Weather Dataset Analysis - Complete Documentation

Overview

This Jupyter Notebook demonstrates comprehensive data analysis techniques using a weather dataset containing hourly weather conditions. The dataset includes temperature, humidity, wind speed, visibility, pressure, and weather conditions data.

Dataset Description

The Weather Dataset is a time-series dataset with per-hour information about weather conditions at a particular location. It contains the following columns:

- **Temperature**: Air temperature measurements
- **Dew Point Temperature**: Dew point temperature
- **Relative Humidity**: Humidity percentage
- **Wind Speed**: Wind speed in km/h
- **Visibility**: Visibility distance in km
- **Pressure**: Atmospheric pressure in kPa
- **Weather Condition**: Categorical weather descriptions (Clear, Fog, Snow, etc.)

Setup and Data Loading

Import Required Libraries

```python
import pandas as pd
```

Purpose: Imports the pandas library for data manipulation and analysis.

Load the Dataset

```python
data = pd.read_csv(r"Weather Dataset.csv")
```

Purpose: Loads the weather dataset from a CSV file into a pandas DataFrame.

Display the Dataset

```python
data
```

Expected Output: Shows the entire dataset with all rows and columns. The output will display the first and last few rows with dimensions information.

DataFrame Analysis Methods

1. .head() - View First N Rows

```python
data.head()
```

Purpose: Shows the first 5 rows of the dataset (default N=5).

Expected Output:

	Temp_C	Dew Point Temp_C	Rel Hum_%	Wind Speed_km/h	Visibility_km	Press_kPa	Weather
0	-1.8	-3.9	86	7	25.0	101.24	Clear
1	-1.8	-3.7	87	7	25.0	101.24	Clear
2	-1.8	-3.4	89	7	25.0	101.26	Clear
3	-1.5	-3.2	88	8	25.0	101.27	Clear
4	-1.5	-3.3	88	8	25.0	101.23	Clear

2. .shape - Dataset Dimensions

```python
data.shape
```

Purpose: Returns the total number of rows and columns.

Expected Output: `(8784, 8)` (example - actual numbers may vary)

3. .index - DataFrame Index

```python
data.index
```

Purpose: Shows the index range of the DataFrame.

Expected Output: `RangeIndex(start=0, stop=8784, step=1)`

4. .columns - Column Names

```python
data.columns
```

Purpose: Displays all column names in the dataset.

Expected Output:

```
Index(['Date/Time', 'Temp_C', 'Dew Point Temp_C', 'Rel Hum_%',
       'Wind Speed_km/h', 'Visibility_km', 'Press_kPa', 'Weather'],
      dtype='object')
```

5. .dtypes - Data Types

python

```
data.dtypes
```

Purpose: Shows the data type of each column.

Expected Output:

```
Date/Time            object
Temp_C              float64
Dew Point Temp_C    float64
Rel Hum_%             int64
Wind Speed_km/h       int64
Visibility_km       float64
Press_kPa           float64
Weather              object
dtype: object
```

6. .unique() - Unique Values in a Column

python

```
data['Weather'].unique()
```

Purpose: Shows all unique values in the Weather column.

Expected Output: `array(['Clear', 'Mainly Clear', 'Partly Cloudy', 'Mostly Cloudy', 'Cloudy', 'Fog', 'Rain', 'Snow', 'Freezing Rain'], dtype=object)`

7. .nunique() - Count of Unique Values

python

```
data.nunique()
```

Purpose: Returns the number of unique values in each column.

Expected Output:

```
Date/Time           8784
Temp_C               122
Dew Point Temp_C     112
Rel Hum_%             84
Wind Speed_km/h       28
Visibility_km         19
Press_kPa            340
Weather                9
dtype: int64
```

8. .count() - Non-null Value Count

python

```
data.count()
```

Purpose: Shows the total number of non-null values in each column.

9. .value_counts() - Value Frequency

python

```
data['Weather'].value_counts()
```

Purpose: Shows the frequency of each unique value in the Weather column.

Expected Output:

```
Clear           2370
Mainly Clear    1088
Cloudy           933
Partly Cloudy    848
Mostly Cloudy    442
Fog              150
Rain             306
Snow             583
Freezing Rain     64
Name: Weather, dtype: int64
```

10. .info() - DataFrame Information

python

```
data.info()
```

Purpose: Provides comprehensive information about the DataFrame including data types, non-null counts, and memory usage.

Data Analysis Questions and Solutions

Q1: Find all unique 'Wind Speed' values

python

```
data.head(2)  # Preview first 2 rows
data.nunique()  # Count unique values in all columns
data['Wind Speed_km/h'].nunique()  # Count unique wind speeds
data['Wind Speed_km/h'].unique()  # Show all unique wind speeds
```

Solution: The `.unique()` method returns an array of all distinct wind speed values in the dataset.

