Rapport de Projet de Fin de Formation

Hugues Bousquet[1]

13 Juin 2025

1. Projet commun réalisé par Jiang Chang, Christopher Dumas, Michael Thomas et l'Auteur

Résumé

Ce rapport présente la conception et mise en œuvre d'un pipeline de données complet, depuis l'extraction jusqu'à la visualisation, dans le cadre d'un projet de fin de formation. Nous avons opté pour une architecture ELT (Extract-Load- Transform) avec une *stack* technique incluant Fivetran pour l'extraction, DBT pour la transformation, BigQuery comme entrepôt de données, et Tableau pour la visualisation. L'orchestration a été assurée par Airflow, permettant d'automatiser l'ensemble des *workflows*, y compris l'exécution de modèles de Machine Learning (Prophet pour les prévisions de Chiffre d'affaires, Apriori pour l'analyse des associations de produits et XGBoost pour l'analyse des causes de pannes de machines).

Ce travail a permis de valider des compétences techniques variées (SQL, Python, orchestration, *dataviz*) tout en soulignant l'importance cruciale de la qualité des données et de la collaboration d'équipe dans un projet data. Le pipeline conçu répond aux exigences initiales et pourra facilement s'adapter à de nouveaux besoins analytiques.

Table des matières

Introduction et organisation

Le projet en fin de formation s'inscrivait dans le cadre d'une première mission professionnelle chez Carttrend, entreprise spécialisée dans l'e-commerce. Face à un marché concurrentiel, l'équipe ***Data & Insights*** avait été chargée de construire un pipeline complet pour exploiter des données jusqu'alors sous-utilisées et dispersées dans des fichiers Excel et Google Sheets. L'objectif était alors de transformer ces sources brutes en analyses pertinentes et en tableaux de bord clairs et conviviaux, facilitant ainsi la prise de décisions stratégiques dans quatre domaines critiques : l'analyse des ventes, l'efficacité marketing, l'optimisation logistique et la satisfaction client.

Nous avons rapidement identifié trois impératifs sous-jacents. Premièrement, concevoir des tableaux de bord intuitifs, concis et régulièrement mis à jour. Nous avons donc choisi très tôt de limiter volontairement le nombre de visualisations pour éviter la surcharge cognitive des utilisateurs, privilégiant l'usage quotidien à l'exhaustivité. De plus, il nous a semblé essentiel de proposer une solution clé en main nécessitant une maintenance minimale par des équipes non nécessairement techniques. Enfin, il fallait construire une architecture évolutive, capable de gérer un gros volume de données, mais aussi facilement adaptable à de nouveaux axes d'analyse sans refonte majeure.

Notre équipe, composée de quatre collaborateurs aux compétences diverses, a commencé par analyser les forces et faiblesses de chacun et la manière dont celles-ci pouvaient se compléter au sein du groupe. Certaines forces majoritaires rapidement identifiées, comme une préférence pour SQL dans les opérations de transformation des données, une aisance avec l'écosystème Google Cloud Platform (BigQuery) et une maîtrise de l'outil de visualisation Tableau, ont également orienté le choix de la *stack* technique. Cette diversité de compétences a permis une répartition efficace des tâches, avec un travail parallèle sur la majeure partie du pipeline (principalement pour des raisons pédagogiques), complété par des spécialisations ponctuelles sur des sujets spécifiques comme le Machine Learning ou l'orchestration Airflow.

Un calendrier initial *(Figure 1)* a été rapidement établi pour structurer les dix jours du projet, dans un effort conscient d'éviter l'embourbement dans la partie technique du pipeline et ainsi réserver suffisamment de temps à la visualisation et à l'analyse. L'objectif était également de prévenir le *feature creep* du pipeline, en n'ajoutant des fonctionnalités supplémentaires qu'une fois le cœur du travail réalisé, permettant ainsi d'en évaluer plus précisément la pertinence. Des réunions *stand-up* biquotidiennes ont été mises en place – le matin pour attribuer les tâches et résoudre les blocages immédiats et le soir pour évaluer les résultats de la journée, documenter

collectivement les avancées et ajuster les objectifs du lendemain. Cette organisation, combinée à des revues d'étape bihebdomadaires avec l'enseignant jouant le rôle du chef de projet, a assuré une progression fluide sans recours au *crunch*.

Mardi 20/05	Mercredi 21/05	Jeudi 22/05	Vendredi 23/05	Samedi	Dimanche	Lundi 26/05	Mardi 27/05	Mercredi 28/05	Jeudi 29/05	Vendredi 30/05
Organisation Générale Debut Nettoyage Données (DBT) Configuration Airbyte	Nettoyage Données (DBT Staging)	Conception Marts Implementation Marts (DBT Marts)	Debut Dataviz (Tableau) Machine Learning (Python)	24/05	25/05	Dataviz (Tableau) Fin Machine Learning (Python)	Dataviz (Tableau)	Fin Dataviz (Tableau) Analyses et Recommanda-tions	Orchestration (Airflow) Finitions et Debugage	Finitions et Debugage

FIGURE 1 – Plan d'organisation initial

Chapitre 1

Conception et Sélection de la Stack Technique

Ce chapitre présente les choix fondamentaux qui ont guidé la conception de notre pipeline de données pour Carttrend. Dans le but de transformer des sources disparates en informations exploitables, nous avons dû prendre plusieurs décisions techniques stratégiques concernant l'architecture globale. La conception de notre pipeline de données a ainsi débuté par une réflexion fondamentale sur son architecture.

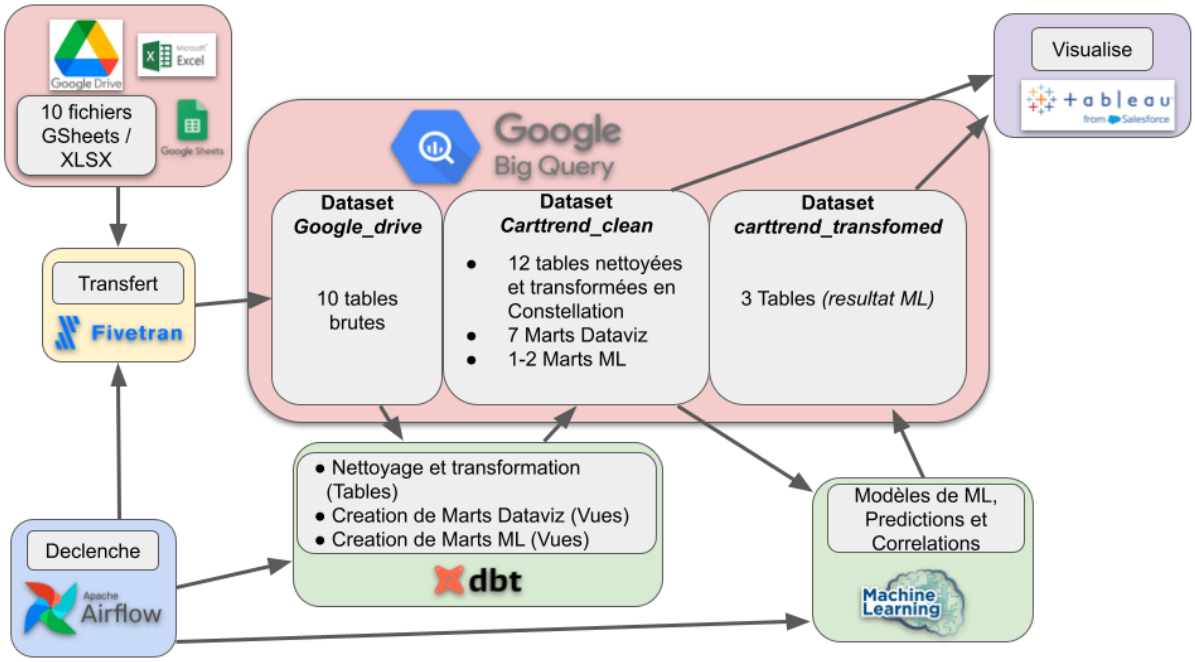


FIGURE 1.1 – Diagrame de l'architecture finale séléctionnée

1.1 Modèle - Extract-Load-Transform

Après analyse, nous avons opté pour un modèle ELT (Extract-Load-Transform) plutôt qu'une approche ETL traditionnelle. En premier lieu, le modèle ELT offre une meilleure scalabilité pour faire face à une potentielle croissance du volume de données chez Carttrend. Contrairement à un modèle ETL où les transformations sont appliquées avant le chargement, travailler en ELT nous permet de charger d'abord les données brutes dans le *data warehouse*, puis d'y appliquer les transformations nécessaires. Cette approche réduit les goulots d'étranglement potentiels lors des phases d'extraction et de chargement. De plus, cette architecture préserve une flexibilité essentielle pour l'analyse des données. En conservant les données originales dans le *data warehouse* avant transformation, nous maintenons la possibilité de retraiter ces informations selon de nouveaux axes d'analyse pas toujours identifiés immédiatement. Enfin, ce choix technique correspondait aux compétences globales de notre équipe, plus à l'aise avec SQL qu'avec Python pour les opérations de transformation complexes. L'approche ELT nous a permis d'implémenter l'ensemble de nos logiques métier en SQL via DBT, tout en bénéficiant de la puissance de calcul distribuée du *data warehouse*. Cette orientation ELT a constitué le fil directeur dans le choix de notre stack technique, qui est présentée dans les sections suivantes.

1.2 Data Warehouse - BigQuery

BigQuery de Google s'est naturellement imposé comme socle de notre infrastructure. En tant que *data warehouse* SaaS, BigQuery réduit en effet la nécessité d'interventions manuelles pour résoudre les problèmes de provisionnement de capacité et abstrait la gestion des ressources physiques (puissance de calcul, espace de stockage, accès réseau...). D'autre part, notre équipe manifestait une aisance particulière avec l'écosystème Google Cloud Platform. Cette familiarité technique, couplée à la large adoption de BigQuery dans l'industrie, a permis de réduire significativement les difficultés de configuration des différents outils du reste de la *stack* (DBT, Fivetran, scripts Python...). Enfin, le choix de BigQuery répondait à une exigence de sécurité des données. Cette solution délègue les problématiques sensibles de chiffrement et de contrôle d'accès à Google, nous permettant de se concentrer sur les données plutôt que sur l'infrastructure sous-jacente.

1.3 Extraction et Chargement - Outils Managés ou Scripts Python

Pour l'extraction et le chargement des données, nous avons délibérément choisi d'utiliser une solution managée (Airbyte Cloud ou Fivetran, le choix exact de l'outil n'ayant pas été fait à ce stade) plutôt que de développer des scripts Python spécialisés. Ces outils managés "cloud-native" offrent en effet une fiabilité supérieure et une maintenance simplifiée, avec une scalabilité immédiate, alors que des scripts Python maison nécessiteraient des ajustements et

une optimisation régulière pour s'adapter aux besoins évolutifs. Sur le plan de la sécurité, ces solutions permettent – comme pour BigQuery – de déléguer les problématiques complexes de chiffrement et de gestion des accès aux experts, plutôt que de les faire reposer sur notre propre implémentation. Enfin, leur configuration via une interface simple et intuitive réduit considérablement le temps de développement tout en assurant une maintenance simplifiée à long terme.

1.4 Transformation - DBT Cloud

Nous avons choisi DBT Cloud (Data Build Tool) comme solution de transformation des données. Cet outil s'intègre parfaitement avec BigQuery et permet d'exécuter les transformations directement dans le *data warehouse*, exploitant ainsi les performances de l'infrastructure Google tout en réduisant les mouvements de données. La flexibilité de matérialisation (*view*/table) offre un contrôle précis sur le compromis entre puissance de calcul et espace de stockage. L'architecture modulaire de DBT (*staging*, *marts*) facilite l'adaptation du pipeline à la fois aux nouvelles sources de données et aux besoins évolutifs des outils d'analyse (Machine Learning, visualisation...). La documentation intégrée et les tests faciles à rédiger garantissent la qualité des données après transformation. Enfin, l'utilisation du SQL – particulièrement adapté aux compétences de notre équipe – combinée aux fonctionnalités collaboratives de l'outil, ont définitivement confirmé notre choix.

1.5 Orchestration - Airflow

Pour l'orchestration de notre pipeline, deux solutions ont été envisagées. La première, plus basique, aurait utilisé les fonctionnalités natives d'orchestration de Fivetran, permettant un déclenchement régulier des synchronisations (par exemple quotidiennes) et l'exécution automatique d'un job DBT après chaque chargement. Cette approche, plus simple à mettre en œuvre et à maintenir par des équipes moins techniques, aurait pu suffire à répondre au cahier des charges. Les scripts Python de Machine Learning auraient alors été exécutés séparément via des *jobs 'cron'* sur un serveur dédié.

Nous avons finalement opté pour Airflow, solution plus complexe initialement mais offrant une gestion bien plus fine des dépendances entre tâches, comme par exemple l'enchaînement conditionnel de l'entraînement des modèles de machine learning seulement après un transfert et une transformation réussis des données. D'autre part, Airflow offre une flexibilité accrue pour l'ajout de nouvelles fonctionnalités, comme l'intégration ultérieure d'un script de géolocalisation des clients via leurs adresses IP. Ces atouts justifient a nos yeux la complexité initiale de mise en place, et nous avons tenté de remédier à la difficulté de maintenance en utilisant une architecture de code simple.

1.6 Visualisation - Tableau

Notre choix s'est porté sur Tableau pour la visualisation des données, principalement en raison de ses avantages en contexte gratuit. La version de démonstration de Tableau Cloud autorisait une collaboration efficace entre membres de l'équipe, avec des fonctionnalités de partage et d'édition conjointe supérieures à ce qu'offre Power BI sans licence payante. L'intégration native avec BigQuery a constitué un autre atout décisif, permettant un accès simple aux données et une mise à jour des *dashboards* automatique et régulière. Enfin, la familiarité préexistante de l'équipe avec Tableau a accéléré notre productivité, nous permettant de se concentrer sur la création de visualisations claires et efficaces.

Conclusion

Cette première partie a présenté les choix architecturaux fondamentaux qui structurent notre pipeline de données. L'architecture ELT retenue, reposant sur une *stack* technique cohérente (BigQuery, Fivetran, DBT Cloud, Airflow et Tableau), forme un pipeline robuste et évolutif. Un diagramme global de cette architecture illustrant les interactions entre ces différents outils *(Figure 2.5)*, est proposé en page 4.

Chapitre 2

Implémentation du Pipeline

Cette section présente la concrétisation technique de nos choix architecturaux en un pipeline opérationnel. Nous détaillons les étapes clés de construction : configuration de BigQuery, intégration des solutions d'extraction (Fivetran), transformation via DBT, enrichissement par scripts Python, et orchestration globale avec Airflow.

2.1 Configuration du *data warehouse* - BigQuery

La mise en place de notre environnement BigQuery a débuté par la création d'un compte Google unique doté des droits d'administrateur sur Google Cloud Platform. Après avoir initialisé un projet BigQuery, nous avons structuré l'espace de stockage en trois *datasets* distincts *(Figure 2.1)*, chacun dédié à une étape spécifique du pipeline : données brutes, données transformées, et résultats des modèles de Machine Learning.

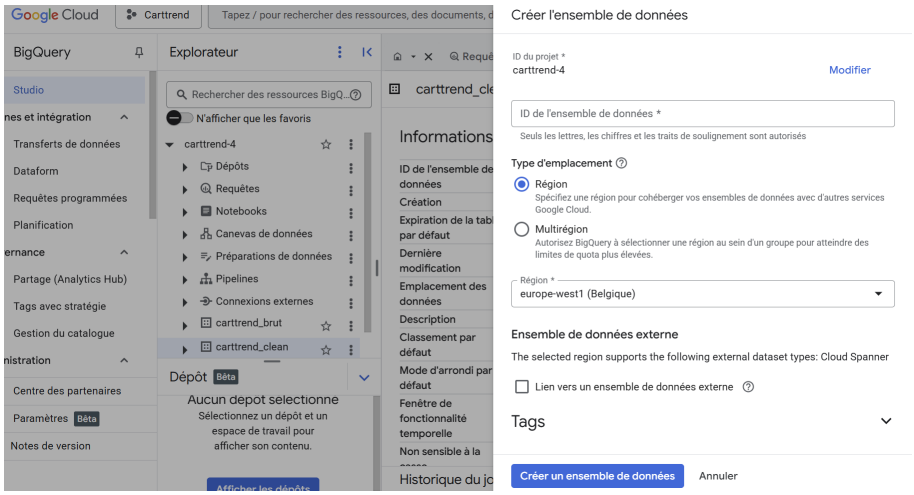


FIGURE 2.1 – Creation des datasets sur BigQuery

L'automatisation des processus a nécessité l'établissement de comptes de service individuels pour chaque outil *(Figure 2.2)*. DBT et les scripts Python de Machine Learning exécutés via Airflow bénéficient ainsi d'identifiants spécifiques avec des permissions limitées à leurs fonctionnalités respectives. Bien que des comptes aient été initialement configurés pour Airbyte durant la phase de tests, ceux-ci ont été soigneusement désactivés après l'adoption définitive de Fivetran comme solution d'extraction.

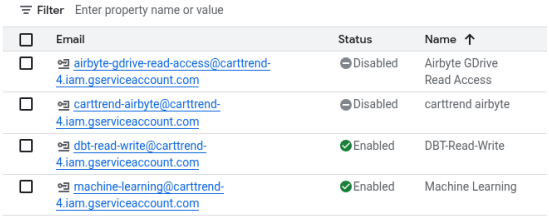


FIGURE 2.2 – Comptes de services

Pour accélérer l'implémentation, nous avons opté pour une authentification OAuth via le compte administrateur Google afin d'autoriser Fivetran et Tableau à accéder aux données. Bien que pratique, cette approche présente des risques de sécurité en accordant des privilèges excessifs à ces outils – qui n'ont pas besoin d'un accès administrateur complet – tout en créant une dépendance à un compte administrateur unique.

Cette configuration permet d'ores et déjà d'utiliser BigQuery dans un pipeline fonctionnel et performant, mais nécessiterait des améliorations avant un déploiement à plus grande échelle.

2.2 Configuration du Chargement et de l'Extraction - Fivetran

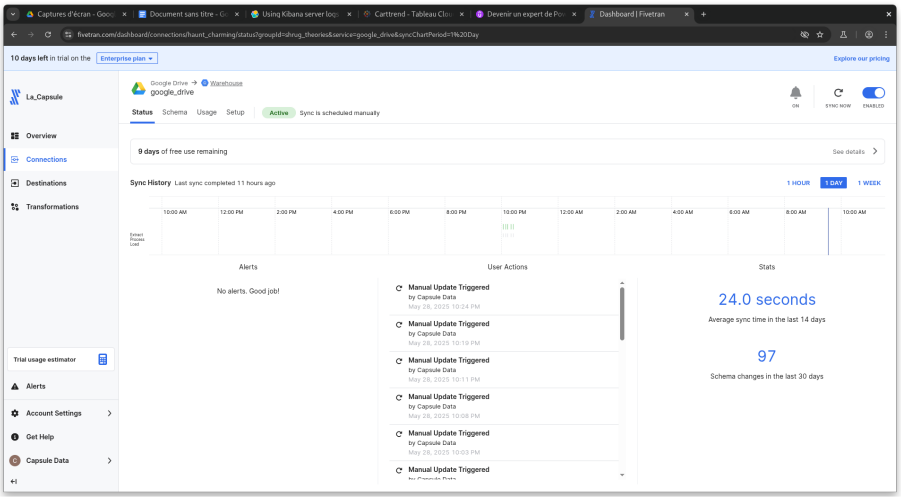


FIGURE 2.3 – Interface de Gestion de Fivetran

Initialement, Airbyte Cloud avait retenu notre attention pour son aspect open source et son interface simplifiée. Cependant, des difficultés techniques sont rapidement apparues, notamment avec le connecteur Google Drive développé par la communauté, qui ne permettait pas d'accéder efficacement aux fichiers partagés – c'est-à-dire ceux dont le compte Google utilisé

par Airbyte n'était pas propriétaire. Bien qu'un contournement ait été envisagé (chargement manuel de chaque fichier dans une source Airbyte distincte), cette solution à été jugée peu flexible. Face à ces difficultés, nous avons initialement procédé à un chargement manuel des données dans BigQuery, permettant ainsi à une partie de l'équipe de commencer les étapes de nettoyage et de transformation, tandis que l'autre pouvait opérer la migration vers Fivetran sereinement.

La configuration de Fivetran *(Figure 2.4)* a débutée par la création d'un compte d'essai, et la connexion à BigQuery et Google Drive via OAuth. Bien que cette méthode d'authentification présente des limites (comme mentionné précédemment), elle a permis une intégration rapide. Fivetran s'est montré remarquablement efficace pour lire sans difficulté les fichiers Google Drive partagés, traitant de manière transparente et uniforme les formats Google Sheets et Excel. L'outil a même fait preuve d'intelligence en supprimant automatiquement les diacritiques des noms de colonnes incompatibles avec BigQuery. En prévision de l'automatisation, nous avons généré une clé API et étudié la documentation de l'API REST [1] afin de pouvoir déclencher les synchronisations via Airflow, une fonctionnalité essentielle pour l'orchestration future du pipeline.

