Data Exam

TURING TEST EXTRA CREDIT: CONVINCE THE EXAMINER THAT HE'S A COMPUTER.
YOU KNOW, YOU MAKE SOME REALLY GOOD POINTS.
I'M ... NOT EVEN SURE WHO I AM ANYMORE.

This exam helps us evaluate your software engineering & data skills. There's **no need for you to complete all of the tasks - but the more, the better.**

We **strongly encourage you to** use Google, Stack Overflow or any other similar websites when you need to unblock yourself in the task or when you need further understanding. This would be very ordinary on any given day of work. 📈
Obviously, we ask you to **refrain from** getting outside help from friends, colleagues or any other third-party people. 🧐

Environment Setup

For the following exercises you will need to install different services.

We **strongly recommend** you read the following installation guides, but you can do as you please. They might not be plug and play, but should work with some simple edits.

PostgreSQL

The best option for this will probably be via Docker using the following Postgres image: https://hub.docker.com/_/postgres

Docker should be installed, if you find that Docker is not part of your terminal's commands, then you might find this guide helpful to install it. This answer might be relevant too.

Remember to **push this setup code** to the repo.

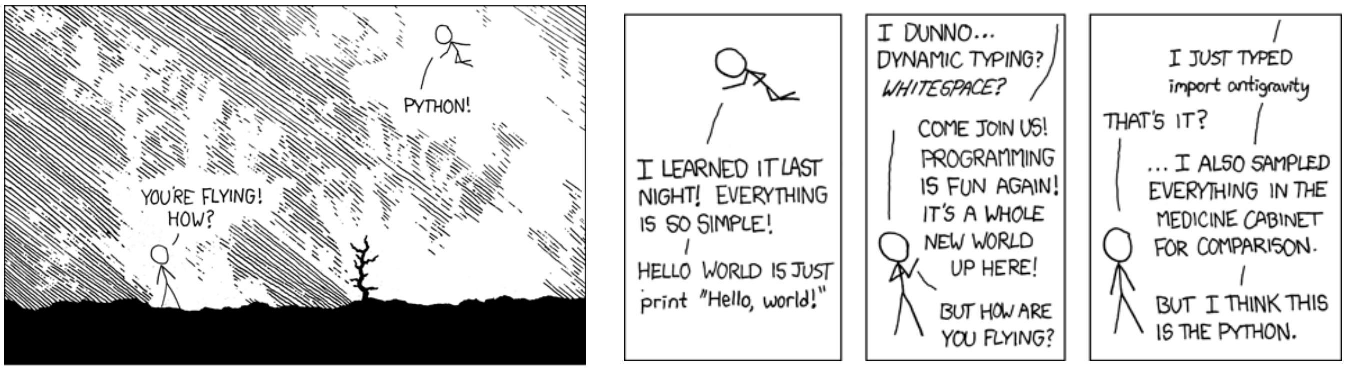
Relevant for later: setup the PostgreSQL server to listen on port 5432, which has inbound traffic permissions.

Python 3.8+

Here is a guide to install Python 3.8 that will work on an AWS EC2 server operating under Red Hat Enterprise LInux (RHEL):

https://techviewleo.com/how-to-install-python-on-amazon-linux/

1. Getting crypto token data 💸


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Context

This exercise tries to evaluate developer experience in Python software. We'll be using a free API called **CoinGecko**, which tracks prices, volume and other information for more than 3K+ different crypto assets. The API doesn't require an access key.

For reference please consult the official API documentation.

To develop the following Python3 tasks, we recommend using the **requests, click (or argparse/typer)** and **logging** python libraries, but please use any libraries of your choice.

Tasks

1. Create a command line Python app that receives an ISO8601 date (e.g. 2017-12-30) and a coin identifier (e.g. `bitcoin`), and downloads data from */coins/{id}/history?date=dd-mm-yyyy* endpoint for that whole day and stores it in a local file. To start, this app should store the API's response data in a local file. Save it in whichever file-format you judge as best for your problem with the corresponding date in the file's name.

   An example endpoint would be :
   https://api.coingecko.com/api/v3/coins/bitcoin/history?date=30-12-2017

