

Activities 2: Working with Python pandas DataFrame

Table of contents

Introduction and pre-requisites

1. Working with Pandas

2. Use pandas to open .csv and .xlsx files and create a DataFrame

3. Use pandas to describe the contents of a DataFrame

4. Identify missing values

5. Introduction to charts with pandas.DataFrame.plot()

6. Explore distributions of values

7. Timeseries data

8. Identifying values in categorical data

9. Data preparation

10. Referencing specific columns and/or rows in a DataFrame

11. Removing columns

12. Resolve missing or incorrect values

13. Changing datatypes

14. Adding new columns

15. Joining dataframes

16. Save the prepared dataset

17. Using the prepared data

18. Next steps

Introduction and pre-requisites

Theme: Using Python to work with data

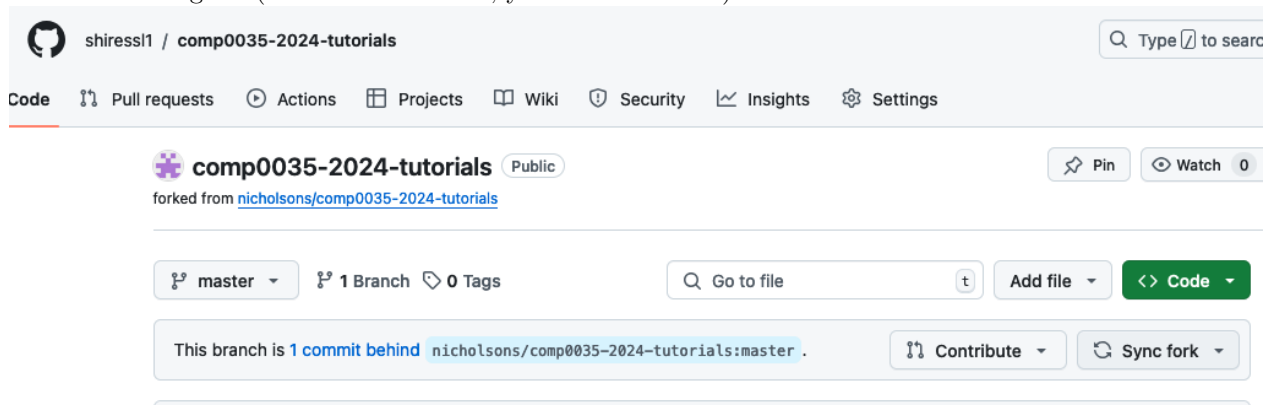
Pre-requisites

1. Completed coding activities 1
2. Forked the [COMP0035 tutorials 2025 repository \(https://github.com/nicholsons/COMP0035-tutorials-2025\)](https://github.com/nicholsons/COMP0035-tutorials-2025), cloned it to your computer, and set up a project with a virtual environment within your IDE (VS Code or PyCharm).

1. Update the forked tutorials repository

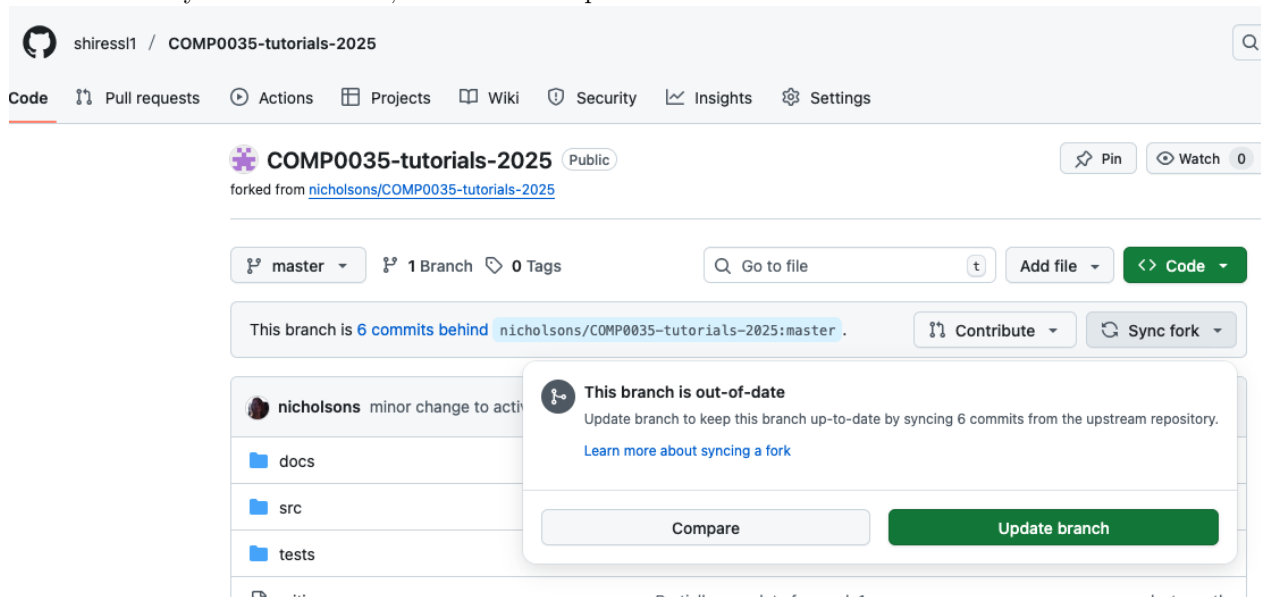
Login to GitHub and navigate to your forked copy of the [COMP0035 tutorials 2025 repository \(https://github.com/nicholsons/COMP0035-tutorials-2025\)](https://github.com/nicholsons/COMP0035-tutorials-2025).

Check whether any changes have been made. For example, the image below shows 1 new commit has been made to the original (2024 version shown, you must use 2025).



If changes have been made, you will need to update your forked repository.

Click on the “Synch fork” button; and then on “Update branch”.



Now, open your IDE (VS Code, PyCharm) and update the local copy of the repository. This assumes you have integrated your IDE with your GitHub account in week 1. You may be prompted to log in to GitHub before you can carry out the following.

- In PyCharm try menu option Git > Pull
- In VS Code click on the source code control icon on the left side panel, then when the source code control pane opens, click on the three dots and select Pull.

There are other methods, look in the Help for either PyCharm or VSCode.

2. Check you have the virtual environment activated

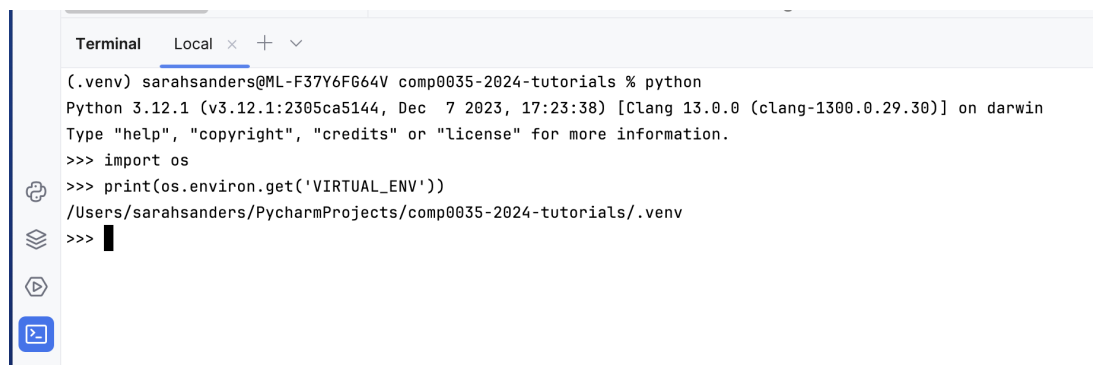
Open a terminal window within your IDE in the project directory.

