Tomáš Sobotík · Follow
Nov 2, 2021 · 12 min read

How to ensure data quality with Great Expectations

Data Quality — one of the crucial parts of the DWH development lifecycle but still I believe part which usually lies on the bottom of the backlog with the lowest priority. Simply because it does not bring immediate business value — no new dashboard or KPI is coming on stage because of DQ, no new customer intake, or canceled churn because of the data quality solution in place. But in long term I think DQ brings many benefits, starting with potential data issues being discovered before they are spotted and reported by business users which is already late because it can have an impact on business or decisions which are being made. Another unquestionable benefit of having DQ in place is building confidence. Confidence in data, confidence in the data team who is preparing them. Last but not least having DQ as part of the project can speed up the development and make it cheaper because once you have an automatic system which is ensuring that your business rules are applied correctly and data have the shape which they are supposed to have then you do not need to dedicate a time to do such task manually as part of development.

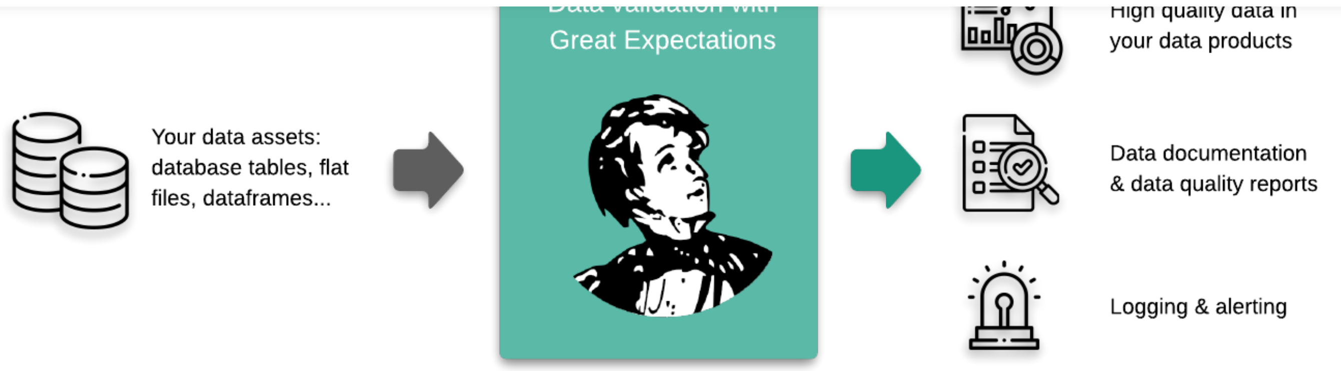
During my career and many projects which I have been part of we sooner or later have made it to the point when we have started thinking about some automatic DQ solution. But always we have started with some bits and pieces, custom-made code which was trying to ensure some minimal data quality. Many times also the target architecture has contained some info about the DQ solutions that should be implemented *some times*.

Today, I would like to have a look at Data Quality, not from an isolated view when developers try to ensure the DQ at least on the level of database constraints. But I would like to have a look at the matter from a complex view, introduce a great tool that has the word **GREAT** in the name and try to introduce complex architecture how it might be integrated into a solution including **Snowflake** and used for automatic data quality verification.

Today I am going to introduce a data quality tool called **Great Expectations** and solution which I have developed around it. We will cover parts like how to prepare infrastructure and development environment for the whole team and then we will have a look at how to integrate the tool with Snowflake and automate the data verification process through **Github Actions**. Let's start! 🚀

Great Expectations introduction

The great expectation is an open-source tool built in Python. It has several major features including data validation, profiling, and documenting the whole DQ project. Automation together with proper logging and alerts on top of data quality solutions are essential parts of having a successful and reliable solution. Great Expectations (GE) offers many integrations (Airflow, Slack, Github Actions, etc.) which supports those tasks and makes them possible.


