Price prediction data pipeline

The submission document of a data engineer task challenge

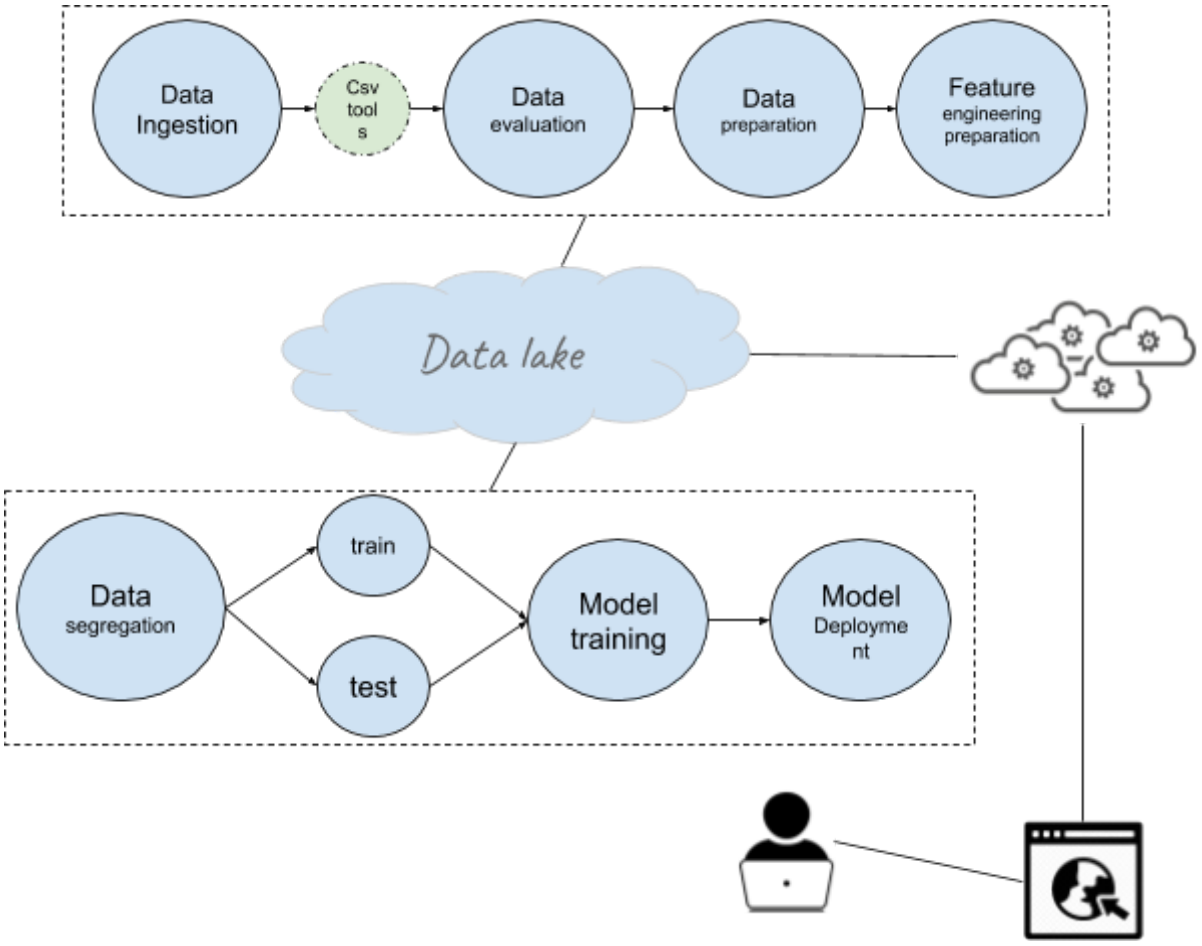
Masoud Allahyari

https://www.linkedin.com/in/masoud-allahyari-b4566556/

Masoud.Allahyari@Gmail.com

Overview

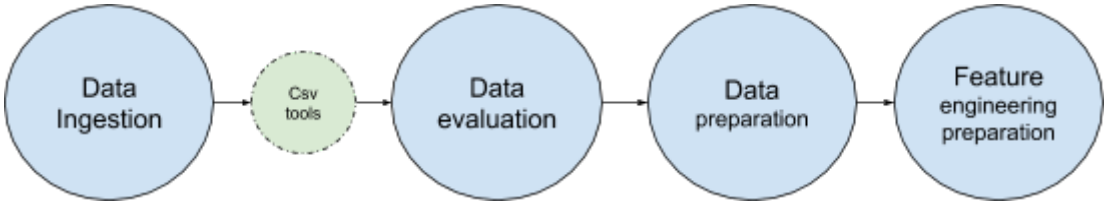
As the goal of this task is data preparation of some data for data scientists or a model. My assumption is that this data are coming continuously , therefore, I designed a data pipeline that triggers daily and process the new data for being used by data scientists. Furthermore, it can evaluate the input data and clean them for further usage. In addition, another pipeline was designed to take the cleansed data generated by the first pipeline and train the model with them.



