Data Engineer at 42matters
Interview Task

General Requirements:

We are looking for good code structure and clear communication style, the future tasks of the data engineer will be very important for our company and we want to be able to rely 100% on the results of the job. Commands in the READMEs must work out of the box and code should not contain passwords, credentials and hardcoded paths. Python as a language has naming conventions that should be followed. Documentation should be easily readable, Markdown (what Github uses) is a good way to do that. Code should be commented. Tools and commands used should contain a very clear instruction on how to install and configure. A good and platform-independent way is to do this via Docker, but we leave that open to the candidates in case they have a better way to do that.

1. **Python script for moving data from Postgresql to Redshift**
   The goal of this task is to generate some random data, store it in Postgresql and than create a Python script for moving the data from Postgresql to Amazon Redshift (https://aws.amazon.com/documentation/redshift/). Postgresql needs to be installed locally by yourself. The tasks to do are:
   a. Create a Python script that generates some random data and stores it into a Postgresql table with the following schema:
      - Table name: `apps`
      - Columns:
        - `pk`: Primary key (integer)
        - `id`: A unique identifier of the app on the app market (varchar(256))
        - `title`: Title of the app (varchar(256))
        - `description`: Description of the app (varchar(2000))
        - `published_timestamp`: Timestamp when the app was published on the app market (TIMESTAMP)
        - `last_update_timestamp`: Timestamp when the app was last updated on the app market (TIMESTAMP)
      - The script should generate 5 million random rows.

   b. Create a python script or a shell script which dumps the data from the table `apps` in Postgresql to a CSV file with GZIP compression.

   c. Create an account on Amazon AWS (https://aws.amazon.com/) and implement a python script for uploading the file created in step b. to Amazon S3.

   d. Create a Python script that loads the data from Amazon S3 to Redshift. (AWS

provides free trial for both S3 and Redshift).

DELIVERABLES:

- SQL scripts to generate the table on Postgresql and Redshift
- Python script which allows us to generate the random source data (part a.)
- Python or shell script that allows to dump the data from Postgresql to a gzipped CSV file
- Python script for uploading the gzipped CSV to Amazon S3
- Python script which allows to load the data from Amazon S3 to Redshift
- Documentation that explains how to use the two python scripts

2. **Word count in Python streaming or PySpark**

The goal of this task is to write map-reduce job in Hadoop Streaming using python (http://hadoop.apache.org/docs/stable/hadoop-streaming/HadoopStreaming.html), i.e. the mapper and reducer have to be written in Python. Note that the Hadoop Streaming map-reduce job can be tested on a local Hadoop cluster or by faking it with the following shell command:

```
cat dataset.txt | python your_mapper.py | sort -k1,1 | python your_reducer.py
```

Alternatively you can also use PySpark if you prefer.

The tasks to do are:

a. Download the data set over which to run word count from the following link: https://s3.amazonaws.com/products-42matters/test/biographies.list.gz b. Implement a mapper and reducer in python that counts the number of occurrences of each word in the provided file. Only lines starting with the “BG:” should be considered, and a whitespace tokenizer should be used for tokenizing the text.

DELIVERABLES:

- Mapper and reducer written in Python as two separate python script files or PySpark script
- The result of the word count
- Documentation that explains how to use the map-reduce job

3. **Rows deduplication using SQL**

In this task you are required to write an SQL query to find the most recent version of each app. The tasks to do are the following:

a. Write the SQL code for creating the following table `apps`:

pk	id	title	rating	last_update_date
1	com.facebook.katana	Facebook	4.0	2016-09-12
2	com.whatsapp	WhatsApp	4.5	2016-09-11

3	com.whatsapp	WhatsApp	4.4	2016-09-12
4	com.nianticlabs.pokemongo	Pokémon GO	4.6	2016-09-05

5	com.nianticlabs.pokemongo	Pokémon GO	4.3	2016-09-06
6	com.nianticlabs.pokemongo	Pokémon GO	4.1	2016-09-07

b. Write an SQL query which for each app returns one row, i.e. the row with the most recent `last_update_date`. (e.g. for WhatsApp it is the row with pk = 3), i.e. the rows with `pk` equal to 1, 3 and 6 should be returned by the query and all the fields (i.e. `pk`, `id`, `title`, `rating`, and `last_update_date`).

DELIVERABLES:

- SQL script for creating the table above (part a.)
- SQL query for finding the most recent version of each app (part b.). Explain why this solution is good and what is its space complexity

4. Convert JSON data from one structure to another (Optional)

In this task you are asked to convert data from one JSON structure to another by using command line tools and the solution must be 1-liner. The input data can be found here - https://s3.amazonaws.com/external.42matters.com/1/42apps/v0.2/sample/playstore/top/latest/top-charts-playstore-daily-04.tar.gz

It consists of newline-delimited json Top Charts objects, which contain a list of apps (their identifiers are called package_name), whose order shows the position of the app in the respective top charts.


```
{
  "list_name": "movers_shakers",
  "country": "US",
  "cat_key": "OVERALL",
  "date": "24-09-2019",
  "app_list": [
    {
      "package_name": "com.handmark.sportcaster"
    },
    {
      "package_name": "com.gotv.nflgamecenter.us.lite"
    },
    {
      "package_name": "com.target.ui"
    },
```

```
    {
      "package_name": "com.SPSoftwareProductions.TheRealJuggle"
    },
    {
      "package_name": "com.yahoo.mobile.client.android.sportacular"
    }],
"category_name": "Overall"
}
```


Your task is to convert all top charts to app ranks, where each line is an app, together with its rank and the additional attributes found in the top chart json object renamed accordingly. E.g.

{"p":"com.handmark.sportcaster","r":0,"c":"US","l":"movers_shakers","cat":"OVERALL","ts":"24-09-2019"}
{"p":"com.gotv.nflgamecenter.us.lite","r":1,"c":"US","l":"movers_shakers","cat":"OVERALL","ts":"24-09-2019"}
{"p":"com.target.ui","r":2,"c":"US","l":"movers_shakers","cat":"OVERALL","ts":"24-09-2019"}
{"p":"com.SPSoftwareProductions.TheRealJuggle","r":3,"c":"US","l":"movers_shakers","cat":"OVERALL","ts":"24-09-2019"}
{"p":"com.yahoo.mobile.client.android.sportacular","r":4,"c":"US","l":"movers_shakers","cat":"OVERALL","ts":"24-09-2019"}

Hint: You can use JQ - https://stedolan.github.io/jq/ to achieve this, but feel free to explore other options.