Student Name: Sowmiya Kumar Student ID: s4040536

Data Preparation

Data Preparation is one of the crucial steps in the Data Science process. It includes cleansing, transforming, and combining raw data. Accurate and complete data is essential for algorithms to learn effectively and produce reliable results. After loading the dataset, a comprehensive understanding of its structure is imperative for subsequent steps. Utilizing Pandas attributes such as *'columns', 'dtypes', 'describe',* and *'info()'* aids in comprehending both the data's layout and statistical characteristics. Notably, despite the "Month" and "Day" columns containing numerical values, they are stored as objects rather than integers. Added to that, the maximum and minimum rainfall amount values seem to be inconsistent. This discrepancy necessitates scrutiny for any inaccuracies or missing values before conversion to integer data type, thereby initiating the data cleansing process. Data cleaning was performed in this assignment to identify errors in the dataset and to ensure a clean dataset is available for exploration and processing, with details of encountered errors and methods utilized provided below.

Error 1: Handling errors from data entry

The *'value_counts()*' method in Pandas provides a Series displaying counts of unique values. When applied to the "Month" column, it's evident that the majority of months are represented numerically. However, 'April' and 'Jan' are spelled out as strings. Similarly, in the "Day" column, one entry appears as 'nine'. To ensure consistency and facilitate data processing, these inconsistencies were addressed. By using the *'replace()*' method, the string representations ('April', 'Jan') were replaced with ('4','1') respectively and 'nine' with '9'.

Error 2: Handling physically impossible values

From the results of '*value_counts()*', we can infer that the "Day" column has improbable values like '48' and '200' which are impractical for a day value. To better understand the incorrect values, the rows containing these incorrect values are selected using the "==" condition and it is displayed. Added to that, the index values from the previous results are used to display a few rows preceding and succeeding the identified erroneous index using the '.loc[]' attribute. The results indicate that the "Day" column should contain incremental values, with '10' instead of '48' and '20' instead of '200'. This inference is corroborated by the count of unique values in the Day column, which shows one fewer count for 10 and 20. These values are then corrected with the help of the '*replace()*' method.

Error 3: Handling missing values

The *'isnull()*' method of Pandas finds the missing values and returns a Boolean mask indicating the locations of missing values. By applying this method to all the columns of the dataframe, all columns have null values except Year. Before addressing these null values, it's essential to look into the nature of the columns. Month and Day columns are ordinal data with a meaningful order depicting the real-world Months and Days. However, Rainfall amount (mm) values represent continuous quantities measured in millimeters instead of classifying or ordering, and the difference between any two values is meaningful. So, the

missing values of the “Rainfall amount” column need to be handled with care using mean/median/Standard deviation.

The **Month column** contains one ‘*NaN*’ value. Using the '*isnull*' method, the row with the null value is identified, and its index value is noted. To gain insight into this null value, several rows preceding and following this index are printed. Since the data is in time series format, it becomes apparent from the results that the missing value should be '1', as all values before and after in this column represent January. This is further confirmed by the unique value count of the Month column, which is one less. To address this, the Pandas *'dataframe.ffill()*' function is applied. *'ffill*', short for forward fill, propagates the last valid observation forward, thereby filling the missing value.

While counting the unique values of the **Year column** using ‘*value_counts()*’ the result showed one less count for 2017 and 2013. Added to that 2027 is present in the dataset which ideally shouldn’t be there, as per the dataset description which covers years from 2013-2023. Firstly, to identify the missing count of 2017, the unique values of Months in 2017 were displayed, which showed Month 4 with one lesser day. With these combined results, filtered Month 4 and Year 2017, which rooted to the finding of the missing row for day 6 i.e. 2017/4/6. Secondly, when the extra year 2027 is printed it shows the data for Month 4 day 6, which leads to the conclusion that the year value was wrongly typed as 2027 instead of 2017. This was then replaced using ‘*replace()*’ method. Finally, to address the lesser count issue in 2013, the values of that year were printed. The results showed that this year consists of data from Month 6, and Month 6-day 1 row was not recorded. As per dataset description, this dataset consists of years from 2013-2023. However, 2013 year has only 6 months of data, this is a limitation while trying to analyse the dataset for the given period. It's not possible to create 6 months of data and add to the dataset, which may not be valid and lead to bias So, to get a full picture of the given dataset a new row for the missing day in one month alone i.e. 2013/6/1 is added with initially setting the rainfall amount as null. (which will be handled later). After adding this row, the dataframe contains 3866 rows and 4 columns. The Month column is then converted to an ‘int’ datatype for further processing.