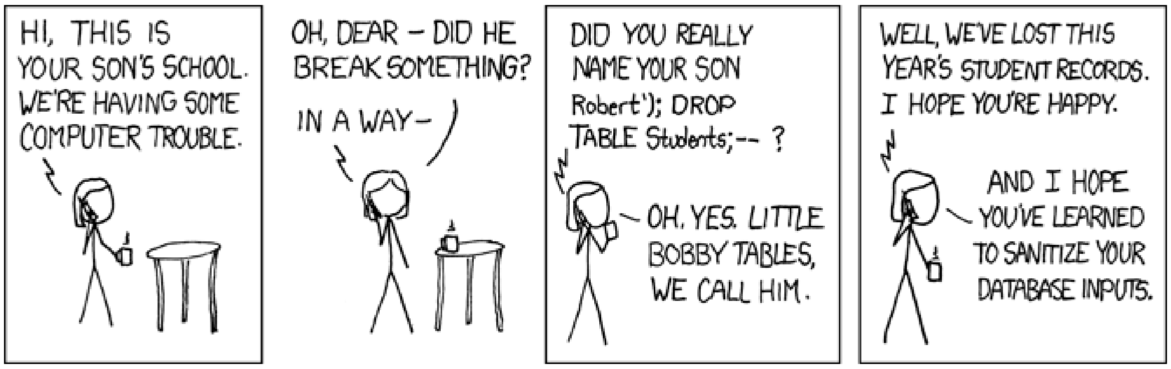
Note: If you are having trouble working with the API and processing its response, then it is also valid to do this exercise with a mocked response.

2. Add proper Python logging to the script, and configure a CRON entry in the instance that will run the app every day at 3am for the identifiers `bitcoin`, `ethereum` and `cardano`. Document this in the README.md file of the repo.

3. Add an option in the app that will bulk-reprocess the data for a time range. It will receive the start and end dates, and store all the necessary days' data. Use whichever patterns, libraries and tools you prefer so that the progress for each day' process can be monitored and logged. Here you can bring your own assumptions if they simplify your work, just remember to comment them in the code or in the README.md.

   Bonus: run the above process in a concurrent way (with a configurable limit), use any library/ package of your preference for this such as **asyncio** or **multiprocessing**.

2. Loading data into the database


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Context

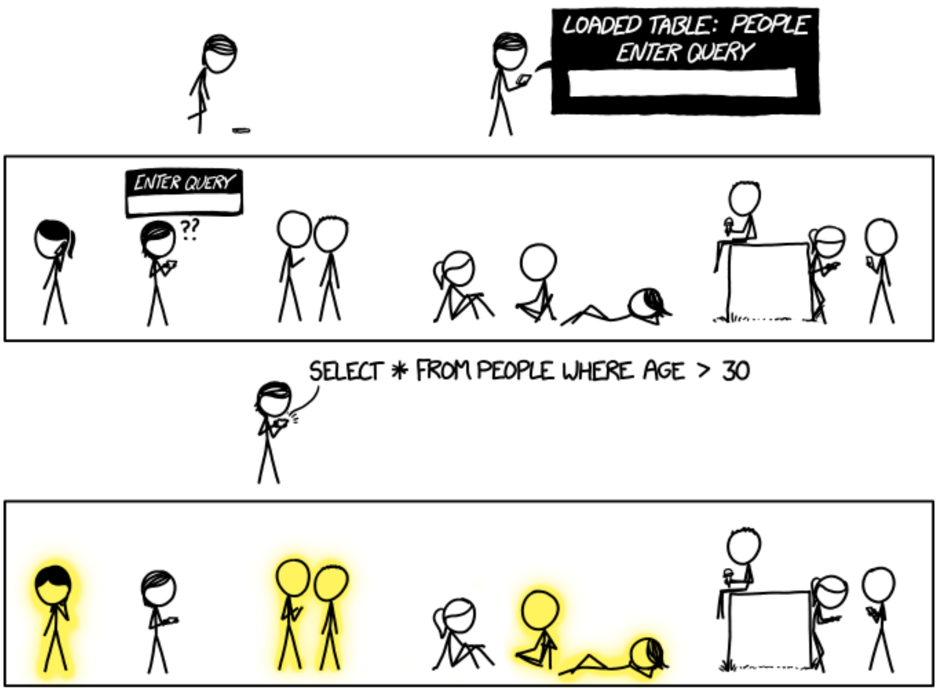
Data's ephemeral in a script without a database. For this exercise we'll be using the command line app that downloads the data and we will store it in the Postgres instance that we previously created to persist data.
Remember to follow the recommended practices for dealing with databases in Python,

such as using the **sqlalchemy** python library (and **alembic** if you find it helpful), but use whichever libraries you prefer.

Tasks

1. Create two tables in Postgres. The first one should contain the coin id, price in USD, date and a field that stores the whole JSON response. We'll be using a second table to store aggregated data, so it should contain the coin id, the year and month and the maximum/minimum values for that period.
2. Modify the command line app so that an optional argument enables storing the data in a Postgres table, and updates the given month's max and min prices for the coin. Remember to add the code logic that deals with pre-existing data.

