Rapport du projet de Data science

Kelun Chai
Oussama Bouzaouit
Djaber Solimani
Nogaye Gning

Année scolaire 2019 / 2020

Table des matières

1 - Introduction

Ce rapport a pour objectif de présenter le travail que nous avons réalisé dans le cadre du projet de science des données de l'UE 17. Nous parlerons d'abord du jeu de données que nous avons choisi. Puis, nous verrons la phase de conception de l'application avant de nous intéresser à la phase de développement. Nous montrerons enfin un aperçu de l'application.

2 - Thème de l'application

Nous avons choisi d'étudier les données de l'épidémie du coronavirus car elle préoccupe tout le monde à l'heure actuelle. Il était facile de trouver des données de bonne qualité. Nous avons utilisé les données de Kaggle. Nous nous intéressons en particulier au nombre de cas confirmés et de décès en France. Notre objectif est de développer un dashboard composé de quelques diagrammes. L'application récupère et nettoiera les données au début avant de les afficher. Il sera également possible de voir des prédictions sur le nombre de cas confirmés et le nombre de décès dans le futur.

3 - Conception de l'application

Nous avons utilisé une architecture Big Data pour stocker et traiter les données, et nous avons choisi de structurer le code de façon modulaire. Il s'agit d'une application client-serveur. Figure 1 montre l'architecture de l'application.

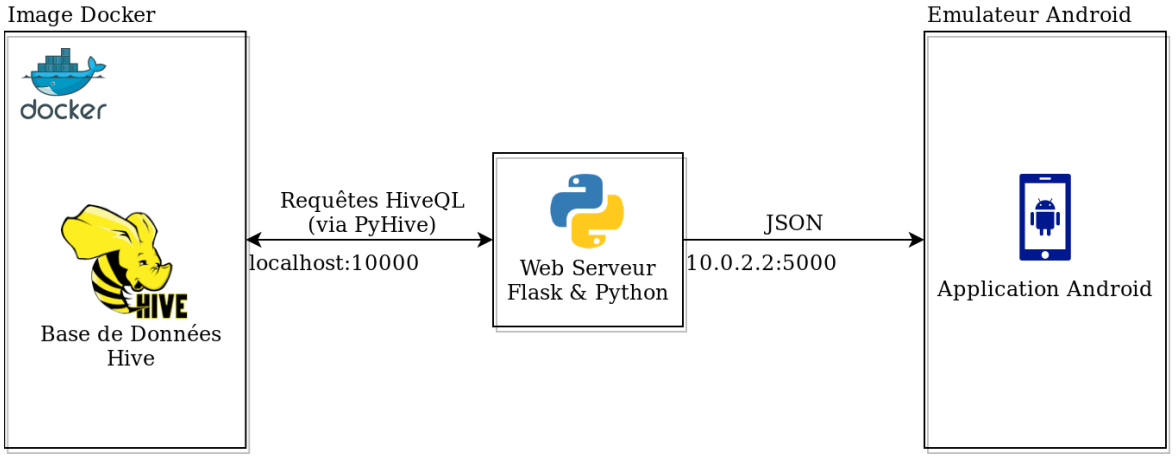


Figure 1 - Architecture de l'application

- Le **client** a été développé en Android, c'est lui qui contient l'interface utilisateur. Il fait des requêtes ajax au serveur web (1 pour chaque graphique à afficher). Il reçoit en retour un objet JSON contenant les données récupérées.
- Le **serveur web** a été développé en python avec l'API Flask. Il fait des requêtes HiveQL au serveur hive via l'API PyHive puis fournit au client les résultats
- Le **serveur hive** contient les données et les fournit au serveur web. Il tourne dans un ensemble de conteneurs lancés avec docker-compose. Nous avons trouvé un fichier de configuration décrivant ces conteneurs sur github, les images Docker utilisées étant publiques.

En ce qui concerne la partie Machine Learning, nous avons utilisé le modèle Python Prophet et ARIMA(**A**uto**r**egressive **I**ntegrated **M**oving **A**verage model), qui est particulièrement efficace pour les séries temporelles comme celles sur le coronavirus.

