Subject: Python- Programming for Data Processing

Assignment Task 2: Report

Name: Kunal Taneja

Student ID: 24995882

Professor: Ali Braytee

INTRODUCTION

Python Assignment 2 is based on fundamentals of python and libraries for Data processing that includes 3 parts i.e**. Python Jupyter Notebook, a report, and a video**.

This report is based on Forest Fires, the analysis of Fire weather Index, with the motive to predict the area of forest fires, in the northeast region of Portugal.
The report will cover the months that are much likely to be the victim of fire using **FFMC**(Fine Fuel Moisture Code), **DMC**(Duff Moisture Code), **DC**(Drought Code), **ISI**(Initial Spread Index) and **FWI**(Fire Weather Index).
This report includes the following:

- Synopsis of the dataset
- Data Processing Techniques
- Data Cleaning
- Data Processing outcomes

Synopsis of the Dataset

The dataset focuses on predicting the burned area of forest fires in the northeast region of Portugal. It includes various meteorological and other data points, such as temperature, humidity, wind speed, and rain, which are used as features for the regression task.

Source: *https://archive.ics.uci.edu/dataset/162/forest+fires*

Key Features of the dataset:

- ❖ Multivariate dataset with **12** real-valued features and **517** instances
- ❖ Represents Various **Meteorological** and **Environmental** factors.
- ❖ Aim to predict the burned area of forest fires.

---

12 Real-valued Features with its data types:

```
X          int64
Y          int64
month     object
day       object
FFMC     float64
DMC      float64
DC       float64
ISI      float64
temp     float64
RH         int64
wind     float64
rain     float64
area     float64
dtype: object
```

Brief explanation of each data type:

int64- Integer values represented with 64 bits of memory, typically used for whole numbers without decimals.

Object- A generic data type used for text or mixed numeric and non-numeric data.

Float64- Floating-point numbers represented with 64 bits of memory, used for numeric values with decimals.

Data Processing techniques

- **Importing Data**: Libraries like Pandas in Python play a vital role in simplifying the data import process. Whether the data is stored in CSV files, Excel Spreadsheets, Databases, or other formats, Pandas helps to import the data efficiently into the analysis environment.

```
#import the pandas as pd
import pandas as pd
#importing the dataset from Google Drive location
dataset_path = '/content/drive/MyDrive/Assignment 2/forestfires.csv'
df = pd.read_csv(dataset_path)
print(df)
```

- **Data Exploration**: Exploring the data is a crucial step in any data analysis as it provides insights into the structure patterns and characteristics of the dataset. Displaying the first few and last few rows of the dataset using functions like 'head()' and 'tail()'. Understanding the dimensions of the dataset using the 'shape' attribute.
Additionally, examining the data types of each column using the 'dtypes' function enables one to understand the nature of the data.

```
# Print first few rows of the dataset.
df.head()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

```
# Print the last few rows of the dataset.
df.tail()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
512	4	3	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44
513	2	4	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29
514	7	4	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00

- **Descriptive Statistics:** Descriptive statistics provide a summary of central tendency, and a shape of a dataset's distribution.
The 'Describe()' function in Pandas generates descriptive statistics for numeric columns in the Data Frame, including count, mean, standard deviation, minimum, maximum and quartiles. The statistics provide the insights into the distribution and variability of the numeric data, helps in understanding the overall pattern and behaviour.

```
[ ] # Descriptive statistics of the Dataset.
    df.describe()
```

	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
count	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000
mean	4.669246	4.299807	90.644681	110.872340	547.940039	9.021663	18.889168	44.288201	4.017602	0.021663	12.847292
std	2.313778	1.229900	5.520111	64.046482	248.066192	4.559477	5.806625	16.317469	1.791653	0.295959	63.655818
min	1.000000	2.000000	18.700000	1.100000	7.900000	0.000000	2.200000	15.000000	0.400000	0.000000	0.000000
25%	3.000000	4.000000	90.200000	68.600000	437.700000	6.500000	15.500000	33.000000	2.700000	0.000000	0.000000
50%	4.000000	4.000000	91.600000	108.300000	664.200000	8.400000	19.300000	42.000000	4.000000	0.000000	0.520000
75%	7.000000	5.000000	92.900000	142.400000	713.900000	10.800000	22.800000	53.000000	4.900000	0.000000	6.570000
max	9.000000	9.000000	96.200000	291.300000	860.600000	56.100000	33.300000	100.000000	9.400000	6.400000	1090.840000

➢ Separating Numeric and Non-numeric columns:

Separating columns into numeric and non-numeric (categorical or text) allow us to perform specific analysis like, numeric columns typically contain quantitative data that can be subject to mathematical operations and help to explore relationships between numeric variables such as correlation coefficients.

```
# Separating non-numeric (Categorial or text) columns from the dataset
non_numeric_columns = df.select_dtypes(exclude=['number'])
print(non_numeric_columns)
```

```
[ ] # Sepatating the numeric columns from the dataset
    numeric_columns = df.select_dtypes(include=['number'])
    print(numeric_columns)
```

```
[ ] # Finding the correlation mattrix of the numeric columns
    correlation_mattrix = numeric_columns.corr()
    print(correlation_mattrix)
```

```
                X         Y      FFMC       DMC        DC       ISI      temp
X        1.000000  0.539548 -0.021039 -0.048384 -0.085916  0.006210 -0.051258
Y        0.539548  1.000000 -0.046308  0.007782 -0.101178 -0.024488 -0.024103
FFMC    -0.021039 -0.046308  1.000000  0.382619  0.330512  0.531805  0.431532
DMC     -0.048384  0.007782  0.382619  1.000000  0.682192  0.305128  0.469594
DC      -0.085916 -0.101178  0.330512  0.682192  1.000000  0.229154  0.496208
ISI      0.006210 -0.024488  0.531805  0.305128  0.229154  1.000000  0.394287
temp    -0.051258 -0.024103  0.431532  0.469594  0.496208  0.394287  1.000000
RH       0.085223  0.062221 -0.300995  0.073795 -0.039192 -0.132517 -0.527390
wind     0.018798 -0.020341 -0.028485 -0.105342 -0.203466  0.106826 -0.227116
rain     0.065387  0.033234  0.056702  0.074790  0.035861  0.067668  0.069491
area     0.063385  0.044873  0.040122  0.072994  0.049383  0.008258  0.097844

                RH      wind      rain      area
X        0.085223  0.018798  0.065387  0.063385
Y        0.062221 -0.020341  0.033234  0.044873
FFMC    -0.300995 -0.028485  0.056702  0.040122
DMC      0.073795 -0.105342  0.074790  0.072994
DC      -0.039192 -0.203466  0.035861  0.049383
ISI     -0.132517  0.106826  0.067668  0.008258
temp    -0.527390 -0.227116  0.069491  0.097844
RH       1.000000  0.069410  0.099751 -0.075519
wind     0.069410  1.000000  0.061119  0.012317
rain     0.099751  0.061119  1.000000 -0.007366
area    -0.075519  0.012317 -0.007366  1.000000
```

- **Average Monthly Statistics**: The DataFrame is grouped by the 'month' column using the groupby() function. Then, the columns 'FFMC', 'DMC', 'DC', and 'ISI' are selected, and their mean values within each group (month) are computed using the mean() function again.

  This Average monthly Statistics help is the following ways:

  - Temporal Analysis: By grouping data based non 'month' column and calculating average values within each group helps to identify patterns, trend, and seasonality in the dataset.
  - Decision Making: This helps to make decisions effectively by providing summary of the dataset. Like, policymakers and fire management agencies can utilize these statistics to assess the effectiveness of fire prevention measures and allocate resources more effectively based on seasonal variations in fire risk.

```
[ ]  # Grouping the 'month' column and calculating average (FFMC, DMC, DC, ISI)

     Average_monthly_stats = df.groupby('month')[['FFMC', 'DMC', 'DC' , 'ISI']].mean()
     Average_monthly_stats
```

month	FFMC	DMC	DC	ISI
apr	85.788889	15.911111	48.555556	5.377778
aug	92.336957	153.732609	641.077717	11.072283
dec	84.966667	26.122222	351.244444	3.466667
feb	82.905000	9.475000	54.670000	3.350000
jan	50.400000	2.400000	90.350000	1.450000
jul	91.328125	110.387500	450.603125	9.393750
jun	89.429412	93.382353	297.705882	11.776471
mar	89.444444	34.542593	75.942593	7.107407
may	87.350000	26.700000	93.750000	4.600000
nov	79.500000	3.000000	106.700000	1.100000
oct	90.453333	41.420000	681.673333	7.146667
sep	91.243023	120.922674	734.615698	8.577326

For and While Loop

```
[56] # Finding the Sum and Average of 'Temperature' column using for loop
     temp_sum = 0
     count = 0

     # Iterate through the "temp" column
     for value in df['temp']:
         temp_sum += value
         count += 1

     # Calculate average
     temp_avg = temp_sum / count

     print("Sum of 'temp':", temp_sum)
     print("Average of 'temp':", temp_avg)

     Sum of 'temp': 9765.699999999999
     Average of 'temp': 18.88916827852998
```

This Python code calculates the sum and average of a column named 'temp' in a DataFrame using a for loop. It initializes variables for the sum and count, then

iterates through each value in the 'temp' column, adding it to the sum and incrementing the count. After the loop, it calculates the average by dividing the sum by the count and prints both the sum and average.

```
[55] # Finding the sum and Average of 'Wind Column' using While loop
     wind_sum = 0
     count = 0
     index = 0

     # Iterate through the "wind" column using while loop
     while index < len(df['wind']):
         wind_sum += df['wind'][index]
         count += 1
         index += 1

     # Calculate average
     wind_avg = wind_sum / count

     print("Sum of 'wind':", wind_sum)
     print("Average of 'wind':", wind_avg)

     Sum of 'wind': 2077.1000000000004
     Average of 'wind': 4.017601547388782
```

This Python code uses a while loop to find the sum and average of a column named 'wind' in a DataFrame. It iterates through the column, adding each value to the sum and counting the elements. After the loop, it calculates the average and prints both the sum and average.

