Summary

Acme Groceries runs a small chain of grocery stores and has hired you to build software to help analyze their sales. Every month, their sales team submits a report to management with an overview of the last 6 months of sales. You have been tasked to build a backend application to ingest these reports and generate some outputs for managers to use.

Task

Build an application to ingest monthly reports, generate sales summaries by category and output consolidated reports.

Inputs

The input files may come in .xlsx (Excel) or .txt (tab delimited) format. They contain the following columns:

- SKU – a unique identifier for each product (always a 7 digit number)
- Category – classification that generally corresponds to which section of the grocery store the item goes in (e.g. Snacks)
- Units and Gross Sales for the prior 6 months

All files have the same headers (with the exception that the years and months in the Units and Sales columns may change), but the columns may not be in the same order. See files 201904.xlsx and 201905.csv for more detail. If the sales file is in Excel format, the data will be in the sheet labelled "Sales".

Each input file gives sales data for the previous 6 months. For example, file 1 might give sales from January-June 2018 and then file 2 would give sales for February-July 2018. Occasionally, the sales team will make small adjustments to sales numbers from prior months. If Units or Gross sales data for a given month conflict from one file to the next, use the numbers from the most recent ingested file when generating numbers for the `generate_report` and `summary` commands (more info about these commands below).

Additionally, a given ingestion file may contain duplicate SKUs, in which case you should combine units and sales for the two SKUs (even if categories differ). The same SKU might have different categories within or between files. If this is the application's first time encountering this SKU, use the category associated with the highest number of sales in the most recent month in the given file. However, once an item is assigned a category, the category doesn't change, even if it's different on a new input file.

Storage

You are going to need to store the data that gets ingested from the input files. While in real life you would want to use a persistent store such as a database to keep track of state, for this exercise, you can simply store the state in memory, filesystem, or other convenient local storage.

Commands

The interface allows the following commands:

ingest <filename>

This ingests a file (either .xlsx or .txt) and then shows a message with "Success" or "Error" based on the file being ingested properly. If there is an error in the file, roll back the transaction (in other words, if you reach an error on line 5 of the file where the sales value isn't a valid numeric type, then roll back any changes you made based on lines 1-4 of that file).

Example:

```
>>> ingest foo1.xlsx
Success
>>> ingest foo2.txt
Error
```

summary <category name> < year > <month>

Shows a summary of the given grocery category for the given year and month. Summary includes the category name, total units, and total gross sales.

```
{category_name} - Total Units: {total_units}, Total Gross Sales: {total_sales}
```

For example:

```
>>> summary _summary Produce 2019 1
Produce - Total Units: 35012, Total Gross Sales: 103015.65
>>> summary NotACategory 2019 1
No data available
>>> summary Produce 2150 4
No data available
>>> summary Produce 2018 12
Produce - Total Units: 32124, Total Gross Sales: 99517.22
```

generate_report <filename>

Generates a CSV with the following columns:

Year	Month	SKU	Category	Units	GrossSales
2018	12	1111111	Produce	60	671.2
2019	1	1111111	Produce	70	740.5
2019	2	1111111	Produce	70	740.5
…	…	…	…	…	…
2018	12	2222222	Freezer	85	1025.0
2019	1	2222222	Freezer	84	1012.0
2019	2	2222222	Freezer	89	1101.1
…	…	…	…	…	…

and exports it to <filename> (e.g. "foo.csv"). Include all available historical data in the report. Don't include any rows where units and sales are zero. The file should be ordered by SKU, then year and month.

exit

Quits the program

Notes

You can make the following assumptions:

- The input files have headers in the first row and the data starts on the second row
- No empty rows in the middle of the data. Ignore empty rows at the end of the file.
- You can use any 3rd party library installable via pip, npm/yarn, or nuget.

Submission

You can submit a zipped copy via email, or a link pointing to your GitHub or similar repo. If your file is too large for email, you can use DropBox or a similar service as an alternative.

Include your package manager's package file for any 3rd party dependencies, as well as any additional instructions needed to run your sample in a README.md.

You will be judged on understanding of the problem and adherence to spec and general code quality and organization. Completeness of all functionality is secondary.

Tips

Read the instructions carefully to understand the problem and map out a solution. There are some edge cases around duplicate values and error handling that need to be considered.

Consider writing unit tests around the core functionality, which will make refactoring and testing go faster. Include any relevant tests in your submission.

Performance and scalability are not a strong consideration. You can assume all data comfortably fits in memory and is scannable.

It is better to have a solid 2/3rds of the functionality complete with an obvious path to the rest, rather than hurrying to complete all functionality. If you felt particularly challenged for time or complexity on a portion, drop a note in the README and move on.

In keeping with the spirit of "real world code", you can make reasonable assumptions and modifications to the requirements as needed, so long as your application continues to meet the needs of the fictitious users. Eg., requiring category names to be a maximum length or contain no special characters, or using a shell script to convert XLSX to CSV as part of your command line application, etc. Please document any such changes or assumptions in your README.