BriteCore

Data Engineer Test

Eduarda Goulart

This test was implemented using Python, Django, Django-rest, Django-rest-pandas, pandas and it's online on AWS cloud services.

Minas Gerais, Brazil

August, 2019

Contents

1 Introduction

The main purpose of this document is to give an overview of the technologies I used to build a analytics platform for a *Fake Insurance* company using the Kaggle dataset *Agency Performance Model*. Additionally, this document also details some design decisions I made during the implementation of the proposed data pipelines and Representational State Transfer (REST) application programming interface (API) thereof. However, it is worth mentioning that this document is not a replacement for a full-blown documentation. Rather, as mentioned, my goal was to give the interested reader a general understanding of the technologies and elements that comprise my solution to the proposed analytics platform problem. Essentially, the remainder of this document is organized into the following sections:

- Section 2 lists all the project's dependencies.
- Section 3 describes the elements that comprise the proposed solution, i.e., data pipelines and REST API. Specifically, this section outlines usage and functionality.

2 Dependencies

The proposed solution taps into several well-established Python libraries and frameworks:

- Pandas
- Django 2.2
- Django-Rest-Framework
- Django-Rest-Pandas

Also, it is worth mentioning that the proposed solution runs on Python 3.6. Installation and troubleshooting of the aforementioned dependencies is not within the scope of this document.

3 Proposed Solution Implementation

3.1 Data Pipelines Overview

All the data pipeline related operations are included in the `csv_to_database.py` file. There is no prerequisite for running this data pipeline (apart from having Python

3.6 properly installed). To run this data pipeline that extracts data from a CSV file and sends it to an sql3 database. To run the data pipeline script:

```
$python3 csv_to_database.py
```

Before sending the extracted data to the database, the script also checks for "standard missing values", that is, missing values that Pandas can detect (i.e., NaN). During this phase, all missing values are removed from the dataset so that no missing values are sent to the database.

Additionally, after saving all data to the database, the automated data pipeline is also able to create a summarized view of the data. The summarized view contains only data I deemed potentially relevant to analysis. Essentially, it is possible to obtain different set of data centered around *losses* and *gains*. Specifically, it is possible to filter/summarize data according to the following fields:

- *Agency id*;
- *Month*;
- *Year*; and
- *State*.

So, for instance, it is possible to gain insight into the gains and losses in a given year. Similarly, it is also possible to probe into which month the *Fake Insurance* company had more profit.

3.2 REST API

The proposed REST API is broken down into four parts. The first, which is in the initial screen, is named *insurance*. This part lists all data, from all agencies: basically, this part lists all information in the Kaggle dataset.

The second part is centered around providing summarized views. This second part also appears in the initial screen. Through this part of the API it is possible to query for data that matches a given criteria. To do so, the user needs to write `'/search/?query=value'`. where query should be replaced by a valid column name. For instance, to search for all information regarding the fifth month of the year: `'/search/?months=5'`.

The third part of the API makes it possible to filter the data by *month*, *state_abbr*, *agency_id*, or *stat_profile_date_year*, allowing for a more user-friendly form of data filtering. For instance, it is possible to filter the whole dataset according to a valid

agency_id: making it easier to focus on specific information. It is worth mentioning that when an invalid *agency_id* is provided as parameter, the API returns `null`.

The forth part of the API generates reports on demand: all information in reports are filtered by year. This part of the API can be accessed at `'/report/?data=year'`. It is also possible to export the reports' details to CSV files.

4 Difficulties

After I finished implementing pretty much the whole functionality described in the test, I now realize that the most complicated part was to deploy the application to AWS. In fact, at first I adopted PostgreSQL to store all information regarding the Fake Insurance company. However, I faced several technical problems while trying to deploy a version of the application that used PostgreSQL. So, eventually, I gave up on trying fixing these deploy-related problems and settled on using SQLite3. That said, I know SQLite has scalability problems and was not designed to be used in production. (I had to comprise in order to turn in the test in a timely fashion)

For some reason, the report function, which exports all data into a CSV file does not work when the app is hosted on AWS. It works fine when running local though.

http://britecoreproject.xnm6eb7mad.us-west-2.elasticbeanstalk.com/