Pipelines

Two pipelines were designed for this challenge as mentioned before. The first one prepare the data for the model and for other use cases and the other pipeline train the model.

1- ETL Pipeline:



Data ingestion: In this step all the csv files in the provided path will be read and parsed to a spark dataframe

Csv tools: In order to evaluate the number of rows parsed by spark, csv tools is used to compare the row count of input from an external environment.

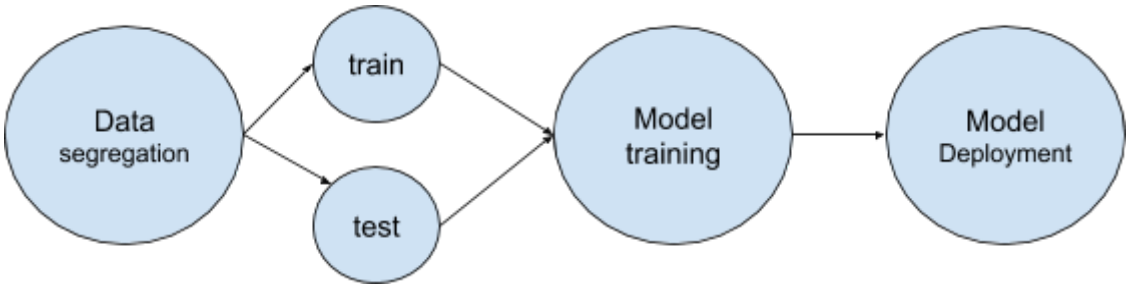
Data evaluation: As the quality of data are very important, this task will evaluate not only the schema but also the values of fields. I explain this part with more detail in the next chapters.

Data preparation: Data will be prepared for not only the model but also other use cases such as Data warehouse or data exploration or analyzation.

Feature engineering data preparation: Data will be prepared specifically for the ML model. For example, the columns which are not required for the model will be dropped or some important fields which contain null, Zero or empty values will be also dropped.

Schedule interval: Daily

2- ML Pipeline:



***Note**: This pipeline is only a demo to present how we can continuously train a model based on the historical data. Therefore, the implementation and available pipeline is much simpler than this.*

Data segregation: All the required ML functions will be applied on the data and will be splitted to train and test

Model training: The model or models serve the test and train. A Metric report will be generated next to the model.

Model deployment: Model will be saved with a new version.

Schedule interval: Weekly or monthly

Main Tasks

Data Ingestion

Class: task.main.DataIngestor

After parsing the data to spark dataframe we should save them to a local or cloud storage. In order to increase the performance I used parquet format.

Data quality

Class: task.main.DataQuality

Row count check: As each line of data is important for us, we need to be sure if pipeline will ingest the whole data and would not skip or forget some lines. Therefore using some external tools, I count the rows number and compare them with the rows number which pipeline will parse and in case of differences task will be failed.

Data structure: Because it is very important to have a reliable structure of data, the new incoming data structures should be checked with what we already have. For that reason the schema has been defined in a json format and been placed in the resources of the project. The ingestion task, after reading and parsing the CSVs to spark dataframe checks the dataframe schema against the already defined one and will reject the ingestion in case of differences in column's types or number of columns.

Data integrity: data integrity can be considered a crucial part of data quality checks. One of the most important areas to check is whether the ingested raw data is correct or not.
There are lots of different methods to check the data quality. In this task I used Deequ which is an open source tool developed and used at Amazon. It is built on top of Apache Spark and measure data quality in large datasets. It can handle the following dimensions: completeness measures, uniqueness looks, Timeliness ensures, validity measures, accuracy measures, consistency evaluates.
In this task only some of the measurements were used to demonstrate this step. To make this part easy for the production development, the fields and their measurements were defined in a config file in json format which is placed under resources folder. For example, the following rules

will check the id values against the null or empty and the other rule means id should not have any negative values:

```
{ "field": "id", "rule": "Complete", "type": "StringType", "extra": ""    },
{ "field": "id", "rule": "NonNegative", "type": "StringType", "extra": ""    }
```

