Student Name: Jigisha Tomar Student ID: s4084915

Data Preparation

To begin with my assignment, I imported pandas into my working directory. Next, I specified the file path of the CSV file that I intended to use. I then used the read_csv() function to read the CSV file, specifying the destination of the file and the separator to be ‘,’ since it was a comma-separated file. During the process, I encountered various types of errors in different columns, which I have listed below for reference:

Bad Lines Error

There’s no bad line error in the data as no extra column is generated while reading the CSV file using read_csv(). So, I proceeded to check other different errors that might be present in the data.

Redundant White Spaces

After importing the data into the workspace, I renamed the columns of the data frame as 'day', 'month', 'year', and 'rainfall'. I modified the first letter of the 'year', 'month', and 'day' columns for convenience, and I changed the name of the 'Rainfall amount (millimetres)' column because it contained white spaces, which is not a valid format for a column name.

Incorrect Data Types

I imported Numpy into my project and used the dtype() function to check the data type of the columns. After running the function, I discovered that the ‘month’ and ‘day’ columns had incorrect data types. I attempted to convert them to integer values but encountered an error message due to the presence of string values in the data.

Data Entry Errors

With the help of the value_counts() function I found the total number of counts for each value in the ‘day’ and ‘month’ columns of the dataframe.

There were two manual errors in the month column i.e. ‘Jan’, and ‘April’. I retrieved the records with the error and converted Jan to 1 and April to 4 with the help of the loc function. Similarly, in the ‘day’ column, only one data entry was not correctly stated, 9 was registered as ‘nine’. After extracting the index the day was correctly replaced as 9 with the help of loc function.

Impossible values

While I was looking for incorrect data entries with the help of the value_counts() function, I also found some impossible values in the ‘day’ and ‘year’ columns of the dataframe.

In the ‘year’ column, I found the year ‘2027’ which was not a meaningful entry as the ongoing year is 2024 and the data stored must be for the early years only. With the help of loc, I located the index of the row and deleted the record using the drop() function because I was uncertain if it was ‘2017’ or another year.

Later on, when I was working with ‘day’, I found two entries that were not between 1 and 31, all months can have a maximum of 31 days only. I deleted the entries which contained ‘48’ and ‘200’ in the day column to eliminate impossible values.

Missing values

The datatypes for ‘month’ and ‘day’ were recognized as objects rather than integers so I scanned for the missing values with the help of the .isnull() function in the columns.

I found a missing value in the ‘month’ column at index 2067. The ‘date’ column stored 29 in it, whereas the month column was empty. When dealing with missing data, it's important to avoid simply deleting records with missing values, as this can lead to a biased subset of data. To address this problem, I created a solution to provide a list of months excluding February to the random.choice() function of the numpy model as 2019 is not considered a leap year. This will help ensure that the pattern of missing values is not systematic. The list is passed on to the fillna() function with the inplace argument to make the changes permanent.

Later on, I found another missing value in the ‘day’ column at the index 1736 which had 3 in the ‘month’ column. March consists of 31 days so I generated a list with the help of the randint() function of the random module of numpy ranging from 1 to 32, 32 being excluded. I passed the list to the fillna() function and to make the changes permanent inplace argument was used. It always generates a new input to eliminate any type of bias.

There were a bunch of missing values in the ‘rainfall’ column of the dataframe. Those were replaced by the mean of the year and month the record was registered, with the help of groupby() and mean() aggregate functions. This eliminated the NaN values with meaningful data.

Outliers

To identify any outliers in the 'rainfall' column of my dataset, I imported the numpy package as the value_counts() function did not provide an effective solution. By utilizing the group by() and min() aggregate functions, I was able to obtain a table that contained different rainfall amounts along with their corresponding minimum values. During my analysis, I discovered two outliers - '-10.0' and '100000.0'. As rainfall can never be negative and the rainfall percentage cannot exceed 100, I concluded that these values were incorrect and treated them as missing values. Subsequently, I removed them from the dataset to ensure accurate analysis.

Conclusion

After completing the data preparation process, I converted the data types of the columns 'day' and 'month' to integers using the astype() function. Finally, I used the to_csv() function of the pandas dataframe to create a new CSV file named 'cleaned_version.csv', which contained the cleaned data. By converting the data types and creating a new CSV file containing the cleaned data, I ensured that the data was in the correct format and ready for any further analysis or processing.

Data Exploration

Task 2.1

I created a dataframe named 'clean_df' by importing the cleaned CSV file generated in the first part. Then, I extracted the data entries for the year '2014' from the dataframe and saved them in a variable called 'year2014'.

After that, I used the 'groupby()' function to group the data by 'day' and 'month'. Then, I applied the 'agg()' function to the 'rainfall' column to obtain the sum of rainfall for each day. Furthermore, I used the 'unstack()' function to pivot the 'month' column as column headers. I saved the resulting table as a dataframe named 'dataframe_2014'.

