Python, Pandas

Hugo MATH, Clément PRUNOT

ESIREM 4A IT ILC

https://github.com/Mathugo/f1-1950-2021-datascience

Table of Contents

Présentation des technologies utilisées

Backend

Python 3

Nous avons choisi Python pour sa facilité de mise en place ainsi que sa comptabilité avec un bon nombre de packages dédiés pour le traitement de données. La version 3 rajoute des syntaxes supplémentaires pour avoir un code plus propre ainsi que la comptabilité avec nos packages pip. Python est spécialement utile pour notre **ETL**, en effet nous avons créé une classe *ETL* permettant de télécharger notre dataset depuis kaggle, ensuite de le transformer en jetant les tables et colonnes inutiles et enfin de push le résultat dans notre serveur Postgresql. L'utilisation d'un logiciel ETL nous semblait inutile vu la simplicité d'extraction et de traitement de données. Dans une optique de production, on pourra très facilement déployer notre script sur n'importe quelle plateforme.

Pandas

Pandas nous sert essentiellement pour le traitement de nos données sous formes de *dataframe* pandas. Celle-ci pourra transiter efficacement dans notre base de données avec **sqlalchemy**. Grace à de simple fonction comme to_sql() de **pandas**, on pourra charger nos fichiers csv dans **Postgresql**

Sqlalchemy

Cette librairie servira de lien entre pandas et Postgres, nous permettant de communiquer avec notre sgbd et d'y exécuter nos requêtes.

Postgresql

Postgresql nous permet un temps raisonnable de traitement de nos requêtes (très largement suffisant dans notre cas). De plus, Postgresql est très utilisé dans des contextes de datamining et data warehouse ce qui est notre cas. Il y est d'ailleurs très facile d'exécuter des requêtes et sous requêtes directement grâce à pg admin. Postgresql dispose notamment d'une grande communauté et d'une forte maintenabilité, ne dépendant pas d'un organisme privé comme Oracle. Sa documentation est facile d'accès et très varié, ce qui facilite grandement la mise en production et le développement.

Frontend

Streamlit

Streamlit est une petite librairie python assez récente permettant de construire des applications data en python directement. Supportant les libraires de charts les plus connus (Plotly, Altair, Matplotlib, Bokeh ..) elle permet une mise en production rapide sans passer par du css, html et js pour construire l'interface graphique.

Altair

Altair est un libraire de graphiques sous python. Nous l'avons choisi pour sa grande documentation et sa compatibilité avec Streamlit. On peut notamment montrer des relations entre données à l'aide de nœuds mais aussi des graphiques plus simples comme des BarChart ou encore des PieChart.

ETL et code python

ETL

Comme dit précédemment, **l'ETL** se fait simplement grâce à **python** et **pandas** en quelques lignes de code. Le dataset est prélevé directement sur **kaggle** à l'aide d'un token d'identification et est ensuite chargé en mémoire, traité puis « load » vers le serveur **Postgresql**. Le code source est disponible sur github.

Le script python en charge de **l'ETL** est : **load.py,** il y a les 3 fonctions distinctes du ETL ainsi que la connexion à la base de données **Postgresql** grâce à **sqlalchemy** et à un fichier de configuration **database.ini.**

Data Vizualisation

Du côté de la visualisation, streamlit permet de gérer un bon nombre de chart et la mise en page est plutôt simple. Le fichier **data_viz.py** détaille toute l'interface.

Requêtes SQL

Nous avons créé une classe **Queries** dans le fichier **data_viz.py,** celle-ci contient toutes les requêtes **SQL** de notre programme. Grâce à **pandas**, on peut directement récupérer les résultats des requêtes sous forme de *dataframe* pour ensuite les analyser ou les afficher efficacement sur l'interface.

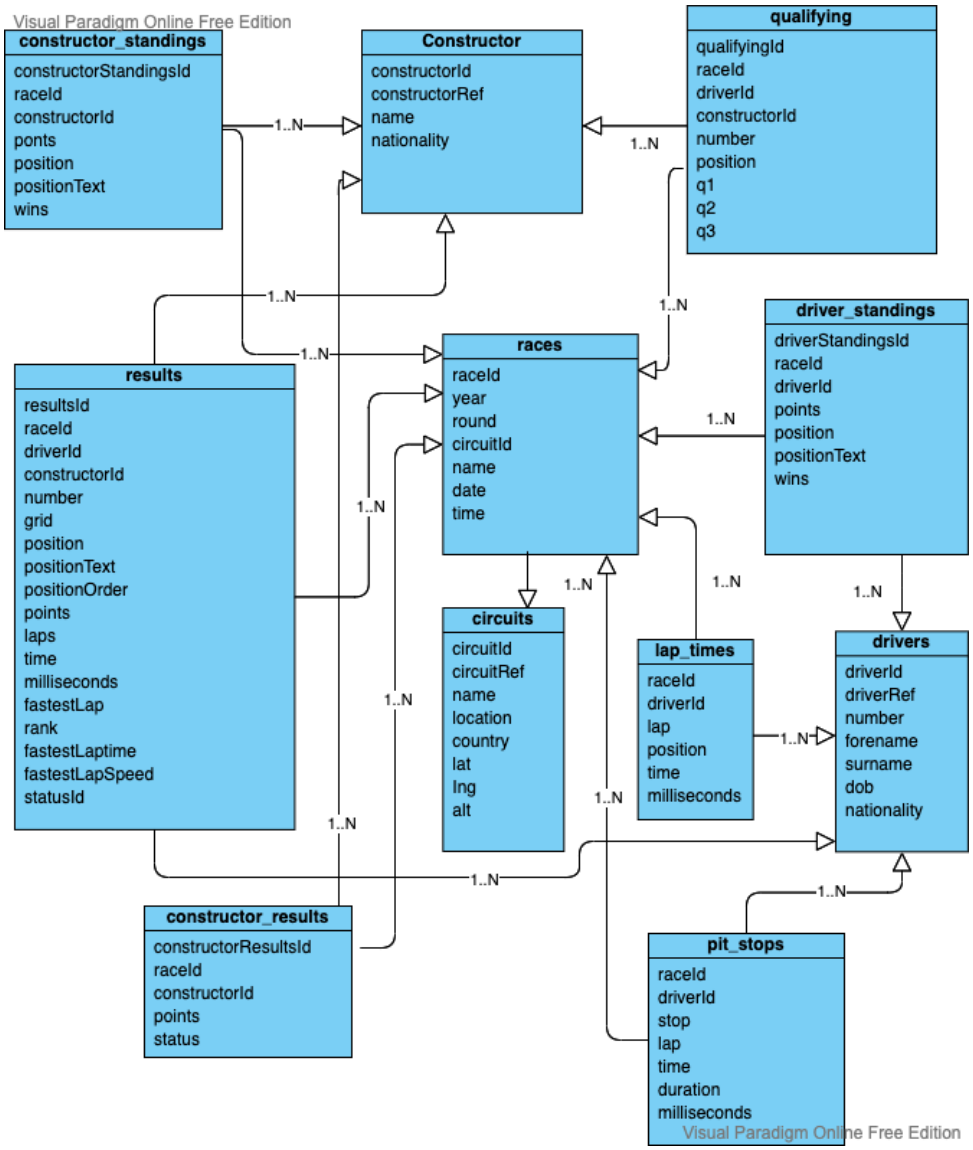
```
"""select * from (
        select count(position) as total_wins, ds."driverId"
        from driver_standings ds where position = 1
        group by ds."driverId"
        order by total_wins desc
        limit 5) as ds
        inner join drivers d ON ds."driverId" = d."driverId"
        """
```

Exemple de requête nous permettant de récupérer le coureur avec le plus de victoires.

Data Lake

Première couche

Juste après l'extraction des données, nous arrivons à ce diagramme UML. On voit qu'il y a beaucoup de clef étrangères et de classes que l'on pourrait surement diminuer sans perdre d'information, et augmenter la clarté de nos tables pour mieux les exploiter par la suite.

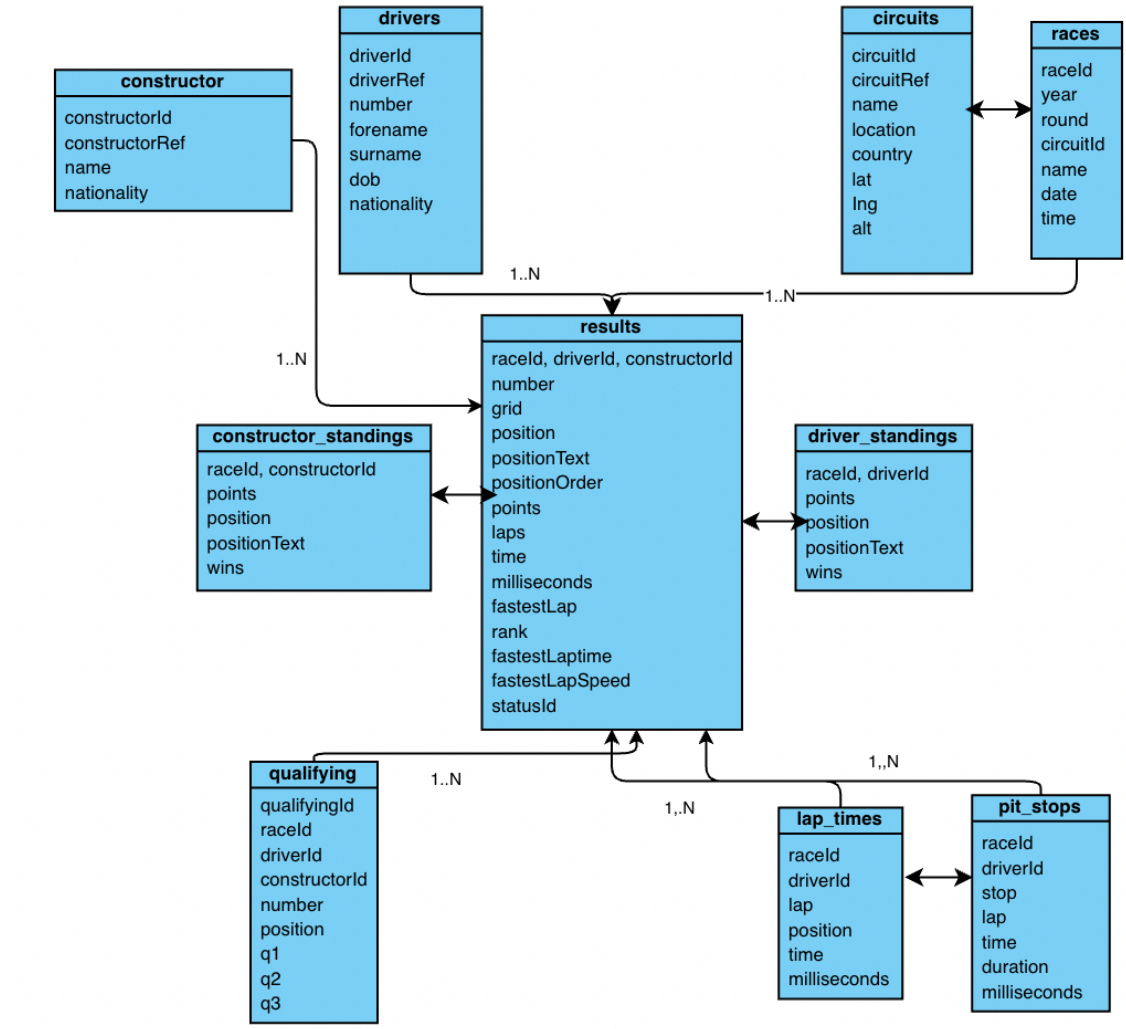


Deuxième couche

Après l'implémentation du paradigme **dimension/fait,** nous obtenons ceci :

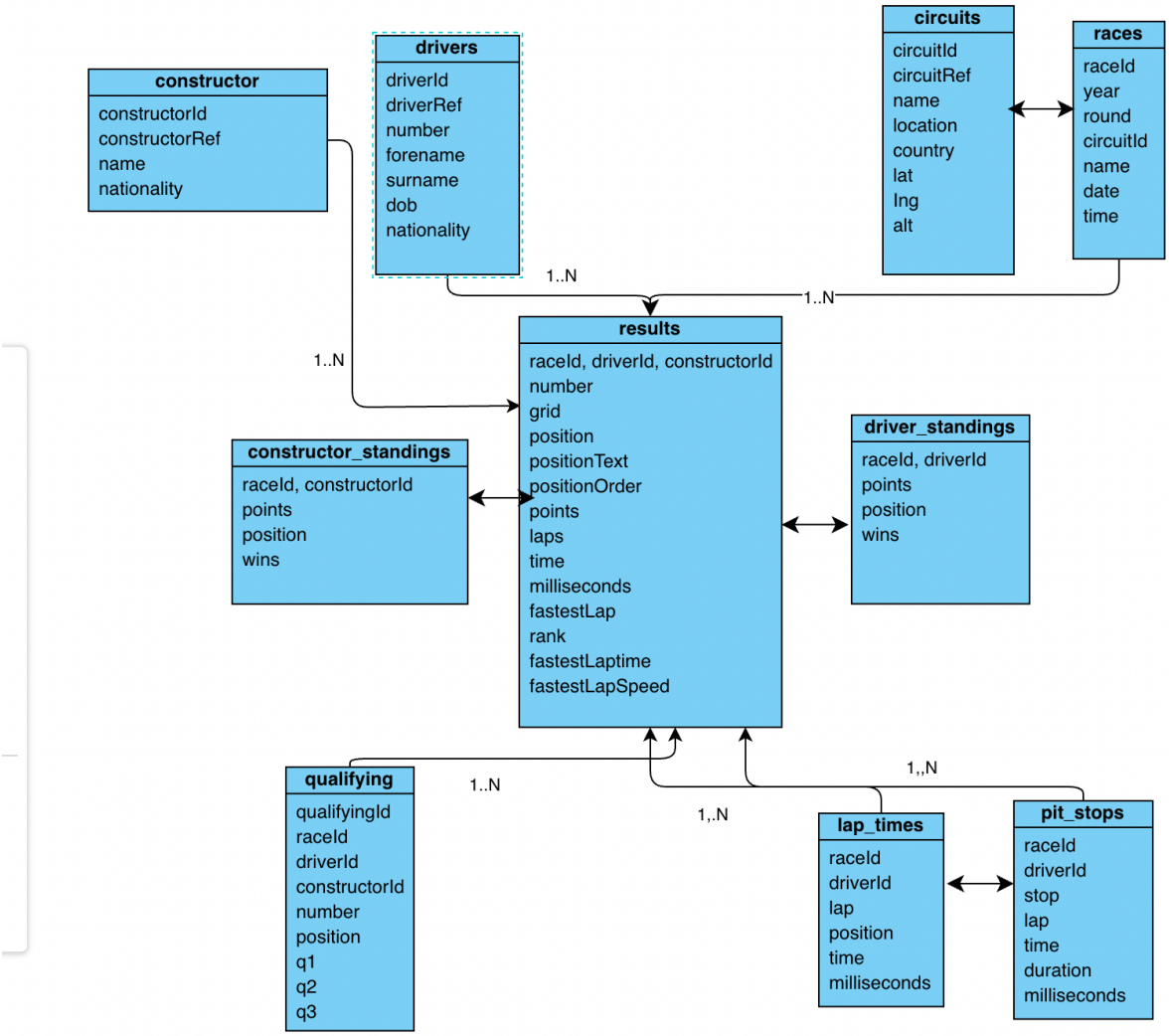
Les classes **drivers, constructor, races, circuits, lap_times, pit_stops, qualifying** sont donc des dimensions tandis que **results, constructor_standings, driver_standings** sont des faits.

On nettoie aussi un peu les tables notamment constructor_results très similaire à constructor_standings. Nos 3 faits se trouvent au milieu, dans l'ordre des tables de dimension-i jusqu'à 4 (pit stops, lap times). Le code transformant nos données est disponible dans la classe **Secondlayer**



Troisième couche

Le but de notre troisième couche est de fournir nos données traitées directement à notre application au format utilisateur et prêtes à l'emploi. Le code est disponible dans la classe **ThirdLayer.** Cette couche sert essentiellement à avoir les bonnes informations pour notre interface, au niveau du code python elle se distingue par des modifications au niveau des tables (suppression de certains attributs, formatage d'autres) elle ne change fondamentalement que très peu comparé à la deuxième.

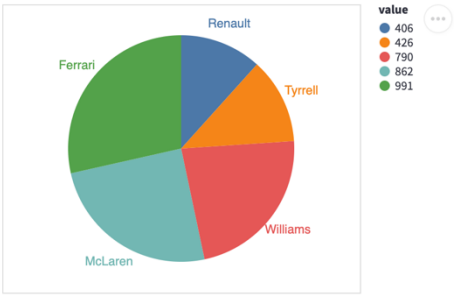


Application Finale

Exemples de requêtes SQL

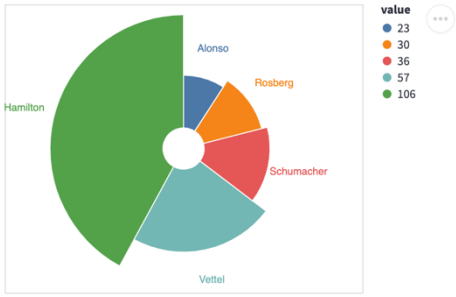
Dominant racing team in f1 history

Without a doubt, the winnner is still ferrari



Top 5 Drivers with the most pole position

Again, Sir Lewis Hamilton ..



Requêtes assez simples avec des graphiques Pie permettant de voir la répartition des victoires de constructeurs depuis 1950 en formule 1 ainsi que les plus grands nombres de pôles (1er en qualification).

Drivers with the most wins of all time

	total_wins	driverRef	forename	surname	dob	nationality
0	125	hamilton	Lewis	Hamilton	1985-01-07	British
1	52	alonso	Fernando	Alonso	1981-07-29	Spanish
2	63	vettel	Sebastian	Vettel	1987-07-03	German
3	121	michael_schumacher	Michael	Schumacher	1969-01-03	German
4	79	prost	Alain	Prost	1955-02-24	French

Une autre requête nous permet de voir le top 5 des coureurs avec le plus grand nombre de victoire de grand prix depuis 1950.

Dû à un manque de temps de notre part et d'organisation pour implémenter tous les graphiques et requêtes, d'autres seront disponible pendant la présentation.

Possible améliorations

Cloud

L'avantage de nos scripts python est qu'ils sont simples à déployer sur n'importe quel services cloud et ainsi créer une pipeline de données, un data warehouse et une application BI efficace en remote.

Nous avons essayé de mettre le projet sur Google Cloud Platform, la connexion à la base de données marchait bien et Streamlit était également en remote. Jusqu'à un moment où nous avons reçu une facture cloud de 367€ (heureusement remboursé quelques jours après).

Analyse statistique & Prédiction

On aurait pu essayer d'appliquer des méthodes statistiques pour détecter des liens de corrélations et de causalité dans nos données. Est-ce que le challenge entre deux pilotes d'une même écurie a un effet de boost sur les performances de l'équipe ? Est-ce qu'un grand prix se gagne aux arrêts de stand ou en conduisant ? Toutes ces questions auraient pu faire l'objet d'une plus grande analyse mais par manque de temps et d'organisation nous n'avons pas pu les réaliser.