Projet de l’équipe D pour le cours de Big Data au seins de l’UPEC : Flight Catcher

Groupe D

Massil Kouchit   
massil.kouchit@etu.u-pec.fr

Yanis Ben Aied  
yanis.ben-aied@etu.u-pec.frRayane Slimani  
aissa-rayane.slimani@etu.u-pec.fr

Rayan Dansou  
yao-rayan.dansou@etu.u-pec.frAito Meignan  
aito-florentin.meignan@etu.u-pec.fr

# Introduction

## Contexte et enjeux du transport aérien moderne

Dans un monde de plus en plus interconnecté, le transport aérien occupe une place centrale, permettant de relier les continents et de faciliter les échanges économiques, culturels et humains. Cependant, cette importance s'accompagne de défis significatifs pour les compagnies aériennes et les gestionnaires d’aéroports, notamment la gestion des retards de vols, qui représente une source majeure de perturbations pour les passagers et les opérations. Ces retards, souvent causés par des facteurs tels que les conditions météorologiques, les contraintes de gestion du trafic aérien ou des problèmes techniques, engendrent des coûts économiques et sociaux conséquents.

Face à ces enjeux, le développement de solutions basées sur l’analyse des données aéronautiques devient crucial. C’est dans ce contexte que s’inscrit le projet **Flight Catcher**, qui exploite le jeu de données **Flight Status Prediction** de Kaggle. L’objectif est de concevoir une interface utilisateur intuitive et performante, capable de traiter et de visualiser des données complexes afin d’aider les acteurs du secteur à anticiper et réduire les retards. Les passagers des vols pourront également utiliser l’interface afin de choisir le vol qui leur correspond le plus. Ce projet ambitionne de transformer des données brutes en informations exploitables pour optimiser la prise de décision et améliorer l'expérience utilisateur dans le domaine du transport aérien.

## Structure du document

Le présent rapport s’articule autour de plusieurs sections détaillant chaque étape clé de notre projet. La **section II** introduit les choix techniques initiaux, avec une présentation de l'architecture backend basée sur FASTAPI et DuckDB, ainsi que les premières analyses réalisées sur le jeu de données. La **section III** explore le remplacement de DuckDB par Apache Spark, en mettant en lumière les avantages de cette solution pour des analyses à plus grande échelle et des applications de machine learning. La **section IV** se concentre sur le développement du frontend, où l’utilisation de Dash Plotly a permis de concevoir une interface utilisateur interactive et intuitive pour visualiser les données. La **section V** évalue les performances de notre système, notamment en termes de rapidité et de scalabilité, tandis que la **section VI** propose des scénarios d'application concrets illustrant l'utilité de notre interface dans divers contextes. Enfin, nous concluons avec une synthèse des résultats obtenus et des perspectives d’amélioration pour ce projet.

## Utilisateurs potentiels

L’interface développée dans le cadre du projet **Flight Catcher** a été conçue pour répondre aux besoins d’une large variété d’utilisateurs, qu’ils soient passagers occasionnels ou professionnels du secteur aérien.

D’une part, les **passagers individuels** constituent un groupe clé d’utilisateurs. Grâce à cette interface, ils pourront accéder à des informations claires et précises sur les vols, comme les prévisions de retards ou les tendances générales pour une destination donnée. Ces données leur permettront de mieux planifier leurs déplacements et de choisir les options les plus fiables pour leurs voyages, optimisant ainsi leur expérience.

D’autre part, les **professionnels du secteur aérien** bénéficieront d’outils puissants pour améliorer leurs opérations. Les **aéroports** pourraient utiliser cette interface pour optimiser la gestion des ressources au sol, telles que les pistes ou les portes d’embarquement, en anticipant les retards. Les **compagnies aériennes**, quant à elles, pourront se servir de ces analyses pour ajuster leurs plannings, réduire les impacts des retards sur les connexions, et améliorer la satisfaction de leurs clients. En outre, les **organismes de régulation** ou les prestataires de services liés au transport aérien (comme les gestionnaires de bagages ou les équipes de maintenance) pourraient exploiter ces données pour coordonner leurs activités et réduire les inefficacités opérationnelles.