A similar approach is used to identify the ‘*NaN*’ value in The **Day column** and concluded that the Month number 4 should be present in the place of ‘*NaN*’. This can also be cross-verified from the count of number 4 in the Day column which is 126 and it is supposed to be 127. After replacing the Day value, the column is then converted to int datatype for further processing.

Since, the **Rainfall amount** column is continuous data, the missing values should be replaced with mean/median/Standard deviation which can help preserve the overall distribution of the data and reduce the impact of outliers. Considering the nature of the dataset, which is real-world rainfall data, calculating the mean for a particular month in a year and using it to fill in missing values would be more efficient i.e. calculating the mean of rainfall amount in 2013 month 6 and using it to fill the missing value. Before proceeding with filling missing values with mean, it is efficient to check for outliers in that column. Before proceeding with the missing values in the “Rainfall amount” column, it is better to check for outliers.

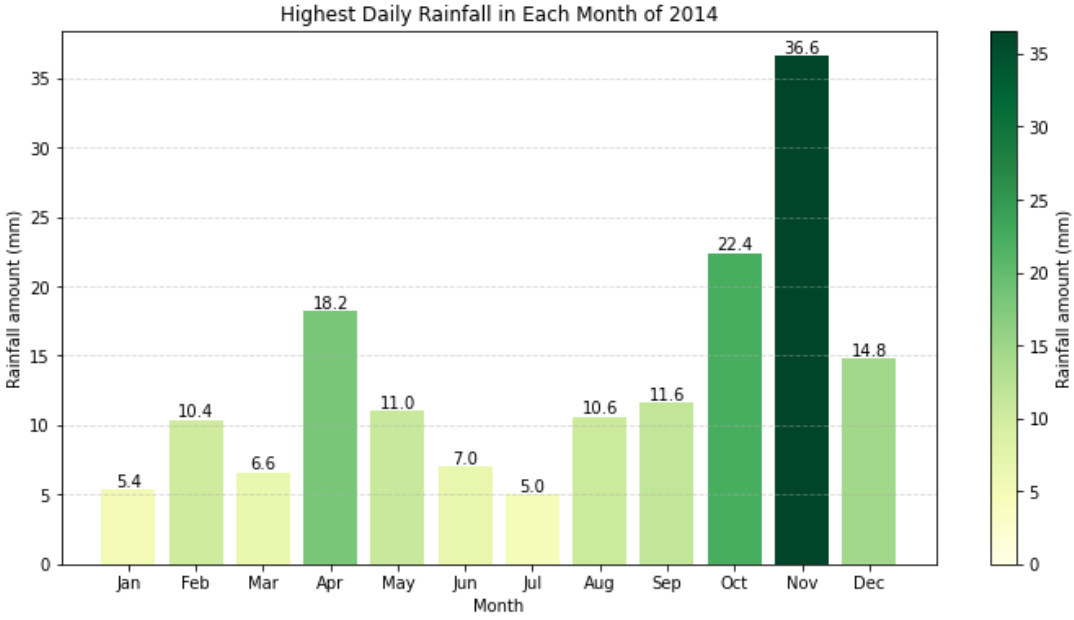
Error 4: Handling outliers

The '*describe()*' attribute describes the selected column which consists of values like count, mean, min, max, etc., The minimum and maximum values of the "Rainfall amount" are -10 and 100000 which are outliers. So initially this value was replaced as null. Post this mean value was calculated by grouping the Year and Month and the calculated value was filled in place of the missing values. The dataset was then checked for duplication and converted into a CSV file.

