Case Study - Xinyuan

Analytics Engineer - Full Stack Case Study

Hello! Congrats on making it to this round of the interviewing process for the Analytics Engineer position at Fluence Digital.

This case study consists of several parts, and is meant to test your abilities writing SQL and Python, your software engineering skills in terms of design and deployment, and your ability to both tell data stories and help stakeholders tell their own data stories.

If you knew every piece of this puzzle going in and how they fit together, a perfectly good response might take you a few hours. However, this case study is meant to meet you wherever you are in your skills, and you are not expected to know every piece of the puzzle going into it. In fact, if you knew none of the pieces, there is enough material online that you might be able to accomplish it in a weekend starting from scratch.

You will likely need to make design decisions along the way that minimize pitfalls, or play to your strengths. Some components have suggestions for bonus points (a harder, but more robust path) and shortcuts (a less robust, but much faster solution). Explanations of your design decisions are critical for success. If you take a shortcut, or try for bonus points, explain your choice in your write-up.

Case Study

Prompt - "Which forecast is better: ours, or the market operators?"

Your task is to answer the question of "which forecast is better?" as if a business stakeholder asked it, and to provide them a deployed webapp that shows them the result.

Part 1 - The Data

1. Inside of a Snowflake instance, there are 3 tables.
    1. `our_forecast`
    2. `their_forecast`
    3. `actual_prices`

Each contains a year's worth of price data, with timestamps and prices in $/kWh of electricity.

Snowflake instance located at: https://ke74435.snowflakecomputing.com/

Your credentials are as follows:

- Login: `XCHENG`
- Password: `jq-JNWJxLgP-7p`
- Compute Warehouse: `CASESTUDY_WH`
- Database.Schema: `CASESTUDY.CASESTUDY_XINYUAN`

Login to the instance and view the data (you can do this through your browser - it will prompt a password change). You may do whatever you want to the schema or data (create tables, add stages, add columns etc etc).

FYI The permissions are fully locked down and I can see all your queries. :)

Part 2 - The "Better" Metric

You get to decide on how you want to measure which forecast (the Fluence forecast or the market forecast) is "better". Decide on a metric to evaluate the forecasts against the actual prices (to tell which forecast is "better"), and explain your choice. There are many ways to show how a forecast compares to an actual price, so the most important part is that you can explain and defend your choice.

Reminder: Compare each forecast to the actual prices first! Then compare their performance on that metric to each other.

Part 3 - The App

If this were an analyst position, the case study might stop there. But this is an *engineering* position. Scripts and CSVs are not what we are here for. We are here for deployment. We are here to *ship*.

Build a small applet that does the following:

1. Pulls the data from Snowflake
2. Calculates your "better" metric, and indicates which forecast is better
3. Displays it to the user

Some key features:

1. You'll need to pull data from the database using SQL. *Your SQL must contain at least one JOIN* (no dumping to pandas and doing your join there, even if that might be the smarter option...I want a SQL example)
2. *Calculate your metric for each forecast in python*. Here is where you can mash some dfs together. You may do it from scratch or use any library you want.
3. *Deploy your solution to the web*. I need to be able to go to a URL and interact with your app.

Hints and Tips for each of those key elements are below...

Options for Object-Relational Mapping (ORM)

You'll need to get the data out of the database and into a python object to manipulate it. For that you'll need an ORM.

You can use any ORM, such as SQLAlchemy. Many have Snowflake compatibility explicitly in the library. However, there is a library included here that you might find useful for your app development, called `pandas_to_snowflake.py` It does what the name says. It converts your pandas dataframes to rows and columns in Snowflake, and (more importantly), vice versa. Using this library, you can write SQL against the data and get back a dataframe, which you can then use for your analysis, plotting, etc. Pretty handy.

- Default Option - Just pull the data and calculate your metric each time, ephemerally in the app.
- Bonus Option(s) - Cache or persist the metric back to the database, and then pull from that cache or persisted table intelligently.
- Shortcut option - Just output your metric result once to a flat-file with `df.to_csv()` or similar.

Options for data visualization

- Default Option - Visualize with a library in Streamlit. Streamlit has some pretty straightforward plotting tools, and support libraries like matplotlib and seaborn. Show which forecast is better in the browser. Interactivity for the user in Streamlit here is encouraged.
- Bonus Points Option(s) - Add in interactivity and exploration with a library like Bokeh, or build a whole React app with Django and CSS. These elements can be mixed and matched if you know them (or always wanted to learn!)
- Shortcut option - Sometimes over-engineering is cool, and sometimes it's spaghetti. Just display the data with matplotlib in a Jupyter notebook and call it a day.

Options for deployment

- Default Option - Deploy with Streamlit. When you are developing locally with streamlit, there is an option to deploy to the web directly from your local environment. This is very handy and can be used here.
- Bonus Points Option(s) - While deploying with Streamlit is a fine option, sometimes apps are doing things that Streamlit doesn't support. For a more generalized, scalable solution, use Docker and a Platform-as-a-Service tool like Heroku to deploy your app. Or go wild and write terraform and deploy with k8s! *Careful, these options can get difficult*.
- Shortcut Option - Sometimes we don't have time to write infrastucture-as-code. Plus, who's going to maintain it? Us? Just put your notebook on JupyterHub or Github, link to it in the write-up and call it a day.

Conclusion

If you take every shortcut, you might be able to crank it out in an hour in a few dozen lines, but that probably won't be good enough to get the job. If you write a fully-permissioned interactive React app with k8s infrastructure-as-code and send me my login to compare forecasts on 5 different metrics, I'll refer you over to the platform engineering team (kidding, probably don't do that).

Include a link to your app and a small write-up on your design decisions, explanation of your "better" metric and why you chose it, and any others thoughts or usage notes you want me to know. Send your writeup and navigation to your app to me via email.

`ryan.kladar@fluenceenergy.com`

GOOD LUCK!

Resources

1. Devops
    - It would be rare for one to be developing full stack, engineered apps in a corporate environment without devops support. Fear not! I will be your devops support for this project, so if you can't seem to login, or don't have the right permissions to access a resource, ping me and I will respond promptly. However, just like in a real corporate environment, be mindful of the other person's time in your requests, and formulate them so that the issue can be resolved quickly. I will NOT be here for app design assistance or guidance on the assignment, only technical support for use of the data and resources, just like an actual devops department.
    - Email: `ryan.kladar@fluenceenergy.com`
2. Docs
    - Snowflake Docs
    - Streamlit Docs
    - Heroku w/ Docker Docs