Enfin, cette interface s’adresse également à l’ensemble des **acteurs de l’écosystème aéronautique**, tels que les analystes de données ou les décideurs stratégiques, qui peuvent s’appuyer sur les informations fournies pour identifier des tendances, évaluer les performances et prendre des décisions éclairées visant à améliorer la fiabilité et l’efficacité globale du transport aérien.

En rendant les données accessibles et exploitables pour ces divers profils d’utilisateurs, **Flight Catcher** se positionne comme une solution universelle au service de la fluidité et de l’efficacité du transport aérien.

Outre les passagers et les professionnels directs du secteur aérien, l'interface **Flight Catcher** pourrait également intéresser les **investisseurs** et actionnaires cherchant à optimiser leurs choix d'investissement dans l'industrie aérienne. Ces acteurs financiers, qu’ils soient intéressés par les **compagnies aériennes**, les **modèles d’avions**, ou même les **aéroports**, pourraient utiliser les données fournies par l'interface pour identifier les acteurs les plus performants en termes de ponctualité et de gestion des retards. En analysant les tendances de performance, ils pourront prendre des décisions éclairées sur les investissements les plus prometteurs, en privilégiant les compagnies aériennes qui démontrent une gestion optimale des flux et une réduction significative des perturbations. Cela pourrait aussi les aider à repérer des zones géographiques ou des infrastructures (aéroports, compagnies ou modèles d'avions) particulièrement efficaces, réduisant ainsi les risques financiers et maximisant les retours sur investissement.

Au-delà des acteurs traditionnels du secteur aérien, plusieurs autres profils pourraient être intéressés par l'interface **Flight Catcher**. Par exemple, les **fournisseurs de services technologiques** (tels que les entreprises spécialisées dans les systèmes de gestion du trafic aérien ou les technologies de suivi de vol) pourraient exploiter ces données pour enrichir leurs propres solutions et proposer des services plus efficaces. De même, les **consultants en gestion des risques** pourraient se servir des analyses pour aider leurs clients à minimiser les impacts des retards et à optimiser la planification des ressources.

Les **voyagistes et agences de voyages** pourraient également trouver une utilité directe dans cette interface, leur permettant de mieux conseiller leurs clients en fonction des performances des compagnies aériennes et des aéroports, ou encore d'offrir des services personnalisés, comme des recommandations sur les horaires de vol les plus fiables.

Enfin, les **chercheurs et universitaires** travaillant dans des domaines tels que l’ingénierie du transport, les sciences des données, ou l’économie des transports pourraient tirer parti de l'interface pour mener des études sur la ponctualité des vols, l’impact des retards sur les performances économiques, ou encore l’évolution des technologies aéronautiques.

L’interface **Flight Catcher**, par son approche analytique et ses fonctionnalités adaptées à une diversité d’utilisateurs, pourrait donc se révéler un outil précieux pour une large gamme d'acteurs investis dans l’amélioration de l'efficacité et de la rentabilité du secteur aérien.

## Le jeu de donnée initial

Le jeu de données utilisé dans le projet **Flight Catcher** provient de la plateforme Kaggle, un site de référence pour les compétitions et analyses de données. Ce jeu, intitulé **Flight Delay Dataset 2018-2022**, a été créé pour permettre l'étude des retards et des performances des vols commerciaux aux États-Unis sur une période de cinq ans, de 2018 à 2022. Il contient des informations détaillées telles que les retards, les compagnies aériennes, les aéroports d’origine et de destination, les horaires de vol, le temps de trajet, ainsi que les distances parcourues.

Avec un volume total de 2 Go, le fichier est fourni au format Parquet, un format bien adapté aux grandes volumétries de données. Ce choix offre plusieurs avantages : une structure orientée colonnes qui optimise les analyses par lots et des capacités avancées de compression, ce qui réduit considérablement la taille des fichiers tout en maintenant des temps d'accès rapides. Ces caractéristiques font du format Parquet une solution idéale pour le traitement efficace des données volumineuses, limitant ainsi les ressources nécessaires tout en conservant une grande flexibilité pour les analyses.