3. Analysing coin data with SQL 👓



Context

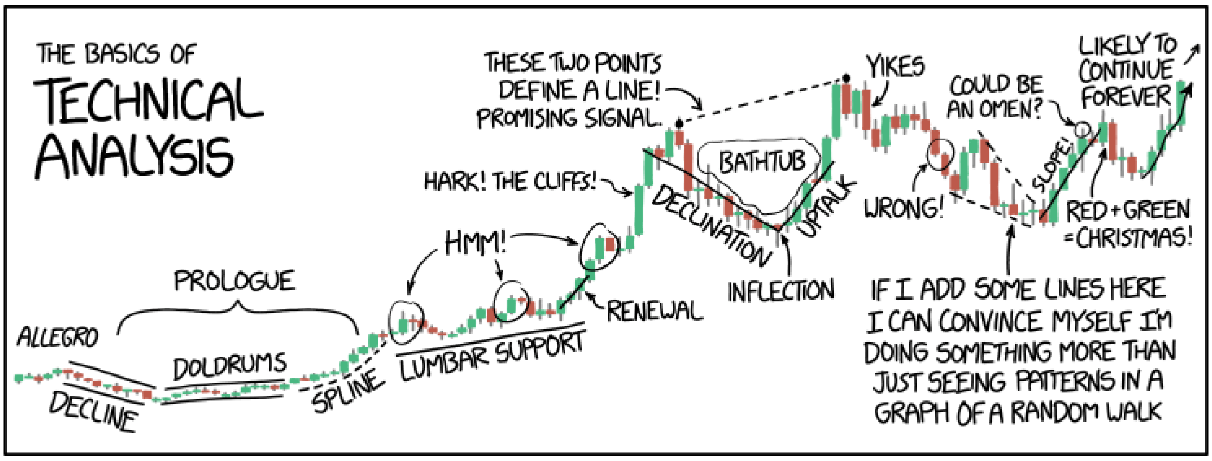
This exercise tries to evaluate expertise in SQL language, specifically on DQL statements.
For the next task, we would like you to write SQL queries for certain cases/analysis and save them in one (or more) .sql files inside your repo.

Tasks

Create SQL queries that:

1. Get the average price for each coin by month.
2. Calculate for each coin, on average, how much its price has increased after it had dropped consecutively for more than 3 days. In the same result set include the current market cap in USD (obtainable from the JSON-typed column). Use any time span that you find best.

4. Finance meets Data Science 🏛️


THE BASICS OF
TECHNICAL ANALYSIS
ALLEGRO
DECLINE
PROLOGUE
DOLDRUMS
SPLINE
HMM!
LUMBAR SUPPORT
RENEWAL
HARK! THE CLIFFS!
THESE TWO POINTS DEFINE A LINE! PROMISING SIGNAL.
BATHTUB
DECLINATION
UPTALK
INFLECTION
YIKES
WRONG!
COULD BE AN OMEN?
SLOPE!
LIKELY TO CONTINUE FOREVER
RED+GREEN =CHRISTMAS!
IF I ADD SOME LINES HERE I CAN CONVINCE MYSELF I'M DOING SOMETHING MORE THAN JUST SEEING PATTERNS IN A GRAPH OF A RANDOM WALK

Context

Using the data stored in the Postgres database, we'd like to perform some further analysis with **pandas**. You may do the tasks in a Jupyter Notebook or as standalone scripts, whatever you prefer.

Tasks

Develop python code to:

1. Plot the prices of `bitcoin`, `ethereum` and `cardano` for the last 30 days, commit those plots images in your repo.
   Hint: You can use `pandas`' read_sql_query, or read_sql_table to query data from Postgres.
2. Loading all data from the daily stock value table as a Dataframe, generate the following new features:
   a. Define 3 types of coins: "High Risk" if it had a 50% price drop on any two consecutive days, during a given calendar month, "Medium risk" if it dropped more than 20%, and "Low risk" for the rest (in this same fashion of consecutive days).
   b. For each row-day, add a column indicating the general trend of the price for the previous 7 days (T0 vs. T-1 through T-8), and the variance of the price for the same 7 days period.

3. To perform predictions on this time series we'll need to have a slightly different structure in the dataframe. Instead of having only the price and date for each row (plus the newly generated features), we'll also want the price of the last 7 days as columns as well as the next day's price, which will be used as a target variable.
    a. You can now add extra features if you'd like, for example with feature scaling or with the skewness of the stock.
    b. Add any other time features you'd like such as what day of the week is this? Is this a weekend or weekday? What week of the year is this? What month? Etc.
    c. Play with the data regarding the volume transacted to create new features.
    d. **Bonus**: Add extra features for both China's and US' holidays using Python's `holiday` package.
4. **Regression:** For every (coin, date) row in your dataset with date T0, predict the next day's price (T1) based on the previous 7 days of the stock's price (T-7 through T-1).
   You can use a simple linear regression model for this task, there's no need to get fancy with the ML model.

   Note, the seven rows with earliest date will not have any previous dates on which to predict!
   So you need to skip these predictions.

Bonus Section

So you are feeling adventurous? 🤓 Choose one of the following:

1. Add one unit-test to your previous python API code, for whichever function you find fit to be tested.
2. Use Poetry to manage library dependencies for the first and fourth exercise.