Data Cleaning

Data cleaning is a critical step in the data analysis process as it ensures that the dataset is accurate, consistent, and ready for analysis.

Identifying the duplicates:

- Identifying the Duplicate Rows: The ‘duplicated()’ function is used to identify duplicate rows in the dataset.
  Four duplicate rows were found based on all columns i.e. rows with indices 53, 100, 215, and 303.
- The total number of duplicate rows was calculated to quantify the extent of duplication in the dataset.

```
[6]  duplicate_rows = df.duplicated()
     print("Duplicate Rows except first occurrence based on all columns are :")
     print(df[duplicate_rows])

     Duplicate Rows except first occurrence based on all columns are :
          X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain   area
     53   4  3   aug  wed  92.1  111.2  654.1   9.6  20.4  42   4.9   0.0   0.00
     100  3  4   aug  sun  91.4  142.4  601.4  10.6  19.8  39   5.4   0.0   0.00
     215  4  4   mar  sat  91.7   35.8   80.8   7.8  17.0  27   4.9   0.0  28.66
     303  3  6   jun  fri  91.1   94.1  232.1   7.1  19.2  38   4.5   0.0   0.00

     num_duplicate_rows = duplicate_rows.sum()
     print(f"Number of duplicate rows: {num_duplicate_rows}")

     Number of duplicate rows: 4
```

Identifying the null values:

```
# Code to find the null values in the dataset.

print(df.isnull().sum())
```

```
X        0
Y        0
month    0
day      0
FFMC     0
DMC      0
DC       0
ISI      0
temp     0
RH       0
wind     0
rain     0
area     0
dtype: int64
```

The code snippet utilizes 'isnull()' function to identify the null values in the dataset. It then applies the 'sum()' function to count the number of null values present in each column of the dataframe.

In this dataset there are no null values.

CONVERTING ALL THE STRING VALUES TO LOWERCASE

```
# Converting all the string values in Lower case using If statement
for column in df.columns:
    if df[column].dtype == 'object':  # Checking if column contains string values
        df[column] = df[column].str.lower()

print(df)
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.00
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.00
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.00
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.00
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.00
..	..	..	...	...	...	...	...	...	...	..	...	...	...
512	4	3	aug	sun	81.6	56.7	665.6	1.9	27.8	32	2.7	0.0	6.44
513	2	4	aug	sun	81.6	56.7	665.6	1.9	21.9	71	5.8	0.0	54.29
514	7	4	aug	sun	81.6	56.7	665.6	1.9	21.2	70	6.7	0.0	11.16
515	1	4	aug	sat	94.4	146.0	614.7	11.3	25.6	42	4.0	0.0	0.00
516	6	3	nov	tue	79.5	3.0	106.7	1.1	11.8	31	4.5	0.0	0.00

[517 rows x 13 columns]

The code iterates through each column in the dataframe and checks if the column contains string values (i.e. if its data type is 'Object'). If a column contains string values, it converts all

the string values in that column to lowercase using the 'str.lower()' function. This ensures that all the string values in the dataframe are consistently formatted in lowercase.

Importance of converting to Lowercase:

- Consistency: Converting all string values to lowercase ensures that string values are consistently formatted, making it easier to compare and manipulate it.
- Normalization: Lowercasing string values can help normalize the data, reducing the impact of case sensitivity in subsequent analysis.
- Searching and filtering: Converting string values to lowercase can simplify search and filtering operations, as it eliminates the need to consider case sensitivity.

Data Processing Outcomes:

- Identification of Duplicates: With this data processing number of duplicates are identified.
- Missing Values: With the data processing, we came to know that there is no missing values in the dataset.
- Converting string values to lowercase enhances consistency by ensuring uniformity in text representation, regardless of original case usage. This normalization simplifies comparisons and operations, reducing the complexity of case-sensitive tasks such as searching and filtering.
- Descriptive Statistics provide a snapshot of central tendencies, showing average levels of variables such as **FFMC and DMC**. Standard deviations indicate the spread of data around these averages, with higher values suggesting greater variability, as seen in ISI. Percentiles further detail data distribution, aiding in understanding the spread and presence of outliers within the dataset.
- The average monthly statistics provide valuable insights into the seasonal trends of fire weather indices. For example, higher **FFMC, DMC, DC, and ISI** values are typically observed during the summer months, particularly in August and September, indicating increased fire risk. In contrast, lower values are seen during the winter months, such as January and December.
- Data Quality: The data quality appears to be relatively high post-processing. Additionally, the conversion of string values to lowercase enhances consistency, simplifying subsequent operations. These efforts contribute to a dataset that is more reliable and usable for further analysis tasks.