Data Exploration

Task 2.1

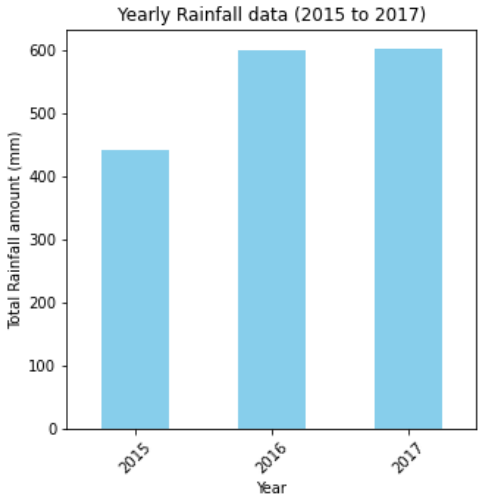
The task is to explore the highest daily rainfall in each month of the year 2014. The filtered data containing 2014 data is converted into a dataframe, with rows as the days in a month and columns as the months in a year using '*.pivot()*'. The *'.max'* function is applied to the result to find the highest daily rainfall in each row. The resulted values are then plotted in a bar graph.



The chart shows a clear seasonal variation in rainfall. Rainfall peaks in November with the highest daily rainfall of 36.6 mm which is 22.9% of total rainfall in that year followed by October and December, suggesting that this might be during a rainy season. The beginning of the year except April which shows a sudden spike of 18.2, specifically January and July, experienced the lowest amounts of rainfall, with the highest daily totals being 5.4 mm and 5.0 mm respectively, indicating a drier season during these months. Starting from June, there is a general increasing trend in the amount of rainfall. There is a high variability in the highest daily rainfall, which indicates that the location might experience both very dry and very wet conditions throughout the year. The distribution of rainfall is uneven throughout the year, with the second half of the year generally wetter than the first half.

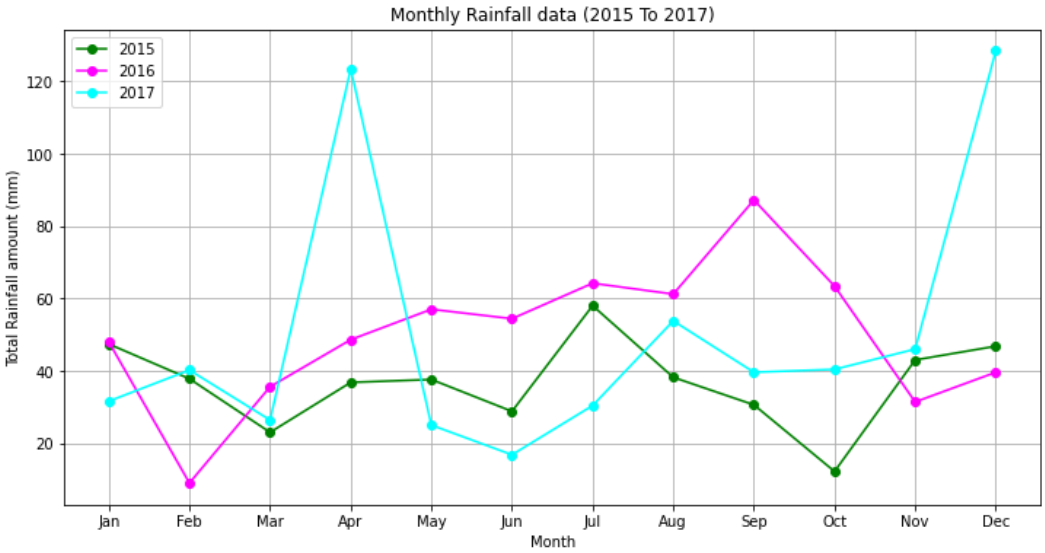
Task 2.2

The task is to explore the data between 2015 and 2017 on monthly and yearly basis. These years are filtered from the dataset using comparison (>=, <=) and logical operators (&). The *'.groupby'* method is used to group the data based on the year and each of the year's total rainfall is calculated using '*sum()*' function. The result is then plotted to a bar graph.



The graph shows an increasing trend in total rainfall over the selected three years. In 2015, the total rainfall was just over 400 mm. This increased to slightly above 500 mm in 2016, and in 2017, the total was close to 600 mm.