Check that your virtual environment is activated. There are various ways to do this, IDEs vary, usually a quick visual way is to check whether the prompt starts with (.venv) or the name of your venv folder if not .venv. You can also use Python in the Terminal:

```
import os

print(os.environ.get('VIRTUAL_ENV'))
```

The following screenshot shows this in PyCharm on macOS:

A screenshot of a PyCharm terminal window. The terminal title bar shows 'Terminal' and 'Local'. The prompt is '(.venv) sarahsanders@ML-F37Y6F664V comp0035-2024-tutorials %'. The command 'python' has been entered, showing the Python version '3.12.1 (v3.12.1:2305ca5144, Dec 7 2023, 17:23:38) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin'. The user has entered 'import os' and 'print(os.environ.get('VIRTUAL_ENV'))', resulting in the output '/Users/sarahsanders/PycharmProjects/comp0035-2024-tutorials/.venv'.

0-1. Check for active venv

If you are not in a venv, refer to activating a virtual environment in [Week 1 activity 2 \(../1_structure/1-02-environments.md\)](#).

3. Check you have the required libraries installed in the virtual environment

Open the Terminal in your IDE.

At the prompt, enter: `pip list`

This outputs a list of installed Python packages and their versions. Check for `pandas` and `openpyxl`.

If missing, install these using `pip` e.g. `pip install pandas openpyxl`

¹ The default installation of pandas can only open Excel .xlsx files with an additional library called openpyxl ¹.

Complete the activities

This set of activities use Python pandas to describe, explore and prepare data from one or more raw data files (data set). The activities also introduce you to writing code in a way that is potentially re-usable in a project.

The activities in weeks 2, 3 and 4 are not evenly balanced, with more in week 2 and less in week 4. If you can't complete this set of activities within the week, continue in a later week.

1. [Create a package and module \(2-01-python-structure\)](#)
2. [Open .csv and .xlsx files and create a DataFrame \(2-02-pandas-df\)](#)
3. [Describe the dataframe \(2-03-pandas-describe\)](#)
4. [Identify missing values \(2-04-missing-values-identify.md\)](#)
5. [Overview of plots \(2-05-plot-overview.md\)](#)
6. [Plots to show distributions \(2-06-plot-distribution.md\)](#)
7. [Plots for timeseries data \(2-07-plot-timeseries.md\)](#)
8. [Columns with categorical values \(2-08-categorical-data\)](#)
9. [Data preparation \(2-09-data-prep.md\)](#)
10. [Locating rows and columns \(2-10-locating-rows-cols.md\)](#)
11. [Remove columns \(2-11-removing-columns.md\)](#)
12. [Remove and replace values \(2-12-resolve-missing-incorrect-values.md\)](#)
13. [Change data types \(2-13-change-datatypes.md\)](#)
14. [Add new data by computing \(2-14-new-column.md\)](#)
15. [Add new data by combining dataframes \(2-15-joining-dataframes.md\)](#)
16. [Save dataframe to file \(2-16-save-df-to-file.md\)](#)
17. [Check the prepared data suits the purpose \(2-17-questions.md\)](#)
18. [Next steps \(2-18-next-steps.md\)](#)

1. Working with Pandas

Read and complete actions before moving to the next activity.

Introduction

This minor focuses on developing data-driven software applications using Python.

Writing code for applications differs to simply writing code that functions. COMP0035 focuses on the supporting practices.

In this tutorial, you are asked to write code considering how the code could be re-used as part of an application. This implies also that other developers may use your code so your code needs to run for anyone, not just yourself.

This tutorial expects you to:

- Use modules, packages and functions to structure your application code
- Import modules, packages and functions in your code
- Access data files in a way that is not specific only to your computer

Project directory structure

Last week you cloned the activities repository to a project in your IDE and created a virtual python environment.

Check if you already installed the tutorial code by running `pip list`. The output should have something like:

Package	Version	Editable project location
astroid	3.3.11	
comp0035-tutorials	2025.0.1	/path_on_your_computer/your_projects/
COMP0035-tutorials-2025		
contourpy	1.3.3	

If not then install an editable version of your own code in it using `pip install -e ..`

This tells your IDE about the structure of your COMP0035 tutorials project, that you have a directory called 'src' which contains packages. Packages by default are auto discovered in the root of a project, not in a non-package subdirectory.

The code package location is defined in `pyproject.toml`.

The project structure can contain the following:

- A Python module (<https://docs.python.org/3/tutorial/modules.html>) is “a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended.”
- A Python package (<https://docs.python.org/3/tutorial/modules.html#packages>) is “a way of structuring Python’s module namespace by using ‘dotted module names’. For example, the module name `A.B` designates a submodule named `B` in a package named `A`.” There is a more advanced concept, ‘namespace packages’, in Python that is not covered in this minor.

- A Python function (<https://docs.python.org/3/glossary.html#term-function>) is “A series of statements which returns some value to a caller. It can also be passed zero or more arguments which may be used in the execution of the body.”
- Directories (or folders) and files that are not Python code files, e.g. data files such as `.csv`, `.xlsx`; or database files such as `.sqlite` or `.db` files.
- Project configuration and information files e.g. `README.md`, `.gitignore`, `pyproject.toml` etc.

ACTIVITY: Create a package and module for the tutorial activities

In the `src/activities` folder create a package (folder) for your work. Give it a meaningful name so you know this is your code solutions to the activities, e.g. `'solutions'`, `'student'`, `'my_code'`.

Inside the folder you just created, add a python file named exactly like this: `__init__.py` (you might already have this, some IDEs add it for you when you add a new package)

Now create a new python file, a module, in that package folder for the tutorial activities. Note: Do not call it `pandas.py` as it could conflict with the `pandas` package!

File locations

The problem with file paths

You will be using data files that are stored in your project's directory structure. You will need to be able to reference the location of these files in your Python code.

If you hard-code the file paths that are specific to your computer when writing code that others will use issues will arise due to:

1. Windows and Unix/Mac file paths are different.

Consider a Mac/Unix style file path `/Users/jojo/py_project/test.py` and a Windows file path `C:\\Documents\\py_project\\test.py`. The syntax and structure are different. Further where you have the files for a project on your computer is likely different to where someone else does, so using shared code where file paths are written using a particular operating system format using a given person's directory structure quickly becomes problematic.

2. The root folder can differ depending on your code editor.

If you are writing code in VSCode and PyCharm then filepath roots are typically the project root, that is you can add paths that are relative to your project root and ignore the preceding local file system directory structure. As an example a file in the data folder would be `data/file.csv` and not `C:\\Documents\\py_project\\data\\file.csv`. Different environments and editors may set the project root differently.

3. The relative path can differ depending on where the code is called from.

If you use a file path in a code file so that it is relative to the current file, you are likely to get issues if you then import that file and execute it from another. For example: You have the following directories and files: `/data/datafile.csv`, `/module_a/code_a.py` and `/module_a/module_b/code_b.py`. In `code_a.py` if you reference the datafile using `../data/datafile.csv` in a function that you then import and use inside `code_b.py` you might get an issue as relative to `code_b.py` the data file is not in `../data/datafile.csv` but in `../../data/datafile.csv`. It's a little more complex than this, however, using relative file paths can lead to problems.

When you are working with packages in Python then relative paths are relative to the current working directory rather than the code file it is written in.

4. Referencing files in web apps introduces further complexity.

In web apps you will also need to consider that where files are located on a development platform for example is likely different to that of the deployed version. This isn't covered in COMP0035.

Using Python libraries to determine the path

There are different ways you can do this in Python, and differing opinions on which is best.

Using pathlib

One approach is to use the `os` or `pathlib` package, both are part of the base Python installation. These allow you to define relative paths to the file i.e. locate a file relative to another. Pathlib is newer, it:

- Avoids the `\\` versus `/` issue by using the Pathlib `joinpath` method.
- Has methods that let you determine the current working directory e.g. `pathlib.Path.cwd()`.
For example: `my_file_path = pathlib.Path.cwd().joinpath('data', 'datafile.csv')` instead of `data/datafile.csv` or `data\datafile.csv`
- Allows you to code relative to the current code file, whatever that file is
e.g. `pathlib.Path(__file__).parent` would go to the directory that is the parent of the current file.