Partitioning strategy

Partitioning will improve scalability and performance for the large scale data. However, the partitioning strategy must be chosen carefully. As we have a daily batch pipeline and also model will use the data for a period of time , the best strategy is to partition the data by date. Most common partitioning case is year/month/day but in order to query this data in the future by some data catalog tools such as Hive or Spectrum or etc., there will be some cases in which this partitioning strategy can not be helpful. Assume that we want to query some data between this date ranges 28-01-2019 to 03-02-2019 then it should be parsed to this query: `select * from table where year=2019 and (month =1 or month =2) and (day >=28 and day <= 03).` The better strategy to solve this issue would be `year/yearMonth/yearMonthDay  year=2018/month=201801/day=20180101.`

Data preparation

Class: task.main.ListingsTablePreprocessor

Regardless of our assumption for the usage of the data, in this task we prepare the data in a reliable and standard format for further uses.

Boolean fields: in the new coming data boolean values are defined by “t” or “f” which this should be converted to true false and column type should be casted to a boolean.

Non values fields: Some fields are holding none which is equivalent to null in spark dataframe and this should be converted to null.

Removing brackets and parentheses: list values are wrapped into brackets, parentheses or curved brackets. Using regex, I remove these and split the values into an array. This can help us in the future to be able query or aggregate these values more optimised.

Trimming big text: \r \n or \t were replaced by [NewLine] and [Tab], to be able to visualize the data frame.

Numeric fields: $ and % were removed from the numeric fields and were casted to Double type.

Model data preparation / Feature engineering

Class: task.main.ListingsModelDataPreparation

This task as I explained before will focus only on preparing data which are useful for the model only. As I focus more on the data engineer parts of the project, I implemented only some parts of feature engineering task here. There are a bunch of other algorithms that one can apply in this task but it is beyond the requirements, time and my skills. For example, applying some NLP algorithm such as Bag of words on the long text fields, we can get some useful information for the model.

Dropping non required fields: Selecting only the required fields which the ML model would consume.

Concatenate the list fields: Since the model can not use list type, these fields concatenated with a comma.

Cast to boolean: Some of the fields are important if they have values, therefore I consider true if they have value and if they don’t I consider fasle for them. For example, picture_url will change to has_picture.

Deleting bad rows: The rows that some of their important fields are holding null, empty or zero will be deleted. For example, price will be considered as the label of our data and it should not be zero or null.

ML Model pipeline

CLASS: task.main.MLListingsModel

My assumption was that we already have a code which can train the model with good accuracy metrics and we want to use it in the prod environment. So, the model algorithm and parameters can be changed over time and will be trained with new data to be always a fresh model. Depends on the configurations, model use the data for a period of time to train itself. Finally, it will save to a new directory with the execution date . As it is already mentioned, this pipeline will be triggered weekly or even monthly. There are lots of tools that we can use for versioning the model or making the model more configurable such as Mlflow or Kuberflow. Using Linear Regression, I implemented a very simple and stupid model only to be able to use it in the pipeline. So please don’t judge the model implementation.

Microservice and restful API

To be able to use the model for prediction, we can implement some microservices for getting input from the user, using the model and sending them the prediction results via a restful API. This part is not implemented.

Testing

There are already some unit testing for two methods but I consider this chapter as future works.

Airflow

I used airflow to design and manage the pipelines. Pipelines code can be found under airflow directory in the root of the project. All the variables can be configurable in Variables section under Admin menu. I used bash operators and spark-submit operators to communicate with the project and xCom objects for communicating between the tasks.

Task dependencies:

Tasks will be executed with the following dependencies:

1- build project and test
2- get_row_numbers, get_files_numbers
3- ingest_data
4- archive_csvs, preprocessing_data
5- preprocessing_data
6- model_data_preparation

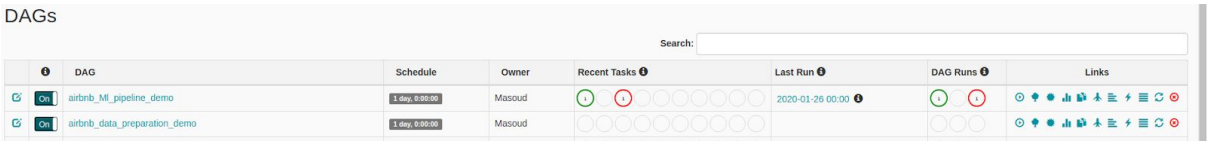
Variables:

Key	***Value***
csv_path	/[root of project directory]/data/airbnb/csvs_files
days_before	2000 day [airflow will find 2000 days before the execution date and send it as start date filter for training the model]
extra_jars	/[root of project directory]/extra/deequ-1.0.2.jar
model_data_path	/[root of project directory]/data/airbnb//model_data
model_path	/[root of project directory]/data/airbnb//ml_model
preproccessed_path	/[root of project directory]/data/airbnb//preprocessed
project_file_path	/[root of project directory]/extra/testproject.jar