Ce jeu de données sera exploré en détail dans une section ultérieure du rapport, où nous expliquerons comment ses différentes caractéristiques ont guidé nos choix techniques et analytiques, ainsi que les premières observations tirées des données.

# Mise en place d’un backend "minimal" et premières requêtes

## Choix techniques

Afin de développer l’interface d’analyse des données, nous avons opté pour l’utilisation de **FASTAPI** en tant que framework pour le développement de l’API côté serveur. Ce choix repose sur plusieurs avantages clés de FASTAPI : sa légèreté, ses performances élevées, et sa capacité à générer automatiquement une documentation des points d’accès (endpoints) via Swagger UI, ce qui facilite le développement collaboratif. Par rapport à des alternatives comme **Flask** ou **Django**, FASTAPI se distingue par sa prise en charge native d’asyncio, qui optimise la gestion des requêtes simultanées. Cette caractéristique est particulièrement utile pour notre projet, où des requêtes fréquentes et intensives sur les données sont attendues. Cependant, si Django offre un écosystème riche et intégré pour des projets nécessitant des fonctionnalités web complexes, il aurait pu être perçu comme trop lourd pour notre besoin spécifique, centré principalement sur des appels API performants.

Pour la partie base de données, nous avons choisi **DuckDB** comme moteur analytique principal. DuckDB s’est révélé particulièrement adapté au traitement de fichiers volumineux, notamment au format **Parquet**, grâce à son intégration native qui permet d’effectuer des requêtes SQL directement sur ces fichiers. Cela évite d’avoir à importer les données dans une base intermédiaire, réduisant à la fois le temps de prétraitement et l’utilisation de ressources mémoire. Par exemple, des benchmarks indiquent que DuckDB consomme en moyenne 40 à 70 % moins de mémoire que des bases relationnelles classiques comme **PostgreSQL** ou **MySQL** pour des opérations similaires sur des fichiers de grande taille. Comparé à des solutions distribuées comme **Apache Spark**, DuckDB est limité par son exécution locale mais se montre beaucoup plus rapide et efficient pour des volumes de données jusqu’à plusieurs dizaines de gigaoctets, ce qui correspond à notre cas.

En revanche, une limite notable de DuckDB réside dans sa scalabilité : il n’est pas conçu pour traiter des charges massives sur des clusters distribués. Si nos besoins venaient à évoluer vers des analyses nécessitant une parallélisation étendue ou une intégration avec des flux de données en temps réel, Spark pourrait être une alternative plus appropriée, malgré son coût en ressources et sa complexité d’installation.

Ainsi, le choix initial de **FASTAPI** et **DuckDB**, optimisé pour la légèreté et l’efficacité mémoire, reflétait un compromis judicieux entre performance, simplicité et adaptation à l’échelle de notre projet. Cependant, face à des besoins croissants en termes de scalabilité et d’analyse distribuée, notamment pour des volumes de données dépassant les capacités locales, nous avons dû basculer sur **Apache Spark**. Spark permet une gestion plus robuste des données massives grâce à sa capacité à exécuter des calculs sur des clusters distribués, offrant ainsi une solution adaptée à des traitements plus intensifs et complexes.

## Chargement des données

Les données du jeu Flight Status Prediction sont stockées au format Parquet, ce qui offre des avantages significatifs en termes de compression et de performance de lecture. Ce format a été choisi en raison de sa capacité à réduire la taille des fichiers tout en maintenant un accès rapide aux données, ce qui est essentiel pour traiter les 2 Go d’informations contenues dans ce jeu de données.

Grâce à DuckDB, le fichier Parquet peut être chargé directement dans une table temporaire et interagit avec des requêtes SQL classiques. Cette méthode supprime le besoin de transformations complexes et permet de tirer parti de la simplicité des requêtes SQL pour récupérer les informations pertinentes.

