:warning: Please read these instructions carefully and entirely first

- Clone this repository to your local machine.
- Use your IDE of choice to complete the assignment.
- When you have completed the assignment, you need to push your code to this repository and mark the assignment as completed by clicking here.
- Once you mark it as completed, your access to this repository will be revoked. Please make sure that you have completed the assignment and pushed all code from your local machine to this repository before you click the link.

Table of Contents 1. Before you start, a brief explanation for the exercise and software prerequisites/setup.
2. Tips for what we are looking for provides clear guidance on solution qualities we value 3. The Challenge explains the data engineering code challenge to be tackled. 4. Follow-up Questions related to the challenge which you should address
5. Your approach and answers to follow-up questions is where you should include the answers to the follow-up question and clarify your solution approach any assumptions you have made.

Before you start

Why complete this task?

We want to make the interview process as simple and stress-free as possible. That's why we ask you to complete the first stage of the process from the comfort of your own home.

Your submission will help us to learn about your skills and approach. If we think you're a good fit for our network, we'll use your submission in the next interview stages too.

About the task

You'll be creating an ingestion process to ingest files containing vote data. You'll also create a means to query the ingested data to determine outlier weeks.

There's no time limit for this task, but we expect it to take less than 2 hours.

Software prerequisites

To make it really easy for candidates to do this exercise regardless of their operating system, we've provided a containerised way to run the exercise. To make use of this, you'll need to have Docker (or a free equivalent like Rancher or Colima) installed on your system.

To start the process, there is a `Dockerfile` in the root of the project. This defines a linux-based container that includes Python 3.11, Poetry and DuckDB.

To build the container:

```
docker build -t ee-data-engineering-challenge:0.0.1 .
```

To run the container after building it:

Mac or Linux, or Windows with WSL:

```
docker run --mount type=bind,source="$(pwd)/",target=/home/dataeng -it ee-data-engineering-c
```

Windows (without WSL):

```
docker run --mount type=bind,source="%cd%",target=/home/dataeng -it ee-data-engineering-chal
```

Running the container opens up a terminal in which you can run the poetry commands that we describe next.

You could also proceed without the container by having Python 3.11 and Poetry directly installed on your system. In this case, just run the poetry commands you see described here directly in your own terminal.

Poetry

This exercise has been written in Python 3.11 and uses Poetry as a dependency manager. If you're unfamiliar with Poetry, don't worry – the only things you need to know are:

1. Poetry automatically updates the `pyproject.toml` file with descriptions of your project's dependencies to keep track of what libraries are being used.
2. To add a dependency for your project, use `poetry add thelibraryname`
3. To add a "dev" dependency (e.g. a test library or linter rather than something your program depends on) use `poetry add --dev thelibraryname`
4. To resolve dependencies and find compatible versions, use `poetry lock`
5. *Please commit any changes to your `pyproject.toml` and `poetry.lock` files and include them in your submission* so that we can replicate your environment.

In the terminal (of the running docker image), start off by installing the dependencies:

```
poetry install --with dev
```

> **Warning** If you're a Mac M1/M2 (arm-based) user, note that as of June 2023, DuckDB doesn't release pre-built `aarch64` linux wheels for the Python `duckdb` library (yet). This means that the dependency installation in the running container can take some time (10mins?) as it compiles `duckdb` from source. If you don't feel like waiting, you can build an `amd64` Docker container with by adding the `--platform`

`amd64` flag to both the `docker build` and `docker run` commands above (i.e. you'll have to re-run those). This image will run seamlessly on your Mac in emulation mode.

Now type

```
poetry run exercise --help
```

to see the options in CLI utility we provide to help you run the exercise. For example, you could do

```
poetry run exercise ingest-data
```

to run the ingestion process you will write in `ingest.py`

Bootstrap solution

This repository contains a bootstrap solution that you can use to build upon.

You can make any changes you like, as long as the solution can still be executed using the `exercise.py` script.

The base solution uses DuckDB as the database, and for your solution we want you to treat it like a real (OLAP) data warehouse. The database should be saved in the root folder of the project on the local disk as `warehouse.db`, as shown in the `tests/db_test.py` file.

We also provide the structure for: * `equalexperts_dataeng_exercise/ingest.py` - the entry point for running the ingestion process. * `equalexperts_dataeng_exercise/outliers.py` - the entry point for running the outlier detection query. * `equalexperts_dataeng_exercise/db.py` - is empty, but the associated test demonstrates interaction with an DuckDB database.

Tips on what we're looking for

1. **Test coverage**

   Your solution must have good test coverage, including common execution paths.

2. **Self-contained tests**

   Your tests should be self-contained, with no dependency on being run in a specific order.

3. **Simplicity**

   We value simplicity as an architectural virtue and a development practice. Solutions should reflect the difficulty of the assigned task, and shouldn't be overly complex. We prefer simple, well tested solutions over clever solutions.

   Please avoid:

- unnecessary layers of abstraction
- patterns
- custom test frameworks
- architectural features that aren't called for
- libraries like `pandas` or `polars` or frameworks like `PySpark` or `ballista` - we know that this exercise can be solved fairly trivially using these libraries and a Dataframe approach, and we'd encourage appropriate use of these in daily work contexts. But for this small exercise we really want to know more about how you structure, write and test your Python code, and want you to show some fluency in SQL – a `pandas` solution won't allow us to see much of that.

4. **Self-explanatory code**

   The solution you produce must speak for itself. Multiple paragraphs explaining the solution is a sign that the code isn't straightforward enough to understand on its own. However, please do explain your non-obvious *choices* e.g. perhaps why you decided to load data a specific way.

5. **Demonstrate fluency with data engineering concepts**

   Even though this is a toy exercise, treat DuckDB as you would an OLAP data warehouse. Choose datatypes, data loading methods, optimisations and data models that are suited for resilient analytics processing at scale, not transaction processing.

6. **Dealing with ambiguity**

   If there's any ambiguity, please add this in a section at the bottom of the README. You should also make a choice to resolve the ambiguity and proceed.

Our review process starts with a very simplistic test set in the `tests/exercise_tests` folder which you should also check before submission. You can run these with:

```
poetry run exercise check-ingestion
poetry run exercise check-outliers
```

Expect these to fail until you have completed the exercise.

You should not change the `tests/exercise-tests` folder and your solution should be able to pass both tests.

Download the dataset for the exercise

Run the command

```
poetry run exercise fetch-data
```

which will fetch the dataset, uncompress it and place it in `uncommitted/votes.jsonl` for you. Explore the data to see what values and fields it contains (no need to show how you explored it).

Begin the two-part challenge

There are two parts to the exercise, and you are expected to complete both. A user should be able to execute each task independently of the other. For example, ingestion shouldn't cause the outliers query to be executed.

Part 1: Ingestion

Create a schema called `blog_analysis`. Create an ingestion process that can be run on demand to ingest files containing vote data. You should ensure that data scientists, who will be consumers of the data, do not need to consider duplicate records in their queries. The data should be stored in a table called `votes` in the `blog_analysis` schema.

Part 2: Outliers calculation

Create a view named `outlier_weeks` in the `blog_analysis` schema. It will contain the output of a SQL calculation for which weeks are regarded as outliers based on the vote data that was ingested. The view should contain the year, week number and the number of votes for the week *for only those weeks which are determined to be outliers*, according to the following rule:

NB! If you're viewing this Markdown document in a viewer where the math isn't rendering, try viewing this README in GitHub on your web browser, or see this pdf.

> **A week is classified as an outlier when the total votes for the week deviate from the average votes per week for the complete dataset by more than 20%.**
> For the avoidance of doubt, *please use the following formula*:
>
> > Say the mean votes is given by $\bar{x}$ and this specific week's votes is given by x_i. We want to know when x_i differs from $\bar{x}$ by more than 20%. When this is true, then the ratio $\frac{x_i}{\bar{x}}$ must be further from 1 by more than 0.2, i.e.: $\left|1 - \frac{x_i}{\bar{x}}\right| > 0.2$

The data should be sorted in the view by year and week number, with the earliest week first.

Running `outliers.py` should recreate the view and just print the contents of this `outlier_weeks` view to the terminal - don't do any more calculations after creating the view.

Example

The sample dataset below is included in the test-resources folder and can be used when creating your tests.

Assuming a file is ingested containing the following entries:

```
{"Id":"1","PostId":"1","VoteTypeId":"2","CreationDate":"2022-01-02T00:00:00.000"}
{"Id":"2","PostId":"1","VoteTypeId":"2","CreationDate":"2022-01-09T00:00:00.000"}
{"Id":"4","PostId":"1","VoteTypeId":"2","CreationDate":"2022-01-09T00:00:00.000"}
{"Id":"5","PostId":"1","VoteTypeId":"2","CreationDate":"2022-01-09T00:00:00.000"}
{"Id":"6","PostId":"5","VoteTypeId":"3","CreationDate":"2022-01-16T00:00:00.000"}
{"Id":"7","PostId":"3","VoteTypeId":"2","CreationDate":"2022-01-16T00:00:00.000"}
{"Id":"8","PostId":"4","VoteTypeId":"2","CreationDate":"2022-01-16T00:00:00.000"}
{"Id":"9","PostId":"2","VoteTypeId":"2","CreationDate":"2022-01-23T00:00:00.000"}
{"Id":"10","PostId":"2","VoteTypeId":"2","CreationDate":"2022-01-23T00:00:00.000"}
{"Id":"11","PostId":"1","VoteTypeId":"2","CreationDate":"2022-01-30T00:00:00.000"}
{"Id":"12","PostId":"5","VoteTypeId":"2","CreationDate":"2022-01-30T00:00:00.000"}
{"Id":"13","PostId":"8","VoteTypeId":"2","CreationDate":"2022-02-06T00:00:00.000"}
{"Id":"14","PostId":"13","VoteTypeId":"3","CreationDate":"2022-02-13T00:00:00.000"}
{"Id":"15","PostId":"13","VoteTypeId":"3","CreationDate":"2022-02-20T00:00:00.000"}
{"Id":"16","PostId":"11","VoteTypeId":"2","CreationDate":"2022-02-20T00:00:00.000"}
{"Id":"17","PostId":"3","VoteTypeId":"3","CreationDate":"2022-02-27T00:00:00.000"}
```


Then the following should be the content of your `outlier_weeks` view:

Year	WeekNumber	VoteCount
2022	0	1
2022	1	3
2022	2	3
2022	5	1
2022	6	1
2022	8	1

Note that we strongly encourage you to use this data as a test case to ensure that you have the correct calculation!

Follow-up Questions

Please include instructions about your strategy and important decisions you made in the README file. You should also include answers to the following questions:

1. What kind of data quality measures would you apply to your solution in production?
2. What would need to change for the solution scale to work with a 10TB dataset with 5GB new data arriving each day?
3. Please tell us in your modified README about any assumptions you have made in your solution (below).

Your Approach and answers to follow-up questions

Please provide an explaination to your implementation approach and the additional questions ***here***