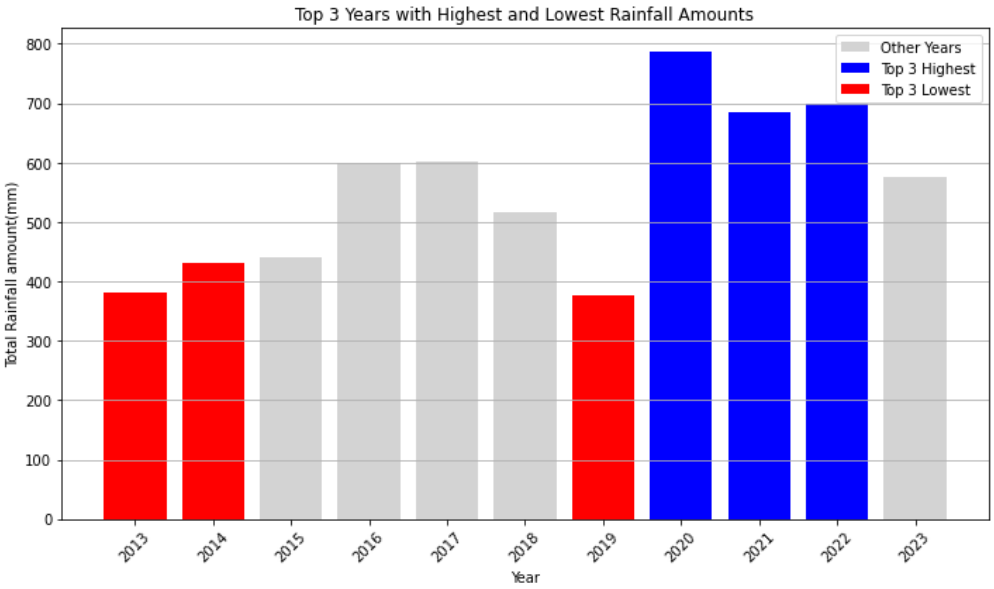
To analyse the total rainfall on a monthly basis, the dataframe is grouped by Year and Month using '*groupby*' and the total is calculated using *'sum()*'.



It can be inferred from the graph that, there is a similar rainfall pattern between 2015 and 2016, which started with the same rainfall amount of nearly 50 in Jan, and 2016 recorded a slightly higher amount in the consecutive months till Dec except Feb. The highest monthly rainfall in these three years occurred in April 2017. The beginning of the year tends to be drier, particularly February across all three years. The end of the year tends to have higher rainfall, particularly noticeable in 2017.

Task 2.3

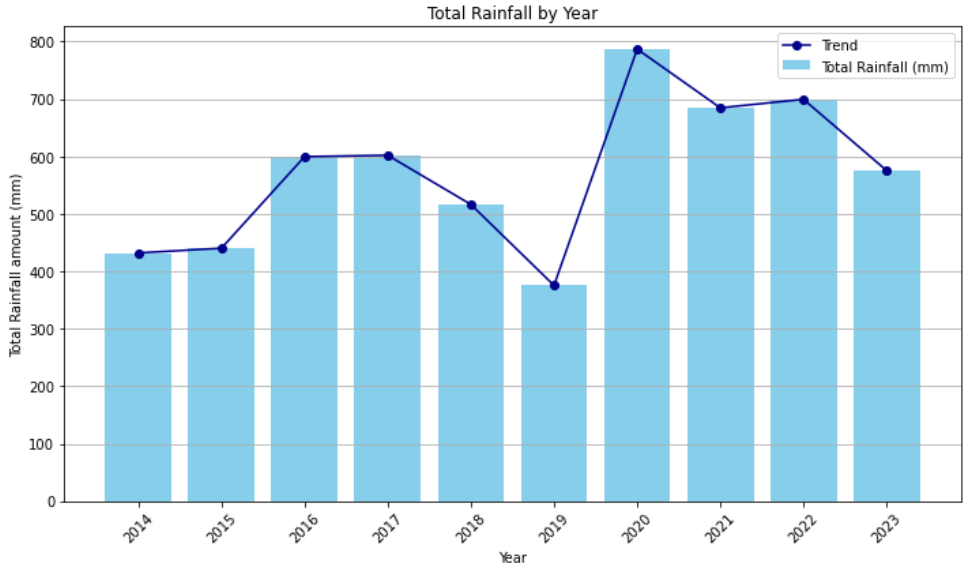
The task is to find the top 3 years with the highest and lowest rainfall amount and compare the results. The whole dataset is grouped by Year using '*groupby*' method and aggregated using '*sum()*' function. By using methods like '*nlargest()*' and '*nsmallest()*' the highest and lowest 3 values are calculated respectively. The data is then plotted into a bar graph.