4 - Développement de l'application

4.1 - Organisation

Tout au long du projet nous nous sommes partagés les tâches pour être plus efficace. Par exemple, pendant que quelqu'un développait des diagrammes dans l'interface graphique quelqu'un d'autre s'occupait des requêtes au serveur web pour récupérer les données à afficher. Nous avons utilisé la plateforme github pour versionner l'application. Pour communiquer nous avons utilisé Discord ainsi que Google meet pour faire des visioconférences. On faisait des points hebdomadaires pour voir ce que chacun a fait durant la semaine, où en est l'état d'avancement du projet et ce que nous devons faire pour la semaine prochaine. En ce qui concerne la rédaction du rapport, tous les membres du groupe y ont participé. Le tableau ci-dessous montre la répartition des tâches au sein du groupe.

BD Hive & Docker	Oussama, Kelun, Djaber
Serveur Python	Djaber, Kelun
Traitement de donnée	Kelun, Nogaye
Prédiction	Kelun
Partie client Android	Oussama, Nogaye

4.2 - Serveur web

Nous avons d'abord développé un serveur web basique en Flask qui renvoie les résultats au format json via une URL spécifique. Ces liens peuvent être modifiés et ajoutés en fonction des besoins du client. Le tableau ci-dessous montre pour chaque URL le type de données demandées par le client.

/deathbyday	Nombre de décès par jour en France (valeur agrégée)
/confbyday	Nombre de cas confirmés par jour en France (valeur agrégée)
/recovbyday	Nombre de personnes guéries par jour en France (valeur agrégée)

/compose/<day>	Situation épidémiologique en France à un jour <day> donné (nombre de cas confirmés, guéries, décès)
/pred_conf	Prévision du nombre de cas confirmés en France dans les 30 prochains jours - Modèle Prophet
/pred_conf_arima	Prévision du nombre de cas confirmés en France dans les 30 prochains jours - Modèle Arima
/pred_death_P	Prévision du nombre de morts en France dans les 30 prochains jours - Modèle Prophet
/pred_death_A	Prévision du nombre de morts en France dans les 30 prochains jours - Modèle Arima

Le serveur web utilise l'API PyHive pour manipuler Hive afin de retourner les résultats de recherche souhaités. Nous utilisons le curseur pour exécuter les requêtes HiveQL, puis nous emballons les valeurs renvoyées au format json et les envoyons au client.
La requête suivante de HiveQL sélectionne la situation de l'épidémie française à une date précise dans la base de données.

```
cursor.execute("SELECT * FROM (SELECT jour, sum(confirm),
sum(recov), sum(death) from covid WHERE country=/"France/" group
by jour ORDER BY jour)a
   WHERE a.jour="%s"" % day)
```

Ensuite, nous avons transformé la valeur envoyé par Hive en JSON.

```
res=cursor.fetchall()
res_json=[{"confirm":row[1],"recov":row[2],"death":row[3]} for
row in res]
```

4.3 - Hive et Nettoyage de données

Avant d'importer les données dans la base de données, nous devons les traiter. Nous avons utilisé le jeu de données Coronavirus 2019-nCoV sur Kaggle(Coronavirus 2019-nCoV), la taille du fichier csv est de 13,2 Mo et contient 212 120 lignes de données.
Nous avons utilisé Python pour remplir les valeurs NaN dans les données et supprimant les données inutiles (par exemple la localisation géographique du pays). Voici une comparaison des tableaux avant et après traitement:

Country/Region	Province/State	Latitude	Longitude
Confirmed	Recovered	Deaths	calendar_today

↓↓↓

Country	Province	Confirmed	Recovered	Deaths	Date

Nous avons ensuite exporté les données traitées sous la forme d'un nouveau fichier csv et l'avons copié sur le serveur Hive dans le conteneur du Docker via une commande système. Ce processus est effectué automatiquement par le script python.

```
os.system(docker cp res.csv
server_hive-server_1:/opt/hive/bin/res.csv)
```

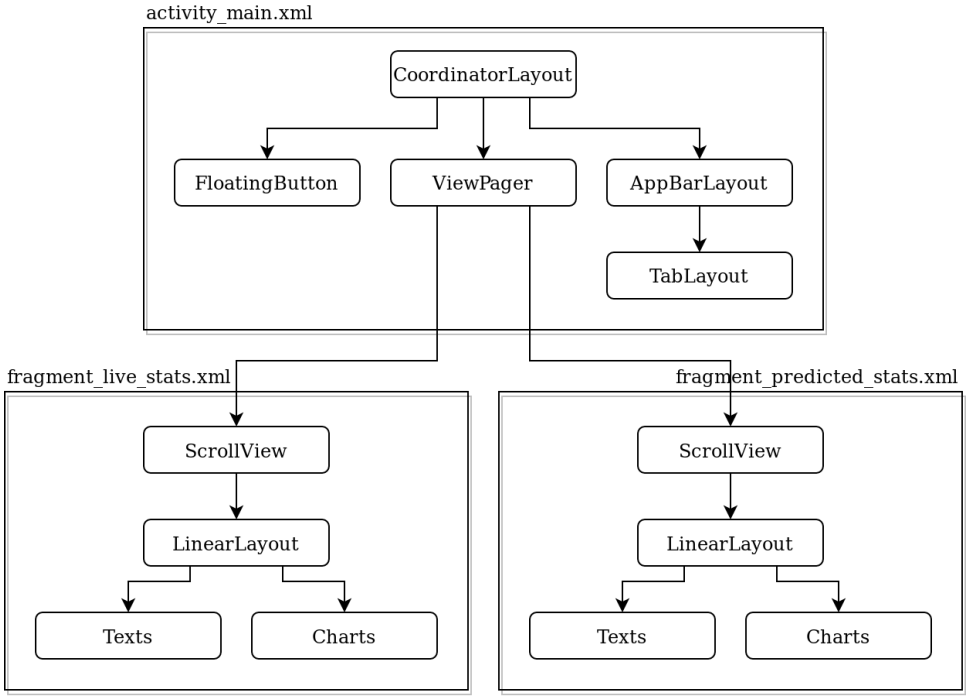
Avant d'importer les données, nous vérifions si le tableau nommé *covid* existe déjà dans la base de données, et si c'est le cas, nous supprimons les anciens tableaux, et en créer de nouveaux. Le format du tableau correspond strictement au format du fichier csv. Ici, nous utilisons le point-virgule comme division et ignorons le nom de l'en-tête de la première ligne.

```
cursor.execute(CREATE TABLE IF NOT EXISTS covid(country
STRING,prov STRING,confirm INT, recov INT, death INT,jour STRING)
       ROW FORMAT DELIMITED
       FIELDS TERMINATED BY ";"
       tblproperties("skip.header.line.count"="1"))
```

4.4 - Client Android

4.4.1 - Structure de l'interface graphique

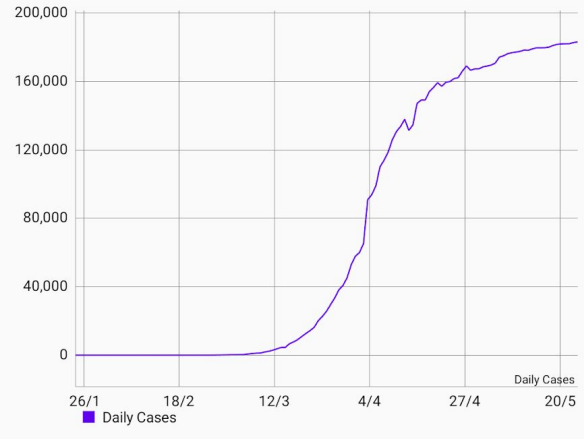
L'interface client de l'application est composée de plusieurs éléments graphiques, générés depuis 3 fichiers XML, et structurés selon la hiérarchie suivante :



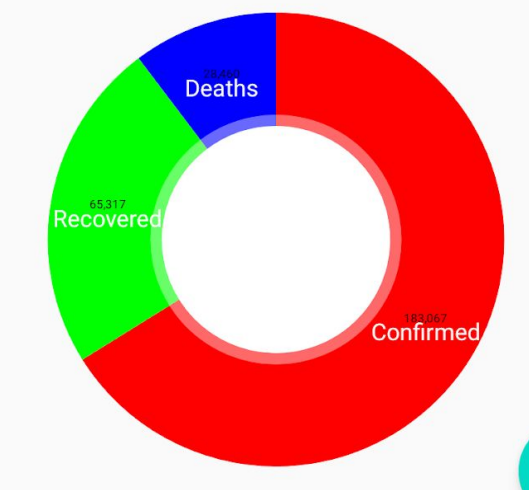
Cette hiérarchie imbriquée permet d'inclure plusieurs graphiques en une seule interface, tout en offrant une navigation conviviale grâce au glissement de pages (Scrolling) et au changement d'onglets (Tabs).

4.4.2 - MPAndroidChart

La visualisation des statistiques se fait grâce à des graphiques générés par la bibliothèque open source MPAndroidChart : https://github.com/PhilJay/MPAndroidChart
Parmi les graphiques offerts par cette bibliothèque, on a le LineChart, qui est le type de graphique idéal pour la représentation d'une série temporelle, il permet de mieux comprendre l'évolution d'un phénomène dans le temps.

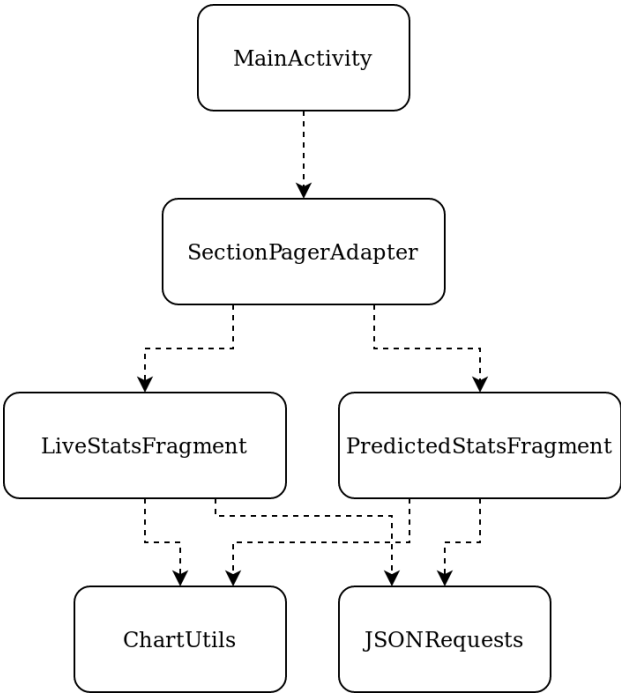


Pour la comparaisons de plusieurs propriétés entre eux, le PieChart est à son tour le meilleur type de graphiques dans ce cas.



4.4.3 - Diagramme de classes

La partie client de l'application est codé en Java, l'ensemble du code est réparti sur plusieurs classes qui forment l'architecture suivante :



MainActivity.java : La classe principale, qui lance l'application au début, instancie les éléments principaux de l'interface graphique, et enregistre les callbacks nécessaires.

SectionPagerAdapter.java : Cette classe permet de structurer l'interface sous forme d'onglets "Tabs", et gère le contenu de chaque onglet, en insérant dans le ViewPager les Fragments : fragment_live_stats.xml et fragment_predicted_stats.xml.

LiveStatsFragment.java : Cette classe gère le contenu de l'onglet affichant les statistiques réelles du COVID-19. à la création de cet onglet, ses différents graphiques sont initialisés à partir des données issues de la base de données Hive, et ce en envoyant au web serveur une requête HTTP Get et en parsant la réponse en JSON.

PredictedStatsFragment.java : Cette classe fait la meme chose que la classe précédente, mais pour l'onglet des statistiques prédites par les méthodes utilisées dans le serveur.

ChartUtils.java : Cette classe contient des fonctions fréquemment utilisées par LiveStatsFragment et PredictedStatsFragment, qui servent à factoriser le code et à simplifier la manipulation des éléments graphiques de la bibliothèque MPAndroidChart. Cette bibliothèque n'utilise pas directement les types de bases Java pour tracer les graphiques, mais elle a ses types propres à elle, la fonction *ChartUtils.getLineChartData(json, date_column, value_column)* construit à partir d'un objet JSON, une liste de Entry qui peut être utilisée pour remplir le contenu d'un LineChart. La fonction

ChartUtils.refreshLineChart(activity, line_chart, json, …) permet à son tour de prendre un élément graphiques de type LineChart et un objet JSON, et remplit le LineChart par les données contenus dans ce JSON.

JSONRequests.java : Cette classe permet de requêter un URL en HTTP GET et retourner la réponse sous forme de String. La requête pouvant prendre longtemps, de ce fait, l'attente de réponse est faite dans un nouveau Thread.
L'utilisation de cette classe est simple, il suffit de l'instancier : *X = new JSONRequests();* et de lire la réponse de la requête comme suit : *String result = X.execute("URL").get();*

4.5 - Prédiction

Pour la partie prédiction, nous n'avons sélectionné que les données concernant la France.

4.5.1 - ARIMA

ARIMA peut être utilisé pour prévoir des séries temporelles. L'idée du modèle est d'entraîner le modèle à partir de données historiques, et une fois appris, d'utiliser ce modèle pour prédire l'avenir.
ARIMA(p, d, q) se compose de trois éléments.

AR(p): AR est autorégressif, c'est-à-dire une régression dans laquelle la valeur présente est égale à la valeur à plusieurs moments dans le passé. p indique une dépendance à la p valeur historique la plus récente dans le passé.
En général, les variables des séries temporelles sont corrélées dans le temps. Par exemple, la vitesse d'une voiture, lorsque l'intervalle de temps est suffisamment petit, si la vitesse au moment précédent est lente, le moment suivant a également tendance à être assez lentement.

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \ldots + \phi_p y_{t-p} + \varepsilon_t$$

$$c : valeur\ initiale, \phi : Autocorrélation$$

$$E_{\varepsilon_t} = 0, Var(\varepsilon_t) = \sigma_\varepsilon^2, E(\varepsilon_t \varepsilon_s) = 0, \varepsilon_t \neq \varepsilon_s$$

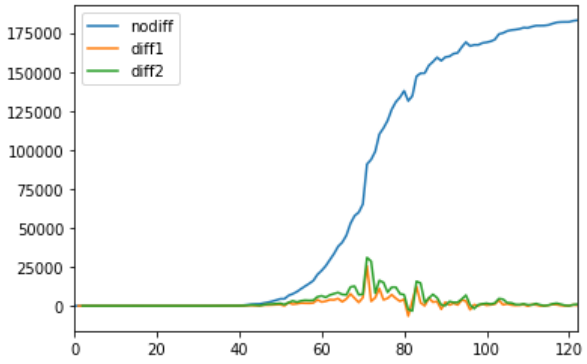
εt est un bruit en distribution normale avec moyenne=0 et variance=1.
Nous définissons ici **p=2**, c'est-à-dire que la prédiction est basée sur deux valeurs historiques.

I(d): I est integrated. L'analyse des séries temporelles nécessite des données stables, et les séquences instables doivent être transformées en séquences stables par calcul différentiel. Ici nous choisissons **d=2**.

$$d = 0, y_t = Y_t$$

$$d = 1, y_t = Y_t - Y_{t-1}$$

$$d = 2, y_t = (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2})$$



Si nous choisissons d=1, le modèle indiquera que les données ne sont pas assez stables.

```
ValueError: The computed initial AR coefficients are not stationary
You should induce stationarity, choose a different model order, or you can
pass your own start_params.
```

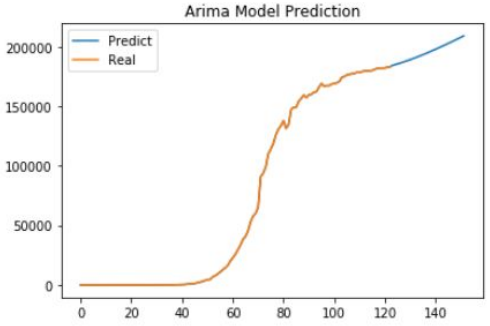
MA(q): MA est moving average, la valeur de q est le nombre de décalées de l'erreur de prédiction. Le modèle MA utilise des erreurs de prédiction historiques pour créer un modèle de régression.

Erreur de prédiction = valeur de prédiction du modèle - valeur réelle.

Nous définissons ici **q=1**.

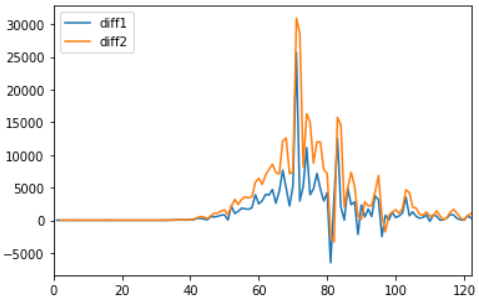
$$Y_t = \epsilon_t + \beta Y_{t-1} - \beta^2 Y_{t-2} + \beta^3 Y_{t-3} - \beta^4 Y_{t-4} + \ldots$$

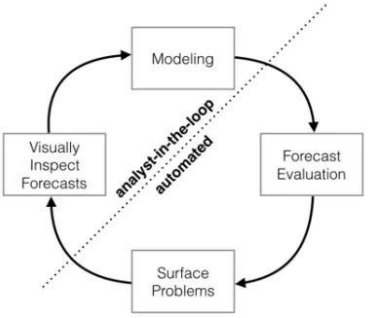
$$Y_t = \epsilon_t + \beta \epsilon_{t-1}$$



4.5.2 - Prophet

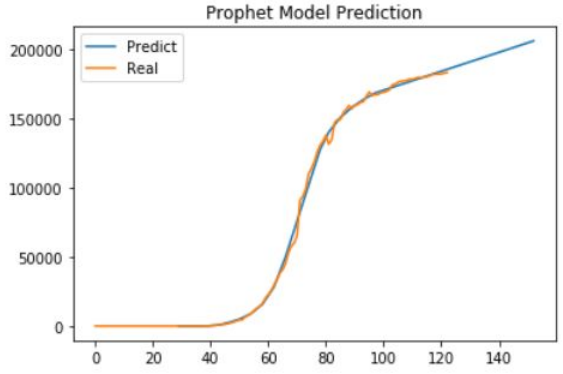
Le modèle ARIMA est imparfait et exige que les données temporelles soient stables. Comme nous pouvons le constater, même après différenciation, les données sur le nombre de cas confirmés ne sont pas encore assez stables.

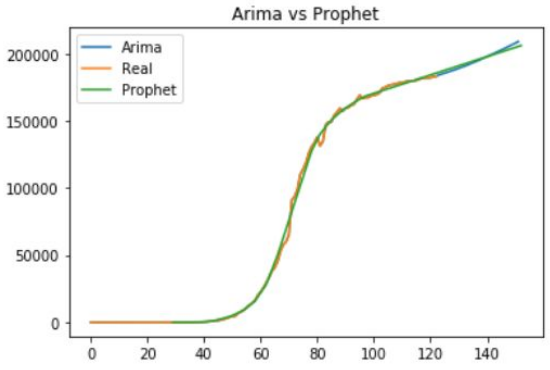




Nous utilisons le modèle Prophet par Facebook. L'ensemble du processus est divisé en quatre parties: Modeling, Forecast Evaluation, Surface Problems, Visually Inspect Forecasts.

L'algorithme Prophet divise la série temporelle en termes de saison, de tendance et de résidu. Nous avons entraîné le modèle en utilisant que les termes de tendance et avons produit des prévisions pour les 30 prochains jours.





En raison de la quantité de données, nous n'avons pas vu de différences significatives entre les deux algorithmes.

5 - Difficultés rencontrées

5.1 - Dataset

Au début, nous avons choisi les données officielles de Santé Publique et les mettre à jour en utilisant le script python. Mais nous avons constaté que les données ne sont pas suffisamment claires: le nombre d'hospitalisations ne représente pas précisément le nombre de cas confirmées. En outre, certaines des données sont cumulatives et d'autres ne le sont pas. Le traitement de ces données est complexe.
Finalement, nous avons choisi le jeu de données de Johns Hopkins sur Kaggle, qui contient des données sur les épidémies du monde entier. Bien que nous devions également traiter les données, par exemple en complétant les valeurs NaN, cet ensemble de données est plus facile à utiliser et plus précis.
En raison de la limitation de l'API du Kaggle,nous n'avons pas pu mettre à jour les données automatiquement en utilisant une méthode générique. D'autre part, comme les anciennes et les nouvelles données du modèle de prédiction de ARIMA doivent être concaténé manuellement, nous n'utilisons pas de mise à jour automatique des données.