Your data assets: database tables, flat files, dataframes...
Data validation with Great Expectations
High quality data in your data products
Data documentation & data quality reports
Logging & alerting

What are Great Expectations about

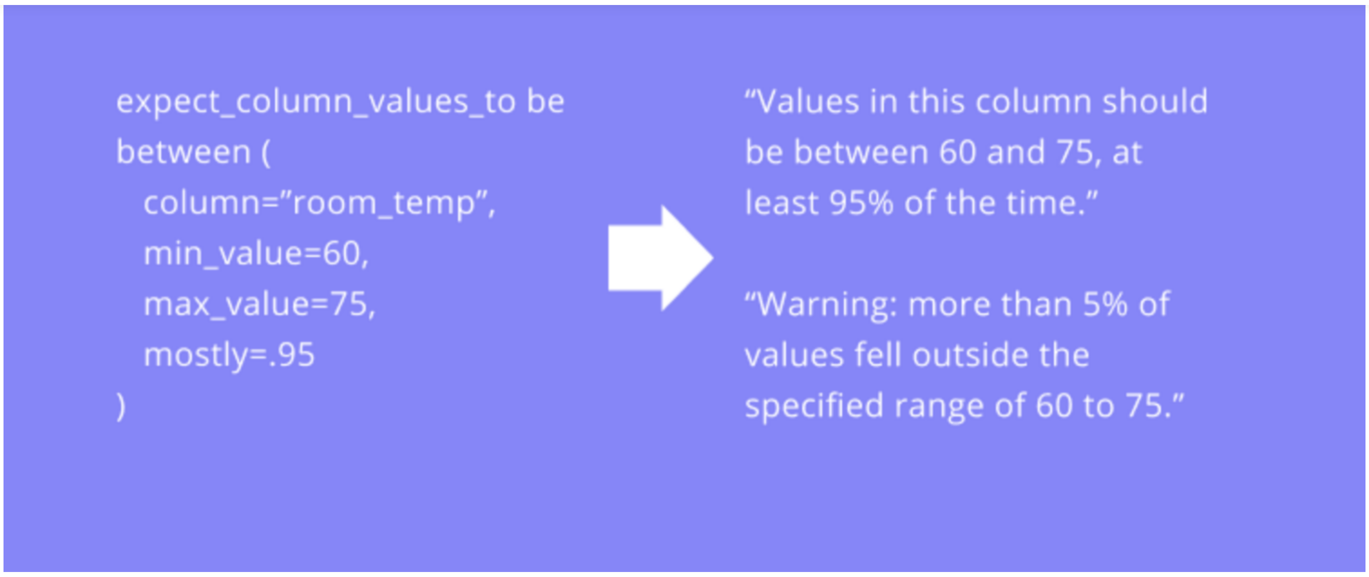
Expectations

Are one of the tool's key features used for data assertions. There is a lot of prebuilt expectations where not all of them have been implemented for all sources. If you need an expectation that does not exist you are free to develop it on your own! Meaning that the whole concept behind expectations is very flexible and easy to extend. How does such an expectation look? Please look below. Basically, they cover common data issues.

```
expect_column_values_to_not_be_null

expect_column_values_to_be_unique

expect_table_row_count_to_be_between

expect_column_to_exist

expect_column_values_to_be_in_set
```

Data docs

Automatically generated documentation about your expectations same as about result of their verification is called Data docs. It is generated into a human-readable format in form of web pages available to anyone who needs it. Documentation is always up to date because it is generated based on your tests and tests run against new data. Documenting is always a pain for each team so providing automatically generated documentation makes life easier for developers and provides value to data consumers and maintainers.



```
expect_column_values_to be
between (
  column="room_temp",
  min_value=60,
  max_value=75,
  mostly=.95
)
```