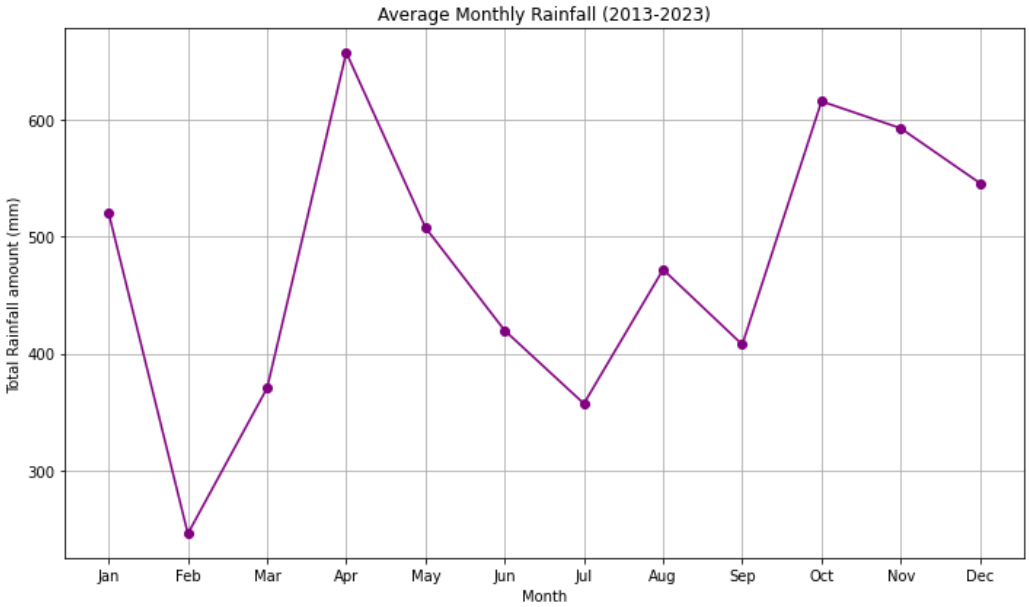
It can be inferred from the graph that the Year 2020 recorded the highest rainfall of 786.8mm and 2019 recorded the lowest rainfall of 381.2. The red bars indicate the years with the highest rainfall. These years stand out significantly compared to the others, suggesting unusually high annual rainfall amounts. The blue bars show the years with the lowest rainfall. They are markedly lower than the average, indicating particularly dry years.

Task 2.4

The task is to explore the changes in rainfall in the last 10 years. The last 10 years are filtered and grouped by year using the '*groupby*' method and then summed using '*sum()*' function to arrive at the results. The results are then plotted into a bar graph to show the trend.



There is considerable variability in rainfall from year to year, with some years experiencing significantly more rainfall than others. The trend line indicates an overall upward trend in total rainfall over the ten years, although there are fluctuations from year to year. There was a noticeable dip in rainfall in 2019 (375.4 mm) compared to the surrounding years. Similarly, there's a significant peak in 2020, suggesting a year with exceptionally high rainfall of 786.8 mm). The total rainfall has been relatively high in the most recent years (2021 to 2023). Despite the fluctuations, the general trend appears to be an increase in annual rainfall over the period.



The above graph shows the total rainfall recorded in each month for all 10 years. It is evident from the graph that, April month recorded the highest rainfall over 600mm which may be due to the season. The pattern of peaks and troughs seems consistent, suggesting that the weather has a regular seasonal cycle affecting rainfall.

References

1. https://pandas.pydata.org/docs/
2. https://matplotlib.org/
3. https://matplotlib.org/stable/api/cm_api.html
4. https://www.geeksforgeeks.org/python-pandas-dataframe-describe-method/
5. https://www.geeksforgeeks.org/python-pandas-dataframe-ffill/
6. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.merge.html