For example, given the file structure below:

```
my_python_project/  (root)
|— this_script.py
|— data/
|   |— example.csv
```

Then `pathlib` can be used in `this_script.py` to locate the `example.csv` file:

```
from pathlib import Path

# This script is located in the project root, so find the path to the
# current file and then go to the parent of that file
project_root = Path(__file__).parent

# Find the .csv file relative to the project root and join to that path
# the data folder and then the example.csv file
csv_file = project_root.joinpath('data', 'example.csv')
# csv_file = project_root / 'data' / 'example.csv' # this notation
# would also work, even though you think the '/' is only unix/
# macOS

# Check if the file exists, this will print 'true' if it exists
print(csv_file.exists())
```

`Path(__file__)` refers to the file in which the code is i.e. `this_script.py`. `.parent` refers to the parent that the `this_script.py` is in, so `my_python_project` which is the project root, the highest folder in the project. `.joinpath` takes the current location in this case `my_python_project` and joins to that the `data` directory and then the `example.csv` so will be `my_python_project/data/example.csv`

You can use the same technique to navigate up through multiple hierarchical directories, 'parents', so given the next scenario where the script and the data are in different subdirectories:

```
my_python_project/  (root)
├── src/
│   ├── this_script.py
├── data/
│   └── example.csv
```

then you can use `pathlib` like this:

```
from pathlib import Path

# This script is located in a subfolder so you need to navigate up to
# the parent (src) and then its parent (project root), then down
# to the 'data' directory and finally the .csv file
csv_file = Path(__file__).parent.parent.joinpath('data', 'example.csv')
csv_file_v2 = Path(__file__).parent.parent / 'data' / 'example.csv' #
# also works

# Check if the file exists
if csv_file.exists():
    print(f"CSV file found: {csv_file}")
else:
    print("CSV file not found.")
```

Using `importlib` resources

An alternative approach that is available from Python v10 onwards is the [method recommended in the setuptools guide](https://setuptools.pypa.io/en/stable/userguide/datafiles.html#accessing-data-files-at-runtime) (<https://setuptools.pypa.io/en/stable/userguide/datafiles.html#accessing-data-files-at-runtime>).

This uses the `importlib.resources` package (<https://docs.python.org/3.11/library/importlib.resources.html#module-importlib.resources>) from Python.

```
from importlib import resources

from activities import data

paralympics_data_file_csv =
    resources.files(data).joinpath("paralympics_raw.csv")
```

ACTIVITY: Add code to reference a .csv file

In the Python module you created earlier add code to locate the data file `paralmpics_raw.csv`.

Try using `pathlib.Path` and the `importlib.resources` approach i.e. do the activity twice using a different approach each time.

Optional, check you can access the file by opening and printing a line e.g.:

```
import csv

if __name__ == '__main__':
    data_file = # Your code that locates the file

    with open(data_file) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        first_row = next(csv_reader)
        print(first_row)
```

Delete the csv reader code once you've demonstrated your file is accessed. You won't be using `csv` to read the files.

Next activity (2-02-pandas-df)

2. Use pandas to open .csv and .xlsx files and create a DataFrame

Read: Data and questions

The data in the activities is from the following source: [Paralympic medals and event info \(https://www.paralympic.org/london-2012/results/medalstandings\)](https://www.paralympic.org/london-2012/results/medalstandings)

The data is saved in the `paralympics_raw.csv` file. Some data quality issues have been introduced such that it can be used for the tutorial activities.

In the activities you will be describing and exploring a dataset, and then preparing it for use in a scenario.

Pandas DataFrame overview

You may have seen spreadsheets that contain data in rows and columns, where the columns represent variables and the rows the values of those variables for a particular instance.

The paralympics data has data in rows with variable names in columns:

type	year	country	host	start	end	duration	disabilities_i	countries	events	sports	participants	participants	participants	highlights	URL
summer	1960	Italy	Rome	18/09/1960	25/09/1960	7	Spinal injury	23	113	8				209 First Games	https://www.paralympic.org/london-2012/results/medalstandings
summer	1964	Japan	Tokyo	08/11/1964	12/11/1964	4	Spinal injury	21	143	9	195	71	266 Weightlifting	https://www.paralympic.org/london-2012/results/medalstandings	
summer	1968	Israel	Tel Aviv	05/11/1968	14/11/1968	9	Spinal injury	29	188	10	578	196	774 Lawn Bowls	https://www.paralympic.org/london-2012/results/medalstandings	
summer	1972	Germany	Heidelberg	02/08/1972	11/08/1972	9	Spinal injury	43	188	10	654	268	922 First quadrag	https://www.paralympic.org/london-2012/results/medalstandings	

0-1. Raw data set table

The data values can be a mix of data types: integers, text, dates, etc.

The pandas library has many functions for analysing, cleaning, exploring and manipulating data. Pandas works with one-dimensional data array using pandas Series, and with two-dimensional data using the pandas DataFrame. Each of these has functions. These activities use Pandas DataFrame since the data is two-dimensional.

You can create a pandas DataFrame in a number of ways - see [pandas documentation \(https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html\)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html). In this course you will typically create it by reading data that is stored in a `.csv` or `.xlsx` file.

ACTIVITY: Read .csv and .xlsx into a DataFrame

There are two data files in the `src/activities/data/` package:

- `paralympics_raw.csv` ([../src/tutorialpkg/data/paralympics_raw.csv](https://www.paralympic.org/london-2012/results/medalstandings)) - a `.csv` file with data about paralympic games
- `paralympics_all_raw.xlsx` ([../src/tutorialpkg/data/paralympics_all_raw.xlsx](https://www.paralympic.org/london-2012/results/medalstandings)) - an Excel file with 2 worksheets, the first with the data about paralympic games, the second with the medal tables of the competing teams

Use a different Pandas function for each type of file:

- `pandas.read_csv` (https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html) for `csv` files
- `pandas.read_excel` (https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html) for `xlsx` files

The linked documentation for each function shows the options that can be applied such as selecting which columns to import, setting data types for columns, etc.

In the ‘main’ section:

1. You already have a variable that locates the file `/activities/data/paralympics_raw.csv`. Add a second variable to locate the `.xlsx` file `activities/data/paralympics_all_raw.xlsx`.
2. Add code to the function to read the contents of the `.csv` file into a pandas DataFrame using `variable_name_for_df = pd.read_csv(your_filepath_variable)`.
3. Add code to read the `.xlsx` file into two further pandas DataFrames. The Excel file has several worksheets, by default `read_excel` will open the first worksheet. You can change this by specifying the worksheet e.g. `pd.read_excel(paralympics_datafile_excel, sheet_name=1)` reads the second worksheet (counts from 0), `pd.read_excel(paralympics_datafile_excel, sheet_name="name_of_sheet")` to read a sheet called ' name_of_sheet'. Add code to read in each of the two sheets.
4. Run the code. It won't return anything if all is well. If you have errors, read the error messages to try and identify what is wrong and correct your code. Some common issues:

```
FileNotFoundError: [Errno 2] No such file or directory: '../src/data/paralympics_raw.csv' - check if the directory structure printed matches your directory structure
```

```
Traceback (most recent call last):
  File ".../src/tutorialpkg/data_prep/analyse_dataframe.py",
line 55, in <module>
    paralympics_excel_df =
pd.read_csv(paralympics_datafile_excel)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
<lines omitted here for brevity>
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xde in
position 16: invalid continuation byte
```

This one is more tricky to detect as the issue is that `read_csv` has been used for a file that is an Excel data file instead of `read_excel`

```

ImportError: Missing optional dependency 'openpyxl'. Use pip or conda to
install openpyxl. - you forgot to install openpyxl which is needed to read an Excel file. Install
it in the virtual environment using pip install openpyxl

```

Next activity (2-03-pandas-describe)

3. Use pandas to describe the contents of a DataFrame

In this activity you will use some of the pandas functions to describe the contents of the DataFrames:

- `DataFrame.shape` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html>) - Returns the number of rows and columns in the DataFrame
- `DataFrame.head` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html>) and `DataFrame.tail` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.tail.html>) - Returns the first / last 5 rows of the dataframe
- `DataFrame.columns` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.columns.html>) - Returns the column labels
- `DataFrame.dtypes` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dtypes.html>) - Returns the data types in the columns of the dataframe. Columns with mixed types are stored with the object dtype.
- `DataFrame.info` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html>) - Prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.
- `DataFrame.describe` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html>) - descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution

Activity: Create a function stub and docstring

Write the code in a way that makes it potentially re-usable in an application. This is expected for the coursework.

