# Data Format for Wind Energy Applications

## Organizing data

Effective data organization is essential for maximizing the utility of datasets in computational analyses. While spreadsheets are commonly used for data entry, their role often extends to creating tables for publications, generating summary statistics, and producing figures. However, these tasks can be challenging in spreadsheets due to the need for special formatting and the potential for unpredictable results when transferring data to document software. Additionally, the graphical interface of spreadsheets makes it difficult to replicate complex calculations, increasing the risk of inconsistencies.

Command-line-based programs like Python or R offer more reliability by ensuring uniform application of calculations across datasets. This section will outline best practices for organizing and formatting your data for seamless use in these programming environments.

## Formatting Data for Programming Languages

Proper data formatting is crucial when working with programming languages like Python to avoid issues related to data manipulation and analysis. Consistent and clean data formatting ensures that your code runs smoothly, without errors related to data types or mismatched formats.

In this section, we will explore how to format your data in text format for use in Python, including tips on data cleaning and preparation. Ensuring your data is properly formatted before importing it into Python can save a significant amount of time and prevent potential errors.

We will guide you through the process of preparing your data in a text-based format ready for analysis. This includes:

* Data Cleaning: Removing any inconsistencies, duplicates, and errors in your dataset to ensure accuracy.
* Data Formatting: Ensuring all data is in a consistent format, such as CSV, which is easily readable by Python.
* Data Validation: Checking for and correcting any issues related to data types, missing values, and formatting errors.
* Preliminary Analysis: Generating summary statistics to understand the basic properties of your dataset and identify any potential issues early on.

By following these best practices, you can enhance the reliability and efficiency of your data analysis workflows, ultimately leading to more robust and reproducible results.

### Importance of Proper Data Formatting

A common mistake is treating data files like lab notebooks, relying on context, notes, and spatial layout to convey information. While humans can interpret these cues, computers require explicitly defined structures. Properly formatted data allows you to harness the power of programming languages for effective and efficient data analysis. Consistently formatted data ensures that computational tools can accurately and reliably process your information, making your analyses more robust and reproducible.

### Best Practices for Data Formatting

#### Save each data file separately

Ensure each text file contains only one table. Each row should represent a distinct observation, and each column should represent a variable.

If the data is saved in spreadsheet format with many tabs, combine data from all tabs into a single text file, adding additional columns to differentiate data where necessary. Alternatively, separate each tab into individual text files.

#### Including Metadata in Data Table

Adding metadata directly into the data table disrupts data parsing. Store metadata in a separate file. For example, the first 9 lines in the raw file *Turbine\_Data\_Kelmarsh\_1\_2022-01-01\_-\_2023-01-01\_228.csv* include metadata that can create issues when loading the data in Python.

Example:

# This file was exported by Greenbyte at 2023-08-15 14:16:56. Please see https://www.greenbyte.com for more information about Greenbyte.

#

# Turbine: Kelmarsh 1

# Turbine type: Senvion MM92

# Time zone: UTC

# Time interval: 2022-01-01 00:00:00 - 2023-01-01 00:00:00 (365 days)

#

# Data that is missing or is erroneous has been marked with the value "NaN"

#

#### Not Filling in Zeros

Leaving cells blank instead of filling in zeros can lead to misinterpretation of data as missing rather than zero. Explicitly enter zeros where applicable to indicate a measured value of zero.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,0

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

#### Using Problematic Null Values

Using arbitrary values like -999 or 999 to indicate missing data can cause issues in analysis. Use a consistent and appropriate null indicator like NA or leave cells blank.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00, ,1.2872216897689304

#### Consistent Delimiters:

Ensure your text data uses consistent delimiters (e.g., commas, tabs) to separate fields. CSV (Comma-Separated Values) is a widely accepted format.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

#### Uniform Data Types:

Ensure each column contains data of the same type. For example, a Wind\_speed column should only have numeric values.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

#### Avoid Special Characters:

Avoid using spaces, special characters, or starting field names with numbers. Remove or escape special characters that might interfere with data parsing. Use descriptive, alphanumeric field names with underscores or camel case within text fields.

Example:

# Date and time,Wind speed (m/s),"Wind speed, Standard deviation (m/s)"

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

#### Standardize Date Formats:

Storing dates in a single column might initially seem convenient, but it can introduce significant ambiguities and compatibility issues. Different software and programming environments may interpret the same date format differently, leading to potential errors and inconsistencies in data processing. For instance, the date "03/04/2021" might be interpreted as March 4th, 2021, in one context and April 3rd, 2021, in another, depending on regional settings and software defaults.

Separate Year, Month, and Day: To eliminate ambiguity and make date manipulation easier programmatically, it is recommended to store dates as separate fields for the year, month, and day. This approach ensures that each component of the date is clearly defined and less prone to misinterpretation by various software tools.

Example:

Year,Month,Day

2015,03,24

Use Unambiguous Date Formats: When storing dates as strings, use a consistent and sortable format like YYYYMMDD for dates and YYYYMMDDhhmmss for date-times. This format is both unambiguous and easily sortable, which is crucial for maintaining data integrity across different systems and applications.

