***CODE EXPLANATION***

The code is split into a few main parts, like different sections of a cookbook:

**1. parse\_excel\_to\_table(file\_path): The "Reading and Understanding" Recipe**

This is the heart of the code where all the magic happens.

* **wb = openpyxl.load\_workbook('/content/Awesome port.xlsx', data\_only=True)**: This line is like telling the code, "Okay, go find the Excel file named 'Awesome port.xlsx' and open it up!" The data\_only=True part is important; it tells openpyxl to only grab the *values* you see in the cells, not any formulas that might be hiding behind them.
* **ws = wb.active**: Most Excel files have multiple sheets (like tabs in a browser). This line says, "Let's focus on the first (active) sheet in this Excel file."
* **report\_date = ws['B1'].value**: The code expects the "report date" (the date when this information was compiled) to be in cell B1 of your Excel sheet. It reads that value.
* **if isinstance(report\_date, datetime.datetime): ...**: Sometimes, dates in Excel are stored in a special way that Python understands as a datetime object. If it is, this line makes sure to format it nicely as "day/month/year" (e.g., 01/06/2025).
* **data = []**: This creates an empty list. Think of this as an empty basket where the code will start collecting all the organized information it finds.
* **current\_section = None and current\_cargo = None**: These are like temporary sticky notes. As the code reads through the Excel file, it uses these to remember what "section" it's currently in (like "Expected" ships or "At Berth" ships) and what "cargo" type it's currently looking at (like "Crude oil" or "Diesel").
* **for row in ws.iter\_rows(values\_only=True):**: This is where the code starts going through your Excel sheet row by row, from top to bottom. The values\_only=True means it's just interested in the content of each cell, not its formatting.
  + **if all(cell is None for cell in row): continue**: If an entire row is empty, the code just skips it. No need to process empty lines!
  + **if row[0] == "Expected": ... and elif row[0] == "At Berth": ...**: These lines check the very first cell of each row (row[0]). If it sees "Expected" or "At Berth", it knows it's entering a new section of the report and updates its current\_section sticky note.
  + **if row[0] in ["Crude oil", "Diesel"]: ...**: Similarly, if it sees "Crude oil" or "Diesel" in the first cell, it knows it's about to list vessels carrying that specific cargo and updates its current\_cargo sticky note.
  + **if row[0] == "Vessel": continue**: The Excel file probably has headers like "Vessel" or "Date" within its sections. This line tells the code to ignore those header rows because they aren't actual data about a ship.
  + **if current\_cargo and row[0] and isinstance(row[0], str) and row[0].strip():**: This is the main check for actual data rows. It makes sure:
    - It knows what current\_cargo it's looking for.
    - The first cell (row[0]) isn't empty.
    - The first cell is actually text (a ship name).
    - And the text isn't just spaces.
  + **vessel = row[0].strip()**, **date\_val = row[1]**, **quantity = row[2]**: If all those checks pass, it grabs the **vessel name** (from the first column), the **date** (from the second column), and the **quantity** (from the third column).
  + **if isinstance(date\_val, datetime.datetime): ...**: Just like with the report date, it formats any date values to "day/month/year".
  + **quantity\_str = str(int(quantity)) if ... else str(quantity)**: Quantities might be numbers, so this makes sure they're always converted to text, and if they're whole numbers, it removes any decimal points (e.g., 1000.0 becomes 1000).
  + **data.append([...])**: This is where the organized data goes into our data basket! For each valid ship entry, it adds a new row to the data list, containing:
    - The report\_date (from cell B1)
    - The current\_section (Expected or At Berth)
    - The vessel name
    - The ETA/Berthed date
    - The current\_cargo type
    - The quantity
    - And "Awesome port" (since that's the port this report is about!)
* **return data**: Once the code has gone through every row, it hands back the data basket, which is now filled with nicely organized information.

**2. save\_to\_excel(data, output\_file): The "Saving to a New Excel" Recipe**

This function takes all that organized data and puts it into a brand new Excel file.

* **wb = Workbook()**: This creates a brand new, empty Excel workbook.
* **ws = wb.active**: It selects the first (and only, for now) sheet in this new workbook.
* **ws.title = "Parsed Data"**: It gives that sheet a nice name: "Parsed Data."
* **headers = [...] and ws.append(headers)**: It defines the titles for your columns (like "Report date", "Status", etc.) and adds them as the very first row in your new Excel sheet.
* **for row in data: ws.append(row)**: It takes each organized row from your data basket and adds it as a new row in the Excel sheet.
* **for column in range(1, 8): ... ws.column\_dimensions[col\_letter].width = 15**: This is a neat little extra! It goes through each column in your new sheet and makes them wider (to a width of 15) so that all the text fits nicely without being cut off.
* **wb.save(output\_file)**: Finally, it saves the entire new workbook to the specified output\_file name, which will be "Parsed\_Output.xlsx" in our case.

**3. if \_\_name\_\_ == "\_\_main\_\_":: The "Start Here" Button**

This is a standard Python way of saying, "When you run this script directly, start doing these things."

* **input\_file = '/content/Awesome port.xlsx'**: It sets the name of the original Excel file.
* **output\_file = "Parsed\_Output.xlsx"**: It sets the name for the new, organized Excel file.
* **parsed\_data = parse\_excel\_to\_table(input\_file)**: It calls the first function to read and organize the data from the input file.
* **save\_to\_excel(parsed\_data, output\_file)**: It then calls the second function to save that organized data into the new output file.
* **print(f"Data successfully saved to {output\_file}")**