1. Import `pandas` which by convention is imported using the syntax `import pandas as pd`.
2. Create the structure for a Python function in the module. This function will take a DataFrame as its argument. It will print information that describes the data in the DataFrame.
3. Add a docstring to the function. The following is a general format for a multi-line docstring; specific styles include [Google](https://github.com/google/styleguide/blob/gh-pages/pyguide.md#38-comments-and-docstrings) (<https://github.com/google/styleguide/blob/gh-pages/pyguide.md#38-comments-and-docstrings>), reStructuredText (reST), Numpydoc, Epytext, and more. If you are using Co-pilot inline chat try `/doc format as Google-style docstring` or similar. Docstring example:

```
def some_function(argument1):
    """Summary or description of the function

    Parameters:
    argument1 (int): Description of argument1

    Returns:
    int: Description of the returning value

    """
```

NB This is not the only way to structure the code. For example, you could define the function differently, or use a class.

Activity: add pandas functions to describe the dataset

1. Add to the function code that does the following:
 - Print the shape e.g. `print(name_of_your_dataframe.shape)` to find the number of rows and columns
 - Print the first 5 rows and the last 5 rows (you may need to change the Pandas settings to see all columns by using `pd.set_option("display.max_columns", None)`)
 - Print the column labels
 - Print the column datatypes
 - Print the info
 - Print the results of `describe`
2. Add code to your 'main' to call it for each of the three dataframes. It may look something like this, though you may have different names for your variables and functions:

```
if __name__ == "__main__":
    # Filepath of the csv data file (you may have used
    # importlib.resources rather than pathlib.Path)
    paralympics_csv = Path(__file__).parent.parent.joinpath("data",
        "paralympics_raw.csv")

    # Read the data from the file into a Pandas dataframe
    events_csv_df = pd.read_csv(paralympics_csv)

    # Call the function named 'describe_dataframe' - you may have
    # a different name for your function
    describe_dataframe(events_csv_df)
```

3. Run your Python code and look at the output to check you understand the results. The resulting printed information for the two dataframes with the 'events' data should be the same, the 'medal_standings' dataframe info will be different.

[Next activity \(2-04-missing-values-identify.md\)](#)

4. Identify missing values

Missing values or missing data are those values that are not present in the data set.

There are many reasons for values to be missing, for example because they don't exist, or because of improper data collection or improper data entry.

Missing values in data can reduce the accuracy of any resulting data analysis or machine learning model.

In pandas, missing values are represented by None or NaN (Not a Number). Note that empty strings (' ') are not considered NA values.

In data science, missing values are typically categorised as:

- Missing Completely At Random (MCAR). There is no relationship between the missing data and any other values in the dataset. There is no pattern for the missing values. The advantage of such data is that the statistical analysis remains unbiased.
- Missing At Random (MAR). The missing values are related to variables that are complete, i.e. there is a relationship between the missing data and other values. The missing data may be estimated using the related values.
- Missing Not At Random (MNAR). There appears to be a pattern to the missing data; however this pattern does not depend on other variables that are in the dataset but some unknown variable. This can bias the results of any analysis.

This article (<https://www.analyticsvidhya.com/blog/2021/10/handling-missing-value/>) summarises these succinctly.

Activity: Find the missing values

You will need to refer to the pandas documentation:

- `dropna()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html>)
- `fillna()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html>)
- `isnull()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isnull.html>)
- `isna()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isna.html>)

You could add the following to the describe function, or you could create a new function e.g. that explores the data, or that checks data quality.

1. Add code that prints the number of missing values in the merged DataFrame using `isna()` or `isnull()`. 'True' indicates a missing value. Run the code to see the results.
2. Step 1 shows all the data which may not be practical in a large dataset. Instead, create a copy of the dataframe with only the rows that contain any missing values. Example of the syntax to create a DataFrame with missing rows: `missing_rows = original_df[original_df.isna().any(axis=1)]` or columns: `missing_columns = original_df[original_df.isna().any(axis=0)]`.

[Next activity \(2-05-plot-overview.md\)](#)

5. Introduction to charts with pandas.DataFrame.plot()

The aim during the data exploration phase is to get a better understanding of the data to decide if what you have is largely free from data quality issues and that it suits the needs of your project.

These are charts for your own use, not for others, so you don't need to focus on visual aesthetics.

Pandas plot

Pandas has a builtin plotting method, `.plot()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html>) that can be applied to a DataFrame to generate visualisations. These visualisations are also referred to as “charts”, “graphs” or “plots”.

By default, pandas `.plot()` uses `matplotlib` as the engine that renders (draws) the plot. `Matplotlib` (<https://matplotlib.org>) is a Python visualisation library that has more functions than are used by pandas. You won't be learning matplotlib directly in this tutorial, only the `pandas.plot()` interface.

As pandas is built on matplotlib then you will see common terminology used.

- Data (dataframe): The data to be used in the plot.
- Figure: The overall window that the visualisation is drawn on. This can contain multiple plots or ‘axes’.
- Axes: The individual plots within a Figure. Each can have its own labels, titles, ticks, etc. Where there are several plots in one figure, these plots are created as ‘subplots’.

The typical steps to create a pandas plot are:

- Define the data to be plotted in a dataframe
- Create a figure and axes
- Plot the data on the axes
- Optionally, customise the plot e.g. styling, titles, ticks, labels etc.
- Show the plot

For example:

```

import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame
df = pd.DataFrame({
    'A': [1, 2, 3, 4],
    'B': [4, 3, 2, 1]
})

# Using pandas.plot directly creates the figure, axes and allows for
# some customisation
# matplotlib examples typically split this into separate commands
# defining fig and ax then adding customisation
ax = df.plot(title='Sample Plot', xlabel='X-axis Label',
             ylabel='Y-axis Label')

# Save plot (may prefer this to showing the plot)
plt.savefig('sample_plot.png')

# Show the plot
plt.show()

```

You can run this example in `starter/example_plot.py` in your own IDE.

In VS Code the chart usually displays in a separate pop-out window. You need to close this window before running the code again.

In PyCharm it displays in the Plots pane within the interface.

Note that once the chart is displayed, the code stops executing so you may need to close the chart window before running further code. You may prefer to save the chart as an image file instead.

Styling

During the data exploration phase you are not creating a polished chart intended for a particular audience.

You don't need to spend time formatting these charts, however some formatting may be needed beyond the examples included in the activities.

The [pandas visualisation documentation \(https://pandas.pydata.org/docs/user_guide/visualization.html#plot-formatting\)](https://pandas.pydata.org/docs/user_guide/visualization.html#plot-formatting) gives you plot formatting guidance and examples. This includes:

- [x and y labels \(https://pandas.pydata.org/docs/user_guide/visualization.html#controlling-the-labels\)](https://pandas.pydata.org/docs/user_guide/visualization.html#controlling-the-labels)
- [displaying the legend \(https://pandas.pydata.org/docs/user_guide/visualization.html#controlling-the-legend\)](https://pandas.pydata.org/docs/user_guide/visualization.html#controlling-the-legend)
- [subplots \(https://pandas.pydata.org/docs/user_guide/visualization.html#plot-formatting\)](https://pandas.pydata.org/docs/user_guide/visualization.html#plot-formatting)
- [custom formatters for timeseries plots \(https://pandas.pydata.org/docs/user_guide/visualization.html#custom-formatters-for-timeseries-plots\)](https://pandas.pydata.org/docs/user_guide/visualization.html#custom-formatters-for-timeseries-plots)

[Next activity \(2-06-plot-distribution\)](#)

6. Explore distributions of values

It can be useful to understand the range of values, and the distribution of those values, for the numerical columns in your data.