5.2 - Conteneurisation de l’application

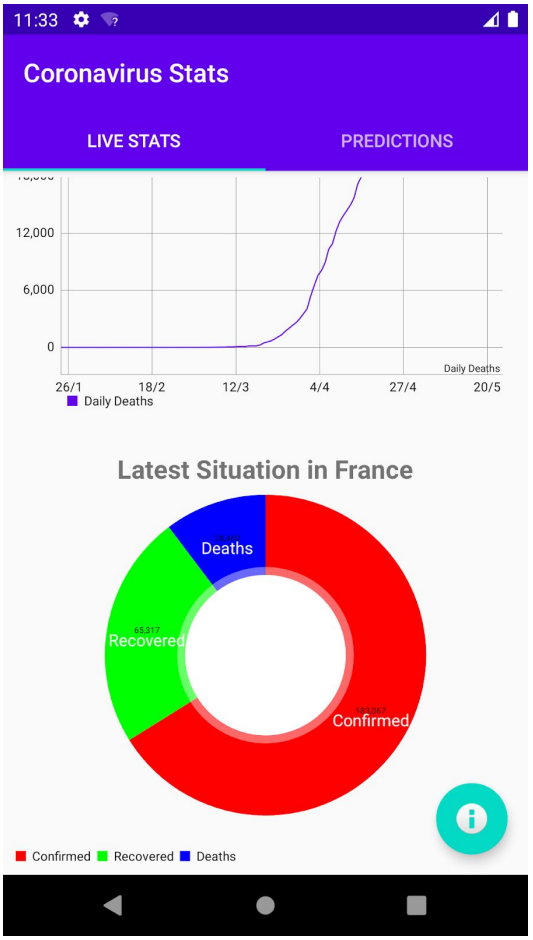
Afin de pouvoir déployer plus facilement l’application nous voulions la conteneuriser dans sa globalité avec Docker. Cependant, nous n’avons pas réussi à intégrer le serveur hive dans une nouvelle image Docker, nous n’avons pas non plus réussi à rajouter de nouveaux conteneurs au fichier de configuration. Pour utiliser l’application, il faut donc installer soi-même les dépendances sur sa machine. Nous avons malgré tout détaillé dans un README les quelques commandes à exécuter pour installer les dépendances et démarrer l’application.