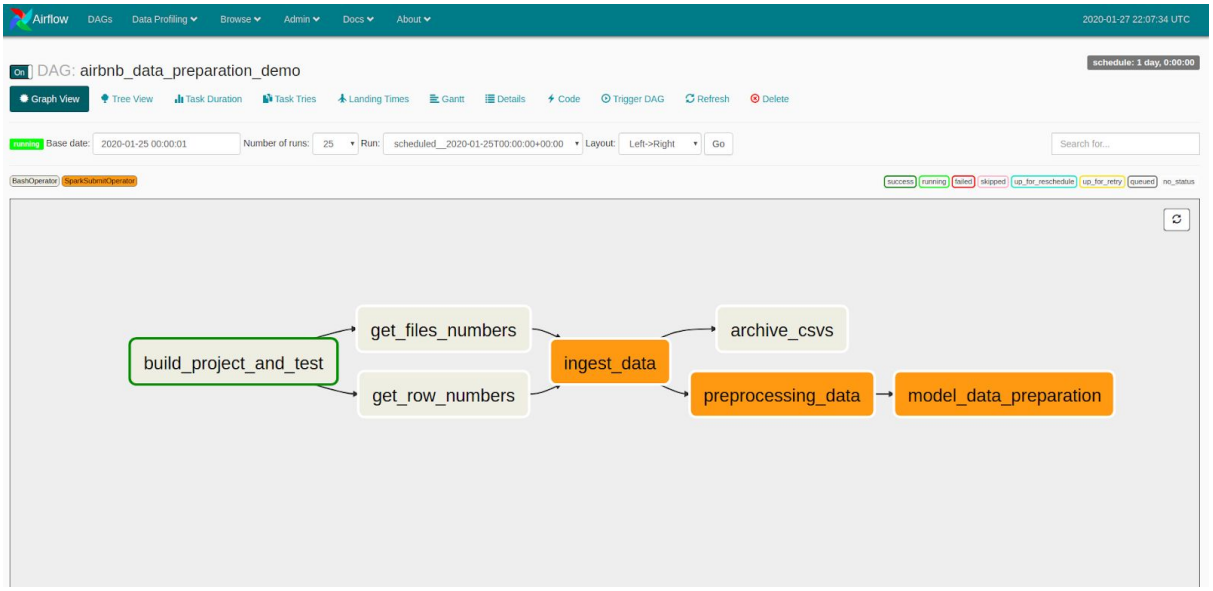
project_path	/[root of project directory]/
run_date	all
spark_config	--master local[*] --jars extra/*.jar
staging_path	/[root of project directory]/data/airbnb//staging
table_name	listings

ScreensScreenshots:

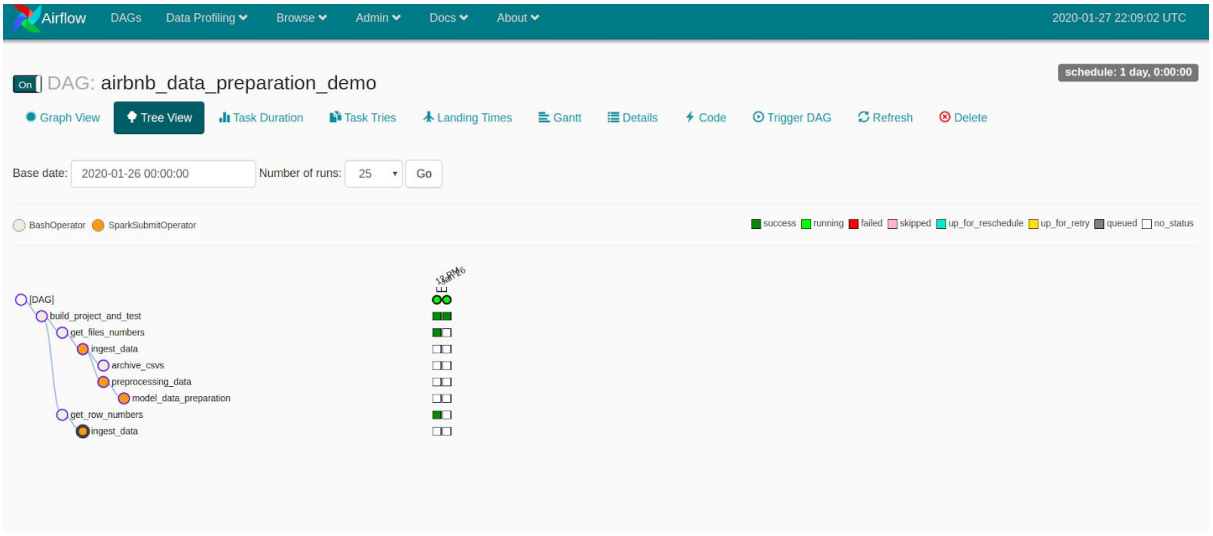
DAGs:



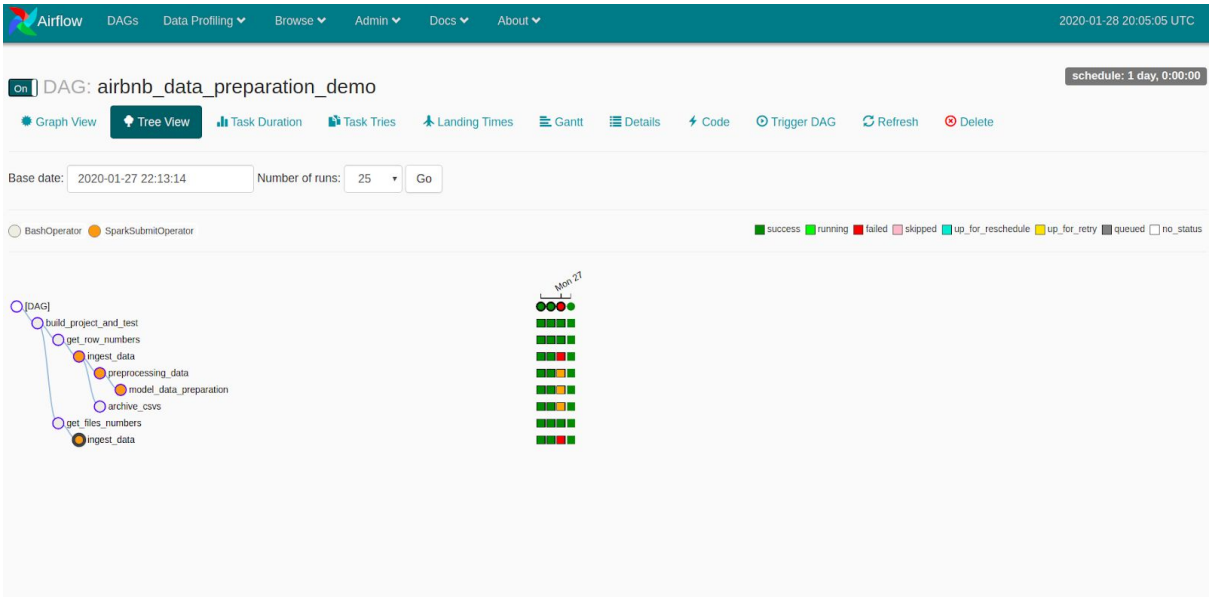
Data preparation dag:



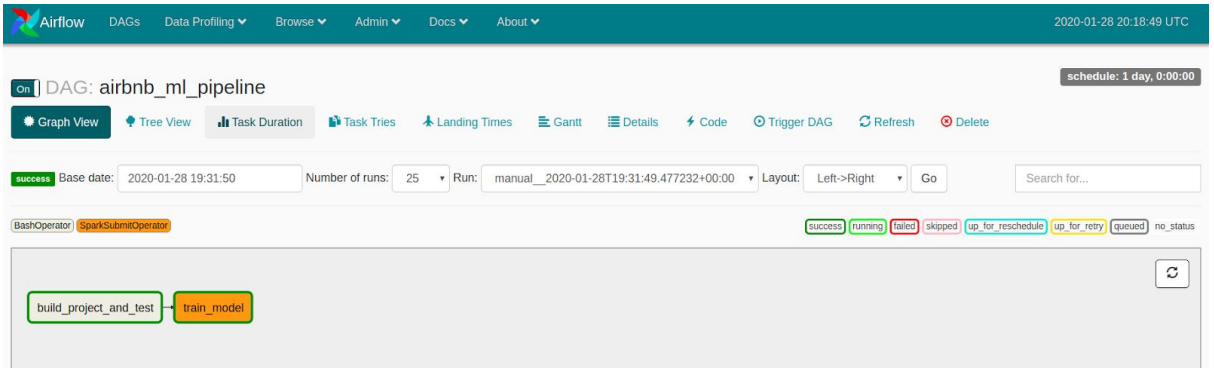
Data preparation tree view:



Showing the completeness of tasks:



Ml pipeline dag:



Ml pipeline treeview: