City	State	LocationLat	LocationLng	Type	Duration
11257	KUIL	Forks	WA	47.9375	-124.5550	Rain	0.028387
10688	KSXT	Wolf Creek	OR	42.6000	-123.3656	Fog	0.030099
8110	KNRS	Imperial Beach	CA	32.5630	-117.1109	Cold	0.033349

[22]:

```python
weather_data_flat = weather_data_grouped.pivot_table(index='AirportCode', columns='Type', values=['Duration']).reset_index().fillna(0)
weather_data_flat.columns = pd.MultiIndex.from_tuples(((' ', 'AirportCode'),(' ', 'Cold'),(' ', 'Fog'),
            (' ',  'Hail'),(' ', 'Precipitation'),(' ', 'Rain'),(' ', 'Snow'),(' ', 'Storm')))
weather_data_flat.columns = weather_data_flat.columns.get_level_values(1)
unique_key = weather_data_grouped[['AirportCode', 'City', 'State', 'LocationLat', 'LocationLng']].sort_values(by='AirportCode').drop_duplicates()
weather = pd.merge(weather_data_flat, unique_key, how='inner', on='AirportCode')
weather.tail(3)
```

[22]:

	AirportCode	Cold	Fog	Hail	Precipitation	Rain	Snow	Storm	City	State	LocationLat	LocationLng
2051	KYNG	0.000018	0.004126	0.000014	0.000147	0.015883	0.013148	0.000005	Vienna	OH	41.2544	-80.6739
2052	KZPH	0.001115	0.002765	0.000000	0.000079	0.007184	0.000000	0.000003	Zephyrhills	FL	28.2281	-82.1559
2053	KZZV	0.000228	0.003248	0.000000	0.000142	0.009986	0.003088	0.000000	Zanesville	OH	39.9444	-81.8921

[23]:

```python
X = weather_data_flat.drop(['AirportCode','Cold', 'Hail'], axis=1)
X.tail(3)
```

[23]:

	Fog	Precipitation	Rain	Snow	Storm
2051	0.004126	0.000147	0.015883	0.013148	0.000005
2052	0.002765	0.000079	0.007184	0.000000	0.000003
2053	0.003248	0.000142	0.009986	0.003088	0.000000

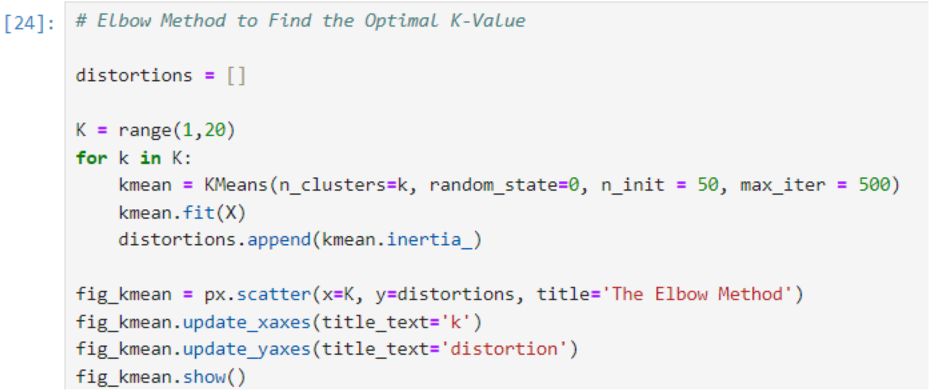
Finding optimal K using Elbow Method:

Elbow method is a popular technique used to determine the optimal number of clusters (K) in K-means clustering. For each value of K (the number of clusters), the sum of squared distances between each data point and its nearest centroid within the cluster is calculated. This gives us a measure of how compact the clusters are.

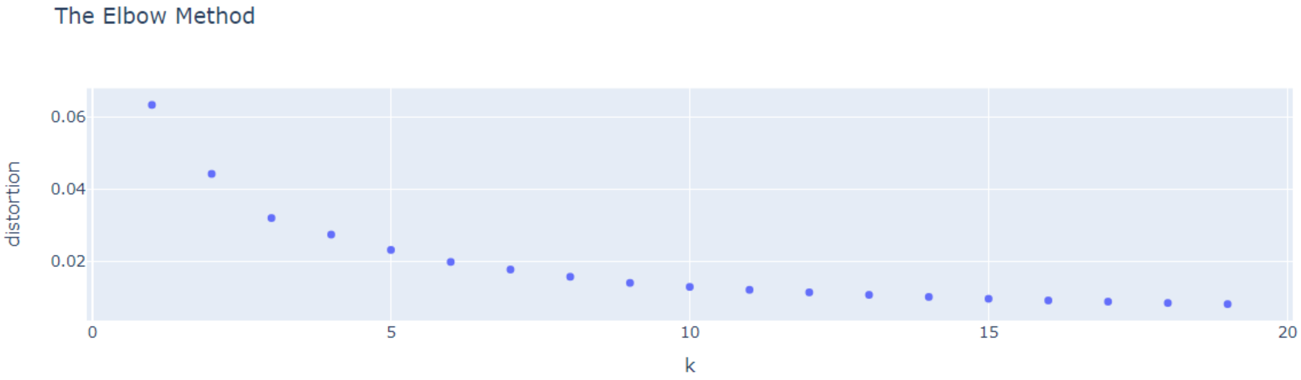
Plot the Elbow Curve: The WCSS values for different values of K are plotted on a graph. As K increases, the WCSS generally decreases because more clusters will naturally reduce the distance between data points and centroids. However, at some point, adding more clusters will not significantly reduce the WCSS.

Identify the "Elbow" Point: The optimal number of clusters (K) is typically chosen at the "elbow" point on the graph. The elbow point is where the rate of decrease in WCSS slows down significantly, forming an elbow-like shape on the plot. This point represents the optimal trade-off between the number of clusters and the compactness of the clusters.

Select the Optimal K Value: Based on the elbow method, the optimal K value was chosen to be 4 where adding more clusters does not lead to a significant improvement in clustering quality.

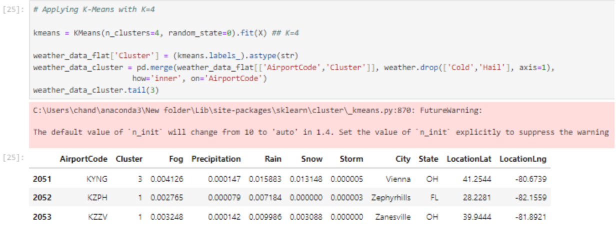


```
[24]: # Elbow Method to Find the Optimal K-Value

      distortions = []

      K = range(1,20)
      for k in K:
          kmean = KMeans(n_clusters=k, random_state=0, n_init = 50, max_iter = 500)
          kmean.fit(X)
          distortions.append(kmean.inertia_)

      fig_kmean = px.scatter(x=K, y=distortions, title='The Elbow Method')
      fig_kmean.update_xaxes(title_text='k')
      fig_kmean.update_yaxes(title_text='distortion')
      fig_kmean.show()
```



APPLICATION OF K-MEANS:

Below code output shows the last three records of the dataset.



```
[25]: # Applying K-Means with K=4

      kmeans = KMeans(n_clusters=4, random_state=0).fit(X) ## K=4

      weather_data_flat['Cluster'] = (kmeans.labels_).astype(str)
      weather_data_cluster = pd.merge(weather_data_flat[['AirportCode','Cluster']], weather.drop(['Cold','Hail'], axis=1),
                                      how='inner', on='AirportCode')
      weather_data_cluster.tail(3)
```

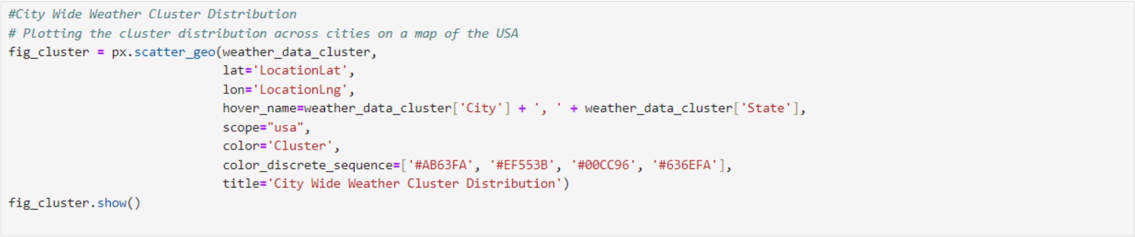
```
C:\Users\chand\anaconda3\New folder\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

[25]:

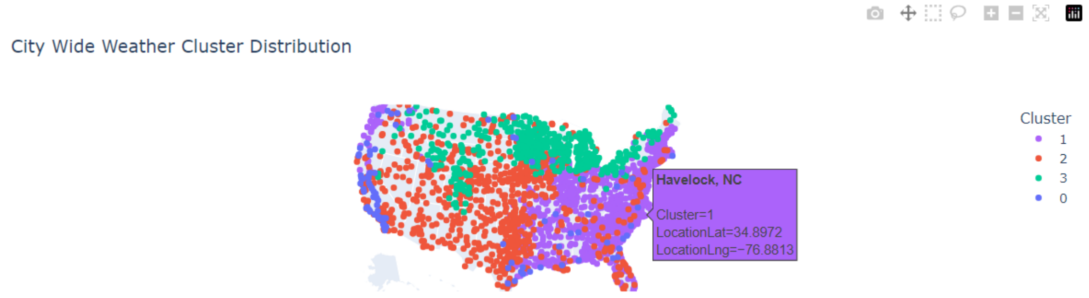
	AirportCode	Cluster	Fog	Precipitation	Rain	Snow	Storm	City	State	LocationLat	LocationLng
2051	KYNG	3	0.004126	0.000147	0.015883	0.013148	0.000005	Vienna	OH	41.2544	-80.6739
2052	KZPH	1	0.002765	0.000079	0.007184	0.000000	0.000003	Zephyrhills	FL	28.2281	-82.1559
2053	KZZV	1	0.003248	0.000142	0.009986	0.003088	0.000000	Zanesville	OH	39.9444	-81.8921

After forming clusters using K-means we plot the clusters in the map graph and display it using dynamic features.

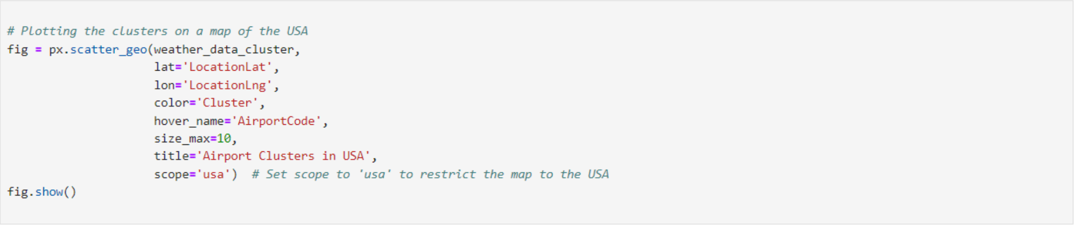
City Wide Weather Cluster Distribution:



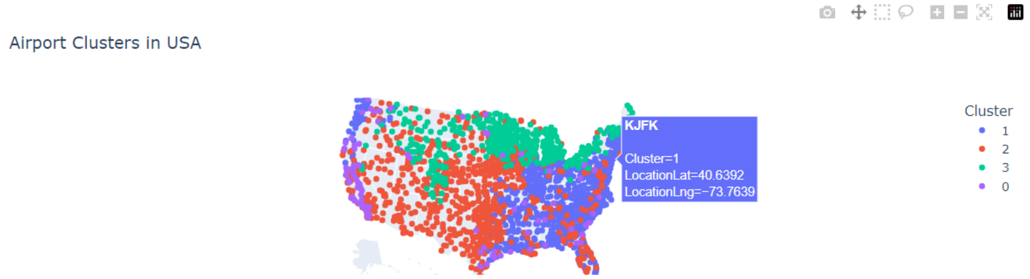
```
#City Wide Weather Cluster Distribution
# Plotting the cluster distribution across cities on a map of the USA
fig_cluster = px.scatter_geo(weather_data_cluster,
                              lat='LocationLat',
                              lon='LocationLng',
                              hover_name=weather_data_cluster['City'] + ', ' + weather_data_cluster['State'],
                              scope="usa",
                              color='Cluster',
                              color_discrete_sequence=['#AB63FA', '#EF553B', '#00CC96', '#636EFA'],
                              title='City Wide Weather Cluster Distribution')
fig_cluster.show()
```



Airport Clusters in USA:

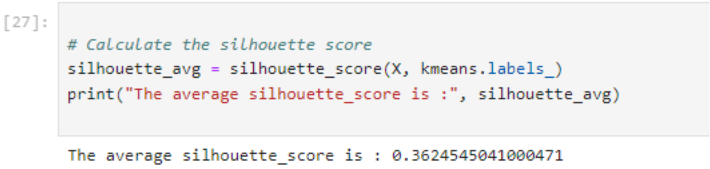


```
# Plotting the clusters on a map of the USA
fig = px.scatter_geo(weather_data_cluster,
                     lat='LocationLat',
                     lon='LocationLng',
                     color='Cluster',
                     hover_name='AirportCode',
                     size_max=10,
                     title='Airport Clusters in USA',
                     scope='usa')  # Set scope to 'usa' to restrict the map to the USA
fig.show()
```



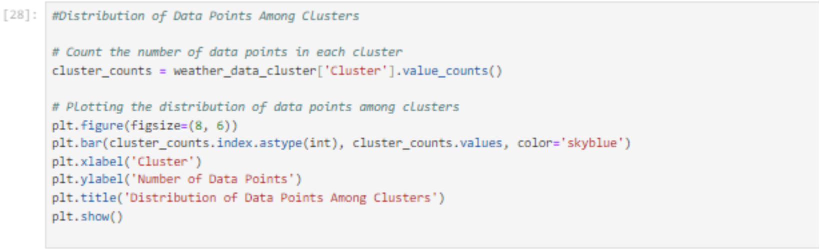
Evaluation of the Cluster division using Silhoutte Score:

The silhouette score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It quantifies the quality and separation of the clusters in a clustering analysis. The silhouette score is a useful metric for evaluating the quality of a clustering solution and determining the optimal number of clusters in a dataset. The overall silhouette score ranging from -1 to 1 for a clustering solution is the average silhouette score across all data points. Higher silhouette scores indicate better-defined clusters, while lower scores suggest overlapping or poorly separated clusters. The silhouette score below indicates a decent clustering.

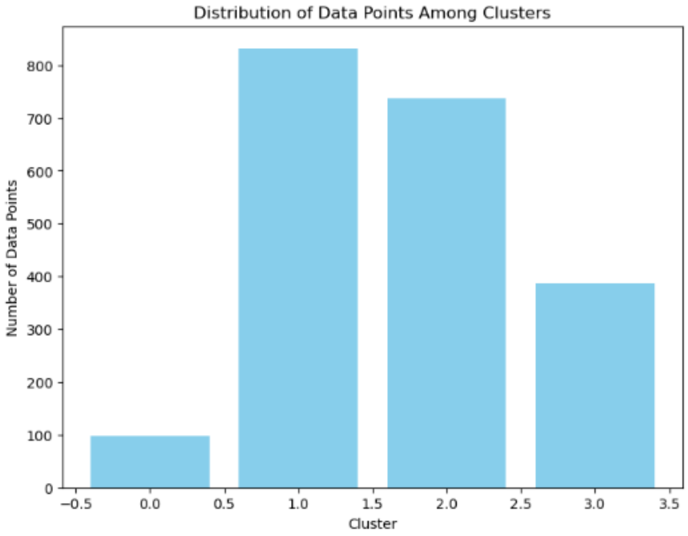


```
[27]:
# Calculate the silhouette score
silhouette_avg = silhouette_score(X, kmeans.labels_)
print("The average silhouette_score is :", silhouette_avg)

The average silhouette_score is : 0.3624545041000471
```

Distribution of Data Points among the Clusters:



```
[28]: #Distribution of Data Points Among Clusters

# Count the number of data points in each cluster
cluster_counts = weather_data_cluster['Cluster'].value_counts()

# Plotting the distribution of data points among clusters
plt.figure(figsize=(8, 6))
plt.bar(cluster_counts.index.astype(int), cluster_counts.values, color='skyblue')
plt.xlabel('Cluster')
plt.ylabel('Number of Data Points')
plt.title('Distribution of Data Points Among Clusters')
plt.show()
```

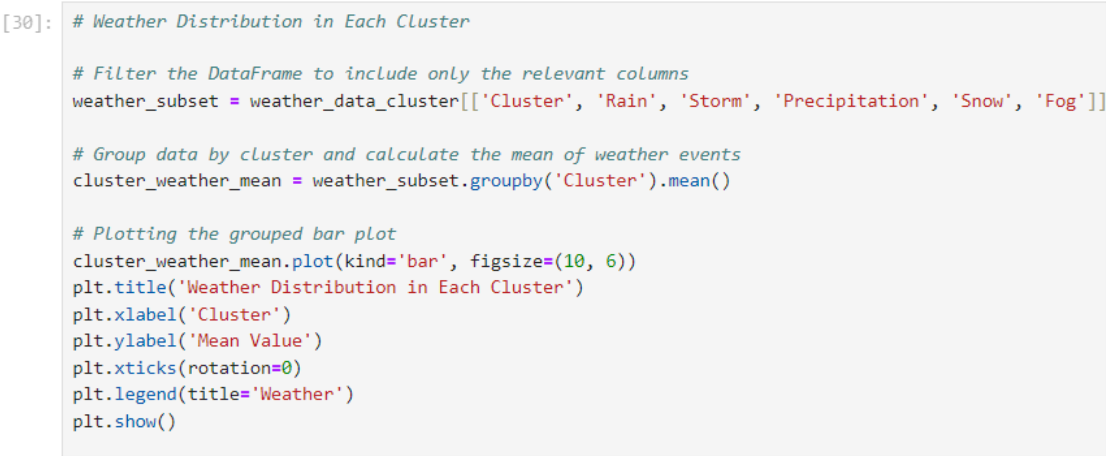


This indicates that clusters 1 and 2 are having more data points than the rest.

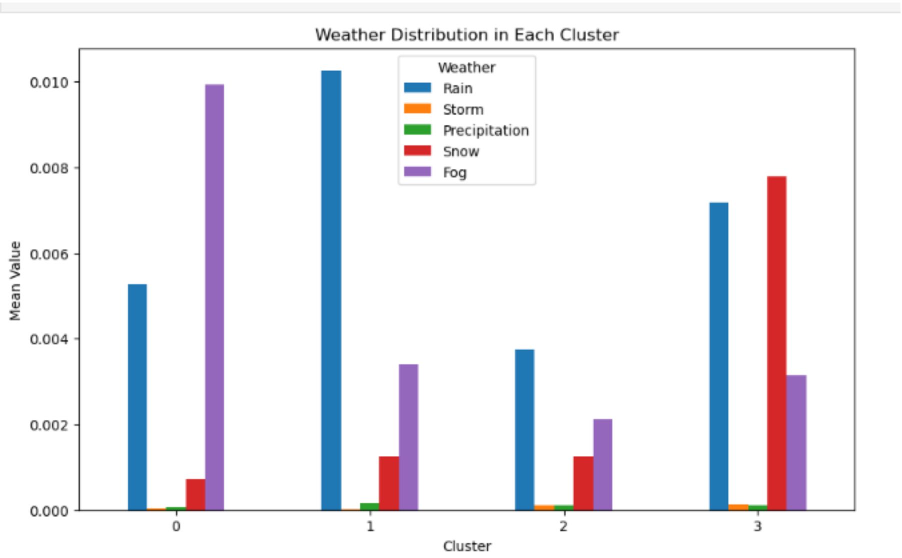
Distribution of Weather Events in Each Cluster:

From the below visualization we can draw conclusions about the highest number of weather events in each cluster.

To our observation we can say that Fog is most occurrent in cluster 0, Rain is prevalent in clusters 1 and 2, and both Snow and Rain are frequent in cluster 3.



```
[30]: # Weather Distribution in Each Cluster

      # Filter the DataFrame to include only the relevant columns
      weather_subset = weather_data_cluster[['Cluster', 'Rain', 'Storm', 'Precipitation', 'Snow', 'Fog']]

      # Group data by cluster and calculate the mean of weather events
      cluster_weather_mean = weather_subset.groupby('Cluster').mean()

      # Plotting the grouped bar plot
      cluster_weather_mean.plot(kind='bar', figsize=(10, 6))
      plt.title('Weather Distribution in Each Cluster')
      plt.xlabel('Cluster')
      plt.ylabel('Mean Value')
      plt.xticks(rotation=0)
      plt.legend(title='Weather')
      plt.show()
```



VI. Future Work

By exploring the following avenues for future work, weather data can contribute valuable insights into understanding and interpreting weather patterns, trends, and impacts for various stakeholders and applications.

Temporal Analysis: Conducting a more in-depth temporal analysis to identify long-term trends, seasonal patterns, and cyclical variations in weather events. This involves time series analysis techniques, such as decomposition, forecasting, and anomaly detection.

Extreme Events Analysis: Focusing on extreme weather events, such as hurricanes, tornadoes, floods, and heatwaves, to understand their frequency, intensity, and spatial distribution. This involves statistical modeling, risk assessment, and impact analysis of extreme weather events on infrastructure, agriculture, and communities.

Multivariate Visualization: Integrating additional variables, such as temperature, humidity, wind speed, and air pressure, into the visualization analysis to explore multivariate relationships and dependencies between different weather parameters.

User Interaction and Dashboard Development: Developing interactive dashboards and visualizations that allow users to explore and analyze the weather data dynamically. This includes features such as filtering, zooming, and drill-down capabilities to facilitate exploration and discovery.

Machine Learning Integration: Incorporating machine learning models for predictive analytics, anomaly detection, and pattern recognition in weather data. This involves building models to forecast weather events, detect unusual patterns, and classify weather phenomena based on historical data.

Collaborative Research and Applications: Collaborating with domain experts, government agencies, and research institutions to apply weather data visualization techniques to specific applications, such as climate change research, disaster management, urban planning, and agriculture.

VII. Conclusion

In conclusion, the "weather data visualization" project has provided valuable insights into understanding and analyzing weather patterns, trends, and phenomena through visual exploration. By leveraging techniques such as data visualization, spatial analysis, and temporal modeling, we have gained a comprehensive understanding of the dynamics and characteristics of weather events across different geographical regions and time periods. Through the development of interactive visualizations, and analytical tools, we have facilitated the exploration and interpretation of weather data by stakeholders, researchers, and decision-makers, enabling informed decision-making and proactive planning in various sectors, including agriculture, infrastructure, disaster management, and climate resilience.

Moving forward, the project sets the stage for further research, innovation, and collaboration in the field of weather data analysis and visualization. With continued advancements in data science, geospatial analytics, and machine learning, there are ample opportunities to enhance the capabilities of weather data visualization techniques, integrate additional data sources, and develop predictive models for forecasting and early warning systems. By fostering interdisciplinary collaboration and engaging with stakeholders from diverse domains, we can harness the power of weather data visualization to address complex challenges related to climate change, extreme weather events, and sustainable development, ultimately contributing to building more resilient and adaptive communities in the face of environmental variability and change.