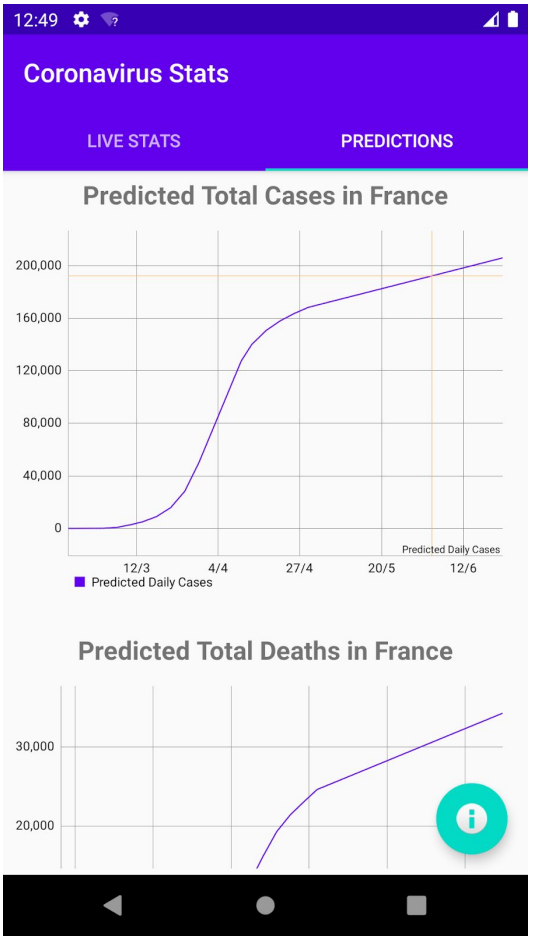
5.3 - Modèle de prédiction

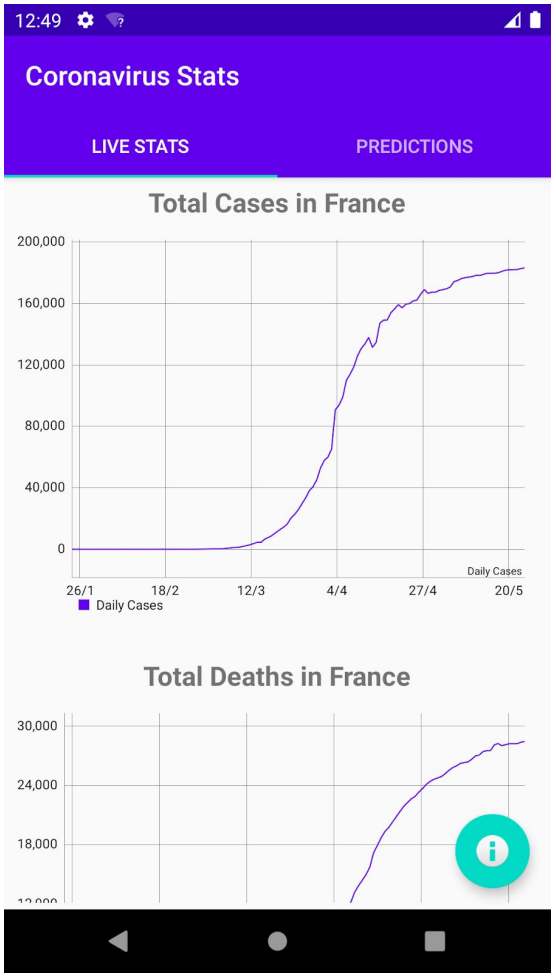
Un des objectifs du projet était de créer un modèle de machine learning afin de prédire les données futures. En ce qui concerne la prévision des épidémies, le modèle le plus applicable actuellement est le modèle SIR et ses variantes (SEIR), auquel on peut ajouter des variables telles que le confinement, le dé-confinement, etc.
Comme nous avons déjà utilisé ce type de modèle dans l'UE20, nous utilisons ici un modèle de prédiction temporelle: le Prophet de Facebook et le modèle de régression ARIMA. Etant donné que notre dataset n’est pas très volumineux (un fichier csv de 212120 lignes), cela complique l’apprentissage du modèle bien qu’elles soient de bonne qualité.

6 - Présentation de l’application

L’interface graphique de notre application contient principalement deux onglets.Un onglet LIVE STATS qui nous permets de visualiser nos données et un onglet PRÉDICTIONS qui nous permet de tracer nos courbes de prédictions.
Sous LIVE STATS, nous avons tracé une courbe représentant l’évolution du nombre de mort en France et une une autre courbes représentant l’évolution du nom de cas confirmés en France.Pour mieux visualiser ces statistiques sur un jour donné, nous avons également tracé un diagramme circulaire qui montre le nombre de cas confirmé, le nombre de mort ainsi que le nombre de guéris pour ce jour.
Deux courbes résultant de l’application de nos modèles de prédiction avec Prophet et ARIMA sont tracées sous PREDICTIONS. Une pour représenter la prédiction du nombre de cas en France et une autre pour prédire le nombre de mort en France.







Vous pouvez actualiser les données en cliquant sur le bouton dans le coin inférieur droit.

7 - Conclusion

Ce projet a été très enrichissant dans l'ensemble. Il nous a permis de mettre en application les compétences acquises à l'université et dans nos entreprises, dans le cadre d'un projet concret de data science avec des données réelles. Nous sommes satisfait de la manière dont nous avons géré le projet, de la phase de conception à la phase de tests. Nous nous sommes familiarisés avec le travail en équipe même si les conditions n'étaient pas idéales car nous travaillions à distance à cause de la crise sanitaire. Une communication régulière était très importante pour avancer efficacement. L'application que nous avons développée est une vraie "application Big Data" qui répond parfaitement aux spécifications du projet.