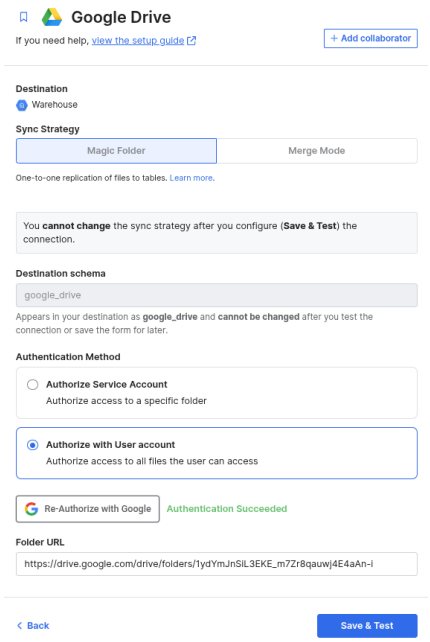


FIGURE 2.4 – Configuration de Fivetran

Au final, Fivetran s'est imposé comme une solution optimale pour l'extraction et le chargement. Ses nombreux connecteurs pré-intégrés, sa maintenance simplifiée et son potentiel d'automatisation en font un composant clé garantissant l'évolutivité et l'adaptabilité future de notre infrastructure de données.

2.3 Transformation de données - DBT Cloud

La transformation des données est une étape capitale durant laquelle les informations brutes sont nettoyées, enrichies et converties en une structure exploitable pour l'analyse. Cette section explore en détail notre mise en œuvre de DBT (Data Build Tool), outil que nous avons retenu pour effectuer cette transformation. Nous commencerons par la configuration de base, puis nous décrirons l'organisation du projet. Cette dernière sera illustrée par l'explication détaillée de quelques modèles représentatifs. Enfin, nous aborderons la configuration avancée de DBT, permettant notamment son interfaçage avec Airflow.

1. Disponible sur le site de Fivetran, `https://fivetran.com/docs/rest-api/api-reference/connections`

2.3.1 Configuration de Base

La configuration intiale de notre environnement DBT a suivi plusieurs étapes. Nous avons débuté par la création d'un compte d'essai sur DBT Cloud, suivie de l'ouverture d'un compte GitHub hébergeant un dépôt dédié au *versioning* de notre code de transformation. Le projet DBT a ensuite été configuré pour lui permettre d'accéder aux données brutes stockées dans BigQuery ainsi qu'à un dataset de sortie, via un compte de service spécifique. Enfin, nous avons établi la connexion entre le projet DBT Cloud et notre dépôt GitHub.

L'utilisation partagée d'un seul compte DBT Cloud partagé par toute l'équipe a cependant révélé des limites organisationnelles. Si le travail sur les modèles SQL restait praticable en parallèle grâce à leur découpage en fichiers distincts, la gestion simultanée des fichiers schema.yaml – essentiels pour la documentation et les tests – posait davantage problème. Pour prévenir les risques d'écrasement des modifications, nous avons instauré un système de rotation informel lors de l'édition de ces fichiers critiques, ce qui a légèrement affecté notre productivité durant les phases de documentation et d'écriture de tests.

2.3.2 Organisation du Projet DBT

Les données sources étaient déjà relativement bien organisées et ne nécessitaient que peu de normalisation supplémentaire. Ce constat nous a conduits à opter pour une architecture simplifiée en seulement deux niveaux : *staging* et *marts*.

Le dossier *staging* regroupe l'ensemble des modèles décrivant des transformations initiales appliquées en une seule étape. Ces modèles sont à la fois responsables du nettoyage des données brutes (par exemple la gestion des valeurs nulles, la standardisation des formats ou le renommage des colonnes) mais aussi d'une normalisation légère (par exemple, l'unification des canaux publicitaires présents dans plusieurs tables sources). A l'issue de cette étape, chaque modèle génère une vue dans BigQuery, optimisant ainsi l'utilisation du stockage. Nous avons délibérément évité une normalisation excessive à ce stade pour éviter de complexifier inutilement les requêtes ultérieures en multipliant les jointures, l'organisation initiale des données source ne le nécessitant pas. Les données transformées adoptent donc une structure proche du modèle en constellation *(Figure 2.5)*, avec :

— Trois tables de fait principales (Campagnes, Posts et Commandes),
— Des tables de dimensions clairement identifiées (ex. Produits, Entrepôts, Canaux...),
— Quelques tables hybrides conservant une nature mixte
(ex. Satisfaction, Entrepots_Machines...).

Le dossier *marts* organise les données transformées selon leurs finalités. D'un côté, les modèles dédiés au machine learning, comme *mart_Previsions_Series_Temporelles_CA.sql*, mettent en œuvre un *feature engineering* spécifiquement conçu pour les algorithmes respectifs. De l'autre, les modèles orientés visualisation, à l'image de *mart_Commentaires.sql*, concentrent les agrégations et calculs d'indicateurs métiers essentiels pour alimenter les *dashboards* Tableau.

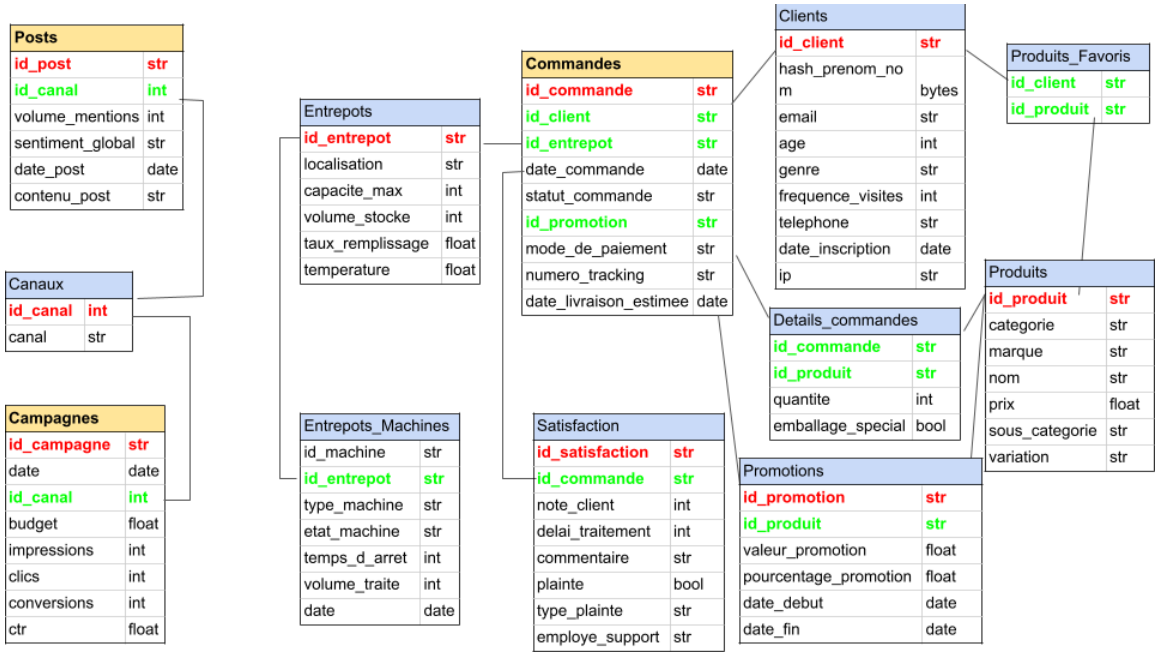


FIGURE 2.5 – Diagramme des vues générées par les modèles *staging*

Contrairement aux vues utilisées dans le dossier *staging*, tous les modèles de type *mart* sont matérialisés en tables dans BigQuery pour accélérer l'accès aux données, améliorant ainsi notablement la réactivité des *dashboards* tout en réduisant la puissance de calcul lors des requêtes fréquentes.

Cette architecture présente cependant des limites structurelles qui sont apparues au fil du développement. Le modèle *mart_Commandes.sql* illustre particulièrement ces contraintes, ayant progressivement accumulé trop de logique de transformation (phénomène de *scope creep*) et aurait mérité un traitement intermédiaire distinct. D'autre part, bien que parfaitement fonctionnelle pour ce projet de taille modérée, l'approche combinant nettoyage et transformation en une seule étape réduit la modularité du code. Une évolution future gagnerait à introduire un dossier *"intermediate"* (conforme aux bonnes pratiques recommandées par la documentation officielle de DBT [1]) qui permettrait d'isoler la construction des modèles complexes, de découpler clairement nettoyage des sources, normalisation et enrichissement métier, et de faciliter la réutilisation des modèles intermédiaires. Enfin, une normalisation plus poussée – qui bénéficierait aussi d'un niveau intermédiaire – pourrait s'avérer nécessaire si la structure des données sources venait à se complexifier ou si de nouvelles sources venaient s'ajouter aux existantes.

Il convient donc de considérer l'architecture actuelle comme une base solide mais perfectible, appelée à évoluer en fonction des besoins futurs.

1. How we structure our dbt projects, `https://docs.getdbt.com/best-practices/how-we-structure/1-guide-overview`

2.3.3 Analyse de code

Cette section présente quelques exemples détaillés de modèles et de fichiers de configuration. Ces exemples complets, ainsi que d'autres sont disponibles, largement commentés [1].

Clients.sql

Ce modèle, situé dans le dossier *staging*, a pour objectif de nettoyer et d'enrichir la table brute des clients. Il effectue notamment un important travail de standardisation des numéros de téléphone et de validation des données.

La première partie est une *Common Table Expression* (CTE) relativement complexe nommée "tel" utilisée dans la standardisation les numéros de téléphone. Elle extrait et le numéro principal au format `000-000-0000` , ainsi que l'indicatif pays (comme `+033` pour la France) avec un ajout de zéros si nécessaire, afin de préparer la standardisation au format international [2]).

```
WITH tel AS (
  SELECT
    id_client,
    -- Formatage standard du numéro (123-456-7890)
    REGEXP_REPLACE(numero_telephone, r'^.*?(\d{3})\D?(\d{3})\D?(\d{4}).*$',
        r'\1-\2-\3') AS numero_telephone,
    -- Left-Padding du code pays (ex: +033 pour la France)
    CASE
      WHEN LENGTH(ac.area_code) = 3 THEN CONCAT('+', ac.area_code)
      WHEN LENGTH(ac.area_code) = 2 THEN CONCAT('+0', ac.area_code)
      WHEN LENGTH(ac.area_code) = 1 THEN CONCAT('+00', ac.area_code)
      ELSE '+033'  -- Valeur par défaut
    END as area_code
  FROM {{source("google_drive",'carttrend_clients_clients')}}
  JOIN (
    -- Sous-Requete pour l'extraction du code pays depuis le numéro
       original
    SELECT
    id_client,
    REGEXP_EXTRACT(numero_telephone, r'^\+?([1-9]{0,3}).*$') AS area_code
    FROM {{source("google_drive",'carttrend_clients_clients')}}
  ) ac USING(id_client)
)
```

La requête principale se concentre sur la sélection et la validation des différents champs clients. Elle inclut notamment l'anonymisation des noms et prénoms via un hachage `SHA256` ainsi que la validation des emails à l'aide d'une expression régulière volontairement permissive, pour minimiser les faux négatifs qui pourraient éliminer des adresses valides. Elle vérifie en outre que les adresses IP respectent le standard IPv4 [3], même s'il est important de noter que

1. cf. Annexe A, page 46
2. ITU-T E.164 §6.1, `https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-E.164-201011-I!!PDF-F`
3. RFC 791, §2.3 `https://www.rfc-editor.org/rfc/rfc791#page-7`

l'expression régulière ne vérifie pas que chaque segment numérique est inférieur à 256.

```sql
SELECT
  -- Identifiant client conservé
  id_client,

  -- Anonymisation des données personnelles avec hashing SHA256
  sha256(CONCAT(prenom_client, nom_client)) as hash_prenom_nom,

  -- Validation de l'email
  CASE
    WHEN REGEXP_CONTAINS(email, r'^[\w\.-]+@[\w-]+.[A-Za-z]+$') THEN email
    ELSE NULL
  END as email,

  -- Suppression des âges négatifs
  CASE WHEN age > 0 THEN age END as age,

  -- Calcul de la tranche d'âge
  CASE
    WHEN age > 0  AND age <= 25 THEN '<=25'
    WHEN age > 25 AND age <= 40 THEN '25-40'
    WHEN age > 40 AND age <= 55 THEN '40-55'
    WHEN age > 55 AND age <= 70 THEN '55-70'
    WHEN age > 70 THEN '>70'
    ELSE NULL
  END as tranche_age,

  -- Normalisation du genre
  CASE
    WHEN lower(genre) = 'homme' THEN 'Homme'
    WHEN lower(genre) = 'femme' THEN 'Femme'
    ELSE NULL
  END as genre,

  -- Validation et correction de la fréquence de visites
  CASE
    WHEN frequence_visites > 0 THEN frequence_visites
    ELSE 0
  END as frequence_visites,

  -- Construction du numéro de téléphone complet
  -- Indicatif + Numero, format final +000-123-456-7890
  CONCAT(tel.area_code, '-', tel.numero_telephone) as telephone,

  -- Validation de la date d'inscription
  CASE
    WHEN DATE_DIFF(date_inscription, CURRENT_DATE(), DAY) <= 0
    THEN date_inscription
    ELSE NULL
  END as date_inscription,

  -- Calcul de l'ancienneté en jours
  CASE
    WHEN DATE_DIFF(date_inscription, CURRENT_DATE(), DAY) <= 0
    THEN DATE_DIFF(CURRENT_DATE(), date_inscription, DAY)
```

```
    ELSE NULL
  END as anciennete_jours,

  -- Validation des adresses IP (format IPv4)
  CASE
    WHEN REGEXP_CONTAINS(adresse_ip, r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$
        ')
    THEN adresse_ip
    ELSE NULL
  END as ip

FROM {{source("google_drive",'carttrend_clients_clients')}}
JOIN tel USING(id_client)
WHERE id_client IS NOT NULL  -- Exclusion des clients sans identifiant
```

Ce modèle fournit une base de données clients propre et standardisée, prête à être utilisée pour toutes les analyses ultérieures.

Canaux.sql

Ce second exemple illustre la normalisation des données communes à plusieurs tables sources. Il s'agit ici de créer une référence unifiée des canaux de communication utilisés dans deux différentes sources : la table "Posts" et la table "Campagnes". Le modèle est d'abord constitué d'une CTE nommée "canal", utilisant opération `UNION DISTINCT` qui agrège les noms des canaux de communication entre les deux sources tout en garantissant l'élimination des doublons. La requête principale ajoute ensuite à cette CTE une clé primaire auto-incrémentée ("id_canal") grâce a la fonction `ROW_NUMBER()`. Le modèle produit une table de dimension avec deux colonnes :

- "id_canal" : Identifiant unique numérique (clé primaire),
- "canal" : Nom original du canal de communication.

```
WITH canal AS (
  -- Extraction des canaux depuis les posts réseaux sociaux
  SELECT canal_social
  FROM {{source("google_drive",'carttrend_posts_posts')}}

  UNION DISTINCT  -- Combine les deux en éliminant les doublons

  -- Extraction des canaux depuis les campagnes marketing
  SELECT canal
  FROM {{source('google_drive', 'carttrend_campaigns_campagnes')}}
)
SELECT
  -- Génération d'un ID unique pour chaque canal,
  -- un chiffre séquentiel auto-incrémenté
  ROW_NUMBER() OVER() as id_canal,

  -- Conservation du nom original
  canal_social as canal

```

```sql
FROM canal
-- Tri alphabétique pour une meilleure lisibilité
ORDER BY canal ASC
```

En centralisant les valeurs communes, le processus de normalisation supprime les redondances et on garantit la cohérence des données. Pour en bénéficier, une mise à jour des tables sources "Posts" et "Campagnes" s'impose : elles devront désormais référencer la table dimension via des clés étrangères ("id_canal") plutôt que de stocker directement les noms de canaux.

mart_Commentaire.sql

Il s'agit cette fois-ci d'un *mart*, transformant les commentaires bruts des clients en données structurées pour alimenter un nuage de mots interactif dans le *dashboard* Satisfaction Client. Il extrait les termes significatifs et calcule leurs indicateurs clés :

— Fréquence : le nombre de fois que ce mot apparait. Détermine la taille du mot dans le nuage,
— Note moyenne : la note moyenne du commentaire dans lequel le mot apparait. Détermine la couleur du mot dans le nuage et permet de filtrer celui-ci afin de n'afficher que les mots apparaissant dans une certaine plage de notes.

La requête s'articule en deux étapes principales, à nouveau avec l'aide de CTEs. La première, nommée "mots", décompose chaque commentaire en mots tout en préservant la note associée. Les fonctions imbriquées `UNNEST(SPLIT(' '))` permettent cette transformation d'une chaine de caractères à une colonne de mots individuels.

```sql
WITH
  mots AS (
    SELECT
    note_client,  -- Note associée (1-5)
    mot
    FROM {{ref("Satisfaction")}},  -- Table staging nettoyée
    UNNEST(SPLIT(commentaire, ' ')) AS mot  -- Désagrégation en mots
  ),
```

Le résultat de cette première CTE est ensuite filtré pour retirer les mots vides puis agrégée par mots pour supprimer les doublons. Les mots sont ensuite nettoyés d'éventuels caractères spéciaux et normalisés en majuscule, puis fréquence et note moyenne sont calculées.

```sql
  mots_aggreges AS (
    SELECT
      UPPER(REGEXP_REPLACE(mot, r'[^\w]', '')) AS mot,
      COUNT(*) AS frequence,
      ROUND(AVG(note_client), 1) AS note_moyenne
    FROM mots
    WHERE mot != ''
      AND lower(mot) NOT IN (
        'a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', '
            an', 'and'
      ) -- liste de mots vides ecourtee pour plus de lisibilite
```

```
    GROUP BY mot
  )
```

Le résultat final est enfin trié par fréquence décroissante, principalement à des fins de *debugage*

```
SELECT *
FROM mots_aggreges
ORDER BY frequence DESC
```

Cette préparation des données est essentielle pour la réalisation d'une visualisation relativement complexe dans Tableau comme celle de cet exemple. Grâce à cette dernière, les commentaires des clients pourront être analysés d'un coup œil, tout en préservant le contexte dans lequel ils ont été écrits.

mart_Prevision_Series_Temporelles_CA.sql

Ce modèle de type *mart* prépare les données historiques de chiffre d'affaires pour alimenter le modèle de prévision de séries temporelles (Prophet). Il garantit un format adapté ses besoins spécifiques (deux colonnes, l'une contenant la date sans interruption et l'autre la série à prédire, ici le chiffre d'affaires). Le modèle utilise à nouveau deux CTEs pour assembler la table finale.

La première CTE, nommée "cal", génère l'ensemble des dates entre la première et la dernière commande enregistrée, créant ainsi un calendrier continu sans interruption.

```
WITH
  cal AS (
    SELECT date
    FROM UNNEST(GENERATE_DATE_ARRAY(
      -- Date minimale
      (SELECT MIN(date_commande) FROM {{ref('mart_Commandes')}}),
      -- Date maximale
      (SELECT MAX(date_commande) FROM {{ref('mart_Commandes')}})
    ) AS date
  ),
```

La seconde CTE, "ca", calcule le chiffre d'affaires journalier à partir des commandes validées, en excluant explicitement les celles qui ont été annulées et ne représentent donc pas un revenu réel. L'agrégation se fait par date de commande, dont la granularité est le jour.

```
    ca AS (
    SELECT
      date_commande AS date,
      SUM(prix_apres_remise) AS ca  -- Somme des montants apres remise
    FROM {{ref('mart_Commandes')}}
    WHERE statut_commande != 'Annulée'  -- Filtre des commandes annulees
    GROUP BY date_commande
    )
```

La requête finale assemble enfin ces deux CTE via une jointure gauche, garantissant que toutes les dates du calendrier apparaîtront dans le résultat final. Aux jours sans commandes sons assignés le chiffre `0` en valeur de CA plutôt que `NULL`, format préféré par Prophet.

```sql
SELECT
  cal.date AS ds,  -- Colonne 'ds' requise par Prophet
  ROUND(IFNULL(ca.ca, 0), 2) as y  -- Colonne 'y' requise par Prophet
FROM cal
  -- Jointure gauche sur les dates pour garder toutes les dates
  LEFT JOIN ca USING(date)
ORDER BY date  -- Tri chronologique obligatoire
```

Grâce à cette préparation minutieuse, Prophet peut ingérer les données de chiffre d'affaires et ajuster un modèle de prédiction avec un minimum de transformation de données préalable.

2.3.4 Configuration avancée

La configuration du projet DBT a par la suite été modifiée pour garantir la qualité des données et de la documentation, autoriser l'automatisation du processus et gagner en performance de requêtage.

Le fichier *dbt_project.yml* a tout d'abord été configuré pour implémenter les stratégies de matérialisation mentionnées dans le paragraphe 2.3.2 en page 11.

Une attention particulière a été portée aux fichiers *schema.yaml* qui jouent un double rôle essentiel. Ils documentent précisément chaque modèle (description des tables et colonnes) tout en définissant des tests rigoureux pour assurer la qualité des données. Ces tests vérifient divers aspects critiques : intégrité des relations via les clés étrangères, présence obligatoire de données, unicité des valeurs, ou encore leur conformité à des plages acceptables. Nous avons étendu ces capacités de validation grâce à des extensions (notamment `dbt_utils`) installées via *packages.yml*, permettant l'implémentation de contrôles plus avancés.

Le bloc de code suivant est un extrait commenté de l'un de ces fichiers, situé dans le dossier *"staging"*, et qui illustre le sérieux avec lequel les modèles ont été documentés et validés. Chaque colonne qui le nécessite bénéficie d'une explication claire de son rôle et format (ex. valeur de promotion exprimée en euros ou dates au format ISO[1]). Il est aussi possible d'observer les vérifications plus complexes évoquées ci-dessus. Par exemple, la validation de l'identifiant d'une promotion via une expression régulière garantit le respect du format 'P' suivi de trois chiffres. Le test de validité du pourcentage de promotion, configuré comme un avertissement plutôt qu'une erreur, montre une attention particulière aux besoins métier en permettant des promotions à zéro pour-cent sans bloquer le traitement. La contrainte de clé étrangère assure quant à elle la cohérence relationnelle avec la table des produits.

```yaml
version: 2

models:
  # [...]
  # Promotions ------------------------------------------------------
  # -----------------------------------------------------------------
  - name: Promotions
```

1. ISO 8601, `https://www.iso.org/iso-8601-date-and-time-format.html`

```
    tests:
    - dbt_utils.expression_is_true:
        # Format clé primaire P000
      expression: 'REGEXP_CONTAINS(id_promotion, r"^P\d{3}$")'
    - dbt_utils.expression_is_true:
        # Alerte en cas de promotion négative
      expression: 'pourcentage_promotion > 0'
      config:
        severity: warn
    columns:
      - name: id_promotion
        tests:
        - not_null # Garantie d'existence de la clé primaire
        - unique # Garantie d'unicite de la clé primaire
      - name: id_produit
        tests:
        - relationships:
            to: ref('Produits') # Controle de cle etrangere
            field: id_produit
      - name: valeur_promotion
        description: en euro
      - name: pourcentage_promotion
        description: promotion en pourcentage (float entre 0 et 1)
      - name: date_debut
        description: Date de debut de la promotion 'yyyy-mm-dd'
        tests:
        - not_null # Garantie d'existence d'une date de début
      - name: date_fin
        description: Date de fin de la promotion 'yyyy-mm-dd'
  # [...]
```

Cet effort visant à garantir une documentation des modèles de qualité et l'intégrité des données a porté ses fruits, détectant au plus tôt des erreurs dans les données brutes qui auraient pu faire boule de neige et entrainer des problèmes catastrophiques par la suite.

Enfin, l'automatisation des *build*s via Airflow a requis une configuration spécifique. Après avoir créé un environnement de production dédié dans DBT Cloud *(Figure 2.6)* – isolant ainsi les exécutions planifiées des développements en cours – nous avons configuré un *job* minimaliste exécutant la commande `dbt build`. Il assure la reconstruction complète des modèles tout en exécutant l'ensemble des tests décrits dans la section précédente. Pour permettre son déclenchement par Airflow, nous avons généré un *token* d'accès personnel (PAT) héritant des permissions de notre unique compte utilisateur DBT, puis étudié la documentation de l'API REST[1] de DBT Cloud.

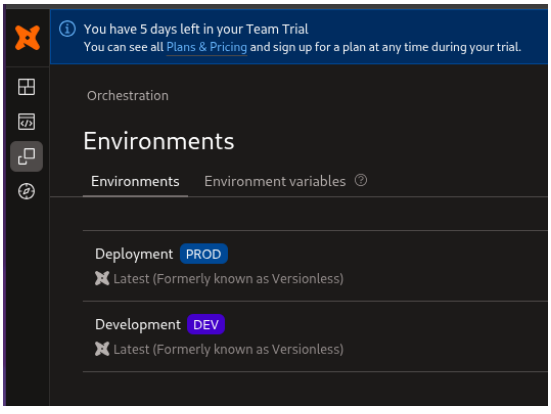


FIGURE 2.6 – Environements du projet DBT

1. Disponible sur le site de DBT, nous avons utilisé la version 2, `https://docs.getdbt.com/dbt-cloud/api-v2#/`

Conclusion

L'intégration de DBT dans notre pipeline de
données s'est globalement déroulée avec aisance, malgré les défis collaboratifs mentionnés précédemment. La structure modulaire du projet a permis à l'équipe de travailler simultanément sur différents modèles, augmentant ainsi notre productivité. Les modèles spécialisés du dossier *marts*, bien que complexes pour certains, ont permis une grande adaptabilité – la plupart ayant été fréquemment ajustés et optimisés pour s'adapter aux évolutions des besoins en visualisation de données et en Machine Learning.

DBT s'est donc imposé comme un composant central de notre pipeline. Son architecture modulaire garantit la maintenabilité du code, et que son approche basée sur SQL a facilité son adoption par l'équipe. La solution mise en place, bien qu'ayant révélé certaines limites, constitue une base opérationnelle, solide et prête à être adaptée aux besoins futurs.

2.4 Scripts Python - Machine Learning et Géolocalisation Clients

Cette section présente les modèles de Machine Learning développés en Python et intégrés au pipeline de données. Ces composants démontrent comment enrichir un pipeline ELT classique avec des fonctionnalités analytiques avancées. Chaque script suit une organisation similaire : extraction des données mises à jour depuis BigQuery, traitement et génération de nouvelles données, puis réinjection des résultats dans BigQuery.

Pour répondre au cahier des charges, et au regard des données disponibles, nous avons déployé trois algorithmes :

— Prévisions de chiffre d'affaires utilisant Prophet,
— Analyse des associations de catégories de produits via l'algorithme Apriori,
— Détection des causes de pannes machines avec XGBoost.

En complément, nous avons implémenté un système de géolocalisation des clients par adresse IP, en exploitant une API gratuite aux capacités limitées. Bien que secondaire, ce système soulève des défis techniques intéressants (mécanisme de *caching* et intégration d'une longue tâche dans un DAG Airflow, entre autres).

Les sections suivantes détailleront chaque composant, à l'exception du modèle de *Gradient Boosting* développé par un collaborateur. Les implémentations finales commentées sont disponibles [1].

1. cf. Annexe B, page 59

2.4.1 Prévision du Chiffre d'Affaires avec Prophet

L'utilisation de Prophet (développé par Facebook) pour la prévision du chiffre d'affaires s'est imposée principalement par à sa facilité de mise en œuvre et d'intégration dans notre pipeline. La bibliothèque propose une API claire et bien documentée, fonctionnant naturellement avec des *DataFrame*s Pandas, ce qui simplifie l'interfaçage avec nos processus d'extraction depuis BigQuery et de réinjection des résultats.

Son adoption large a permis de facilement trouver des exemples pour nous inspirer. Par ailleurs, l'interprétation aisée des résultats – présentés sous forme de décomposition en série de Fourier montrant la tendance globale, la saisonnalité annuelle et hebdomadaire – facilite l'analyse des résultats. La possibilité d'ajouter simplement des régresseurs externes a également été considérée dans notre évaluation, même si cette fonctionnalité n'a finalement pas été utilisée dans notre cas particulier. La flexibilité offerte par ses ajustements manuels ciblés permet une adaptation fine aux spécificités de nos données, comme nous le verrons dans l'analyse de code suivante. Enfin, sa rapidité d'exécution préserve la performance de notre pipeline Airflow. L'ensemble de ces caractéristiques ont fait pencher la balance en faveur Prophet plutôt que d'autres solutions plus modernes et peut-être plus performantes (Neural Prophet, Réseaux Neuronaux de type LSTM...) aussi considérées.

Analyse de Code

Les données de chiffre d'affaires sont préparées par un *mart* dédié [1]. Le chargement des données dans un *DataFrame* Pandas s'effectue donc par l'envoi d'une simple requête à BigQuery.

```
# imports standards retires pour plus de lisibilite
credentials = service_account.Credentials.from_service_account_file(
  os.path.dirname(__file__) + '/Compte Service Machine Learning.json',
)
sql = '''
  SELECT *
  FROM `carttrend_clean.mart_Prevision_Series_Temporelles_CA`
'''
df = pandas_gbq.read_gbq(
  sql,
  project_id="carttrend-4",
  credentials=credentials
)
# Conversion de 'dbdate' en 'datetime'
df['date'] = pd.to_datetime(df['date'])
```

L'entraînement du modèle et la génération des prédictions suivent une implémentation standard de Prophet.

```
from prophet import Prophet
# Initialisation du modele Prophet
model = Prophet()
```

1. cf. §A.10, page 57

```
# Entrainement du modele (il attend des noms de colonnes specifiques)
model.fit(df.rename(columns={'date': 'ds', 'chiffre_affaire': 'y'}))
# Creation d'un dataframe avec 30 jours supplementaires
future = model.make_future_dataframe(periods=30, freq='D')
# Prediction du chiffre d'affaire sur ces 30 jours
forecast = model.predict(future)
```

Les résultats peuvent facilement être visualisés *(Figure 2.7)*.

```
model.plot(forecast)
```

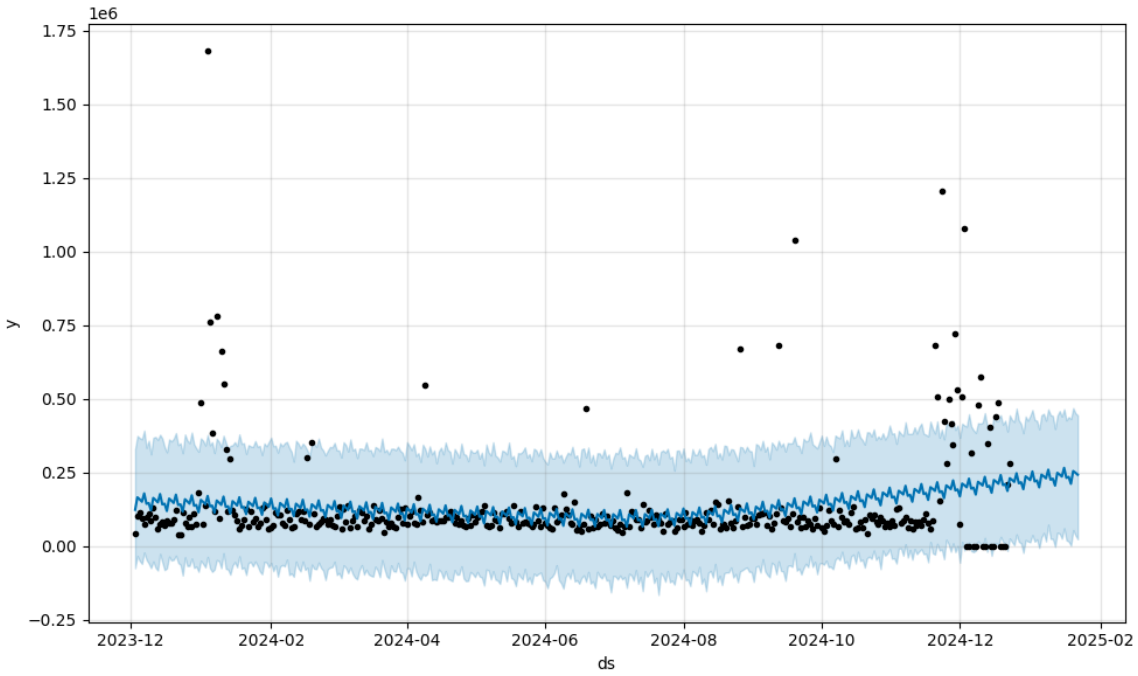


FIGURE 2.7 – Visualisation de l'analyse Prophet brute

L'analyse de ces résultats révèle des performances décevantes du modèle : intervalle de confiance très large et détection erronée d'une tendance haussière. L'examen des données montre en réalité une stabilité globale du chiffre d'affaires, perturbée par deux périodes atypiques :

— La première quinzaine de janvier,
— Entre fin novembre et mi-décembre.

Ces événements ne présentent pas de caractère saisonnier et se distinguent par leur irrégularité (pics de chiffre d'affaires, alternés avec des jours sans vente pour la deuxième période).

Pour améliorer les prédictions, nous avons appliqué l'une des stratégies recommandées dans la documentation de Prophet[1], en traitant ainsi ces périodes comme des vacances ponctuelles. Les résultats sont visibles en Figures 2.8 et 2.9.

```
vacances = pd.DataFrame([
```

1. Spécifiquement la partie *Treating COVID-19 lockdowns as a one-off holidays* de la section *Handling Shocks*, https://facebook.github.io/prophet/docs/handling_shocks.html

```
{
  # Premiere periode de pics, du 30/12/23 pour 20 jours
  'holiday': 'choc 1',
  'ds': pd.to_datetime('2023-12-30'),
  'lower_window': 0,
  'upper_window': 20,
},
{
  # Deuxieme periode de pics, du 20/11/24 pour 28 jours
  'holiday': 'choc 2',
  'ds': pd.to_datetime('2024-11-20'),
  'lower_window': 0,
  'upper_window': 28,
}
])

model = Prophet(holidays=vacances)
# Le reste du code est inchange
```

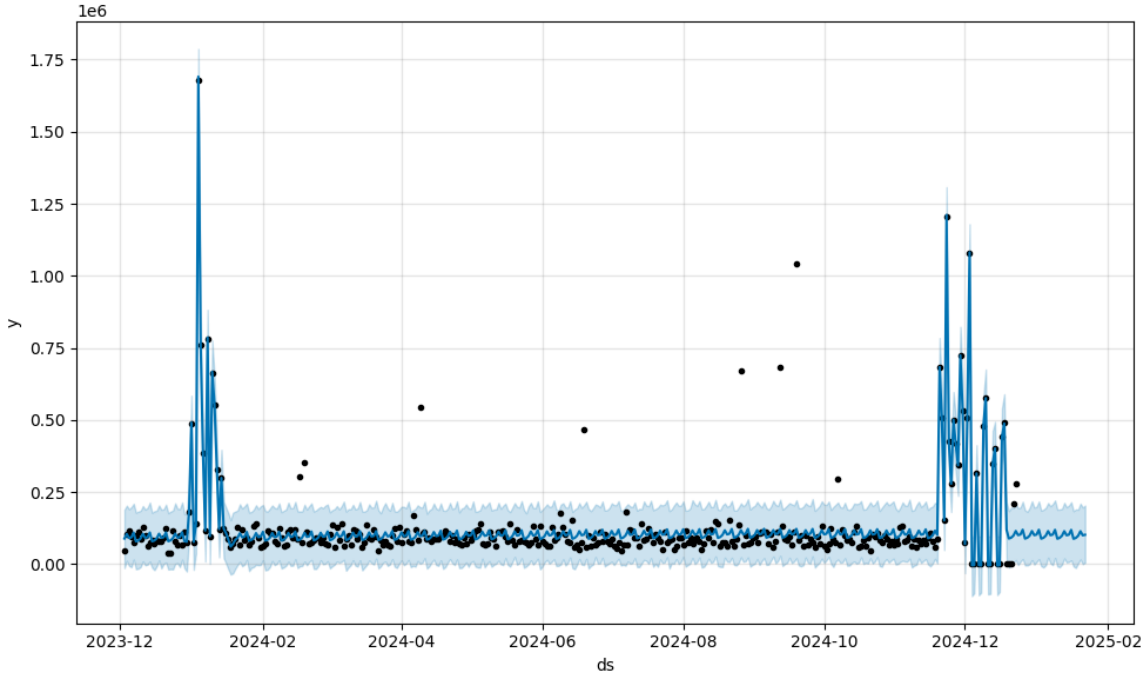


FIGURE 2.8 – Visualisation de l'analyse Prophet après ajustements

Les résultats après ajustement montrent des améliorations limitées. Bien que le modèle parvienne désormais à neutraliser correctement les deux périodes atypiques et à identifier une tendance globale plate plus conforme aux résultats attendus, ses performances restent globalement médiocres. L'intervalle de confiance reste large, reflétant l'incapacité du modèle à réaliser une prédiction fiable. La détection d'une légère saisonnalité hebdomadaire est plausible, mais trop faible pour être réellement utilisable.

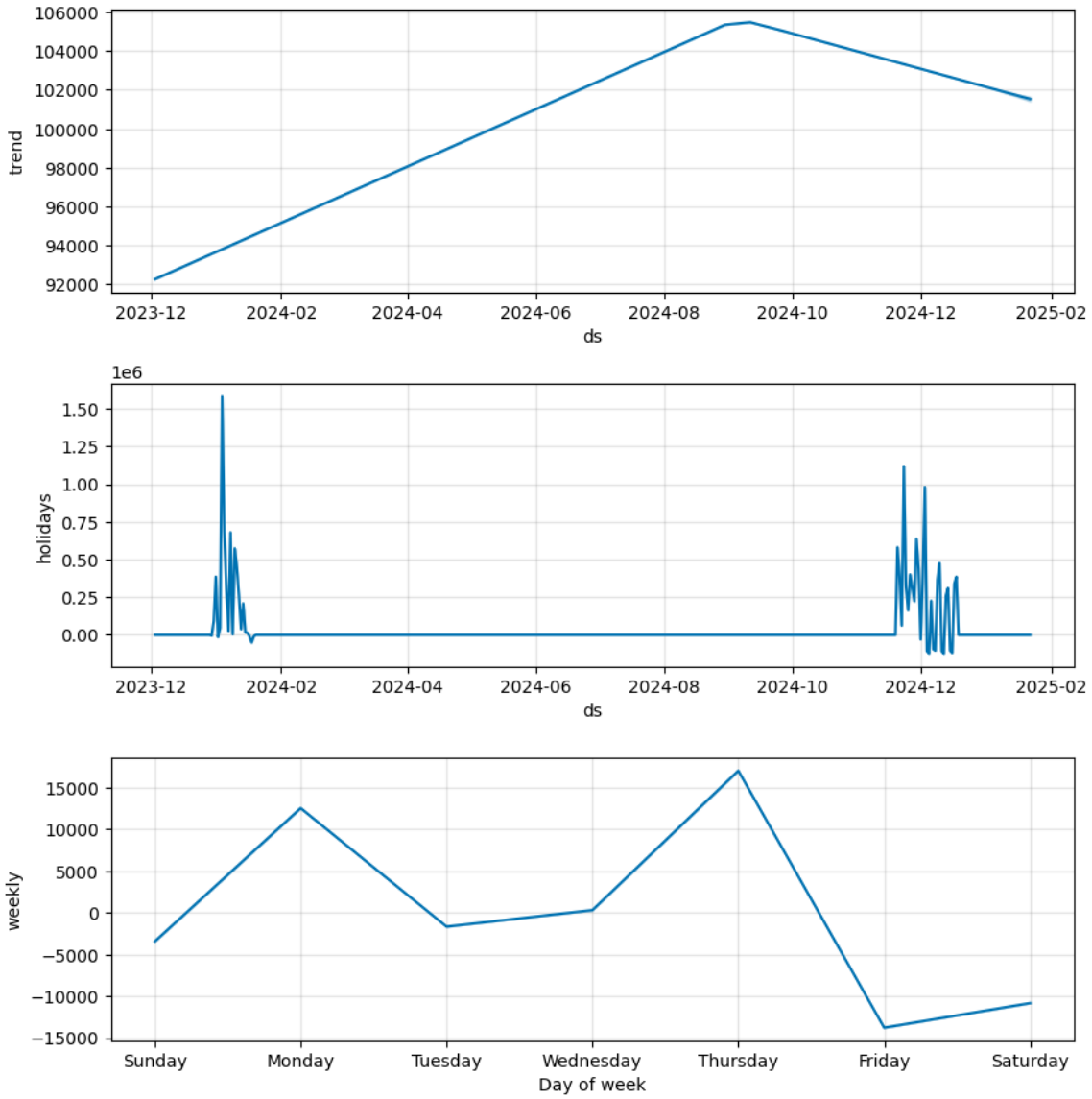


FIGURE 2.9 – Décomposition du chiffre d'affaires

Interprétation

Bien que techniquement correcte dans son implémentation, le modèle se révèle très décevant en terme qualité de prédiction. Cette contre-performance s'explique entièrement par les caractéristiques de nos données d'entrée. Généré aléatoirement pour les besoins de l'exercice, notre jeu de données présente des pics imprévisibles et une absence de tendances saisonnière. Pour la même raison, il est impossible de trouver des régresseurs externes pertinents pour améliorer la performance du modèle. Enfin, le volume historique disponible est limité (à peine plus d'une année de données).

Avec des données réelles, plusieurs pistes d'amélioration pourraient être explorées. L'intégration de régresseurs externes (campagnes marketing, indicateurs économiques généraux, vacances scolaires...) pourrait améliorer le modèle Prophet existant. Il serait aussi possible de

recourir à des algorithmes plus récents comme NeuralProphet ou, si des moyens techniques significatifs étaient disponibles, plus sophistiqués (réseaux neuronaux à mémoire par exemple LSTMs ou Transformers). Enfin, simplement attendre l'accumulation d'un volume suffisant de données historiques avant de déployer un système de prédiction pourrait aider à réaliser des prédictions plus fiables.

Cette expérience souligne avant tout le caractère purement académique de cet exercice. Un modèle aux performances aussi limitées ne devrait en aucun cas dépasser le stade exploratoire, et encore moins servir de base à des décisions stratégiques. Ce travail aura cependant permis de démontrer l'intégration technique complète de Prophet dans notre pipeline de données. Plus fondamentalement, il rappelle l'importance cruciale d'une analyse rigoureuse des données, tant en amont qu'en aval de toute modélisation.

2.4.2 Analyse des associations de catégories de produits avec l'algorithme Apriori

L'objectif initial de cette analyse était d'identifier des groupes de produits fréquemment achetés ensemble afin de pouvoir générer des recommandations pertinentes. Cependant, la structure particulière de nos données – chaque commande ne contient qu'un seul produit – a immédiatement posé problème pour une analyse classique des paniers d'achat. Face à cette contrainte, nous avons adapté notre approche en considérant l'ensemble des achats historiques par client comme un unique "panier étendu dans le temps". L'algorithme Apriori a alors été appliqué pour trouver des combinaisons de produits fréquemment associées au sein de ces historiques clients, même lorsque les achats étaient espacés dans le temps.

Malheureusement, cette analyse a été peu concluante, les métriques *support*, *lift* et *confidence* restant toutes décevantes. Cette impasse nous a conduits à modifier la granularité de notre analyse en appliquant Apriori au niveau des catégories de produits plutôt qu'aux produits individuels.

Analyse de Code

Comme pour les autres modèles, un *mart* dédié a été implémenté. Chaque ligne représente un *Array* SQL de catégories de produits achetés par client, dans l'ordre chronologique. L'obtention des données se fait toujours de la même manière.

```
# imports standards retires pour plus de lisibilite
credentials = service_account.Credentials.from_service_account_file(
  'Compte Service Machine Learning.json',
)
sql = '''
  SELECT *
  FROM `carttrend_clean.mart_Apriori_Combinaisons_Produits`
'''
```

```
df = pandas_gbq.read_gbq(
  sql,
  project_id="carttrend-4",
  credentials=credentials
)
df
```

```
  categorie
0 [Jouets, Mode, Sports, Beaute]
1 [Mode, Livres]
2 [Mode, Maison, Livres, Electronique, Beaute]
3 [Mode, Electronique, Livres, Sports]
4 [Mode, Maison, Sports]
```

Cette *dataframe* est ensuite transformée au format attendu par l'objet *apriori* d'*mlxtend*, soit une liste de liste de paniers. Celle-ci est ensuite encodée, puis le modèle est entrainé.

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# Transformation des donnees en format liste de listes
paniers = [list(row[0]) for row in df.itertuples(index=False)]

# Encodage des "paniers" au format one-hot
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
dfml = pd.DataFrame(te_ary, columns=te.columns_)

# Execution de l'algorithme Apriori avec seuil minimal de support
frequent_itemsets = apriori(dfml, min_support=0.1, use_colnames=True)
frequent_itemsets
```

```
  support   itemsets
0   0.122124  (Alimentation)
1   0.116323  (Beaute)
2   0.100320  (Jouets)
3   0.897580  (Livres)
4   0.234247  (Maison)
5   0.962392  (Mode)
6   0.638328  (Sports)
7   0.564513  (Electronique)
8   0.108822  (Livres, Alimentation)
9   0.117023  (Mode, Alimentation)
... ...         ...
```

À titre exploratoire, il est intéressant d'essayer d'extraire des règles d'association (ex : X est souvent acheté avant Y...)

```
from mlxtend.frequent_patterns import association_rules

rules = association_rules(frequent_itemsets, metric="lift", min_threshold
    =1)
rules.sort_values(by=['lift'], ascending=False)
rules
```

La sortie ci-dessous a été altérée pour augmenter la lisibilité.

```
    antecedents     consequents     ac supp  cs supp  supp    conf    lift
```

```
40  (Sports,...)    (Maison,...)    0.571    0.225    0.130    0.228    1.016
43  (Maison,...)    (Sports,...)    0.225    0.571    0.130    0.580    1.016
15  (Sports,...)    (Maison)        0.571    0.234    0.136    0.238    1.016
18  (Maison)        (Sports,...)    0.234    0.571    0.136    0.580    1.016
17  (Sports)        (Livres,...)    0.638    0.209    0.136    0.213    1.016
... ...             ...             ...      ...      ...      ...      ...
```

Les résultats n'étant que peu concluant (voir section suivante), et la question posée par l'énoncé ne portant que sur les catégories de produits achetées ensemble, il a finalement été décidé de limiter l'analyse aux associations courantes. Ces dernières ont été sélectionnées si elles étaient composées d'au moins deux éléments, et si elles jouissaient d'un support important. La visualisation sur le tableau de bord est enfin simplifiée en transformant la sortie du script en une table de chaines de caractères.

```
29 # Filtrage des resultats :
30 # - Combinaisons d'au moins 2 categories
31 # - Support minimum de 40%
32 frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(
33   lambda x: len(x)
34 )
35
36 frequent_itemsets = frequent_itemsets[
37   (frequent_itemsets['length'] >= 2) &
38   (frequent_itemsets['support'] >= 0.4)
39 ][['support', 'itemsets']]
40
41 # Formatage des resultats (frozenset -> string), facilite la visualisation
42 frequent_itemsets['itemsets'] = frequent_itemsets['itemsets'].apply(
43   lambda x: ' - '.join(list(x))
44 )
45
46 frequent_itemsets
```

```
    support   itemsets
13  0.864573  Mode - Livres
14  0.571214  Sports - Livres
15  0.507401  Livres - Electronique
19  0.613523  Mode - Sports
... ...         ...
```

Interprétation

Cette analyse visait initialement à identifier des combinaisons de catégories de produits fréquemment achetées ensemble pour mieux comprendre les comportements d'achat. L'application de l'algorithme Apriori a effectivement permis d'extraire certaines associations significatives en termes de support, comme les paires "Mode + Livres" (86,5%) ou "Sports + Livres" (57,1%), révélant ainsi des co-occurrences régulières dans les historiques d'achats. Cependant, une analyse plus approfondie des métriques d'association tempère considérablement ces résultats. Si certaines règles présentent une confiance élevée (comme "Électronique + Mode" à 96%), les valeurs de lift systématiquement proches de 1 indiquent en réalité une absence de corrélation significative entre les catégories.

Ici encore, l'analyse est limitée par nos données générées aléatoirement. Mais au-delà de ces résultats, cette étude révèle aussi les limites de performance de l'algorithme Apriori, auxquelles nous avons été confrontés. Pour exploiter un plus gros volume de données, l'investissement dans une puissance de calcul plus importante ou l'exploration d'un algorithme plus efficace, comme FP-Growth, pourrait s'avérer nécessaire. Sur le plan technique, ce travail a néanmoins validé l'intégration fonctionnelle d'Apriori dans notre pipeline. Il met aussi en lumière l'importance d'une analyse rigoureuse des métriques pour éviter les interprétations erronées.

2.4.3 Géolocalisation des Clients avec leur adresse IP

La géolocalisation des clients à partir de leur adresse IP n'était pas une exigence initiale du cahier des charges, mais elle représentait un défi technique intéressant ainsi qu'une illustration de l'intégration d'une nouvelle source de donnée au pipeline. Dans l'absence d'information directe sur la position géographique des clients dans les données, une API de géolocalisation gratuite utilisant leurs adresses IP a été utilisée. Toutefois, comme souvent avec les services gratuits, des limitations importantes étaient à prendre en compte, notamment un temps minimal entre deux requêtes fixé à 0,2 seconde (*rate limit*). Afin de respecter cette contrainte et pour éviter des appels redondants, un système de (*caching*) a été mis en place. Il permet de stocker localement les résultats préalablement obtenus pour une durée définie.

Analyse de Code

Les données sont directement prélevées sur la table Clients nettoyée, en sélectionnant les colonnes *id_client* et *ip*.

```
# imports standards retires pour plus de lisibilite
credentials = service_account.Credentials.from_service_account_file(
  'Compte Service Machine Learning.json',
)
sql = '''
  SELECT id_client, ip
  FROM `carttrend-4.carttrend_clean.Clients`
'''
df = pandas_gbq.read_gbq(
  sql,
  project_id="carttrend-4",
  credentials=credentials
)
df
```

```
    id_client  ip
0   C04367     2.30.247.144
1   C03411     179.123.200.87
2   C02668     101.132.250.223
3   C09565     83.70.211.128
4   C03532     120.217.38.16
... ...        ...
```

Le cache est implémenté en utilisant le module `pickle` de Python [1]. Beaucoup d'objets Python (et en particulier les `DataFrame` Pandas) peuvent être sérialisés de cette manière puis stockée sur le disque dur, pour être chargées en mémoire par la suite. Il est ainsi possible de connaître la date de dernière mise à jour du cache en consultant, via le système d'exploitation, la date de modification du fichier. Si elle est jugée trop ancienne (ici, supérieure à une semaine), le cache est ignoré (et ensuite écrasé par les nouvelles données, comme expliqué ci-après).

```
15 # Initialisation du DataFrame de résultats
16 client_geo = pd.DataFrame(columns=['id_client', 'pays', 'region', 'ville'])
17
18 # Gestion du cache
19 file_age = None
20 max_age = datetime.timedelta(days=7)  # Durée de validité du cache
21 if os.path.isfile(os.path.dirname(__file__) + '/client_geo.pkl'):
22   filetime = os.path.getmtime(
23     os.path.dirname(__file__) + '/client_geo.pkl'
24   )
25   file_age = datetime.timedelta(seconds=time.time()-filetime)
26   if file_age < max_age:  # Utilisation du cache si valide
27     client_geo = pd.read_pickle(
28       os.path.dirname(__file__) + '/client_geo.pkl'
29     )
```

On construit ensuite une `DataFrame` contenant uniquement les clients qui ne sont pas déjà présents dans le cache. Si ce dernier a été ignoré, la `DataFrame` "client_geo" est vide, auquel cas l'ensemble des clients est ajouté à "df_missing".

```
30 df_missing = pd.concat([df, client_geo]).drop_duplicates(
31   subset = ['id_client'],
32   keep=False
33 )
```

On appelle ensuite l'API ip2location pour remplir la `DataFrame` "client_geo". L'API ne permettant pas, à ma connaissance, de traitement en lot, une requête est effectuée pour chaque adresse IP (donc pour chaque client). Si le serveur ne répond pas ou si une erreur se produit, on ignore le client en cours (dont l'IP contient peut etre une erreur) et l'on essaye avec le client suivant, dans la limite de 5 tentatives.

```
34 retries = 5  # Nombre de tentatives en cas d'échec
35 for row in df_missing.itertuples(index=False):
36   try:
37     # Appel à l'API ip2location
38     response = requests.get(
39     f'https://api.ip2location.io/?key=7149995C158ABB38F6BCE6505B006082&ip={
         row.ip}&format=json',
40     timeout=10
41     )
42
43     # Si bonne reponse...
44     if response.status_code == 200:
45       data = response.json()
46       # ...ajout des données de géolocalisation aux résultats
```

1. `pickle` fait parti de la bibliothèque standard de Python, `https://docs.python.org/3/library/pickle.html`

```
      client_geo = pd.concat([
        client_geo,
          pd.DataFrame([[row.id_client,
          data.get('country_name'),
          data.get('region_name'),
          data.get('city_name')]],
          columns=client_geo.columns)
      ], ignore_index=True)
    # Sinon...
    else:
      # ...si il y a encore des tentatives disponibles...
      if retries:
        retries -= 1
        time.sleep(1)  # Attend plus longtemps en cas d'echec
        continue #...on continue avec le client suivant
      else:
        raise Exception(f'API error: {response.status_code}')

  except Exception as e:
    print(f"Error processing IP {row.ip}: {str(e)}")
    break

  time.sleep(0.2)  # Respect du rate limiting de l'API
```

Enfin, si le cache était obsolète, on l'écrase avec les nouvelles données.

```
if not file_age or file_age > max_age:
  client_geo.to_pickle(
    os.path.dirname(__file__) + '/client_geo.pkl'
  )
```

Interprétation

Les résultats obtenus, affichés dans le *dashboard* "Ventes - Produits" [1], correspondent bien aux attentes : les adresses IP des clients étant aléatoires dans les données, les pays les plus représentés ne reflètent pas une réalité commerciale, mais simplement la répartition mondiale des plages d'adresses IP. Les États-Unis et la Chine, qui disposent de larges blocs d'adresses, apparaissent donc très fréquemment dans les résultats. Par ailleurs, même avec des données plus réalistes, la précision de cette approche reste limitée. L'utilisation éventuelle par les clients de VPNs, les erreurs de l'API gratuite ou encore l'évolution constante de l'allocation des adresses IP peuvent facilement fausser les résultats.

D'un point de vue technique, ce script introduit aussi une contrainte non négligeable dans le pipeline : à chaque rafraîchissement complet du cache, il ajoute environ 0,2 seconde par client au temps d'exécution du DAG, en raison de la limitation imposée par l'API utilisée. Cette limitation aurait pu être partiellement contournée par une implémentation plus fine du cache : la table de sortie aurait pu inclure une date de dernière mise à jour pour chaque ligne dans une colonne supplémentaire. On aurait alors la possibilité de ne mettre à jour que les clients dont les données sont devenues obsolètes, grâce à cette granularité accrue. Cela n'aurait pas

1. cf. Annexe D.5, page 75

entièrement résolu le problème de performance, mais aurait permis de limiter la forte surcharge du pipeline lors de la reinitialisation complète des données. Cette approche aurait en outre l'avantage d'éviter le chargement d'un `pickle` depuis le disque dur, opération qui peut être problématique en termes de sécurité [1].

Au final, cet exercice reste majoritairement académique. Une entreprise comme Carttrend disposerait très probablement des adresses postales de ses clients (ne serait-ce que pour pouvoir honorer les commandes...), bien plus précises et robustes que les données dérivées d'IP. Toutefois, cette expérimentation constitue une bonne démonstration de l'intégration d'une source de données externe dans un pipeline de traitement, avec les adaptations techniques et les précautions que cela implique.

Conclusion

Dans l'ensemble, les travaux réalisés autour des scripts Python pour l'analyse prédictive, l'exploration des comportements d'achat et la géolocalisation des clients ont permis de valider des aspects techniques clés du pipeline, tout en mettant en évidence les limites imposées par la nature synthétique des données utilisées. Le modèle de prévision basé sur Prophet, a montré des performances très faibles, l'analyse des associations de produits n'a pu produire que des résultats peu pertinents, révélant l'absence de signal réel dans les données. La géolocalisation des clients, bien qu'extérieure au périmètre du projet, a illustré l'intégration d'un service externe dans un pipeline de données. Là encore, la nature artificielle des données limite toute interprétation opérationnelle.

Un modèle de régression basé sur XGBoost a également été développé par un collaborateur pour tenter d'identifier les principales causes des pannes machines à partir de variables explicatives telles que la localisation de l'entrepôt, la température ou le volume de produits traités. Là encore, les résultats se sont révélés très peu probants, avec une erreur (RMSE) élevée, soulignant le caractère académique de l'exercice et l'absence de lien causalité entre les variables générées aléatoirement.

Dans un contexte réel, une étape de validation automatisée des modèles serait recommandée avant toute intégration des résultats dans BigQuery. Des métriques de performance (RMSE, précision, *recall*, etc.) pourraient être évaluées systématiquement, et seuls les résultats satisfaisants seraient envoyés vers le *data warehouse*. Cela éviterait de présenter à l'utilisateur final des prédictions et analyses peu fiables (comme celles générées dans ce projet). Cette précaution garantirait que seules les analyses pertinentes alimentent les *dashboards*.

Au final, ces travaux rappellent de manière transversale que la qualité des données conditionne directement la pertinence des analyses, et que les algorithmes, aussi perfectionnés soient-

1. Un utilisateur mal intentionné ayant accès au serveur Airflow pourrait théoriquement modifier le fichier `pickle` de telle manière à lui permettre une exécution de code arbitraire avec les droits d'accès octroyés au DAG. Explications et exemples dans un article de Nicolas Lara : `https://lincolnloop.com/blog/playing-pickle-security/`

ils, ne peuvent produire de résultats robustes que s'ils s'appuient sur des fondations solides. Sur le plan pédagogique et technique, ce travail à cependant rempli ses objectifs : il démontre l'intégration de plusieurs briques analytiques dans un pipeline cohérent, avec une attention spéciale portée à la validation et l'interprétation des résultats.

2.5 Orchestration avec Airflow

L'ensemble des traitements développés dans ce projet repose sur une orchestration centralisée assurée par Apache Airflow. Ce composant joue un rôle crucial dans l'automatisation, la planification et le suivi de l'exécution des différentes tâches du pipeline de données. Il permet d'enchaîner les étapes de manière contrôlée, tout en assurant la gestion des dépendances et des erreurs éventuelles.

Le DAG principal mis en place regroupe six tâches, reflétant les différentes étapes de la chaîne analytique. Il commence par déclencher le transfert de données Fivetran, qui alimente le *data warehouse* en données brutes. Une fois ce transfert terminé, Airflow déclenche la commande `dbt build`, qui applique les transformations SQL pour construire les *marts* nécessaires aux analyses. Viennent enfin la tâche de géolocalisation des clients et les trois tâches distinctes de Machine Learning : la prévision du chiffre d'affaires (Prophet), l'analyse des causes de panne des machines (XGBoost), et l'analyse des comportements d'achat (Apriori).

Cette orchestration garantit un traitement complet des données, du chargement initial dans le *data warehouse* jusqu'à leur enrichissement final.

2.5.1 Analyse de code : DAG

Le DAG est implémenté de manière à simplifier l'intégration de nouvelles tâches. Celles-ci correspondent en effet à un simple appel à leur fonction respective. Chacune de ces fonctions appelle ensuite un script Python externe : le processus principal donne naissance à un sous-processus chaque tâche, à l'aide de la fonction `run()` du module `subprocess`. Le DAG commence donc par la définition de ces fonctions.

```
1  from airflow import DAG
2  from airflow.operators.python import PythonOperator
3  from datetime import datetime, timedelta
4  import subprocess
5
6  def start_fivetran_sync():
7    # Déclenche la synchronisation des données sources via Fivetran
8    print("Starting Fivetran Sync Task")
9    subprocess.run(['python3', '../../airflow/scripts/start_fivetran_sync.py'
         ])
10
11 def start_dbt_build():
```

```
12   # Lance les transformations DBT (staging + marts)
13   print('Starting DBT Build Job')
14   subprocess.run(['python3', '../../airflow/scripts/start_dbt_build_job.py'
         ])
15
16 def start_ml_prediction_ca():
17   # Exécute le modèle Prophet de prévision de CA
18   print('Starting ML Prediction CA')
19   subprocess.run(['python3', '../../airflow/scripts/ml_prediction_ca.py'])
20
21 def start_ml_a_priori():
22   # Exécute l'analyse des associations produits
23   print('Starting ML APriori')
24   subprocess.run(['python3', '../../airflow/scripts/ml_a_priori.py'])
25
26 def start_ml_regression_machines():
27   # Exécute le modèle XGBoost d'analyse des pannes
28   print('Starting ML Regression Machines')
29   subprocess.run(['python3', '../../airflow/scripts/ml_regression_machines.
         py'])
30
31 def get_customer_locations():
32   # Appel l'API ip2location pour géolocaliser les clients
33   print('Starting Customer Location API Call Task')
34   subprocess.run(['python3', '../../airflow/scripts/get_customer_location.
         py'])
```

L'objet `DAG` est ensuite instancié avec les paramètres appropriés. Ici, le DAG est programmé pour être déclenché tous les jours à minuit. En cas d'échec, une seconde tentative est effectuée 30 minutes plus tard. Si celle-ci échoue à nouveau, un e-mail d'alerte est envoyé à l'adresse spécifiée, grâce à la configuration préalable du protocole SMTP.

```
35 default_args = {
36   'owner': 'airflow',
37   'start_date': datetime(2025, 5, 27),
38   'schedule_interval': '@daily',  # Exécution quotidienne à minuit
39   'retries': 1,  # Nombre de tentatives en cas d'échec
40   'retry_delay': timedelta(minutes=30),  # Délai entre les tentatives
41   'email_on_failure': True,  # Notification par email (configuration SMTP
         prealable)
42   'email': ['carttrenddata4@gmail.com']  # Destinataires
43 }
44
45 # Initialisation du DAG
46 dag = DAG(
47   'carttrend_data_pipeline',  # Nom du DAG
48   default_args=default_args,
49   description='Pipeline ELT/ML complet pour CartTrend',
50   catchup=False  # Désactive les runs rétroactifs
51 )
```

Les tâches sont ensuite définies. Chacune d'entre elles correspond à l'une des fonctions décrites ci-dessus. Le code ci-dessous est volontairement abrégé pour plus de lisibilité, mais la version complète est fournie [1].

1. cf. Annexe C.1, page 65

```
t1 = PythonOperator(
  task_id="start_fivetran_sync",
  python_callable=start_fivetran_sync,
  dag=dag,
)

t2 = PythonOperator(
  task_id="start_dbt_build",
  python_callable=start_dbt_build,
  dag=dag,
)

t3 = PythonOperator(
  task_id="start_ml_prediction_ca",
  python_callable=start_ml_prediction_ca,
  dag=dag,
)
# ...
```

Enfin, les dépendances entre les tâches sont définies. Le pipeline commence systématiquement par t1 (extraction et chargement des données brutes), suivi de t2 (transformation via DBT). Ensuite, les trois modèles de Machine Learning (t3, t4 et t5) ainsi que la géolocalisation des clients (t6) peuvent s'exécuter en parallèle.

```
t1 >> t2
t2 >> [t3, t4, t5, t6]
```

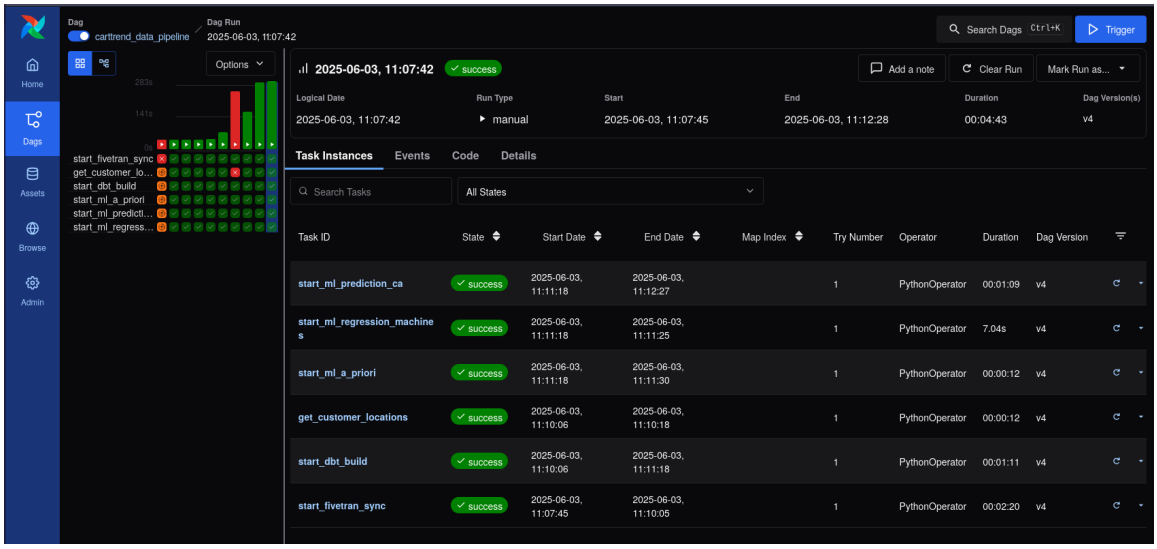


FIGURE 2.10 – Six carrés verts, le DAG s'execute avec succès !

L'architecture retenue pour ce DAG présente plusieurs avantages importants. En déléguant chaque tâche à un script Python externe via `subprocess.run()`, elle permet une évolution simple et modulaire du pipeline : l'ajout d'une nouvelle étape se résume à l'écriture d'un script autonome, suivi de l'ajout d'une fonction et d'une tâche dans le DAG. Cette approche favorise la séparation des responsabilités et facilite la maintenance du code sur le long terme.

Cette structure a toutefois quelques limitations. Le fait d'exécuter chaque étape dans un

sous-processus isolé rend plus difficile la transmission d'informations entre tâches. Par exemple, il n'est pas possible d'utiliser directement le mécanisme de communication natif d'Airflow (XCom). Ce manque peut cependant être partiellement contourné : l'une des possibilités est d'utiliser des `print()` dans les scripts appelés, et de capturer les sorties standard pour les analyser ou les *logguer* depuis le DAG. Ce type de contournement nécessite néanmoins un peu plus d'effort de mise en œuvre si des échanges de données plus complexes sont envisagés.

Par ailleurs, l'ordonnancement défini permet une exécution parallèle de plusieurs traitements indépendants, notamment les trois modèles de Machine Learning et la géolocalisation. Cela optimise le temps d'exécution global du pipeline, en tirant parti de la capacité d'Airflow à gérer plusieurs tâches en parallèle.

2.5.2 Analyse de code : *start_dbt_build_job.py*

start_dbt_build_job.py partage avec le script *start_fivetran_sync.py* une logique similaire : appeler un service externe, attendre la fin de son exécution, et signaler toute erreur en levant une exception. Dans ce rapport, seule l'analyse du premier est détaillée ; l'autre, quasiment identique dans sa conception, est disponible [1].

start_dbt_build_job.py est d'abord de déclencher l'exécution du job DBT configuré précédemment (cf. §Configuration avancée, page 18). Ce dernier va appliquer les modèles de transformation SQL définis dans le projet. Le script attend ensuite patiemment que le processus se termine, vérifie le code de retour, et lève une exception explicite en cas d'échec ou d'absence de réponse, permettant ainsi à Airflow d'identifier correctement une erreur et de réagir selon la configuration du DAG.

Le script commence par les imports nécessaires et la définition de quelques variables globales :

```
1  import requests
2  import time
3  import enum
4
5  # Configuration d'accès à l'API DBT Cloud
6  # Tout ces identifiants ont ete crees sur l'interface DBT Cloud
7  ACCOUNT_ID = '70471823462199' # Identifiant du compte DBT
8  JOB_ID = '70471823467186' # Identifiant du job
9  API_KEY = 'dbtu_57j1bQUrnaKY_XfO3UdF2kHC9g4DkgZNAmyAy2D3QAgQmxIqiM' # Token
       d'API
10
11 # Enumeration des statuts possibles d'un job DBT
12 # Les codes sont documentés sur la doc DBT :
13 # https://docs.getdbt.com/dbt-cloud/api-v2#/operations/List%20Runs
14 class DbtJobRunStatus(enum.IntEnum):
15   QUEUED = 1
16   STARTING = 2
17   RUNNING = 3
```

1. cf. Annexe C.2, page 67

```
18   SUCCESS = 10
19   ERROR = 20
20   CANCELLED = 30
```

Une fonction est ensuite définie pour déclencher le *job* via l'API de DBT. Cette API fourni l'identifiant unique de l'exécution, que la fonction récupère et renvoie.

```
21 def trigger_job():
22   url = f'https://jp371.us1.dbt.com/api/v2/accounts/{ACCOUNT_ID}/jobs/{
         JOB_ID}/run/'
23   headers = {'Authorization' : f'Token {API_KEY}'}
24   payload = {'cause' : 'Triggered from Airflow'}
25
26   # Envoi de la requete POST a l'API DBT
27   response = requests.post(url, json=payload, headers=headers)
28   response.raise_for_status() # Si echec, leve une exception
29
30   # Retourne l'ID d'exécution
31   job_run_id = response.json()['data']['id']
32   return job_run_id
```

Une deuxième fonction est déclarée pour interroger l'état du *job* en cours, toujours via l'API DBT.

```
33 def get_job_status(job_run_id):
34   url = f'https://jp371.us1.dbt.com/api/v2/accounts/{ACCOUNT_ID}/runs/{
         job_run_id}/'
35   headers = {'Authorization': f'Token {API_KEY}'}
36
37   # Envoi de la requete GET a l'API DBT
38   response = requests.get(url, headers=headers)
39   response.raise_for_status()
40
41   # Retourne le statut de l'execution
42   return response.json()['data']['status']
```

Le point d'entrée principal du script se trouve dans le bloc `__main__`. La fonction `trigger_job()` est appelée pour lancer le job, et retourne l'identifiant de son exécution. Cet identifiant est ensuite passé à la fonction `get_job_status()`, appelée régulièrement (toutes les trois secondes) pour suivre l'avancement du job.

Il est important de noter qu'un échec temporaire à ce deuxième appel à l'API DBT (ex. *timeout* ou erreur réseau) ne signifie pas forcément que le *job* a échoué. C'est pourquoi un mécanisme de *retries* est mis en place : si la récupération du statut échoue plus cinq fois, le script considère qu'il est impossible de déterminer le résultat du job et lève une exception.

```
43 if __name__ == '__main__':
44   retries = 5 # Nombre maximum de tentatives d'obtention du statut
45
46   # Phase 1: Déclenchement du Job
47   job_run_id = trigger_job()
48   print(f'DBT Build Job Triggered : {job_run_id}')
49
50   # Phase 2: Verification periodique du statut du Job
51   while True:
52     # Demande l'etat du transfer a Fivetran
```

```
53      try:
54        status = get_job_status(job_run_id)
55
56      # Lève une exception si la demande de statut n'aboutit pas 5 fois
57      except Exception as e:
58        print(f'Could not get job status : {e} - Retries : {retries}')
59        if retries:
60          retries -= 1
61        else:
62          raise Exception('Could not get job status')
63
64      # Si on a une réponse de l'API...
65      else:
66        # ...et qu'il est fini : stop.
67        if status == DbtJobRunStatus.SUCCESS:
68          break
69        # ...et qu'il s'est produit une erreur : exception.
70        elif status == DbtJobRunStatus.ERROR \
71          or status == DbtJobRunStatus.CANCELLED:
72          raise Exception("Job Failed !")
73
74      # ...attendre 3 secs et recommencer.
75      time.sleep(3)
76
77  print('Job Finished')
```

Ce script assure le déclenchement fiable d'un *job* DBT à distance, avec un suivi en temps réel de son exécution. Il gère correctement les erreurs, ce qui le rend tout à fait adapté à son intégration dans notre environnement orchestré par Airflow.

Conclusion

L'intégration d'Airflow dans ce projet s'est révélée être un choix pertinent, notamment en termes de flexibilité et de centralisation de l'orchestration du pipeline de données. Chaque étape – extraction, transformation et enrichissement externe – est orchestrée de manière claire et séquencée, tout en permettant l'exécution parallèle des traitements indépendants. Cette orchestration aurait été plus approximative et moins facile à maintenir en cas d'utilisation des alternatives potentielle évoquées en premiere partie (§Orchestration - Airflow, page 6).

L'architecture retenue, fondée sur l'appel de scripts Python externes, est orientée vers l'évolutivité. Ajouter une tâche ou en modifier une existante ne nécessite qu'un ajustement minimal dans le DAG. En contrepartie, cette approche demande plus d'attention pour la gestion des erreurs, des logs ou encore des transferts de résultats intermédiaires.

Enfin, l'utilisation d'Airflow en mode *standalone*, avec une base SQLite et un exécuteur séquentiel, entraine certaines limitations comme l'absence d'une véritable parallélisation des tâches et une réduction des capacités de gestion des volumes importants. Ces contraintes, pas réellement limitantes dans le contexte de cet exercice, n'existeraient de toute manière pas dans un environnement de production, où une configuration soigneuse avec un *scheduler* dédié, un

exécuteur parallèle et une base de données robuste (PostgreSQL, par exemple) aurait été réalisée.

Conclusion

Cette deuxième partie a permis de détailler, étape par étape, la mise en œuvre technique du pipeline de données. Tout d'abord, l'infrastructure a été posée avec la configuration de BigQuery, utilisée comme entrepôt de données central. Fivetran assure ensuite l'extraction et le chargement des données brutes issues de sources variées. Ces données sont alors structurées, nettoyées et enrichies au moyen de modèles DBT, en facilitant ainsi l'utilisation postérieure. Des scripts Python ont ensuite été intégrés pour effectuer plusieurs traitements spécifiques : prédiction des ventes, analyse des causes de pannes, extraction d'associations fréquentes, ainsi qu'un enrichissement géographique basé sur les adresses IP. Enfin, l'ensemble du processus a été orchestré avec Airflow, garantissant un déclenchement cohérent, un suivi centralisé du pipeline et une exécution efficace des tâches.

Certaines améliorations apparaissent prioritaires si un passage en production était envisagé : remplacer les quelques utilisations d'OAuth par des comptes de service, adopter une configuration Airflow plus robuste (avec exécuteur parallèle et base de données adaptée), et revoir l'architecture des modèles DBT pour la rendre plus modulaire, lisible et maintenable. Cependant, le pipeline construit est dès à présent fonctionnel, adapté aux objectifs pédagogiques du projet, et facilement extensible. Chaque composant peut être remplacé ou amélioré indépendamment des autres, ce qui en fait une base robuste pour de futures évolutions.

La partie suivante s'appuie sur ce pipeline pour visualiser les données dans Tableau, illustrant ainsi l'importance d'un traitement amont rigoureux pour pouvoir construire des tableaux de bord fiables et fluides.

Chapitre 3

Tableaux de bords, analyse et recommandations

L'objectif principal de cette phase du projet a été de concevoir une série de *dashboards* interactifs dans Tableau, permettant une lecture claire, rapide et utile des données pour les utilisateurs finaux. L'accent a été mis sur la simplicité d'utilisation au quotidien, avec des visuels immédiatement compréhensibles et des filtres pertinents, adaptés aux besoins des différents métiers. Un effort particulier a été porté sur l'harmonisation visuelle entre les différents tableaux de bord : couleurs, respect d'une charte graphique, positionnement des filtres et légendes... Ces choix esthétiques, souvent perçus comme secondaires, contribuent pourtant fortement à l'adhésion des utilisateurs et à l'efficacité de l'analyse.

Afin d'éviter de submerger l'utilisateur d'informations, chaque *dashboard* a été volontairement limité à un nombre réduit de visualisations. Néanmoins, des filtres interactifs permettent d'explorer les données plus en profondeur, en adaptant dynamiquement les graphiques affichés selon la période, la catégorie de produits, ou encore d'autres dimensions pertinentes. Cette structure permet de répondre à deux besoins souvent contradictoires : une lecture rapide des indicateurs clés, et une capacité d'analyse plus poussée pour les utilisateurs avancés.

L'intégration des données dans Tableau a été grandement facilitée par l'utilisation des *mart* préparés dans DBT, offrant une structure prête à l'emploi pour la visualisation (un exemple est détaillé en page 16, d'autres sont disponibles [1]). La connexion au *data warehouse* s'est faite facilement (mais pas forcément idéalement) par OAuth. La flexibilité du pipeline a permis des mises à jour régulières des *marts*, adaptant ainsi les données aux besoins évolutifs de la visualisation sans nécessiter de modifications profondes.

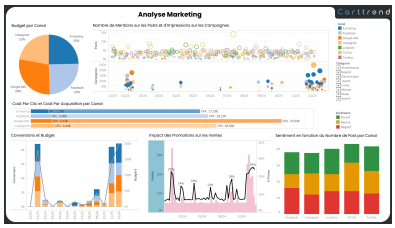
Au total, cinq *dashboards (Figure 3.1)* ont été conçus pour couvrir quatre fonctions métiers principales : deux pour les ventes, un pour la satisfaction client, un pour la logistique et un dernier dédié au marketing. Chacun répond à des problématiques spécifiques énumérées dans

1. cf. Annexe A.3, page 54

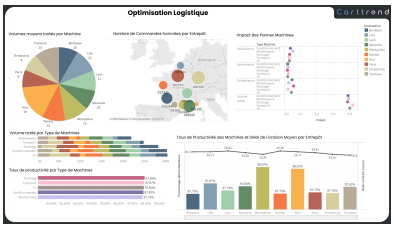
l'énoncé du problème. Des images haute résolution sont proposées[1], mais véritable valeur résidant dans leurs filtres interactifs et pour en mesurer tout le potentiel, le lecteur est invité à consulter les animations vidéo disponibles via les liens ci-dessous.

— Marketing : `https://tinyurl.com/yc4eyf7k`
— Optimisation Logistique : `https://tinyurl.com/mwk9444d`
— Satisfaction Client : `https://tinyurl.com/zbcdjkfb`
— Ventes - Chiffres : `https://tinyurl.com/44tb7eck`
— Ventes - Produits : `https://tinyurl.com/2s3dv3t4`

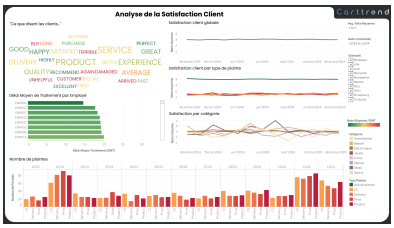
L'ensemble des tableaux de bord a fait l'objet d'une étroite collaboration entre les participants, mais nous nous concentrerons ici sur les deux dédiés aux ventes dont l'auteur était plus particulièrement responsable. Le premier se concentre sur les indicateurs financiers globaux, tandis que le second offre une granularité plus importante, proposant notamment une analyse par produits et pays.



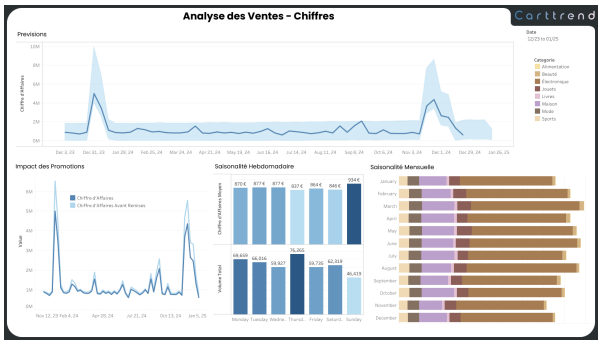
(a) Marketing



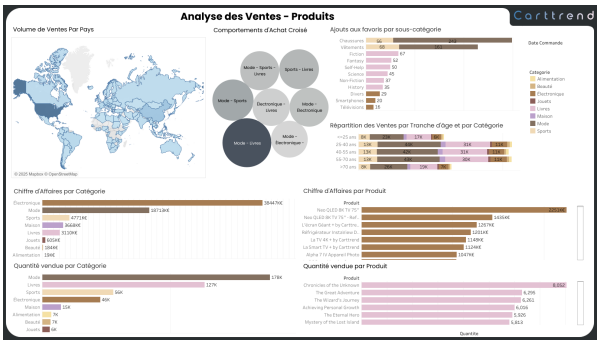
(b) Optimisation Logistique



(c) Satisfaction Client



(d) Ventes - Chiffres



(e) Ventes - Produits

FIGURE 3.1 – Vue d'ensemble des *dashboards* réalisés

3.1 Tableaux de Bord

3.1.1 Ventes - Chiffres

Ce premier tableau de bord vise à fournir une vue d'ensemble claire de la performance commerciale à travers le suivi du chiffre d'affaires (CA) et de ses évolutions. L'élément central

1. cf. Annexe D, page 70

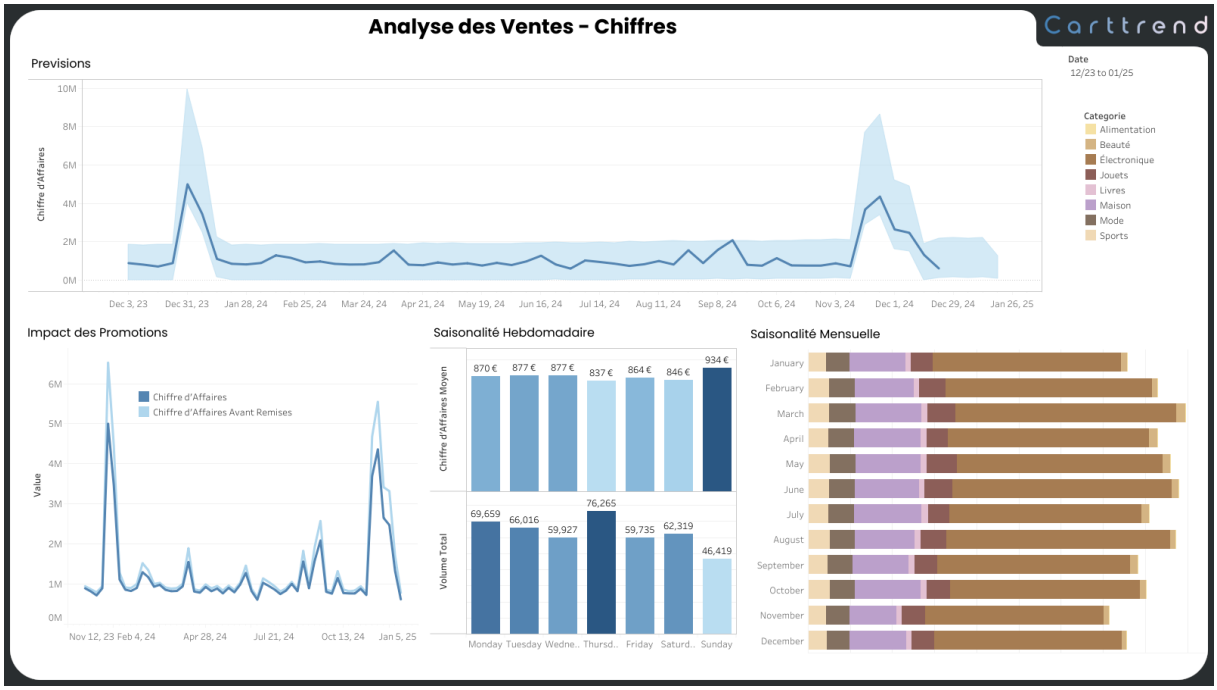


FIGURE 3.2 – Ventes - Chiffres

du *dashboard* est un graphique combiné présentant le CA historique ainsi qu'une projection de son évolution future. La prévision, quant à elle, n'est volontairement pas affichée sous forme chiffrée. Ce choix découle de la qualité jugée insuffisante du modèle prédictif [1], qui aurait pu induire les utilisateurs en erreur quant à la fiabilité des projections. Elle est donc représentée uniquement par un intervalle de confiance au sein duquel la prévision se situe. Cet intervalle est également affiché pour les données historiques, permettant ainsi de se faire une idée visuelle des performances du modèle.

Tous les graphiques présents dans ce *dashboard* sont filtrables selon la période temporelle souhaitée. L'utilisateur peut ainsi facilement isoler une plage de dates spécifique, par exemple pour observer l'impact d'actions commerciales ciblées.

Des filtres supplémentaires viennent enrichir l'analyse, permettant de croiser les données selon différents axes : catégorie de produit, mois de l'année, ou jour de la semaine. Ces filtres ne s'appliquent toutefois pas au graphique de prévision, qui ne bénéficie pas des mêmes possibilités d'interaction. Cette limitation est liée à l'architecture technique du pipeline, qui ne permet pas l'exécution en temps réel du modèle de prévision sur des données filtrées dynamiquement par l'utilisateur. Une solution plus flexible – consistant à intégrer le modèle dans un champ calculé exécuté à la volée – a bien été présentée par un développeur de Tableau lors d'une conférence technique [2], mais elle est inapplicable dans notre environnement Tableau Cloud. Cela limite donc l'interactivité de ce graphique par rapport au reste du *dashboard*.

1. cf. §2.4.1, page 24
2. Conférence donnée par Nathan Mannheimer le 26 oct 2018. Enregistrement disponibe : `https://youtu.be/nRtOMTnBz_Y`

3.1.2 Ventes - Chiffres

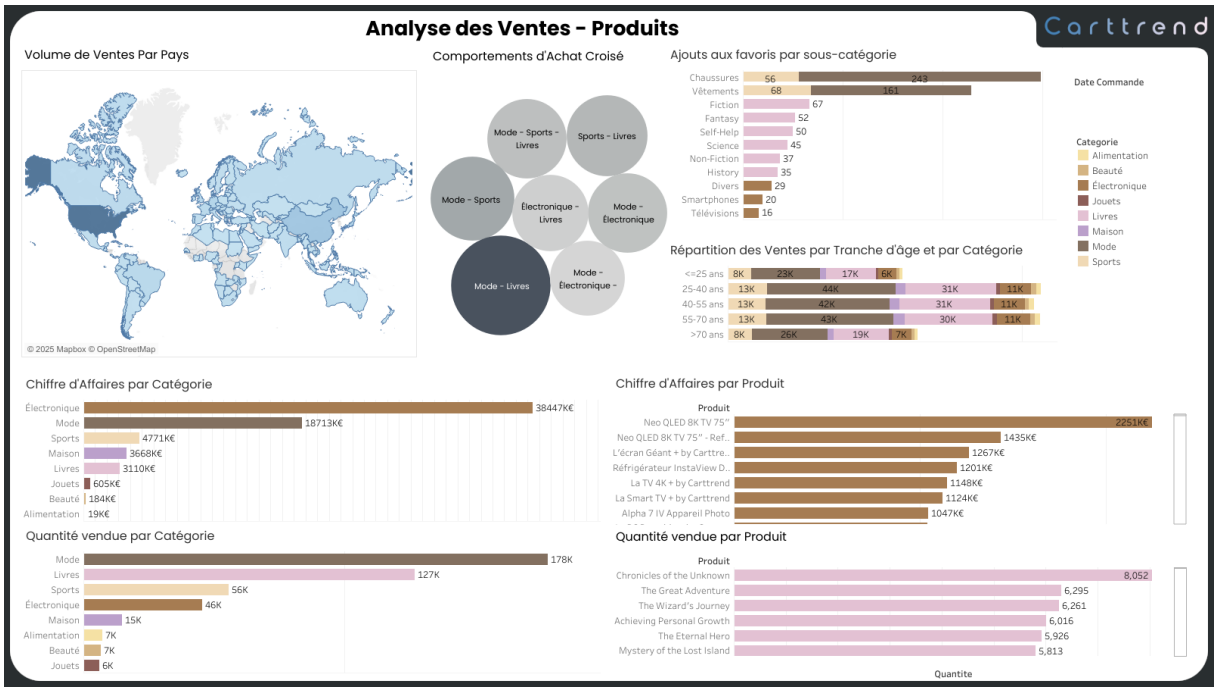


FIGURE 3.3 – Ventes - Produits

Ce deuxième tableau de bord a été conçu pour offrir une analyse plus détaillée des performances commerciales, permettant aux utilisateurs d'explorer les données sous des nouveaux angles. L'ensemble des visualisations réagit là encore dynamiquement aux filtres appliqués, avec des options étendues par rapport au premier dashboard. Outre les filtres temporels et par catégorie de produits, une carte (en haut a gauche) permet un filtre géographique par pays. Cette fonctionnalité facilite l'identification rapide de tendances régionales, bien que la fiabilité des données de géolocalisation (issues d'adresses IP) soit à considérer avec prudence [1].

Le graphique dédié aux comportements d'achat croisés a été délibérément laissé assez vague, en raison des performances médiocres du modèle [2]. La encore, plutôt que d'afficher des résultats peu fiables, la visualisation se concentre sur des représentations qualitatives, évitant ainsi de fournir des *insights* potentiellement trompeurs. La moitié inférieure du *dashboard* propose quant à elle quatre graphiques qui peuvent chacun être réorganisés dynamiquement du plus bas au plus haut. Cette approche permet d'identifier instantanément les meilleures et moins bonnes performances selon différents critères (CA par produit ou catégorie de produits, volume des ventes, etc.), offrant ainsi une vue synthétique et intuitive des *"top"* et *"flop"*. Le graphique en haut à droite est enfin notable par l'utilisation de barres colorées par sous-catégories de produits (par exemple, la catégorie chaussures est séparée en "sport" et "mode"), permettent de visualiser la répartition des favoris au sein des grandes catégories.

1. cf. §2.4.3, page 30
2. cf. §2.4.2, page 27

3.2 Analyse et Recommandations

Les analyses menées dans ce projet se heurtent à une limite fondamentale : la nature aléatoire des données synthétiques utilisées. Cette caractéristique a rendu impossible l'obtention *d'insights* réellement pertinents. Les visualisations ont donc été conçues avec une approche prudente, présentant les données de manière impartiale, et évitant au maximum de forcer des conclusions non soutenues par les informations disponibles. Par exemple, certains graphiques sont délibérément laissés vagues et sans chiffres précis, et les axes ont rarement été tronqués. Cette retenue dans l'interprétation constitue un choix méthodologique conscient.

Il serait en outre très inapproprié de formuler des recommandations opérationnelles concrètes dans ce contexte. Avec un accès limité à des données peu fiables et sans possibilité de consulter l'expertise métier des équipes concernées, toute suggestion concernant par exemple la maintenance des machines ou les stratégies commerciales serait présomptueuse. Cependant, et dans le but exclusivement pédagogique de répondre à l'exercice, certaines observations été réalisées et des recommandations ont été formulées en conséquence. Elles illustrent comment les *dashboards* pourraient orienter des décisions stratégiques.

3.3 Constats et Recommandations

Premièrement, les données semblent indiquer une possible corrélation entre les performances commerciales et les périodes promotionnelles, avec des volumes plus importants en fin d'année. Pour atténuer cette dépendance, des programmes de fidélité pourraient être envisagés, combinés à des événements commerciaux ciblés pendant les périodes creuses et une adaptation des promotions selon les jours de la semaine.

De plus, on observe des disparités entre les catégories de produits. L'électronique semble présenter un ratio CA/volume différent des autres catégories. Cette catégorie pourrait donc présenter une opportunité de *cross-selling*, tandis qu'un rééquilibrage des investissements marketing vers les catégories moins performantes pourrait d'être envisagé.

Certaines combinaisons de catégories de produits fréquemment achetés ensemble semblent se dégager. Une mise en avant stratégique de ces combinaisons, synchronisée avec les moments forts de l'année, comme "Mode + Livres" à la rentrée scolaire, ou "Mode + Sports" en janvier et avant l'été pour les nouvelles résolutions, ainsi que des recommandations intelligentes au panier pourraient potentiellement stimuler les ventes croisées.

Enfin, la répartition par âge montre une distribution gaussienne du chiffre d'affaires, centrée sur les 40-55 ans. Cibler davantage les moins de 25 ans via des canaux comme TikTok (largement sous-exploité, comme le montre le *dashboard* marketing)[1] et développer des offres adaptées aux seniors pourrait élargir la base de clients.

1. cf. Annexe D.1, page 71

Conclusion

Tableau s'est révélé être un excellent choix pour ce projet, permettant une collaboration efficace entre les membres de l'équipe tout en offrant des résultats visuels professionnels, malgré les limitations imposées par la licence d'essai.

Il est important de souligner que ces *dashboards* ne constituent pas une solution figée. Au contraire, leur force réside dans leur capacité à évoluer en fonction des besoins changeants des utilisateurs. La flexibilité de l'écosystème Tableau, couplée à la modularité du pipeline ELT (notamment grâce à DBT), permet des adaptations rapides à tous les niveaux : ajout de nouvelles visualisations, modification des indicateurs clés, ou intégration de sources de données supplémentaires. Cette agilité garantit la pérennité de ces outils face aux évolutions métiers et techniques.

Conclusion

Ce projet a été l'occasion de construire un pipeline de données complet, depuis la collecte jusqu'à l'analyse, en passant par la visualisation. Nous avons opté pour une architecture ELT moderne avec Fivetran pour l'extraction, DBT pour la transformation et BigQuery comme *data warehouse* – des choix qui se sont révélés pertinents pour leur flexibilité et leur facilité d'utilisation. L'automatisation du *workflow* avec Airflow a permis de gérer efficacement l'ensemble du processus, y compris l'exécution de nos modèles de Machine Learning (prévisions avec Prophet, analyse des associations avec Apriori et régression avec XGBoost) et la géolocalisation des clients. Grâce à Tableau, nous avons pu créer des *dashboards* clairs et interactifs qui répondent aux besoins des différents métiers. Le travail sur l'harmonisation visuelle et l'ergonomie a porté ses fruits, présentant les données synthétiquement, esthétiquement et avec intégrité.

En planifiant soigneusement les tâches et en maintenant des *"stand-ups"* quotidiens, nous avons pu avancer de manière constante sans avoir recours au *crunch*. La répartition des rôles selon les compétences de chacun (certains plus orientées data engineering, d'autres sur le Machine Learning ou la *dataviz*) a permis de tirer le meilleur de notre équipe.

Sur le plan pédagogique enfin, ce projet a été extrêmement formateur. Il nous a donné l'opportunité de mettre en pratique des concepts théoriques sur un cas concret, avec les contraintes et les défis d'un vrai projet data. La gestion des dépendances entre les différentes briques techniques, la nécessité de documenter proprement le code et les modèles, l'importance de tests rigoureux – autant d'aspects qui prennent tout leur sens quand on les expérimente en situation réelle.

Annexe A

Code DBT

A.1 Configuration

Listing A.1 – Definition des sources de donnees brutes

```yaml
# [Fichier : sources.yaml]
# Role : Definition des sources de donnees brutes a partir de Google Drive.
# Ce fichier liste les tables sources qui seront utilisees comme input pour les
   transformations DBT.

version: 2

sources:
  - name: google_drive # Dataset BigQuery dans lequel se trouve les tables brutes
    tables:
      - name: carttrend_campaigns_campagnes
      - name: carttrend_clients_clients
      - name: carttrend_commandes_commandes
      - name: carttrend_details_commandes_details_commandes_
      - name: carttrend_entrepots_entrepots
      - name: carttrend_entrepots_machines_machines_entrepots
      - name: carttrend_posts_posts
      - name: carttrend_produits_produits
      - name: carttrend_promotions_promotions
      - name: carttrend_satisfaction_satisfaction
```

Listing A.2 – Documentation des modeles DBT *"staging"*

```yaml
# [Fichier : staging/schema.yaml]
# Role : Documentation des modeles DBT apres etape de staging (nettoyage/validation).
# Decrit la structure des tables transformees, leurs relations et les tests de
   qualite associes.
# Utilise des packages DBT (dbt_utils, dbt_expectations) pour les tests avances.

version: 2

models:
  # Campagne_Marketing ------------------------------------------------------
  #--------------------------------------------------------------------------
  - name: Campagnes
    description: Table des Campagnes Marketing Normalisée et Nettoyée.
    tests:
      - dbt_utils.expression_is_true: # Format clé primaire CAMP000
          expression: 'REGEXP_CONTAINS(id_campagne, r"^CAMP\d{3}$")'
          config:
            severity: warn # Warning plutot qu'erreur
```

```
      - dbt_utils.expression_is_true: # Controle de coherence des metriques
          expression: 'budget > 0 and impressions > 0 and clics > 0 and conversions > 0'
          config:
            severity: warn
    columns:
      - name: id_campagne
        description: Clé Primaire. Pas de test 'not_null', les campagnes n'ayant pas d'id sont automatiquement supprimés
        tests:
          - unique
      - name: date
        description: Jour (DATE) de la campagne
      - name: id_canal
        description: id du Canal Marketing
      - name: budget
        description: Cout de la campagne marketing en E (float). Chiffre superieur a 0
      - name: impressions
        description: Cout de la campagne marketing (int). Chiffre superieur a 0
      - name: clics
        description: Nombre de Clics générés par la campagne marketing (int). Chiffre superieur a 0
      - name: conversion
        description: Nombre de Conversions générés par la campagne marketing (int). Chiffre superieur a 0
      - name: ctr
        description: CTR - Nombre de clics / nombre d'impression. Float entre 0 et 1 ou NULL

  # Canaux ------------------------------------------------------------------
  #--------------------------------------------------------------------------
  # Pas de test car table extraite des donnée d'autres tables. Tests réalisés sur ces autres tables
  - name: Canaux
    description: Table de dimension contenant les canaux publicitaires (résaux sociaux)
    columns:
      - name: id_canal
        description: Clé Primaire. Tests non necessaire car clé crée par rownumber = garantie d'etre unique et non nulle
      - name: canal
        description: Nom du Canal Publicitaire (Facebook, Twitter...)

  # Clients -----------------------------------------------------------------
  #--------------------------------------------------------------------------
  - name: Clients
    description: Table des clients nettoyée. Les produits des favoris des clients ne sont pas normalisés
    tests:
      - dbt_utils.expression_is_true: # Format clé primaire C00000
          expression: 'REGEXP_CONTAINS(id_client, r"^C\d{5}$")'
      - dbt_utils.expression_is_true:
          expression: 'age < 110'
          config:
            severity: warn
    columns:
      - name: id_client
        description: Clé Primaire. Pas de test 'not_null', les clients n'ayant pas d'id valide sont automatiquement supprimés
        test:
          - unique
      - name: hash_prenom_nom
        description: Hash sha264 (pour anonymisation) du prenom et du nom du client
```

```
      - name: email
        description: Email du client valide ou NULL
      - name: age
        description: Age du client (int). Chiffre superieur a 0 ou NULL
      - name: tranche_age
        description: Tranche d'age. String '<=25', '25-40', '40-55', '55-70' ou '>70'
      - name: genre
        description: Genre du client. 'Homme', 'Femme' ou NULL
      - name: frequence_visites
        description: Nombre de visites (int) du client sur le site. Chiffre superieur
            a 0 ou NULL
      - name: telephone
        description: Numero de téléphone au format '+XXX-XXX-XXX-XXXX' ou NULL
      - name: date_inscription
        description: Date d'inscription (date) du client ou NULL. Avant la date d'
           aujourd'hui
      - name: anciennete_jours
        description: Nombre de jours depuis l'inscription du client
      - name: ip
        description: Adresse IPv4 du client ou NULL. Au format 'XXX.XXX.XXX.XXX'

  # Details_Commandes ------------------------------------------------------------
  #-------------------------------------------------------------------------------
  - name: Details_commandes
    description: Information sur la commande - produits, quantitées...
    tests:
      - dbt_utils.expression_is_true:
          expression: 'quantite > 0'
    columns:
      - name: id_commande
        tests:
          - relationships:
              to: ref('Commandes')
              field: id_commande
      - name: id_produit
        tests:
          - relationships:
              to: ref('Produits')
              field: id_produit
      - name : quantite
        description : Nombre du même article par commande. Supérieur strictement à 0
      - name : emballage_special
        description : La commande est emballée avec un emballage spécial. Type boolé
           en

  # Entrepots --------------------------------------------------------------------
  #-------------------------------------------------------------------------------
  - name: Entrepots
    description: Table des Entrepots Nettoyée
    tests:
      - dbt_utils.expression_is_true:
          expression: 'capacite_max > volume_stocke'
          config:
            severity: warn
      - dbt_utils.expression_is_true:  # Format clé primaire E000
          expression: 'REGEXP_CONTAINS(id_entrepot, r"^E\d{3}$")'
    columns:
      - name: id_entrepot
        description: Clé Primaire, au format 'EXXX'
        tests:
          - not_null
          - unique
      - name: localisation
        description: Ville dans laquelle se trouve l'entrepot
```

```
      - name: capacite_max
        description: Nombre de produits (int) maximum que peut contenir l'entrepot.
            Garantie > 0
      - name: volume_stocke
        description: Nombre de produits (int) actuellement stocké dans l'entrepot.
      - name: taux_remplissage
        description: Pourcentage de stockage utilisé. float > 0
      - name: temperature
        description: Temperature moyenne mesurée dans l'entrepot

  # Entrepots_Machines ------------------------------------------------------
  #--------------------------------------------------------------------------
  - name: Entrepots_Machines
    description: Table qui décrit l'etat de fonctionnement des machines des entrepots
    tests:
      - dbt_utils.expression_is_true:  # Format clé primaire ME000000
          expression: 'REGEXP_CONTAINS(id_machine, r"^ME\d{6}$")'
    columns :
    - name: id_machine
      description: identifiant de la machine
      tests:
        - not_null
    - name: id_entrepot
      description: id de l'entrepot contenant la machine. Cle etrangere
      tests:
        - relationships:
            to: ref('Entrepots')
            field: id_entrepot
    - name: type_machine
      description: Fonction de la machine. Garanti 'Tri','Maintenance','Transport','
          Stockage','Conditionnement' ou 'Autre'
    - name: etat_machine
      description: Etat de Fonctionnement de la machine. Garanti 'Service','Panne','
          Maintenance' ou 'Autre'
    - name: temps_d_arret
      description: Duree de non fonctionnement de la machine
    - name: volume_traite
      description: le nombre de produit traités par ma machine dans le mois
    - name: date
      description: date du premier jour du mois évalué

  # Posts -------------------------------------------------------------------
  #--------------------------------------------------------------------------
  - name : Posts
    columns :
    - name : id_post
      tests :
        - not_null
        - unique
    - name : date_post
      description : date de publication du post
      tests:
        - dbt_expectations.expect_column_values_to_be_of_type:
            column_type: date
    - name : id_canal
      description : id du canal utilisé. Clé étrangère, mais pas de test car garantie
          d'exister dans la table Canaux
    - name : volume_mentions
      description : Nombre d'interactions par post (int), garanti > 0
    - name : sentiment global
      description : Sentiment lié au post. Peut être 'Positif', 'Neutre' ou 'Negatif'
          ou 'NULL'
    - name : contenu_post
      description : Contenu du post publié
```

```
191
192   # Produits ------------------------------------------------------------------
193   #-------------------------------------------------------------------------------
194   - name: Produits
195     description: Table des Produits nettoyée
196     tests:
197       - dbt_utils.expression_is_true:  # Format clé primaire P00000
198           expression: 'REGEXP_CONTAINS(id_produit, r"^P\d{5}$")'
199       - dbt_utils.expression_is_true:
200           expression: 'prix > 0'
201           config:
202             severity: warn
203     columns:
204       - name: id_produit
205         description: Clé Primaire.
206         tests:
207           - not_null
208           - unique
209       - name: categorie
210         description: Categorie du produit. N'importe lequel de 'Livres','Alimentation
              ','Mode','Maison','Électronique','Jouets','Beauté','Sports' ou 'Autre'
211       - name: marque
212         description: Marque du produit ou NULL. Premiere lettre en majuscule
213       - name: nom
214         description: Nom du produit ou NULL. Premiere lettre en majuscule
215       - name: prix
216         description: Prix du produit. (float) > 0
217       - name: sous_categorie
218         description : Nom de la sous categorie du produit ou NULL. Premiere lettre en
              majuscule
219       - name: variation
220         description: Nom de la variation (taille, gamme...) ou NULL. Premiere lettre
              en majuscule
221
222   # Produits Favoris ----------------------------------------------------------
223   #-------------------------------------------------------------------------------
224   - name: Produits_Favoris
225     description: Table contenant les identifiants des produits que les clients ont
          ajoutés dans leur favoris
226     columns:
227       - name: id_client
228         description: Clé etrangere client
229         tests:
230           - relationships:
231               to: ref('Clients')
232               field: id_client
233       - name: id_produit
234         description: Clé etrangere produit
235         tests:
236           - relationships:
237               to: ref('Produits')
238               field: id_produit
239
240   # Promotions ----------------------------------------------------------------
241   #-------------------------------------------------------------------------------
242   - name: Promotions
243     tests:
244       - dbt_utils.expression_is_true:  # Format clé primaire P000
245           expression: 'REGEXP_CONTAINS(id_promotion, r"^P\d{3}$")'
246       - dbt_utils.expression_is_true:
247           expression: 'pourcentage_promotion > 0'
248           config:
249             severity: warn
250     columns:
```

```
      - name: id_promotion
        tests:
          - not_null
          - unique
      - name: id_produit
        tests:
          - relationships:
              to: ref('Produits')
              field: id_produit
      - name: valeur_promotion
        description: en euro
      - name: pourcentage_promotion
        description: promotion en pourcentage (float entre 0 et 1)
      - name: date_debut
        description: Date de début de la promotion 'yyyy-mm-dd'
        tests:
          - not_null
      - name: date_fin
        description: Date de fin de la promotion 'yyyy-mm-dd'

  # Satisfaction ----------------------------------------------------------------
  #-----------------------------------------------------------------------------
  - name: Satisfaction
    columns:
      - name: id_satisfaction
        tests:
          - unique
          - not_null
      - name: id_commande # Hugues :
        description: id de la commande a laquelle se rapporte cet avis
        tests:
          - relationships:
              to: ref('Commandes')
              field: id_commande
      - name : note_client
        description : Note laissee par le client de 1 à 5
      - name : commentaire
        description : commentaire laissé par le client
      - name : plainte
        description : le commentaire fait l'objet d'une plainte de la part du client.
            Type booléen
      - name : delai_traitement
        description : Duree de reponse effectuee par l'equipe support
      - name : type_plainte
        description : Type de plainte effectuée par le client
      - name : employe_support
        description : identifiant du membre du service support
```

A.2 Exemples de Moèles *"staging"*

Listing A.3 – Nettoyage et Transformation des donnees brutes des entrepots

```sql
-- [Fichier : Entrepots.sql]
-- Role : Nettoyage et Transformation des donnees brutes des entrepots.
-- Effectue le nettoyage de base et le calcul des indicateurs durant l'etape staging.

SELECT
    id_entrepot,  -- Conserve l'identifiant original

    -- Normalisation de la localisation (mise en majuscule de la premiere lettre)
    INITCAP(localisation) AS localisation,  -- ex. 'paris' => 'Paris'

    -- Capacite toujours >= 0
```

```sql
    CASE
        WHEN capacite_max < 0 THEN 0
        ELSE capacite_max
    END AS capacite_max,

    -- Stock toujours >= 0
    CASE
        WHEN volume_stocke < 0 THEN 0
        ELSE volume_stocke
    END AS volume_stocke,

    -- Calcul du taux de remplissage (derive)
    volume_stocke / capacite_max AS taux_remplissage,

    -- Simplification du nom de la colone temperature
    temperature_moyenne_entrepot AS temperature

-- Table contenant les donnees brutes
FROM {{source("google_drive", "carttrend_entrepots_entrepots")}}
```

Listing A.4 – Normalisation des canaux publicitaires depuis Posts et Campagnes

```sql
-- [Fichier : Canaux.sql]
-- Normalisation des canaux publicitaires depuis Posts et Campagnes
-- Crée une table de référence unique pour tous les canaux marketing

WITH canal AS (
    -- Extraction des canaux depuis les posts réseaux sociaux
    SELECT canal_social
    FROM {{source("google_drive",'carttrend_posts_posts')}}

    UNION DISTINCT  -- Combine les deux en éliminant les doublons

    -- Extraction des canaux depuis les campagnes marketing
    SELECT canal
    FROM {{source('google_drive', 'carttrend_campaigns_campagnes')}}
)
SELECT
    -- Génération d'un ID unique pour chaque canal,
    -- un chiffre séquentiel auto-incrémenté
    ROW_NUMBER() OVER() as id_canal,

    -- Conservation du nom original
    canal_social as canal

FROM canal
-- Tri alphabétique pour une meilleure lisibilité
ORDER BY canal ASC
```

Listing A.5 – Nettoyage des données de posts réseaux sociaux avec intégration des canaux normalisés

```sql
-- [Fichier : Posts.sql]
-- Nettoyage des données de posts réseaux sociaux avec intégration des canaux
   normalisés
-- Montre l'utilisation de la table de dimension Canaux créée précédemment

SELECT
    id_post,  -- Identifiant original du post
    id_canal,  -- Clé étrangère vers la table Canaux normalisée

    -- Garantit volume_mentions >= 0
    CASE
        WHEN volume_mentions < 0 THEN 0
        ELSE volume_mentions
    END AS volume_mentions,
```

```sql

    -- Validation des valeurs de sentiment
    CASE
        WHEN sentiment_global IN ('Positif', 'Négatif', 'Neutre') THEN
            sentiment_global
        ELSE NULL
    END AS sentiment_global,

    -- Garantit date_post <= date actuelle
    CASE
        WHEN date_post <= CURRENT_DATE() THEN date_post
        ELSE CURRENT_DATE()
    END AS date_post,

    -- Texte original du post conservé sans transformation
    contenu_post

FROM {{source("google_drive",'carttrend_posts_posts')}} p  -- Table source des posts
    -- Jointure avec la table Canaux normalisée
    JOIN {{ref('Canaux')}} c ON c.canal = p.canal_social
```

Listing A.6 – Nettoyage et normalisation des données clients

```sql
-- [Fichier : Clients.sql]
-- Nettoyage et normalisation des données clients avec :
-- 1. Anonymisation des données sensibles
-- 2. Standardisation des formats
-- 3. Validation des données
-- 4. Enrichissement léger

WITH tel AS (
    -- CTE pour le traitement des numéros de téléphone
    SELECT
        id_client,
        -- Formatage standard du numéro (123-456-7890)
        REGEXP_REPLACE(numero_telephone, r'^.*?(\d{3})\D?(\d{3})\D?(\d{4}).*$', r'
            \1-\2-\3') AS numero_telephone,
        -- Left-Padding du code pays (ex: +033 pour la France)
        CASE
            WHEN LENGTH(ac.area_code) = 3 THEN CONCAT('+', ac.area_code)
            WHEN LENGTH(ac.area_code) = 2 THEN CONCAT('+0', ac.area_code)
            WHEN LENGTH(ac.area_code) = 1 THEN CONCAT('+00', ac.area_code)
            ELSE '+033'  -- Valeur par défaut
        END as area_code
    FROM {{source("google_drive",'carttrend_clients_clients')}}
    JOIN (
        -- Sous-Requete pour l'extraction du code pays depuis le numéro original
        SELECT
            id_client,
            REGEXP_EXTRACT(numero_telephone, r'^\+?([1-9]{0,3}).*$') AS area_code
        FROM {{source("google_drive",'carttrend_clients_clients')}}
    ) ac USING(id_client)
)
SELECT
    -- Identifiant client conservé
    id_client,

    -- Anonymisation des données personnelles avec hashing SHA256
    sha256(CONCAT(prenom_client, nom_client)) as hash_prenom_nom,

    -- Validation de l'email
    CASE
        WHEN REGEXP_CONTAINS(email, r'^[\w\.-]+@[\w-]+.[A-Za-z]+$') THEN email
        ELSE NULL
    END as email,
```

```sql

    -- Suppression des âges négatifs
    CASE WHEN age > 0 THEN age END as age,

    -- Calcul de la tranche d'âge
    CASE
        WHEN age > 0  AND age <= 25 THEN '<=25'
        WHEN age > 25 AND age <= 40 THEN '25-40'
        WHEN age > 40 AND age <= 55 THEN '40-55'
        WHEN age > 55 AND age <= 70 THEN '55-70'
        WHEN age > 70 THEN '>70'
        ELSE NULL
    END as tranche_age,

    -- Normalisation du genre
    CASE
        WHEN lower(genre) = 'homme' THEN 'Homme'
        WHEN lower(genre) = 'femme' THEN 'Femme'
        ELSE NULL
    END as genre,

    -- Validation et correction de la fréquence de visites
    CASE
        WHEN frequence_visites > 0 THEN frequence_visites
        ELSE 0
    END as frequence_visites,

    -- Construction du numéro de téléphone complet
    -- Indicatif + Numero, format final +000-123-456-7890
    CONCAT(tel.area_code, '-', tel.numero_telephone) as telephone,

    -- Validation de la date d'inscription
    CASE
        WHEN DATE_DIFF(date_inscription, CURRENT_DATE(), DAY) <= 0 THEN
            date_inscription
        ELSE NULL
    END as date_inscription,

    -- Calcul de l'ancienneté en jours
    CASE
        WHEN DATE_DIFF(date_inscription, CURRENT_DATE(), DAY) <= 0
        THEN DATE_DIFF(CURRENT_DATE(), date_inscription, DAY)
        ELSE NULL
    END as anciennete_jours,

    -- Validation des adresses IP (format IPv4)
    CASE
        WHEN REGEXP_CONTAINS(adresse_ip, r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$')
            THEN adresse_ip
        ELSE NULL
    END as ip

FROM {{source("google_drive",'carttrend_clients_clients')}}
    JOIN tel USING(id_client)
WHERE id_client IS NOT NULL  -- Exclusion des clients sans identifiant
```

A.3 Exemples de Moèles *"marts"*

Listing A.7 – Analyse textuelle des commentaires clients pour *dataviz*

```sql
-- [Fichier : mart_Commentaires.sql]
-- Analyse textuelle des commentaires clients pour dataviz :
-- 1. Extraction et nettoyage des mots individuels
```

```sql
-- 2. Filtrage des stopwords
-- 3. Calcul des indicateurs clés (fréquence, sentiment moyen)

WITH
  -- Découpage des commentaires en mots individuels
  mots AS (
    SELECT
      note_client,  -- Note associée (1-5)
      mot
    FROM {{ref("Satisfaction")}},  -- Table staging nettoyée
    UNNEST(SPLIT(commentaire, ' ')) AS mot  -- Désagrégation en mots
  ),

  -- Agrégation et analyse des mots
  mots_aggreges AS (
    SELECT
      UPPER(REGEXP_REPLACE(mot, r'[^\w]', '')) AS mot,  -- Nettoyage (majuscules, ponctuation)
      COUNT(*) AS frequence,  -- Nombre d'occurrences
      ROUND(AVG(note_client), 1) AS note_moyenne  -- Sentiment moyen associé
    FROM mots
    WHERE mot != ''  -- Exclusion des chaînes vides
      -- Filtre des stopwords (mots courants non significatifs)
      AND lower(mot) NOT IN (
        'a', 'about', 'above', ..., 'yourselves'  -- Liste écourtée de stopwords anglais
      )
    GROUP BY mot
  )

-- Résultat final pour dataviz
SELECT *
FROM mots_aggreges
 -- Tri par fréquence (principalement pour debugage)
ORDER BY frequence DESC
```

Listing A.8 – Mart simplifiant l'analyse et la visualisation des notes des clients dans leur contexte

```sql
-- [Fichier : mart_Notes_Clients.sql]
-- Mart simplifiant l'analyse et la visualisation des notes des clients dans leur contexte
-- à l'aide des jointures multiples pour croiser les dimensions pertinentes

SELECT
    p.id_produit,  -- Identifiant produit
    p.nom,         -- Nom du produit
    p.categorie,   -- Catégorie produit
    c.date_commande,  -- Date de commande
    c.date_livraison_estimee,  -- Date de livraison estimée
    s.type_plainte,  -- Type de plainte associée (le cas échéant)
    c.id_client,     -- Identifiant client
    AVG(s.note_client) AS avg_note_client,  -- Note moyenne calculée
    e.localisation AS localisation_entrepot  -- Localisation de l'entrepôt

-- Jointure des différentes tables staging
FROM {{ ref("Produits") }} p
JOIN {{ ref("Details_commandes")}} d USING(id_produit)
JOIN {{ ref("Commandes") }} c USING(id_commande)
JOIN {{ ref("Satisfaction") }} s USING(id_commande)
JOIN {{ ref("Entrepots")}} e USING(id_entrepot)

-- Groupement par dimensions d'analyse
GROUP BY
    p.id_produit,
    p.nom,
```

```
    p.categorie,
    c.date_commande,
    c.date_livraison_estimee,
    s.type_plainte,
    c.id_client,
    e.localisation

-- Tri par note moyenne (plus basse en premier pour identifier les problèmes)
ORDER BY avg_note_client ASC
```

Listing A.9 – Table complète des commandes clients

```
-- [Fichier : mart_Commandes.sql]
-- Table complète des commandes clients avec :
-- 1. Calcul des prix (initial et après remise)
-- 2. Contexte client et produit
-- 3. Promotions éventuelles

SELECT
    co.date_commande,  -- Date de la commande
    p.nom,  -- Nom du produit
    p.prix as prix_unitaire,  -- Prix unitaire de base
    c.quantite,  -- Quantité commandée

    -- Calcul du prix total avant remise (quantité * prix unitaire)
    ROUND(p.prix * dc.quantite, 2) AS prix_initial,

    -- Calcul du prix après application de la remise éventuelle
    ROUND(CASE
        WHEN pourcentage_promotion IS NOT NULL THEN
            p.prix * dc.quantite * (1 - pourcentage_promotion)
        ELSE p.prix * dc.quantite
    END, 2) as prix_apres_remise,

    p.categorie,  -- Catégorie produit
    p.sous_categorie,  -- Sous-catégorie

    -- Flag indiquant si le produit est dans les favoris du client
    CASE
        WHEN EXISTS (
            SELECT *
            FROM {{ref('Produits_Favoris')}} pf
            WHERE pf.id_client = cl.id_client AND pf.id_produit = p.id_produit)
        THEN true
        ELSE false
    END as produit_favori,

    -- Informations promotionnelles (avec valeurs par défaut)
    IFNULL(pr.valeur_promotion, 0) as valeur_promotion,
    IFNULL(pr.pourcentage_promotion, 0) as pourcentage_promotion,

    -- Informations du client
    cl.id_client,
    cl.age,
    cl.tranche_age,
    cl.genre,
    cl.frequence_visites,
    cl.date_inscription,
    cl.ancienneté_jours,

    co.statut_commande,  -- Statut de la commande
    p.id_produit  -- ID produit pour les jointures

-- Tables sources avec jointures LEFT pour conserver toutes les commandes
FROM {{ref('Commandes')}} co
```

```
LEFT JOIN {{ref('Details_commandes')}} dc USING (id_commande)
LEFT JOIN {{ref('Produits')}} p USING (id_produit)
LEFT JOIN {{ref('Promotions')}} pr USING (id_promotion)
LEFT JOIN {{ref('Clients')}} cl USING (id_client)
```

Listing A.10 – Preparation des series temporelles de CA pour Prophet

```
-- [Fichier : mart_Prevision_Series_Temporelles_CA.sql]
-- Preparation des series temporelles de CA pour Prophet :
-- 1. Generation d'un calendrier continu sans trous
-- 2. Aggregation du CA quotidien
-- 3. Formatage specifique pour l'analyse forecasting

WITH
    -- Generation du calendrier complet entre premiere et derniere commande
    cal AS (
        SELECT date
        FROM UNNEST(GENERATE_DATE_ARRAY(
            (SELECT MIN(date_commande) FROM {{ref('mart_Commandes')}}),  -- Date minimale
            (SELECT MAX(date_commande) FROM {{ref('mart_Commandes')}})   -- Date maximale
        ) AS date
    ),

    -- Calcul du CA quotidien (hors commandes annulees)
    ca AS (
        SELECT
            date_commande AS date,
            SUM(prix_apres_remise) AS ca  -- Somme des montants apres remise
        FROM {{ref('mart_Commandes')}}
        WHERE statut_commande != 'Annulée'  -- Filtre des commandes annulees
        GROUP BY date_commande
    )

-- Jointure finale pour serie temporelle complete
SELECT
    cal.date AS ds,  -- Colonne 'ds' requise par Prophet
    ROUND(IFNULL(ca.ca, 0), 2) as y  -- Colonne 'y' requise par Prophet
FROM cal
    LEFT JOIN ca USING(date)  -- Jointure sur les dates, left pour garder toutes les dates
ORDER BY date  -- Tri chronologique obligatoire
```

Listing A.11 – Preparation des donnees pour modele XGBoost de prediction des pannes

```
-- [Fichier : mart_Regression_Fonctionnement_Machines.sql]
-- Preparation des donnees pour modele XGBoost de prediction des pannes

SELECT
  -- Variables Explicatives
  EXTRACT(MONTH FROM m.date) AS month,  -- Saisonnalite (1-12)
  e.localisation,  -- Localisation de l'entrepot (categorielle)
  e.temperature,  -- Temperature moyenne
  m.volume_traite,  -- Charge de travail (continue)

  -- L'analyse sera faite par type de machines
  m.type_machine,  -- Type d'equipement (categorielle)

  -- Variable cible : Uptime
  -- [0-1], 1 = 100% uptime
  1 - m.temps_d_arret / (DATE_DIFF(
    DATE_ADD(m.date, INTERVAL 1 MONTH),
    m.date,
    DAY
```

```
  ) * 24) as pourcentage_uptime,

FROM {{ref("Entrepots_Machines")}} m
  JOIN {{ref('Entrepots')}} e USING(id_entrepot)
```

Annexe B

Scripts Machine Learning et Geolocalisation

Listing B.1 – Implementation du modele Prophet pour la prevision de CA

```python
# [Fichier : ml_prediction_ca.py]
# Implementation du modele Prophet pour la prevision de CA :
# 1. Extraction des donnees depuis BigQuery
# 2. Configuration du modele avec gestion des périodes particulière
# 3. Entrainement et prediction
# 4. Stockage des resultats dans BigQuery

from google.oauth2 import service_account
import pandas_gbq
from prophet import Prophet
import pandas as pd
import os

# Authentification GCP via service account
credentials = service_account.Credentials.from_service_account_file(
    os.path.dirname(__file__) + '/Compte Service Machine Learning.json',
)

# Extraction des donnees historiques
sql = '''
    SELECT *
    FROM `carttrend_clean.mart_Prevision_Series_Temporelles_CA`
'''
df = pandas_gbq.read_gbq(sql, project_id="carttrend-4", credentials=credentials)
df['date'] = pd.to_datetime(df['date'])

# Certaines périodes de ventes sont très particulières, et pas forcement repré
    sentatives
# Elles sont déclarées comme vacances et sont ignorées par le modèle
# cf. https://facebook.github.io/prophet/docs/handling_shocks.html
vacances = pd.DataFrame([
    {
        'holiday': '2023',
        'ds': pd.to_datetime('2023-12-30'),
        'lower_window': 0,
        'upper_window': 20,
    },
    {
        'holiday': '2024',
        'ds': pd.to_datetime('2024-11-20'),
        'lower_window': 0,
        'upper_window': 25,
    }
])

```

```
# Initialisation du modele Prophet
model = Prophet(holidays=vacances)

# Entrainement du modele (il attend des noms de colonnes specifiques)
model.fit(df.rename(columns={'date': 'ds', 'chiffre_affaire': 'y'}))

# Creation d'un dataframe avec 30 jours supplementaires
future = model.make_future_dataframe(periods=30, freq='D')

# Prediction du chiffre d'affaire sur ces 30 jours
# ds : date, yhat : prediction,
# yhat_lower et yhat_upper : bornes de l'intervalle de confiance
forecast = model.predict(future)[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]

# Formatage des resultats
forecast['ds'] = pd.to_datetime(forecast['ds'])
output = forecast.rename(columns={'ds': 'date'})

# Combinaison des données de départ et des prédictions
# Cela simplifie la visualisation par la suite
output = pd.concat([forecast.set_index('date'), output.set_index('date')], axis=1).
    reset_index()

# Envoi vers BigQuery
pandas_gbq.to_gbq(
    output,
    "carttrend_transform.ml_predictions_ca",
    project_id="carttrend-4",
    if_exists="replace"
)
```

Listing B.2 – Implementation de l'algorithme Apriori pour l'analyse des associations produits

```
# [Fichier : ml_a_priori.py]
# Implementation de l'algorithme Apriori pour l'analyse des associations produits :
# 1. Extraction des transactions depuis BigQuery
# 2. Preprocessing des donnees pour ML
# 3. Execution de l'algorithme Apriori
# 4. Filtrage et stockage des resultats

from google.oauth2 import service_account
import pandas_gbq
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
import os

# Authentification GCP via service account
credentials = service_account.Credentials.from_service_account_file(
    os.path.dirname(__file__) + '/Compte Service Machine Learning.json',
)

# Extraction des combinaisons produits depuis BigQuery
sql = '''
    SELECT *
    FROM `carttrend_clean.mart_Apriori_Combinaisons_Produits`
'''
df = pandas_gbq.read_gbq(sql, project_id="carttrend-4", credentials=credentials)

# Transformation des donnees en format liste de listes (format attendu par mlxtend)
transactions = [list(row[0]) for row in df.itertuples(index=False)]

# Encodage des "paniers" au format one-hot
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
```

```
dfml = pd.DataFrame(te_ary, columns=te.columns_)

# Execution de l'algorithme Apriori avec seuil minimal de support
frequent_itemsets = apriori(dfml, min_support=0.1, use_colnames=True)

# Filtrage des resultats :
# - Combinaisons d'au moins 2 categories
# - Support minimum de 40%
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets = frequent_itemsets[
    (frequent_itemsets['length'] >= 2) &
    (frequent_itemsets['support'] >= 0.4)
][['support', 'itemsets']]

# Formatage des resultats (frozenset -> string), facilite la visualisation
frequent_itemsets['itemsets'] = frequent_itemsets['itemsets'].apply(
    lambda x: ' - '.join(list(x))
)

# Envoi des resultats vers BigQuery
pandas_gbq.to_gbq(
    frequent_itemsets,
    "carttrend_transform.ml_categories_achetees_ensemble",
    project_id="carttrend-4",
    if_exists="replace"
)
```

Listing B.3 – Modele XGBoost pour l'analyse des causes de pannes machines

```
# [Fichier : ml_regression_machines.py]
# Modele XGBoost pour l'analyse des causes de pannes machines :
# 1. Preparation des donnees (normalisation, encodage)
# 2. Entrainement d'un modèle par type de machine
# 3. Evaluation des performances pour debugage
# 4. Export des resultats vers BigQuery

from google.oauth2 import service_account
import pandas as pd
import numpy as np
import pandas_gbq
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import os

# Authentification GCP via service account
credentials = service_account.Credentials.from_service_account_file(
    os.path.dirname(__file__) + '/Compte Service Machine Learning.json',
)

# Extraction des donnees depuis BigQuery
sql_machine = '''
    SELECT *
    FROM `carttrend-4.carttrend_clean.mart_Regression_Fonctionnement_Machines`
'''
df_machine = pandas_gbq.read_gbq(sql_machine, project_id="carttrend-4", credentials=
    credentials)

# Normalisation de la variable cible (entre 0 et 1)
scaler_uptime = MinMaxScaler()
df_machine['pourcentage_uptime_normalise'] = scaler_uptime.fit_transform(
    df_machine[['pourcentage_uptime']]
)

```

```
# Standardisation du volume traite (moyenne=0, ecart-type=1)
sta_volume = StandardScaler()
df_machine['volume_traite_standalise'] = sta_volume.fit_transform(
    df_machine[['volume_traite']]
)

# Encodage des variables categorielles (localisation)
le = LabelEncoder()
df_machine['localisation'] = le.fit_transform(df_machine['localisation'])

# Extraction de la liste des types de machines
lst_typesMachines = df_machine['type_machine'].unique()

# Initialisation d'une DataFrame pour stocker les résultats
df_out = pd.DataFrame(columns=[
    'type_machine', 'month', 'localisation', 'temperature',
    'volume_traite_standalise', 'mse', 'rmse'
])

# Entrainement d'un modèle par type de machine
for typeMachine in lst_typesMachines:
    # Filtrage des donnees
    # Ne garde que celles concernant le type de machine en cours d'analyse
    df_filtre = df_machine[df_machine['type_machine'] == typeMachine]

    # Separation variables explicatives / variables cibles
    X = df_filtre.drop(columns=[
        'pourcentage_uptime', 'pourcentage_uptime_normalise',
        'type_machine', 'volume_traite'
    ])
    y = df_filtre['pourcentage_uptime_normalise']

    # Decoupage train/test
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # Configuration et entrainement du modele
    model = xgb.XGBRegressor(
        n_estimators=100,  # Nombre d'arbres
        learning_rate=0.1,  # Taux d'apprentissage
        max_depth=3  # Profondeur maximale des arbres
    )
    model.fit(X_train, y_train)

    # Prediction et evaluation
    y_pred = model.predict(X_test)
    y_test_inv = scaler_uptime.inverse_transform(y_test.to_numpy().reshape(-1, 1))
    y_pred_inv = scaler_uptime.inverse_transform(y_pred.reshape(-1, 1))
    mse = mean_squared_error(y_test_inv, y_pred_inv)
    rmse = np.sqrt(mse)
    rmses.append(rmse)