Q2: Count instances when Weather is exactly 'Clear'

python

```
# Method 1: Using value_counts()
data.Weather.value_counts()

# Method 2: Using filtering
data[data.Weather == 'Clear']

# Method 3: Using groupby()
data.groupby('Weather').get_group('Clear')
```

Explanation: Three different approaches to find Clear weather instances:

- `value_counts()`: Shows frequency of all weather conditions
- Filtering: Returns subset where condition is met
- `groupby()`: Groups data by weather condition and extracts specific group

Q3: Find instances when Wind Speed was exactly 4 km/h

python

```
data[data['Wind Speed_km/h'] == 4]
```

Solution: Uses boolean indexing to filter rows where wind speed equals 4 km/h.

Q4: Find all Null Values

python

```python
data.isnull().sum()  # Count null values per column
data.notnull().sum()  # Count non-null values per column
```

Purpose:

- `isnull().sum()`: Returns count of missing values in each column
- `notnull().sum()`: Returns count of valid values in each column

Q5: Rename Column

python

```python
data.rename(columns={'Weather': 'Weather Condition'}, inplace=True)
data.head()  # Verify the change
```

Explanation:

- `rename()`: Changes column names
- `inplace=True`: Modifies the original DataFrame
- `columns` parameter: Dictionary mapping old names to new names

Q6: Calculate Mean Visibility

python

```python
data.Visibility_km.mean()
```

Expected Output: Returns the average visibility value (e.g., `27.664447368421053`)

Q7: Calculate Standard Deviation of Pressure

python

```python
data.Press_kPa.std()
```

Expected Output: Returns the standard deviation of pressure values.

Q8: Calculate Variance of Relative Humidity

python

```python
data['Rel Hum_%'].var()
```

Expected Output: Returns the variance of relative humidity values.

Q9: Find all Snow instances

```python
# Method 1: Exact match
data[data['Weather Condition'] == 'Snow']

# Method 2: Contains 'Snow' (includes partial matches)
data[data['Weather Condition'].str.contains('Snow')].tail(50)
```

Explanation:

- Exact match: Finds rows where weather condition is exactly 'Snow'
- `str.contains()`: Finds rows containing 'Snow' anywhere in the string

Q10: Complex Filtering - Wind Speed > 24 AND Visibility = 25

```python
data[(data['Wind Speed_km/h'] > 24) & (data['Visibility_km'] == 25)]
```

Key Points:

- Use `&` for AND operation (not `and`)
- Use `|` for OR operation (not `or`)
- Parentheses are required around each condition

Q11: Mean values by Weather Condition

```python
data.groupby('Weather Condition').mean()
```

Purpose: Groups data by weather condition and calculates mean for all numeric columns.

Q12: Min/Max values by Weather Condition

```python
data.groupby('Weather Condition').min()  # Minimum values
data.groupby('Weather Condition').max()  # Maximum values
```

Purpose: Shows minimum and maximum values for each numeric column, grouped by weather condition.

Q13: Filter Fog Records

python

```
data[data['Weather Condition'] == 'Fog']
```

Purpose: Returns all rows where weather condition is 'Fog'.

Q14: OR Condition - Clear Weather OR High Visibility

python

```
data[(data['Weather Condition'] == 'Clear') | (data['Visibility_km'] > 40)].tail(50)
```

Purpose: Finds records meeting either condition (Clear weather OR visibility > 40).

Q15: Complex Boolean Logic

python

```
data[(data['Weather Condition'] == 'Clear') & (data['Rel Hum_%'] > 50) | (data['Visibility_km']
```

Logic:

- (Weather is Clear AND Humidity > 50) OR (Visibility > 40)
- **Note**: Operator precedence matters; use parentheses for clarity

Key Programming Concepts

Boolean Indexing

- Use comparison operators (`==`, `>`, `<`, `>=`, `<=`, `!=`) to create boolean masks
- Combine conditions with `&` (AND) and `|` (OR)
- Always use parentheses around individual conditions

Data Aggregation

- `groupby()`: Groups data by categorical variables
- Aggregate functions: `mean()`, `sum()`, `count()`, `min()`, `max()`, `std()`, `var()`

String Operations

- `str.contains()`: Search for substrings within text data
- Case-sensitive by default; use `case=False` for case-insensitive searches

Missing Data Handling

- `isnull()`: Identifies missing values
- `notnull()`: Identifies non-missing values
- `dropna()`: Removes rows/columns with missing data
- `fillna()`: Fills missing values with specified values

Best Practices

1. **Always preview data** using `head()`, `info()`, and `describe()` before analysis
2. **Check for missing values** using `isnull().sum()`
3. **Understand data types** using `dtypes`
4. **Use meaningful variable names** and comments
5. **Verify results** by cross-checking with different methods
6. **Handle edge cases** like missing data appropriately

Common Pitfalls

1. **Boolean Operators**: Use `&` and `|`, not `and` and `or`
2. **Parentheses**: Required around each condition in complex boolean expressions
3. **Column Names**: Use exact spelling and case-sensitivity
4. **Chaining Operations**: Be careful with method chaining; break into steps for clarity

This documentation provides a comprehensive guide to analyzing the weather dataset using pandas, covering fundamental data exploration techniques, filtering operations, and statistical analysis methods.