## Premiers objectifs d'analyse

Une fois les données accessibles via DuckDB, nous avons défini plusieurs types de requêtes pour répondre aux principaux besoins analytiques. Ces requêtes constituent les fondations de notre backend et permettent d’extraire des statistiques utiles sur les vols commerciaux. Voici les premiers objectifs couverts par ces analyses :

Identification des compagnies aériennes : L’objectif est de récupérer la liste des compagnies présentes dans le jeu de données afin d’avoir un aperçu des acteurs impliqués.

Analyse des annulations : Le calcul du pourcentage de vols annulés a été réalisé selon deux approches :

Par année pour observer l’évolution des annulations au fil du temps.

Depuis une date spécifique pour analyser les tendances plus récentes.

Analyse des retards : De manière similaire aux annulations, l’étude des vols retardés a été menée par année et depuis une date donnée. Ces statistiques permettent d’identifier les périodes critiques où les retards sont les plus fréquents.

Classement des compagnies aériennes : Deux classements ont été mis en place :

Les 10 compagnies aériennes les plus utilisées, en fonction du nombre total de vols enregistrés.

Les 10 compagnies aériennes les plus performantes, en tenant compte à la fois du taux d’annulation et des délais moyens de départ.

Performance par État : Une analyse a été réalisée pour classer les États en fonction de leurs performances, notamment en évaluant le taux d’annulations et les retards moyens des vols.

Analyse des vols détournés : Enfin, le pourcentage de vols détournés a été étudié pour fournir une vision complète des perturbations.

Les requêtes définies avec DuckDB ont été intégrées dans le backend via FASTAPI. Chaque requête est exposée sous forme d’endpoint API, permettant ainsi à l’interface utilisateur de récupérer les données analytiques en temps réel. Cette architecture garantit une communication fluide entre le backend et l’interface utilisateur, tout en tirant parti des performances offertes par DuckDB.

# Remplacement de DuckDB par Spark

À la suite des premières phases d’analyse, certaines limites de DuckDB ont été identifiées, notamment pour le Machine Learning. Pour surmonter ces limites et répondre aux besoins en matière de scalabilité et d’efficacité, le choix a été fait de remplacer DuckDB par Apache Spark.

## Apache Spark

Apache Spark est un framework de traitement de données distribué qui offre des performances exceptionnelles pour les analyses de grandes volumétries. Contrairement à DuckDB, qui opère principalement en local, Spark permet de distribuer les calculs sur plusieurs nœuds, rendant le traitement plus rapide et plus efficace.

De plus, Spark s’avère particulièrement adapté si l’on souhaite intégrer des traitements de Machine Learning dans le futur. Grâce à sa bibliothèque MLlib, Spark propose des outils performants pour l’entraînement et l’exécution de modèles prédictifs sur des données massives.

L’un des principaux avantages de Spark réside dans son support natif des fichiers au format Parquet, tout comme DuckDB. Cette compatibilité a facilité la migration des requêtes existantes vers Spark sans nécessiter de transformations des données. De plus, Spark dispose d’une API SQL très performante, qui nous a permis de conserver une approche basée sur des requêtes structurées tout en bénéficiant de temps de réponse améliorés sur les analyses complexes.

## Nouvelles analyses réalisées avec Spark

Avec Spark, nous avons pu enrichir notre ensemble de requêtes analytiques en introduisant de nouvelles analyses plus avancées. Parmi celles-ci, on retrouve :

L’analyse des retards par heure de départ : Cette étude permet de déterminer les heures où les retards sont les plus fréquents, offrant ainsi une meilleure compréhension des périodes critiques pour les vols.

Le classement des aéroports en fonction de leur performance : Cette analyse classe les aéroports en tenant compte de leur taux de retards et d’annulations. Elle permet d’identifier les infrastructures les plus performantes et celles qui nécessitent des améliorations.

L’étude de la corrélation entre les retards au départ et à l’arrivée : Cette analyse met en lumière les relations entre les retards enregistrés au départ des vols et les retards constatés à l’arrivée, afin de mieux comprendre les facteurs d’impact.