Examples:

Date

20150324

Handle Missing Dates Properly: Missing dates should be clearly indicated to avoid misinterpretation. Use a consistent placeholder such as NA or leave the cells blank. This practice ensures that the absence of data is explicitly recognized as missing rather than misinterpreted as a default or erroneous value.

Example:

Year,Month,Day

2015,03,24

NA,NA,NA

#### By following the best practices, you can ensure that your dates are handled correctly and consistently in programming environments, minimizing errors and enhancing the reliability of your data analysis.

#### Treat Dates as Multiple Pieces of Data: Store year, month, and day as separate fields rather than a single entry to reduce ambiguity.

#### Use Consistent, Unambiguous Formats: Adopt formats like YYYYMMDD for dates and YYYYMMDDhhmmss for date-times to ensure clarity and consistency.

#### Clearly Indicate Missing Dates: Use placeholders like NA or leave cells blank to clearly denote missing values.

#### Utilize Built-in Functions for Date Handling: In programming environments, leverage built-in functions to extract and manipulate date components effectively.

#### Maintain Raw Data Integrity:

Never modify your raw data directly. Always create a copy for any cleaning or analysis to ensure reproducibility.

#### Keep Track of Data Cleaning Steps:

Document all data cleaning steps in a plain text file stored in the same folder as the data file. This practice ensures that your analysis is reproducible and transparent.

#### Using Formatting to Convey Information

Using cell formatting (e.g., color, bold text) to convey information that isn't captured in the data values themselves can be problematic. Use additional columns to encode any extra information.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation, Status

2022-01-01 00:00:00,6.781222248077393,1.182439280264067, checked

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304, unchecked

#### Placing Comments or Units in Cells

Including comments or units directly in data cells can cause issues. Use separate columns for comments and ensure all measurements in a column are in the same unit, specifying units in the column header. You can also save the unit of each column in a separate raw text file.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation, Failure

2022-01-01 00:00:00,6.781222248077393,1.182439280264067,

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304, “Gearbox error”

#### Entering Multiple Pieces of Information in One Cell

Combining multiple pieces of information within a single cell can complicate data analysis. Separate data into distinct columns.

Example:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation, Failure, Component

2022-01-01 00:00:00,6.781222248077393,1.182439280264067, ,

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304, Yes, “Gearbox”

#### Structuring Data for Programming

The cardinal rule for data structure is to keep it “tidy”:

* Columns = Variables: Each column should represent a variable (e.g., Wind\_speed, Power).
* Rows = Observations: Each row should represent a single observation or record.
* Cells = Values: Each cell should contain a single data value.

Example of well-structured data:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

#### Key Points

By following these practices, you ensure that your data is ready for analysis in programming languages like Python, avoiding common pitfalls related to data formatting and quality:

* Never modify your raw data directly. Always work on a copy.
* Use consistent formatting and delimiters.
* Clearly indicate missing values.
* Maintain uniform data types within columns.
* Document all data cleaning steps for reproducibility.

By adhering to these guidelines, you can significantly enhance the reliability and efficiency of your data analysis workflows, ultimately leading to more robust and reproducible results.

## Exporting Data

When preparing data for analysis in programming environments such as Python, exporting it from spreadsheets in a way that is both universal and reliable is crucial. This guide will help you understand why using common spreadsheet formats is problematic and how to properly export your data to formats like CSV or TSV to ensure compatibility and accuracy.

Issues with Proprietary Formats: Storing your data in proprietary formats such as Excel's default (\*.xls or \*.xlsx) can lead to several issues:

* Proprietary Nature: Future technology might not support these formats.
* Compatibility: Other software might not open these files correctly.
* Version Differences: Different Excel versions might handle data differently.
* Repository Requirements: Most data repositories do not accept proprietary formats.

To avoid these issues, use universal, open, and static formats like CSV (Comma-Separated Values) or TSV (Tab-Separated Values). These formats are plain text files that can be opened and read by almost any software, ensuring maximum portability and longevity.

Exporting Data to CSV from Excel: To save a file in CSV format from Excel:

* Open your file in Excel.
* Select "File" > "Save As".
* Choose "Comma Separated Values" (\*.csv) from the format list.
* Check the file name and save location, then click "Save".

You can open CSV files in Excel, but ensure compatibility with other programs by verifying the format after saving.

Example of problematic CSV data:

Date\_and\_time,Wind\_speed,Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00,6.781222248077393,1.182439280264067

2022-01-01 00:10:00,6.936052060127258,1.2872216897689304

TSV as a solution:

Date\_and\_time Wind\_speed Wind\_speed\_Standard\_deviation

2022-01-01 00:00:00 6.781222248077393 1.182439280264067

2022-01-01 00:10:00 6.936052060127258 1.2872216897689304

By following these practices, you ensure your data is in a robust format for analysis in programming environments, minimizing errors and maximizing compatibility.