Histograms

Histograms are ‘designed to show a dataset’s distribution or spread’ [Reference: Depict, Chart chooser \(https://depictdatastudio.com/charts/histograms/\)](https://depictdatastudio.com/charts/histograms/).

Create `histograms` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.hist.html#pandas.DataFrame.plot.hist>) to show the distribution of the ‘participants_m’ and ‘participants_f’ columns.

The histogram function has the following options `DataFrame.plot.hist(by=None, bins=10, **kwargs)`.

Activity: Create histograms

You can either add the code to one of your existing functions, or create a new one.

1. Add import to the top of the module: `import matplotlib.pyplot as plt`.
2. Add code to a function to create a histogram from the data:

```
# Create a histogram of the DataFrame
df.hist()

# Show the plot
plt.show()
```

3. Run the code to generate the figure. It should generate a figure that has histograms for all the columns in the data.
4. Modify the code to use only specified columns `columns = ["participants_m", "participants_f"]` of the DataFrame e.g. `df[columns].hist()`
5. Run the code again to generate the figure.

The distributions don’t really tell you much in this dataset. It may be more useful for larger datasets.

Boxplot

Identifying outliers

An outlier is a data point that significantly differs from the other observations in a dataset. It lies outside the overall pattern of distribution and can be unusually high or low compared to the rest of the data. Outliers can result from variability in the data, measurement errors, or experimental errors.

You can identify outliers using different techniques:

- Plot the data (e.g., histogram, scatter plot, boxplot)
- Use common sense
- Use statistical tests

In statistical terms, outliers are often defined as values that fall below the first quartile (Q1) minus 1.5 times the interquartile range (IQR) or above the third quartile (Q3) plus 1.5 times the IQR. Mathematically, this can be expressed as:

$$\text{Lower Bound} = Q1 - 1.5 \times \text{IQR}$$

$$\text{Upper Bound} = Q3 + 1.5 \times \text{IQR}$$

Where:

- (Q1) is the first quartile (25th percentile)
- (Q3) is the third quartile (75th percentile)
- (IQR) is the interquartile range, calculated as (Q3 - Q1)

Values outside these bounds are considered outliers.

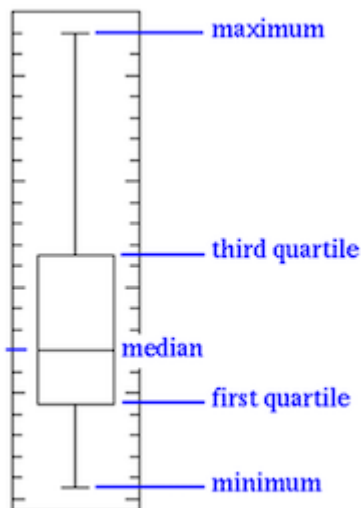
Since this course doesn't expect any knowledge of, nor teach, statistics then we will check for outliers by creating a chart or 'plot'.

In this activity, you will create pandas boxplots (<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.boxplot.html>).

"A Box and Whisker Plot (or Box Plot) is a convenient way of visually displaying the data distribution through their quartiles. The lines extending parallel from the boxes are known as the 'whiskers', which are used to indicate variability outside the upper and lower quartiles. Outliers are sometimes plotted as individual dots that are in-line with whiskers. Box plots can be drawn either vertically or horizontally."

Source: Data Visualisation Catalogue (https://datavizcatalogue.com/methods/box_plot.html)

Box plots have a box from lower quartile to the upper quartile, with the median marked. 25% of the population is below first quartile, 75% of the population is below third quartile.



0-1. boxplot

Source: statinfer (<https://statinfer.com/104-3-5-box-plots-and-outlier-detection-using-python/>)

Box plots can help to get an idea of the data distribution which in turn helps us to identify the outliers more easily.

If the box is pushed to one side, and some values are far away from the box, then it is an indication of outliers.

The following code shows a basic example of creating a boxplot. Numpy is only used in this example to generate the random numbers.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.DataFrame(np.random.rand(10, 5), columns=["A", "B", "C", "D",
        "E"])
df.boxplot()
df.plot.box() # This syntax is also valid
plt.show()
```

Activity: Create box plots

1. Add code to generate a box plot from the paralympics csv dataframe.

Refer to the [pandas boxplot docs \(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.boxplot.html\)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.boxplot.html).

2. Run the code and view the plots. Notice that it plots all the variables on the same scale.
3. Modify the code to see the sports variable in its own plot; or try to create subplots for each variable.

There are no outliers, however if you repeat this step again later after creating a new column for duration, you may spot an outlier.

[Next activity \(2-07-plot-timeseries.md\)](#)

7. Timeseries data

The previous activity considered the use of boxplots and histograms to visualise the distribution of the data.

This example uses a line chart. Line charts are useful for showing “patterns over time”. Source: [Depict](https://depictdatastudio.com/charts/line/) (<https://depictdatastudio.com/charts/line/>). See also [The Data Visualisation Catalogue](https://datavizcatalogue.com/methods/line_graph.html) (https://datavizcatalogue.com/methods/line_graph.html).

In a line plot, the x-axis usually represents the time, and the y-axis represents the variable(s) being measured.

The next charts consider the number of participants over time.

1. Create a new function to plot timeseries. Add code to create and show (or save) a plot (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html>) where `x="start"` and `y="participants"`.

Winter events have far fewer participants than summer, hence the effect you see in the chart.

If you wish to challenge yourself, display the charts showing the split of male and female participants.

You should be able to see from the charts that there appears to be an unusual dip in Winter 1994 Paralympics which may need further investigation.

[Next activity \(2-08-categorical-data.md\)](#)

8. Identifying values in categorical data

Categorical variables are those where the data can be divided into categories, or groups. For example: race, gender, age group, and educational level.

Categorical variables can be problematic for data analysis.

One issue is that many machine learning algorithms cannot handle categorical data. To overcome this, categorical values can be replaced with encoded data. For example: Hot = 1, Warm = 2, Cold = 3. Techniques for addressing this is not covered in this course, however there are Python packages (or functions within packages) that will handle this process of encoding; a popular option is OneHotEncoder (<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwihozWxe2BAxWaUkEAHegCDEYQFnoBlearn.org%2Fstable%2Fmodules%2Fgenerated%2Fsklearn.preprocessing.OneHotEncoder.html&usg=AOvVaw0oQAupueEb>

A second issue is that depending on how the data was collected, categorical data may be entered inconsistently, e.g. in an address where the values “UK”, “Great Britain”, and “United Kingdom” all relate to the same country. However, as they are spelt differently any functions to count, group etc. would not recognise the similarity.

The ‘type’ and ‘disabilities_included’ columns in the event data appear to have categorical data. You could also explore the npc codes and medal standings data.

1. Write code to print unique values for the ‘type’ column. A single column is a Series so you can use `df['ColName'].unique()` (<https://pandas.pydata.org/docs/reference/api/pandas.Series.unique.html>) or `df['ColName'].value_counts()` (https://pandas.pydata.org/docs/reference/api/pandas.Series.value_counts.html).

If you run the code you should see output something like this:

```
Distinct categorical values in the event 'type' column
['Summer' 'summer' 'winter ' 'winter']