Finally, to get the highest daily rainfall for each month, I applied the 'max()' function to the dataframe. This generated a list of all months with each month's highest rainfall. For the graphical representation of the data, I imported matplotlib.pyplot library. I then plotted the points using the 'scatter()' function and later joined them with a line chart using the 'plot()' function of the library, further providing a label for the same. To make the graph complete I also added the labels for the x-axis and y-axis with the help of xlabel() and ylabel() respectively. I used the show() function to display the results.

For references please refer to Figure 1 in assignment1.ipynb

Task 2.2

I have utilized the matplotlib.pyplot library to visualize data using different exploration tools. I extracted the data between 2015 to 2017 from the previously created dataframe named ‘clean_df’ and created a new dataframe with it named ‘df15_17’.

To analyze data based on the year, I used the groupby() function to group the data and then used the sum() function to calculate the total amount of rainfall each year. I represented this data with the help of a pie chart using the pie() function, providing the autopct attribute to generate fraction values to show how each year’s total rainfall percentage is different because in 2016 and 2017 the amount of rainfall was quite similar. I also provided a title and labels for the graph using functions such as title(), and ylabel().

To represent the data on a monthly basis, I utilized a combination of a scatter plot and a line chart. Firstly, I grouped the data for each year and stored them in different variables. Then, I applied the sum() function to each variable to determine the total values for each month. Finally, I plotted the data using the scatter() and plot() functions, providing each line a label, color, and style to distinguish between the years.

To make the data more comprehensible, I used the xlabel() to specify the values at the x-axis, ylabel() to specify the values at the y-axis, xticks() to make sure that each number of months is specified and that it is not vivid, title() to provide a title to the graph, legend() is used to display the labels used while plotting, and grid() to show a grid in the background for easy readability, and show() function to present the graph.

For references please refer to Figures 2 and 3 in assignment1.ipynb

Task 2.3

To begin with, I gathered the data from the ‘clean_df’ for the maximum yearly rainfall, I grouped it by year and applied the max() function to get the maximum amount of rainfall in each year. Later on, I sorted it in descending order using the sort_values() function.

Then, I created a variable named ‘maxdf’ to store the years with maximum rainfall. I created another variable named ‘mindf’ to store the years with the lowest rainfall based on the sorted list.

Afterward, I again grouped the minimum and maximum variables created as ‘maxdf’ and ’mindf’, and applied the max() function to obtain accurate values for each year.

Finally, I used the bar() function from the matplotlib.pyplot library to generate a bar graph. To reduce confusion I specified different colors and labels for the bar graphs. With the help of the index and xticks(), I was able to set the x-axis of the graph from 2013-2023 to represent the years that were used in the visualization.

I made it more presentable by using xlabel() to name the x-axis, ylabel() to name the y-axis, title() to name the graph, legend() to show the labels of the bar graphs, grid() to show a network of line to show the difference between the value points, and show() to display the graph without the list of arrays.

For references please refer to Figure 4 in assignment1.ipynb

Task 2.4

I started by retrieving all the data from the 'clean_df' and using the groupby() function on the 'year' column of the dataframe. I then used the mean() function to get the average value of each year and saved the output in the 'diff' variable.

Later, I defined a list named 'ten_years' to be used as the x-axis while plotting the data. I represented the data in the form of a bar chart by using the bar() function and passing the x and y-axis arguments. I also specified the color of the bars using the color argument.

To make the graph look more presentable, I used other functions from the matplotlib.pyplot library. I added the x-axis and y-axis labels using the xlabel() and ylabel() functions, respectively. I also included a title using the title() function. To make the graph more detailed, I added the grid() function with a dotted line style, so that it doesn't overpower the actual data.

Finally, I used the show() function to display the graph successfully.

For references please refer to Figure 5 in assignment1.ipynb

References

A.Boschetti and L. Massaron, Python Data Science Essentials, Chapter 2

D. Cielen and A. Meysman and M. Ali, Introducing Data Science, Chapter 2

read_csv() arguments
(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

Converting DataFrame To CSV file
(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html)

Python Data Visualization documentation
(https://pandas.pydata.org/docs/user_guide/visualization.html)

Line Plot Formatting
(https://pandas.pydata.org/pandas-docs/version/0.23.4/visualization.html#setting-the-plot-style)

Pie plot documentation
(https://pandas.pydata.org/pandas-docs/version/0.23.4/visualization.html#pie-plot)

Bar plot documentation
(https://pandas.pydata.org/pandas-docs/version/0.23.4/visualization.html#bar-plots)

Scatter Plot documentation
(https://pandas.pydata.org/pandas-docs/version/0.23.4/visualization.html#scatter-plot)