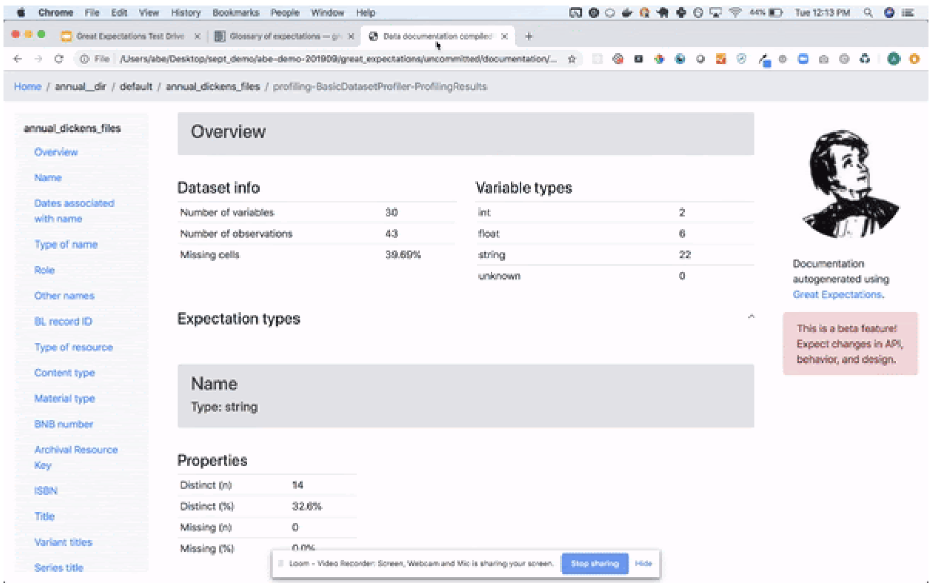
"Values in this column should be between 60 and 75, at least 95% of the time."

"Warning: more than 5% of values fell outside the specified range of 60 to 75."

Expectations translation into a human-friendly format available in data docs

Automated data profiling

Great Expectations can also profile your data automatically and based on results generate the expectations based on profiling. This is a nice feature of how to make understanding the new data easier and faster. Of course, you still need the domain knowledge and verify the proposed expectations as not all of them will be relevant. But as I said it can help you especially with new datasets and maybe it will provide expectations which you would not think of at the early stage of development.



Chrome File Edit View History Bookmarks People Window Help
Great Expectations Test Drive
Glossary of expectations
Data documentation compiled
/Users/abe/Desktop/sept_demo/abe-demo-201909/great_expectations/uncommitted/documentation/...
Home / annual__dir / default / annual_dickens_files / profiling-BasicDatasetProfiler-ProfilingResults
annual_dickens_files
Overview
Name
Dates associated with name
Type of name
Role
Other names
BL record ID
Type of resource
Content type
Material type
BNB number
Archival Resource Key
ISBN
Title
Variant titles
Series title
Loom - Video Recorder: Screen, Webcam and Mic is sharing your screen.
Stop sharing
Hide

Overview

Dataset info

Number of variables	30
Number of observations	43
Missing cells	39.69%

Variable types

int	2
float	6
string	22
unknown	0

Expectation types

Name
Type: string

Properties

Distinct (n)	14
Distinct (%)	32.6%
Missing (n)	0
Missing (%)	0.0%

Documentation autogenerated using Great Expectations.

This is a beta feature! Expect changes in API, behavior, and design.

Profiling example

Various data sources

I want to highlight how many different data sources GE supports. Starting with various SQL databases like **MySQL**, **PostgreSQL** through cloud-based DBs including **Athena**, **BigQuery**, **Redshift,** and **Snowflake**. Next, you can connect to data on **S3**, **GCS**, **Azure Blob Storage** using **Spark** or **Pandas**. Last but not least you can of course validate data on a local filesystem.

Where to get more info and how to learn GE?

simple „*hello world*" application going through different use cases on how to use this and that feature of GE.

Community

I have to mention also the community around GE. There is a growing friendly community around and you can meet others on Slack or GitHub. Slack community is really helpful in cases you will be stuck somewhere. There are support and beginners channels and I have found help there many times.