La distribution des vols par distance parcourue : Cette analyse segmente les vols en plusieurs catégories (courts, moyens et longs trajets) afin d’observer la répartition des distances dans le jeu de données.

La prédiction des retards futurs : Grâce aux capacités d’apprentissage automatique intégrées à Spark, nous avons posé les bases d’un modèle prédictif qui prend en compte des variables comme l’heure de départ, l’origine, la destination et la distance parcourue pour anticiper les retards.

Identify applicable funding agency here. If none, delete this text box.

## Adaptation du backend avec Spark

L’intégration d’Apache Spark dans le backend développé avec FASTAPI a nécessité plusieurs ajustements. Une configuration spécifique de Spark a été mise en place pour assurer une lecture optimale des fichiers Parquet et permettre l’exécution rapide des nouvelles requêtes. Chaque analyse a ensuite été exposée sous forme d’endpoint API, afin de garantir une communication fluide entre le backend et l’interface utilisateur.

# Développement du frontend avec Dash Plotly

Pour le développement du frontend, nous avons sélectionné le framework Dash Plotly, qui constitue une surcouche de React.js pour Python. Dash facilite la création d’applications web interactives pour la visualisation de données, tout en permettant d’intégrer des librairies analytiques comme Pandas pour le traitement des données.

L’interface utilisateur a été conçue pour interagir directement avec le backend FASTAPI, offrant ainsi un tableau de bord interactif capable d’afficher les résultats des requêtes sous forme de tableaux ou de graphiques.

L’application repose sur une structure simple où chaque bouton permet de lancer une requête API spécifique et d’afficher les résultats dans un composant graphique ou tabulaire. Par exemple :

Affichage de la liste des compagnies aériennes : Lorsqu’un utilisateur clique sur le bouton correspondant, l’application récupère les données via un endpoint API et génère un tableau interactif grâce aux composants de Dash.

Calendrier des vols annulés : Une visualisation sous forme de carte thermique (heatmap) est générée pour représenter le pourcentage de vols annulés par mois et par année.

Enfin, Dash Bootstrap Components a été intégré pour assurer un design soigné et une expérience utilisateur optimale grâce à des styles modernes basés sur Bootstrap.

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

# tests de performances

# scénarios d'application

Bibliographie

1. **Flight Delay Dataset 2018-2022**. Kaggle. Disponible à : <https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022>, accédé le 5 novembre 2024.
2. **FastAPI Framework, High Performance, Easy to Learn, Fast to Code, Ready for Production**. FastAPI Documentation. Disponible à : <https://fastapi.tiangolo.com/>, accédé le 15 novembre 2024.
3. **DuckDB: The SQLite for Analytics**. DuckDB Official Website. Disponible à : <https://duckdb.org/>, accédé le 25 novembre 2024.
4. **Apache Spark™ - Unified Analytics Engine for Big Data**. Apache Spark Official Website. Disponible à : <https://spark.apache.org/>, accédé le 5 décembre 2024.
5. **Dash: Low-Code Framework for Dashboards in Python**. Dash Plotly Documentation. Disponible à : <https://dash.plotly.com/>, accédé le 15 décembre 2024.
6. Mühleisen, H., & Raasveldt, M. (2020). DuckDB: Analytical SQL in the Local Environment. VLDB Conference Proceedings.  
   Accédé le 12 novembre 2024. <https://www.vldb.org/pvldb/vol13/p2485-raasveldt.pdf>
7. DuckDB Project Page. Why DuckDB?  
   Accédé le 22 novembre 2024. <https://duckdb.org/why_duckdb>
8. FastAPI Documentation. FastAPI: High Performance Framework.  
   Accédé le 2 décembre 2024. <https://fastapi.tiangolo.com/performance>
9. Databricks Blog. (2023). Spark vs. Local Analytical Databases.  
   Accédé le 12 décembre 2024. <https://databricks.com/blog/2023/01/spark-vs-local-analytical-databases.html>
10. Apache Spark Documentation. Apache Spark Scalability.  
    Accédé le 22 décembre 2024. <https://spark.apache.org/docs/latest/>