Count of each distinct categorical value in the event 'type'
column
  type
summer      17
winter      13
Summer       1
winter       1
Name: count, dtype: int64
```

You can see there are two errors, winter has an extra space in on one row, and in another summer has a capital.

2. Repeat for the ‘disabilities_included’ column.

The results here show a different issue; multiple values are included in a single cell. This will be addressed in a later week in database design.

[Next activity \(2-09-data-prep.md\)](#)

9. Data preparation

The following activities move on from describing the dataframe and exploring its contents, to changing its contents to suit our purposes. These activities are often referred to as ‘data preparation’ or ‘data cleaning’.

In this activity you will prepare the data so that it can be used for a particular purpose.

You are asked to prepare the data for use in a dashboard web application for high school students (the ‘target audience’). The questions the students are interested in are:

- where in the world have paralympic events have been held?
- when have the events been held? (dates)
- how have the number of sports and events included changed over time?
- what are the trends in participant numbers over time? How does this vary by gender? How does this vary by winter and summer events?

In the activities you are guided to create one dataframe that can be used for any of these questions for simplicity; you could choose to prepare the data separately for each question.

In doing so you will also address the issues that were identified during the exploration activities:

- remove columns that are not needed
- resolve issues with missing values
- change data types
- correct categorical data
- add new data: new columns, merge datasets

The following activities focus on the paralympic games data and the country codes, rather than the medal standings. You can use the medal standings data for extra practice.

Activity: Create a new function

The function should take a DataFrame of the raw paralympics event data, perform actions to prepare it for your needs, and then return a dataframe with the prepared data and/or save the data to a .csv or .xlsx file.

You can choose whether the function just deals with all the changes for the data preparation, or whether to have separate smaller functions for each.

Create the outline of the function. You can create it in a new module, or use your existing one.

[Next activity \(2-11-removing-columns.md\)](#)

10. Referencing specific columns and/or rows in a DataFrame

Many of the following activities work with specific columns and rows.

There are many ways to locate columns and rows in pandas. The following lists some of the more common you may see in tutorials and examples online, there are other methods (mask, filter, ...).

- bracket notation to reference a column: `name_of_dataframe['col_name']`
- dot notation to reference a column: `name_of_dataframe.col_name` (works where there are no spaces or special characters in the column name)
- `loc` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html>) to identify a group of rows and columns using integer positions or column labels:
 - `name_of_dataframe.loc[7:9]` - row with index 7 and column with index
 - `name_of_dataframe[:, 'col_a']` - all rows from the column named 'col_a'
 - `name_of_dataframe[:, ['col_a', 'col_b']]` - all rows from columns named 'col_a' and 'col_b'
- `iloc` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html#pandas-dataframe-iloc>) to locate by integers only e.g.
 - `df.iloc[3]` row at index 3
 - `df.iloc[0, 1]` row 0, value from the second column (starts from 0)
 - `df.iloc[1:3, 0:3]` rows 1 and 2, and the first 3 columns
- `iat` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iat.html#pandas-dataframe-iat>) to Access a single value for a row/column pair by integer position e.g. `df.iat[1, 2]`
- `at` (`()`) get or set a value for a specific pair by integer or column label e.g. get `df.at[4, 'B']` or set `df.at[4, 'B'] = 10`
- `query` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.query.html>) used to query columns with an expression e.g. `df_summer = df.query("type == 'summer'")`. You can use expressions such as `==`, `!=`, `<`, `>`, `in`. You can use values of a variable e.g.
`python event_type = "winter" df_winter = df.query("type == @event_type")`

You will need some these approaches in the following activities. Try to use different techniques so you explore more features of pandas.

Here are some examples of how you could use the techniques above to find and then change a value:

```

# Example 1: Find a row/column that matches a certain condition using
            loc with a query or mask
df.loc[df.query("type == 'Summer'").index, 'type'] = 'summer' # query
df.loc[df['type'] == 'Summer', 'type'] = 'summer' # mask

# Example 2: Find the index of the row using `query`, and then use `at`
            to update the value.
# NB Assumes only 1 row matches the criteria, amend to loop through all
            matching indices if more than one result.
index = df.query("type == 'Summer'").index[0]
df.at[index, 'type'] = 'summer'

# Example 3: Uses iloc which only works with integers so you need to
            find the row & column integer references first
row_pos = df.query("type == 'Summer'").index[0]
row_idx = df.index.get_loc(row_pos)
col_idx = df.columns.get_loc('type')
df.iloc[row_idx, col_idx] = 'summer'

```

[Next activity \(2-11-removing-columns.md\)](#)

11. Removing columns

In the function to describe the events dataframe you printed all the column names:

```
['type', 'year', 'country', 'host', 'start', 'end',  
'disabilities_included', 'countries', 'events',  
 'sports', 'participants_m', 'participants_f', 'participants',  
'highlights', 'URL']
```

You don't need the 'URL', 'disabilities_included' and 'highlights' columns in the scenario given.

Options to remove columns:

1. Exclude the columns from the data file when the dataframe is created by specifying the columns to read
2. Removed the columns after the dataframe is created

Create the database with specified columns

Use the `usecols` parameter when you read the data. For example:

```
import pandas as pd  
from pathlib import Path  
from importlib import resources  
  
from activities import data  
  
if __name__ == '__main__':  
    path_para_raw =  
        resources.files(data).joinpath("paralympics_raw.csv")  
  
    cols = ['type', 'year', 'country', 'host', 'start', 'end',  
            'countries', 'events', 'sports',  
            'participants_m', 'participants_f', 'participants']  
  
    df_selected_cols = pd.read_csv(path_para_raw, usecols=cols)
```

Activity: Remove the columns after creating the dataframe

As you already have the columns in the dataframe, use this approach to remove them.

Use the `pandas.DataFrame.drop()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html>) to remove either rows or columns. To remove rows specify `axis=0`; to remove columns you can use `axis=1` or the `columns=` parameter.

`inplace=True` Most methods in pandas return a new object and leave the original untouched. To alter the DataFrame in-place you can sometimes use the attribute `inplace=True`; though most examples suggest allocating the result of an action to a new dataframe object as `inplace=True` can have unwanted implications in some circumstances - [see here \(https://docs.astral.sh/ruff/rules/pandas-use-of-inplace-argument/\)](https://docs.astral.sh/ruff/rules/pandas-use-of-inplace-argument/).

An example of allocating the result of an action to a new dataframe: `df_result = df.drop(columns=['MyCol2', 'MyCol4'])`

An example using `inplace=True` where the result will be accessed in the original `df`: `df.drop(columns=['MyCol2', 'MyCol4'], inplace=True)`

1. Edit the function to drop the following list of named columns ['URL', 'disabilities_included', 'highlights'] and assign the result to a new DataFrame variable, e.g. `df_prepared`.
2. You might also want to add a print of the column names again so you can check the columns were removed. Run the code.

[Next activity \(2-12-resolve-missing-incorrect-values.md\)](#)

12. Resolve missing or incorrect values

This activity addresses some of the data quality issues identified in the exploration stage:

- Missing values
- Mistakes in categorical values

Missing values

There are different ways to deal with missing values. The decision you make will depend on the type of data, the actions you want to perform on the data, and the reasons for the missing values.

Options include:

- **Do nothing.** It may be valid, e.g. the data does not yet exist so its absence is meaningful. It may not have significant impact on your analysis to ignore the missing values.
- **Delete.** If your data set is large enough you might choose to delete missing data without significant impact to the resulting analysis. Likewise, if the percentage of missing values is high, you may prefer to delete those rows or columns.
- **Fill (imputation).** Use an imputation technique to fill the missing values e.g. mean, median, most frequent etc.

Imputation techniques are not covered in this course. This article on [Towards Data Science \(https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779\)](https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779) gives pros and cons of some imputation techniques.

Activity: Drop missing values

You identified the rows and columns with missing values in activity 4. You may want to repeat this now that you have removed columns from the data.

There are missing values in the following rows/columns:

	type	year	country	host	start	end
	countries	events	sports	participants_m	participants_f	
	participants					
0	Summer	1960	Italy	Rome	18/09/1960	25/09/1960
23.0	113.0	8		NaN	NaN	209.0
17	summer	2028	USA	Los Angeles	15/08/2028	27/08/2028
NaN	NaN	22		NaN	NaN	NaN
31	winter	2026	Italy	Milano Cortina	06/03/2026	15/03/2026
NaN	79.0	6		NaN	NaN	NaN

Some pandas functions will function with **NaN** values; and some allow you to specify to ignore **NaN** using the parameter `skipna=True`. Some functions, such as converting column data types, will fail if **NaN** are present, for these you need to fill or remove **NaN**.

Write code to deal with the following:

1. The `participants_m` and `participants_f` data for Rome in 1960 is not available. Imputing the data would be inaccurate so remove row with index 0. To remove a row the syntax: ``your_df_name = your_df_name.drop(index=0)`.

2. Numerous columns are missing from 17 and 31 as these are future events so the values are not known. Again, remove these rows as they aren't needed for the questions in this scenario.
3. Print the first 3 rows of the dataframe. You will see that row 0 is missing - pandas does not reset the index again from 0 unless you specify it should. e.g. `df = df.reset_index(drop=True)`

Activity: Replace values

In activity 8 you found the following issues in the 'type' column:

```
Distinct categorical values in the event 'type' column  
['Summer' 'summer' 'winter ' 'winter']
```

Change 'Summer' to 'summer' and remove the trailing whitespace from 'winter '.

1. Add code to locate the row where `type = "Summer"` using one of the approaches in activity 10. Once located, change it to lower case using `str.lower()` (<https://pandas.pydata.org/docs/reference/api/pandas.Series.str.lower.html#pandas.Series.str.lower>) or set the value to 'summer'.
2. Add code to remove the whitespace from the 'winter' occurrence by stripping whitespace either just from that cell, or from all values in the type column. Use `str.strip()` (<https://pandas.pydata.org/pandas-docs/version/0.24/reference/api/pandas.Series.str.strip.html>).

[Next activity \(2-13-change-datatypes.md\)](#)

13. Changing datatypes

The datatypes in the `events_csv_df` dataframe were output in activity 3; some columns have been removed leaving:

```
type          object
year          int64
country       object
host          object
start         object
end           object
countries     float64
events        float64
sports        int64
participants_m float64
participants_f float64
participants  float64
```

Actions to be taken in this activity:

- Change 'float64' so the values display as '21' rather than '21.0'.
- The pandas reference states: "Columns with mixed types are stored with the object dtype". In this dataframe these are all strings, however the start and end should be dates.

The following summarises the data types needed in this activity. Note the differences for the integer types. See [pandas reference \(https://pandas.pydata.org/docs/reference/arrays.html\)](https://pandas.pydata.org/docs/reference/arrays.html) for other data types.

<i>Data Type</i>	<i>Description</i>	<i>Typical Use Case</i>
float64	Floating-point numbers	Precise values, decimals
int64	Large integers	Whole numbers. 64-bit.
Int64	Large integers	Whole numbers. 64.bit. Pandas-specific data-type that allows for missing numbers 'NaN'
int	General integer type	Usually same as int64. Size can vary.
DatetimeTZDtype	Timezone-aware datetime data	For timezone aware dates/times.

A column's datatype can be specified when the data is read into a dataframe, e.g.

```
import pandas as pd

# Define the data types for the columns
dtype_dict = {
    'column1': 'int',
    'column2': 'float64',
    'column3': 'int',
    # Add more columns as needed
}

# Read the CSV file with specified dtypes
df = pd.read_csv('your_file.csv', dtype=dtype_dict)
```

The datatype of a column can also be changed after the DataFrame is created, e.g.:

```
# Convert a specific column from float64 to int
df['column_name'] = df['column_name'].astype('int')
```

Activity: Change float64 to an integer type

1. Add code to your function to change all the float datatypes to integer (choose either int, int64, or Int64). The columns that need to be changed are `columns_to_change= ['countries', 'events', 'participants_m', 'participants_f', 'participants']`

Did you get an error? e.g. `IntCastingNaNError(pandas.errors.IntCastingNaNError: Cannot convert non-finite values (NA or inf) to integer.`

Pandas cannot convert nulls, or NAs, to int. You will either need to remove, convert them, or convert to a type that handles NaN (“Int64”) e.g.

```
df['col1'].fillna(0).astype(int) # Replaces the missing values with 0
df['events'] = df['events'].astype('Int64') # Handles NaN without
replacing values
```

Activity: Change object to date

The ‘start’ and ‘end’ columns contain dates as strings.

The pandas function `pd.to_datetime()` (https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html) can be used to change strings to a datetime format.

See [Python datetime for the format codes](https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes) (<https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>).

Use `pytz` (<https://pypi.org/project/pytz/>) if you want to list all valid timezones.

1. Print the values from the ‘start’ and ‘end’ columns. You could use ‘`unique()`’ as in activity 8 or print the two columns.

Are there any missing values? There shouldn't be.

2. Add code to change the date format `format='%d/%m/%Y'` using `pd.to_datetime()` e.g.
`df['start'] = pd.to_datetime(df['start'], format='%d/%m/%Y')`. Optionally you can also make the dates timezone aware e.g. `df['start'] = df['start'].dt.tz_localize('Europe/London')`
3. Print the datatype of all columns again to see the difference after your changes. There are still some object data types. Try changing these to strings.

[Go to next activity \(2-14-new-column.md\)](#)

14. Adding new columns

The next two activities add data to the existing data:

- Add a new column by computing a value
- Add a new column by merging data from another dataframe

Adding new data may introduce new issues with NaN, incorrect values, incorrect formats etc. so you would need to repeat many of the previous steps! The new data could have been added before we did the rest of the preparation.

Activity: Add computed values

One way to add new data is to compute new values from the existing data.

Add a new column called `duration` with its values calculated by subtracting the start date from the end date. This should work as you converted the strings to date data types in the last activity.

This uses the `DataFrame.insert()` function (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.insert.html#pandas.DataFrame.insert>).

1. The code to calculate the new column could be written as: `df['duration'] = df['end'] - df['start']`. This would append the column to be the last in the DataFrame with a date datatype expressed in days e.g. '4 days'
2. Instead, add code first calculate the values as an integer format: `duration_values = (df['end'] - df['start']).dt.days.astype('Int64')`
3. Then insert the values as a new column the 'end' column using the `DataFrame.insert()` function e.g. `df.insert(df.columns.get_loc('end') + 1, 'duration', duration_values)`
 - `df.columns.get_loc('end')`: Finds the index of the 'end' column and adds 1 to set the position of duration after the end column.
 - `'duration'`: The name of the new column.
 - `duration_values`: The values for the new column.
4. Run the code. You could `print(df['duration'])` to check the result.

[Next activity \(2-15-joining-dataframes.md\)](#)

15. Joining dataframes

In the `medal_standings` DataFrame there is a field called 'NPC' which contains the three-letter codes used by the National Paralympic Committee (NPC) to describe a team. The teams are usually countries such as "China", or a construct, such as "Neutral Para Athletes".

A list of all the current and historic NPC codes is the data file `activities/data/npc_codes.csv` (the Excel version has the same data).

You don't really need this to answer the questions the scenario; though it may be useful in later analysis to indicate the NPC code for each of the paralympic events. This activity looks at how you can join data from one dataframe to another.

To join the data from two dataframes, there needs to be a column that is common to both. In this case:

- `npc_codes.csv` : "Name"
- `paralympics_events` : "country"

"Name" and "country" both appear to contain the country name.

There are two Pandas functions for combining data:

- To combine two dataframes with a common index or key column (i.e. a column with unique values only) you can use `pandas.DataFrame.join()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html?highlight=join#pandas.DataFrame.join>)
- To combine two dataframes where the common column is not an index or key then you use `pandas.DataFrame.merge()` (<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html?highlight=merge#pandas.DataFrame.merge>)

In this case 'merge' is needed as the values in country are not unique in the events csv.

There are different approaches to combine data. These are:

- **left**: use calling frame's index (or column if on is specified)
- **right**: use other's index.
- **outer**: form a union of calling frame's index (or column if on is specified) with other's index, and sort it lexicographically.
- **inner**: form intersection of calling frame's index (or column if on is specified) with other's index, preserving the order of the calling's one.
- **cross**: creates the cartesian product from both frames, preserves the order of the left keys.

Start with the 'events' dataframe which is the one you want to add the 'NPC' column to. This has ~30 rows. This is the **left** dataframe.

The **right** dataframe is the `npc_codes`. This has ~130 rows.

Using the different methods to combine the data:

- **left**: the resulting dataframe would add the NPC code to the 'events'. The result will be ~30 rows.
- **right**: the resulting DataFrame will have ~130 rows (same as the 'npc_codes' DataFrame). This is because a right join includes all rows from the right DataFrame and matches rows from the left DataFrame based on the join key. If there are no matching rows in the left DataFrame, the result will contain NaN for those columns.
- **outer**: the resulting DataFrame will include all unique rows from both DataFrames. This means it forms the union of the columns ('Name', 'country') from both DataFrames. The exact number of rows in the resulting DataFrame depends on the overlap between the columns of the two DataFrames.

- inner: the resulting DataFrame will only include rows that have matching columns in both DataFrames. The number of rows in the resulting DataFrame depends on the overlap between the columns of the two DataFrames. If there is no overlap, the resulting DataFrame will have 0 rows. If there is complete overlap, the resulting DataFrame will have the same number of rows as the smaller DataFrame.
- cross: if the left DataFrame has 32 rows and the right DataFrame has 132 rows, the resulting DataFrame will have: $\sim 30 * 130 = 4000+$ rows.

Use the ‘left’ method for this activity.

The available parameters in the merge function are: `DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=None, indicator=False, validate=None)`

You need to specify the following:

```
left_dataframe_name.merge(right_dataframe name, how='left',
left_on='col_name_in_left_df', right_on='col_name_in_right_df')
```

1. Open the `npc_codes.csv` into a DataFrame, reading in just the ‘Name’ and the ‘Code’ columns as we only need these. The parameter `usecols=['Code', 'Name']` for the `pd.read_csv` function is useful here.

Note: there is a problematic character somewhere in the `npc_codes.csv` file. You can read the file using the additional parameters from this example: `df = pd.read_csv(path_to_npc_csv_file, encoding='utf-8', encoding_errors='ignore', usecols=['col1', 'col2'])`

2. Add the code to ‘merge’ the two dataframes and create a new merged dataframe with the result.
3. Print the ‘country’, ‘Code’, ‘Name’ columns from the merged dataframe.

This is just the rows with NaNs:

	country	Code	Name
6	UK	NaN	NaN
7	Korea	NaN	NaN
9	USA	NaN	NaN
12	China	NaN	NaN
13	UK	NaN	NaN
17	USA	NaN	NaN
25	USA	NaN	NaN
28	Russia	NaN	NaN
29	Korea	NaN	NaN
30	China	NaN	NaN

There are numerous ‘NaN’s. Why?

To match the values in the columns ‘country’ and ‘Name’ needed to be identical. Unfortunately, countries often have different ways being written in English, so for example UK in the events data, and United Kingdom in the NPC data.

Dealing with NaNs and Nulls is covered in activity 12.

4. The values that are NaN are the country codes for UK, USA, Korea, Russia and China. In the NPC file these are:

```
replacement_names = {
    'UK': 'Great Britain',
    'USA': 'United States of America',
    'Korea': 'Republic of Korea',
    'Russia': 'Russian Federation',
    'China': "People's Republic of China"
}
```

Add code *before* the code to merge the two dataframes. The code should change the values in the ['country'] column to those in the dict given above.

Use the `Series.replace(to_replace=replacement_names)` function (<https://pandas.pydata.org/docs/reference/api/pandas.Series.replace.html#pandas.Series.replace>).

5. Check for NaNs again too check all have been addressed.
6. Remove the 'Name' column as it is a duplicate of the 'country' column
7. Optionally print the dataframe contents to check the results are what you expected.

Next activity (2-16-save-df-to-file.md)

16. Save the prepared dataset

You have now carried out the data cleaning and preparation steps needed for this project so save the prepared dataframe back to a .csv file.

You could also save the data to a database or other format. See [pandas output methods \(https://pandas.pydata.org/docs/reference/io.html#\)](https://pandas.pydata.org/docs/reference/io.html#).

1. Add code add the end of the data preparation function to save the output to file in the 'data' directory before you return the dataframe. You could remove the return dataframe from the end of the function if you prefer. Use the pandas '`to_csv()`' (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html).

Note that the dataframe has an index column with row numbers that you don't need in the csv file. Use the argument `index=False` when you use `to_csv()`

2. Finally, add code to return the dataframe from the function so it can be used by other functions e.g. `return df`
3. Run the code and check that the file is saved.

[Next activity \(2-17-questions.md\)](#)

17. Using the prepared data

The purpose of this activity is to see whether it looks like the prepared data would suit the purposes.

Recap of the purpose of the data

You were asked to prepare the data for use in a dashboard web application for high school students (the ‘target audience’).

The questions the students are interested in are:

- where in the world have paralympic events have been held?
- when have the events been held? (dates)
- how have the number of sports and events included changed over time?
- what are the trends in participant numbers over time? How does this vary by gender? How does this vary by winter and summer events?

This activity is not concerned with answering the questions and interpreting the data; only in confirming if the data preparation is sufficient. Charts are created a visual means of judging if the prepared data appears to be sufficient for use in the intended scenario. You may identify other ways than charts to validate this.

You can create another function for this, or add the code to main.

Where in the world have paralympic events have been held?

The data set contains city locations and the country of that city.

1. Add code to print the unique host-country pairs in which the games have been hosted. You could just print the two columns, or try `pairs = df[['host', 'country']]` and apply the `drop_duplicates()` and `sort_values(by='country')` functions. You can chain these functions together.

Is the prepared data sufficient? This is subjective. The NPC data also had region which might also be added to allow someone to answer. If you wanted to map the locations you would need to find the latitude and longitude of the host cities as well.

When have the events been held?

You could do this by printing the values, or by displaying on a timeline. The following guides you through creating a timeline.

How have the number of sports and events included changed over time?

Over to you, have a go at demonstrating you have the data to allow someone to answer the remaining questions:

- What are the trends in participant numbers over time?
 - How does this vary by gender?
 - How does this vary by winter and summer events?

[Next activity \(2-18-next-steps.md\)](#)

18. Next steps

1. **Further practice using this repo:** Apply the principles to the 'medal_standings' dataframe, e.g.:
 - check for missing values
 - check for issues with categorical data
 - remove columns
 - check and change datatypes
2. **Other tutorials:** Data preparation with pandas is a popular topic, a search shows there are many free tutorials. A few:
 - <https://realpython.com/pandas-plot-python/>
 - <https://www.freecodecamp.org/news/data-cleaning-and-preprocessing-with-pandasbdvhj/>
 - <https://www.w3schools.com/python/pandas/default.asp>
 - https://pandas.pydata.org/docs/user_guide/10min.html
 - <https://www.kaggle.com/learn/pandas>
3. **AI guidance:** Use Copilot or chatGPT and ask for advice on using particular pandas functions. Try the pandasai library.
4. **Find example from others, e.g. Kaggle:** There are lots of data sets and often examples of data preparation others have carried out. For example look at the 120 year of Olympic history (<https://www.kaggle.com/datasets/heesoo37/120-years-of-olympic-history-athletes-and-results>). This is a larger dataset than the one used in this tutorial and there are many code examples from others that use this dataset (some better than others!).
5. **Finally, apply the knowledge to your coursework project!**

As a third year module, the coursework is not meant to be a series of instructions to follow. However, to get you started this week here are some suggestions of what to do:

- Install pandas and openpyxl in the virtual environment of your coursework project in your IDE.
- Use pandas to read the data into a dataframe.
- Write code to describe the dataframe contents e.g. size, column names, datatypes, statistics.
- Write code to explore the data in more detail e.g. consider missing values, incorrect data types, unique values in columns, plot ranges of values in columns, and investigate outliers in columns.