❄ Snowflake and Great Expectations

Now let's focus on our own implementation of Great Expectations, how we have integrated it into our architecture, daily routine and what kind of challenges in relation to infrastructure and overall architecture we have been facing. We have DWH in Snowflake — 30 TB in size, a couple of thousand ETL jobs, various sources including relational DBs, flat files, APIs, etc. We have been aiming for Data Quality Solution which will be:

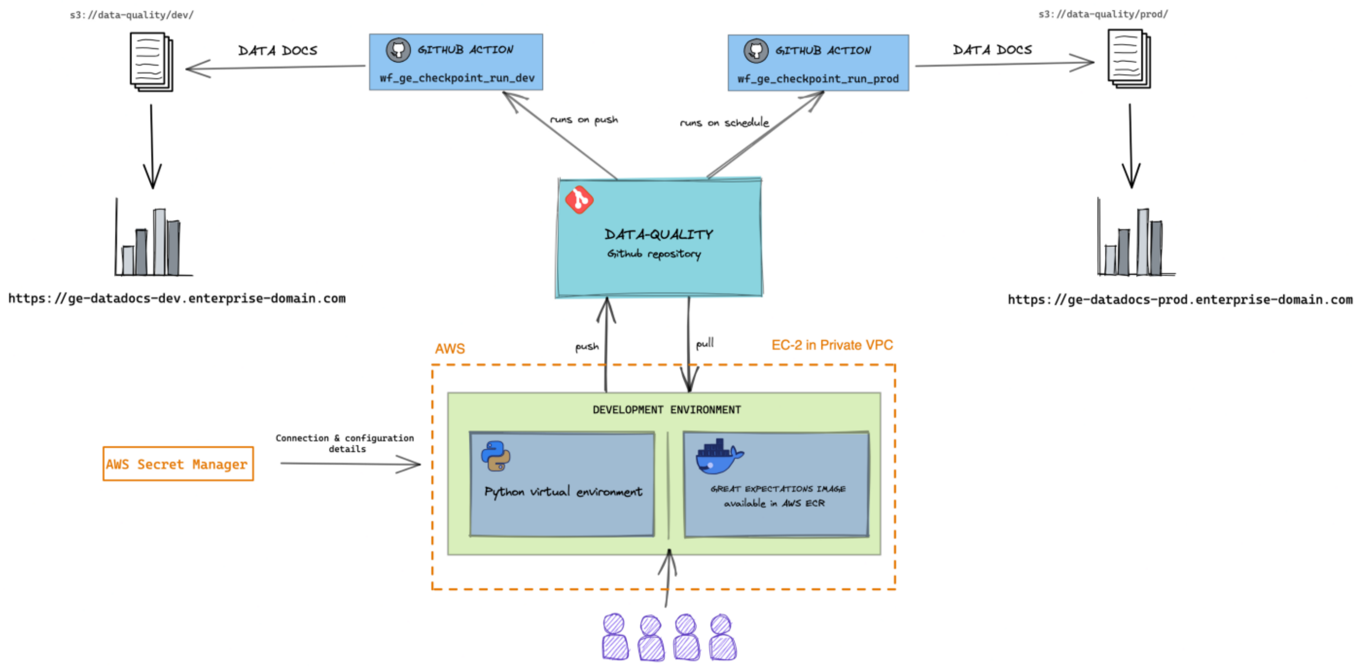
- working as a standalone solution
- independent from the current ETL tool (Dataiku)
- code-based — Python preferably
- working natively with Snowflake
- supporting various orchestration tools like Apache Airflow or GitHub Actions

Great Expectations and Snowflake?

Great Expectations fulfill all the requirements. The first challenge we had to solve is how to prepare the development environment and whole GE set up for the data engineering team. GE is not a client-server where all the developers would be connecting to some sort of server and work there simultaneously on different tasks. The usual pattern is having GE installed locally and working there or using some prepared docker image. Because of internal rules we are not allowed to fetch data into local computers. In the end, we have decided to build the whole infrastructure around the GitHub repository because as an orchestration tool we use GitHub with prepared python's virtual environments per developer or as a second option developers can use prepared docker image with preinstalled tools and dependencies. You can see the architecture on high level schema below. Let's look at it in detail.

Way of working & Architecture

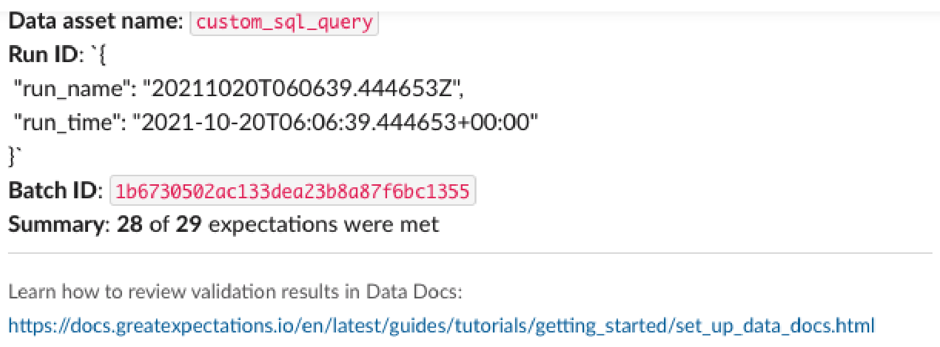
We had to figure out how to have development environment on one side and production (automated) environment on the other side. As I said for development we use an EC2 machine where each developer has its own Python virtual environment or use a prepared docker image which is available in corporate AWS ECR. As you can see in the picture the central point of the whole architecture is GitHub repository. When the developer starts working on the new GE-related task he needs to fetch the latest changes from the repository first to get his own virtual environment updated with the latest changes from others and then create a new branch where the new functionality will be developed.



High-level Data quality solution architecture

Once he is ready with his development locally (inside Python virtual environment or docker image) and everything is working as expected developer can test the automation part by pushing his own branch back into the repository. There is a prepared „DEV" version of Github Actions workflow which is automatically triggered every time there is a push to a non-master branch. This will ensure that newly developed functionality is working also on GitHub Runners when it is triggered via GitHub Actions and it will also ensure that the new part does not break anything already developed.

When new functionality is verified also inside the GitHub environment (successful run of GitHub Actions workflow) developer can create a pull request to merge his changes into the master (production) branch. Pull request needs to be reviewed and approved by admins and then it is merged. Production workflow runs on schedule every morning after daily ETL loads are done to verify all the data in the production Snowflake database. What is not visible in the schema is notification system. We have integrated GE with our Slack and every checkpoint failure is sent into our channel dedicated to DQ issues.



Data asset name: `custom_sql_query`
Run ID: `{
"run_name": "20211020T060639.444653Z",
"run_time": "2021-10-20T06:06:39.444653+00:00"
}`
Batch ID: `1b6730502ac133dea23b8a87f6bc1355`
Summary: **28** of **29** expectations were met

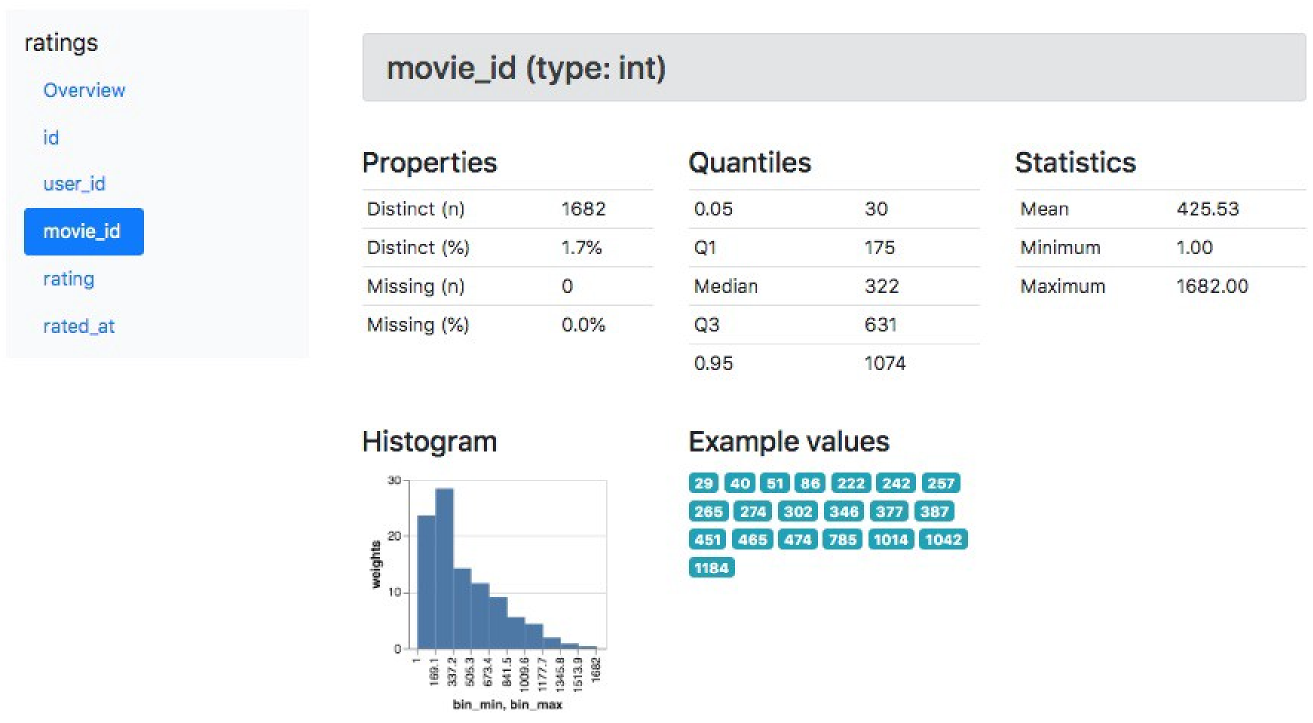
Learn how to review validation results in Data Docs: https://docs.greatexpectations.io/en/latest/guides/tutorials/getting_started/set_up_data_docs.html

Slack notification about expectation suite failure

Last but not least you can see also the integration of **AWS Secret Manager** where we keep needed Snowflake credentials which are rotated regularly. Of course it is because of security reasons as we do not want to have logins and passwords laying in different configuration files in EC-2 machine. And with integration of AWS Secret Manager we read our secrets directly from Secret Manager with no need to keep them anywhere locally. Credentials rotation is done by Lambda function which at the same time updates the credentials in the Manager. By integration of Secret Manager all processes have always access to up-to-date passwords, roles, warehouses, etc. without any difficult distribution of such valuable assets.

Data docs storage and presentation

As you can see we store our data docs on AWS S3. Then we have web servers running as docker images which represent the presentation layer of automatically generated documentation and fetch the content from defined paths in S3.



movie_id (type: int)

Properties

Distinct (n)	1682
Distinct (%)	1.7%
Missing (n)	0
Missing (%)	0.0%

Quantiles

0.05	30
Q1	175
Median	322
Q3	631
0.95	1074

Statistics

Mean	425.53
Minimum	1.00
Maximum	1682.00

Histogram

Example values

29 40 51 86 222 242 257 265 274 302 346 377 387 451 465 474 785 1014 1042 1184

Example of data docs

it properly, and how to do it in best way in relation to our data and processes. We already have tens or even maybe hundreds of expectations stacked into several checkpoints. Most of them verifies source data in terms of basic verifications like not null values and not empty datasets, having a value from the defined set, or checking the data types — e. g. date for all date values. For now, we have just a few expectations that verify business logic but it is slowly growing. Already now by checking source data regularly we can spot potential data issues in source systems very soon before they are reported by business users. Those basic expectations also make a small „pressure" on data developers to be precise in defining correct data types otherwise it is immediately reported. Let me share a few points which I have in my mind how to improve the current solution and move it to the next level. 🚀

Tighter integration with ETL/ELT processes

Right know we run all the checkpoints and all the expectation suites on schedule when our daily loads are done. Processing time is slowly raising as we add more and more expectations. But we do not need to run all of them in same time. We should aim for more parallelism and run those expectations and checkpoints immediately when portion of the data which is going to be verified is refreshed. Thanks to that we will split the whole GE processing into several smaller chunks which will be tightly integrated with that concrete ETL/ELT process that prepares those data. We just need to look at how to trigger the GitHub actions workflow directly from our ETL/ELT tool right after the process is done. GitHub Actions offers also external triggers through their GitHub API so I believe this will be way to go.

Data quality issue classification

This relates to tighter integration with the ETL tool. I can imagine also reverse integration and in case of a critical issue that might corrupt the data in that way, it would have an impact on business the ETL processing could be stopped to preserve current (correct) data. Now we do not use any issue classification and reports everything as a data issue but we also do not stop data processing. I see at least two categories:

- Critical issue = stop the processing
- Data issue = just alert team on slack

More business logic oriented expectations

We have collected feedback from business users about the usual data issue and what they are looking at to ensure the data they use are correct. We should aim for transforming this into expectation suites to ensure analytical tables are correct and follow defined and developed business rules.

Skeleton expectation suite

There are expectations which repeats for majority of the tables. I would like to have in place some kind of skeleton that would contain those basic expectations like key columns are not null, they are unique and exists in the schema, data types verification, or dimensions values from a limited set.

Export area schema detection

As each ordinary DWH we are providing many data exports for our customers and partners. Those integrations are mainly system to system and our partners have built their ETL processes based on schema which we provide. Because of that, I see ensuring that schema is still the same (number of columns, their order, and data types) as a crucial part to keep transfers running smoothly. This is another area where we can deploy Great Expectations. 💪

natively work with Snowflake and fulfill all our other requirements. So far it looks Great Expectations is great candidate. We are still learning and pushing our solution towards being more automated, more reliable and easier to use. Speaking about Snowflake integration it works smoothly, GE is using Python's SQLAlchemy for Snowflake.

From my perspective, there are several selling points why you should at least consider trying Great Expectations if you are looking for a data quality tool, no matter what data sources you have. Let me highlight the most relevant ones:

- community around
- great documentation, learning materials
- integration with modern cloud services (Secret Managers, cloud storage, etc.) and messaging services like Slack
- automatically generated documentation in human-readable format (Data Docs)
- The python-based — easy learning curve

If there are pros, there should be also cons. Let me also share few points which I think could be improved or it is needed to have them in mind when you will be considering using Great Expectations.

There is rapid development behind the tool and new version is out very often. Even though the team is trying to keep continuity some times there are bigger changes in the architecture which will require also changes at your end, in your code. One of those is V2 API vs V3 API. We still have this migration ahead of us but it is good there is some transition period and now both versions are supported. Also sometimes it can happen that any particular part of your solution can be broken by a new library release or anything similar. This has happened to me with new version of snowflake-sqlaclchemy some time ago. But such things are usually solved very quickly by the community. As the new features are adding very quickly it also means that not all parts are updated at the same time and for instance, some new features are firstly available only via code before they will be accessible via Great Expectations CLI or wizards. What I am trying to say is that it should be obvious that the tool is not mature yet and there is very active development behind which requires you to test your code before moving to the newer version.

All in all I think **Great Expectations** is great tool for Data Quality. It is very versatile which make it usable for different use case or environments. Great community around and active development confirms that it is a tool that is worth trying! 👏