    # Stockage des resultats
    importances = model.feature_importances_
    df_out.loc[len(df_out)] = [typeMachine] + importances.tolist() + [mse, rmse]

# Envoi vers BigQuery
pandas_gbq.to_gbq(
    df_out,
    "carttrend_transform.ml_cause_panne_machine",
    project_id="carttrend-4",
    if_exists="replace"
)
```

Listing B.4 – Enrichissement des données clients par géolocalisation IP

```
# [Fichier : get_customer_location.py]
# Enrichissement des données clients par géolocalisation IP :
# 1. Extraction des IP clients depuis BigQuery
# 2. Appels à l'API ip2location.io (limité à 1 requête/0.2s) pour obtenir la position
# 3. Mise en cache des résultats pour 7 jours
# 4. Sauvegarde du résultat dans BigQuery

import requests
from google.oauth2 import service_account
import pandas as pd
import pandas_gbq
import time
import os
import datetime

# Authentification GCP via service account
credentials = service_account.Credentials.from_service_account_file(
    os.path.dirname(__file__) + '/Compte Service Machine Learning.json',
)

# Extraction des clients à géolocaliser et de leurs IP
sql = '''
    SELECT id_client, ip
    FROM `carttrend-4.carttrend_clean.Clients`
'''
df = pandas_gbq.read_gbq(sql, project_id="carttrend-4", credentials=credentials)

# Initialisation du DataFrame de résultats
client_geo = pd.DataFrame(columns=['id_client', 'pays', 'region', 'ville'])

# Gestion du cache
file_age = None
max_age = datetime.timedelta(days=7)  # Durée de validité du cache
if os.path.isfile(os.path.dirname(__file__) + '/client_geo.pkl'):
    filetime = os.path.getmtime(os.path.dirname(__file__) + '/client_geo.pkl')
    file_age = datetime.timedelta(seconds=time.time()-filetime)
    if file_age < max_age:  # Utilisation du cache si valide
        client_geo = pd.read_pickle(os.path.dirname(__file__) + '/client_geo.pkl')

# Selection des IPs dont la position n'est pas dans le cache
df_missing = pd.concat([df, client_geo]).drop_duplicates(
    subset=['id_client'],
    keep=False
)

# Appels à l'API
retries = 5  # Nombre de tentatives en cas d'échec
for row in df_missing.itertuples(index=False):
    try:
        # Appel à l'API ip2location
        response = requests.get(
            f'https://api.ip2location.io/?key=7149995C158ABB38F6BCE6505B006082&ip={
               row.ip}&format=json',
            timeout=10
        )

        # Si bonne reponse...
        if response.status_code == 200:
            data = response.json()
            # ...ajout des données de géolocalisation aux résultats
            client_geo = pd.concat([
                client_geo,
                pd.DataFrame([[row.id_client,
```

```
                                data.get('country_name'),
                                data.get('region_name'),
                                data.get('city_name')]],
                            columns=client_geo.columns)
            ], ignore_index=True)
        # Sinon...
        else:
            # ...si il y a encore des tentatives disponibles...
            if retries:
                retries -= 1
                time.sleep(1)  # Attend plus longtemps en cas d'echec
                continue #...on continue avec le client suivant
            else:
                raise Exception(f'API error: {response.status_code}')

    except Exception as e:
        print(f"Error processing IP {row.ip}: {str(e)}")
        break

    time.sleep(0.2)  # Respect du rate limiting de l'API

# Mise à jour du cache si nécessaire
if not file_age or file_age > max_age:
    client_geo.to_pickle(os.path.dirname(__file__) + '/client_geo.pkl')

# Envoi vers BigQuery
pandas_gbq.to_gbq(
    client_geo,
    'carttrend_transform.Clients_Geo',
    project_id='carttrend-4',
    if_exists='replace'
)
```

Annexe C

Orchestration Airflow

Listing C.1 – DAG Airflow orchestrant le pipeline ELT/ML complet

```python
# [Fichier : carttrend_data_pipeline.py]
# DAG Airflow orchestrant le pipeline ELT/ML complet :
# 1. Synchronisation des données brutes via Fivetran
# 2. Transformation avec DBT
# 3. Exécution des modèles ML
# 4. Enrichissement via API externe

from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime, timedelta
import subprocess

# ----------------------
# Définition des tâches
# ----------------------

def start_fivetran_sync():
    # Déclenche la synchronisation des données sources via Fivetran
    print("Starting Fivetran Sync Task")
    subprocess.run(['python3', '../../airflow/scripts/start_fivetran_sync.py'])

def start_dbt_build():
    # Lance les transformations DBT (staging + marts)
    print('Starting DBT Build Job')
    subprocess.run(['python3', '../../airflow/scripts/start_dbt_build_job.py'])

def start_ml_prediction_ca():
    # Exécute le modèle Prophet de prévision de CA
    print('Starting ML Prediction CA')
    subprocess.run(['python3', '../../airflow/scripts/ml_prediction_ca.py'])

def start_ml_a_priori():
    # Exécute l'analyse des associations produits
    print('Starting ML APriori')
    subprocess.run(['python3', '../../airflow/scripts/ml_a_priori.py'])

def start_ml_regression_machines():
    # Exécute le modèle XGBoost d'analyse des pannes
    print('Starting ML Regression Machines')
    subprocess.run(['python3', '../../airflow/scripts/ml_regression_machines.py'])

def get_customer_locations():
    # Appel l'API ip2location pour géolocaliser les clients
    print('Starting Customer Location API Call Task')
    subprocess.run(['python3', '../../airflow/scripts/get_customer_location.py'])
```

```
46
47 # ----------------------
48 # Configuration du DAG
49 # ----------------------
50
51 default_args = {
52     'owner': 'airflow',
53     'start_date': datetime(2025, 5, 27),
54     'schedule_interval': '@daily',  # Exécution quotidienne à minuit
55     'retries': 1,  # Nombre de tentatives en cas d'échec
56     'retry_delay': timedelta(minutes=30),  # Délai entre les tentatives
57     'email_on_failure': True,  # Notification par email (configuration SMTP prealable
       )
58     'email': ['carttrenddata4@gmail.com']  # Destinataires
59 }
60
61 # Initialisation du DAG
62 dag = DAG(
63     'carttrend_data_pipeline',  # Nom du DAG
64     default_args=default_args,
65     description='Pipeline ELT/ML complet pour CartTrend',
66     catchup=False  # Désactive les runs rétroactifs
67 )
68
69 # ----------------------
70 # Définition des opérateurs
71 # ----------------------
72
73 t1 = PythonOperator(
74     task_id="start_fivetran_sync",
75     python_callable=start_fivetran_sync,
76     dag=dag,
77 )
78
79 t2 = PythonOperator(
80     task_id="start_dbt_build",
81     python_callable=start_dbt_build,
82     dag=dag,
83 )
84
85 t3 = PythonOperator(
86     task_id="start_ml_prediction_ca",
87     python_callable=start_ml_prediction_ca,
88     dag=dag,
89 )
90
91 t4 = PythonOperator(
92     task_id="start_ml_regression_machines",
93     python_callable=start_ml_regression_machines,
94     dag=dag,
95 )
96
97 t5 = PythonOperator(
98     task_id="start_ml_a_priori",
99     python_callable=start_ml_a_priori,
100    dag=dag,
101 )
102
103 t6 = PythonOperator(
104     task_id="get_customer_locations",
105     python_callable=get_customer_locations,
106     dag=dag,
107 )
108
```

```
# ----------------------
# Orchestration des tâches
# ----------------------

# Schéma d'exécution:
# 1. Fivetran (t1) doit terminer avant DBT (t2)
# 2. DBT (t2) doit terminer avant les modèles ML (t3,t4,t5) et geoloc (t6)
t1 >> t2
t2 >> [t3, t4, t5, t6]
```

Listing C.2 – Script de declenchement de synchronisation des données via l'API Fivetran

```
# [Fichier : start_fivetran_sync.py]
# Script de declenchement de synchronisation des données via l'API Fivetran :
# 1. Déclenchement de la synchronisation
# 2. Vérification périodique de l'état
# 3. Gestion des erreurs

import requests
import time

# Déclenche une synchronisation via l'API Fivetran
def start_sync():
    url = "https://api.fivetran.com/v1/connections/haunt_charming/sync"
    headers = {
        "Accept": "application/json;version=2",
        "Authorization": "Basic
           dWY4ZDBmMUZ1WGZ5Z3M5cTpacDJFYjVlRDZzQUp0ZVUxWG1ZUGtlWGV2T1R2Z2Vyag==",
        "content-type": "application/json"
    }

    # Envoi de la requête POST pour démarrer le sync
    response = requests.post(url, json={'force' : False}, headers=headers)
    response.raise_for_status() # Si echec, lève une exception
    return True

# Récupère le statut actuel de la synchronisation
def get_sync_state():
    url = "https://api.fivetran.com/v1/connections/haunt_charming"
    headers = {
        "Accept": "application/json;version=2",
        "Authorization": "Basic
           dWY4ZDBmMUZ1WGZ5Z3M5cTpacDJFYjVlRDZzQUp0ZVUxWG1ZUGtlWGV2T1R2Z2Vyag=="
    }

    # Envoi de la requete GET a l'API Fivetran
    response = requests.get(url, headers=headers)
    response.raise_for_status()
    # Retourne l'etat du transfer ('syncing', 'paused', etc.)
    return response.json()['data']['status']['sync_state']

if __name__ == '__main__':
    retries = 5 # Nombre maximum de tentatives d'obtention du statut

    # Phase 1: Démarrage de la synchronisation
    start_sync()
    print('Sync Requested')

    # Phase 2: Verification periodique du statut de la synchronisation
    sync_started = False
    while True:
        try:
            sync_state = get_sync_state()

        # Lève une exception si la demande de statut n'aboutit pas 5 fois
```

```
        except Exception as e:
            print(f'Could not get sync status : {e} - Retries : {retries}')
            if retries:
                retries -= 1
            else:
                raise Exception('Could not get sync status')

        # Si on a une réponse de l'API...
        else:
            # ...et qu'il est fini : stop.
            if sync_state != 'syncing' and sync_started:
                break

            # ...et qu'il n'a pas encore commence...
            if sync_state != 'syncing' and not sync_started:
                continue

            # ...et qu'il est toujours en cours...
            if sync_state == 'syncing':
                sync_started = True
                print('Syncing')

            # ...et que la synchronisation est un echec : exception.
            else:
                raise Exception('Sync Failed !')

        # ...attendre 3 secs et recommencer.
        time.sleep(3)

    print('Syncing Done')
```

Listing C.3 – Script de declenchement du Job DBT Build via l'API DBT

```
# [Fichier : start_dbt_build_job.py]
# Script de declenchement du Job DBT Build via l'API DBT :
# 1. Déclenchement du job
# 2. Vérification périodique de l'état
# 3. Gestion des erreurs

import requests
import time
import enum

# Configuration d'accès à l'API DBT Cloud
# Tout ces identifiants ont ete crees sur l'interface DBT Cloud
ACCOUNT_ID = '70471823462199' # Identifiant du compte DBT
JOB_ID = '70471823467186' # Identifiant du job
API_KEY = 'dbtu_57j1bQUrnaKY_XfO3UdF2kHC9g4DkgZNAmyAy2D3QAgQmxIqiM' # Token d'API

# Enumeration des statuts possibles d'un job DBT
# Les codes sont documentés sur la doc DBT :
# https://docs.getdbt.com/dbt-cloud/api-v2#/operations/List%20Runs
class DbtJobRunStatus(enum.IntEnum):
    QUEUED = 1
    STARTING = 2
    RUNNING = 3
    SUCCESS = 10
    ERROR = 20
    CANCELLED = 30

# Déclenche l'exécution du job DBT et retourne l'ID d'exécution
def trigger_job():
    url = f'https://jp371.us1.dbt.com/api/v2/accounts/{ACCOUNT_ID}/jobs/{JOB_ID}/run/
        '
    headers = {'Authorization' : f'Token {API_KEY}'}
```

```
    payload = {'cause' : 'Triggered from Airflow'}

    # Envoi de la requete POST a l'API DBT
    response = requests.post(url, json=payload, headers=headers)
    response.raise_for_status() # Si echec, lève une exception

    # Retourne l'ID d'exécution
    job_run_id = response.json()['data']['id']
    return job_run_id

# Récupère le statut actuel d'une exécution DBT
def get_job_status(job_run_id):
    url = f'https://jp371.us1.dbt.com/api/v2/accounts/{ACCOUNT_ID}/runs/{job_run_id}/
        '
    headers = {'Authorization': f'Token {API_KEY}'}

    # Envoi de la requete GET a l'API DBT
    response = requests.get(url, headers=headers)
    response.raise_for_status()

    # Retourne le statut de l'execution
    return response.json()['data']['status']


if __name__ == '__main__':
    retries = 5 # Nombre maximum de tentatives d'obtention du statut

    # Phase 1: Déclenchement du Job
    job_run_id = trigger_job()
    print(f'DBT Build Job Triggered : {job_run_id}')

    # Phase 2: Verification periodique du statut du Job
    while True:
        # Demande l'etat du transfer a Fivetran
        try:
            status = get_job_status(job_run_id)

        # Lève une exception si la demande de statut n'aboutit pas 5 fois
        except Exception as e:
            print(f'Could not get job status : {e} - Retries : {retries}')
            if retries:
                retries -= 1
            else:
                raise Exception('Could not get job status')

        # Si on a une réponse de l'API...
        else:
            # ...et qu'il est fini : stop.
            if status == DbtJobRunStatus.SUCCESS:
                break
            # ...et qu'il s'est produit une erreur : exception.
            elif status == DbtJobRunStatus.ERROR or status == DbtJobRunStatus.
                CANCELLED:
                raise Exception("Job Failed !")

        # ...attendre 3 secs et recommencer.
        time.sleep(3)

    print('Job Finished')
```

Annexe D

Dashboards

FIGURE D.1 – Analyse Marketing

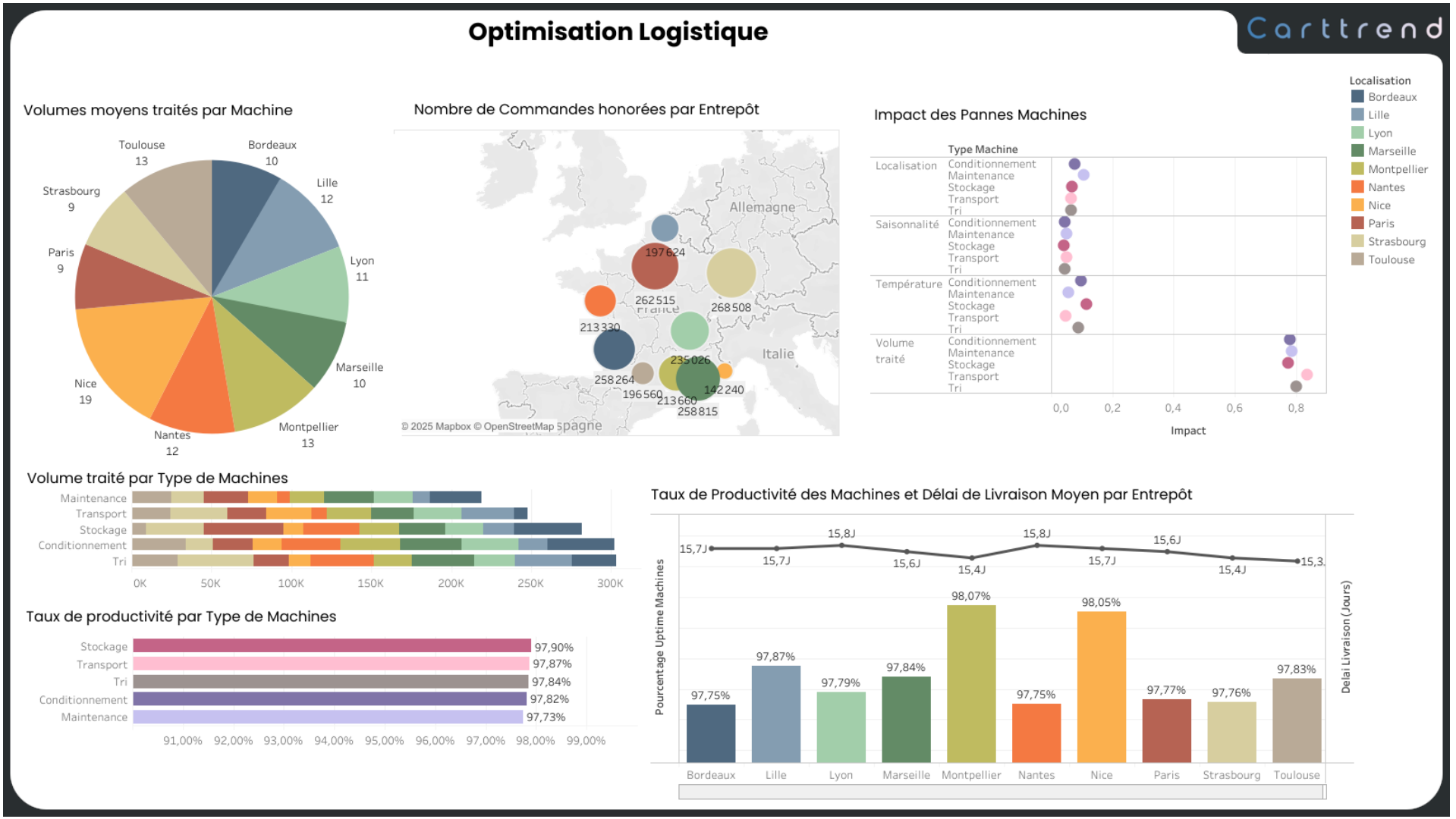


FIGURE D.2 – Optimisation Logistique

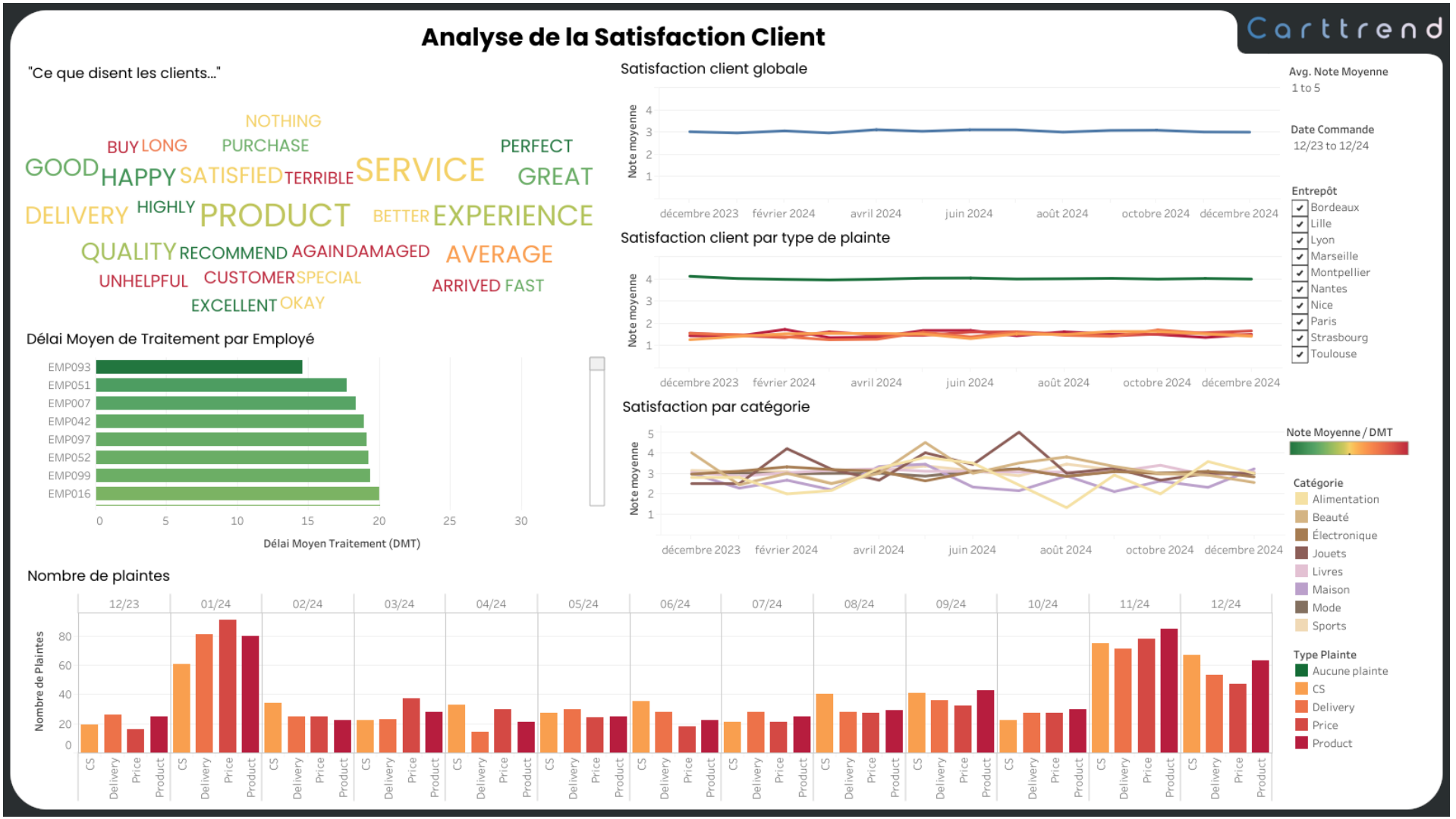


FIGURE D.3 – Analyse de la Satisfaction Client

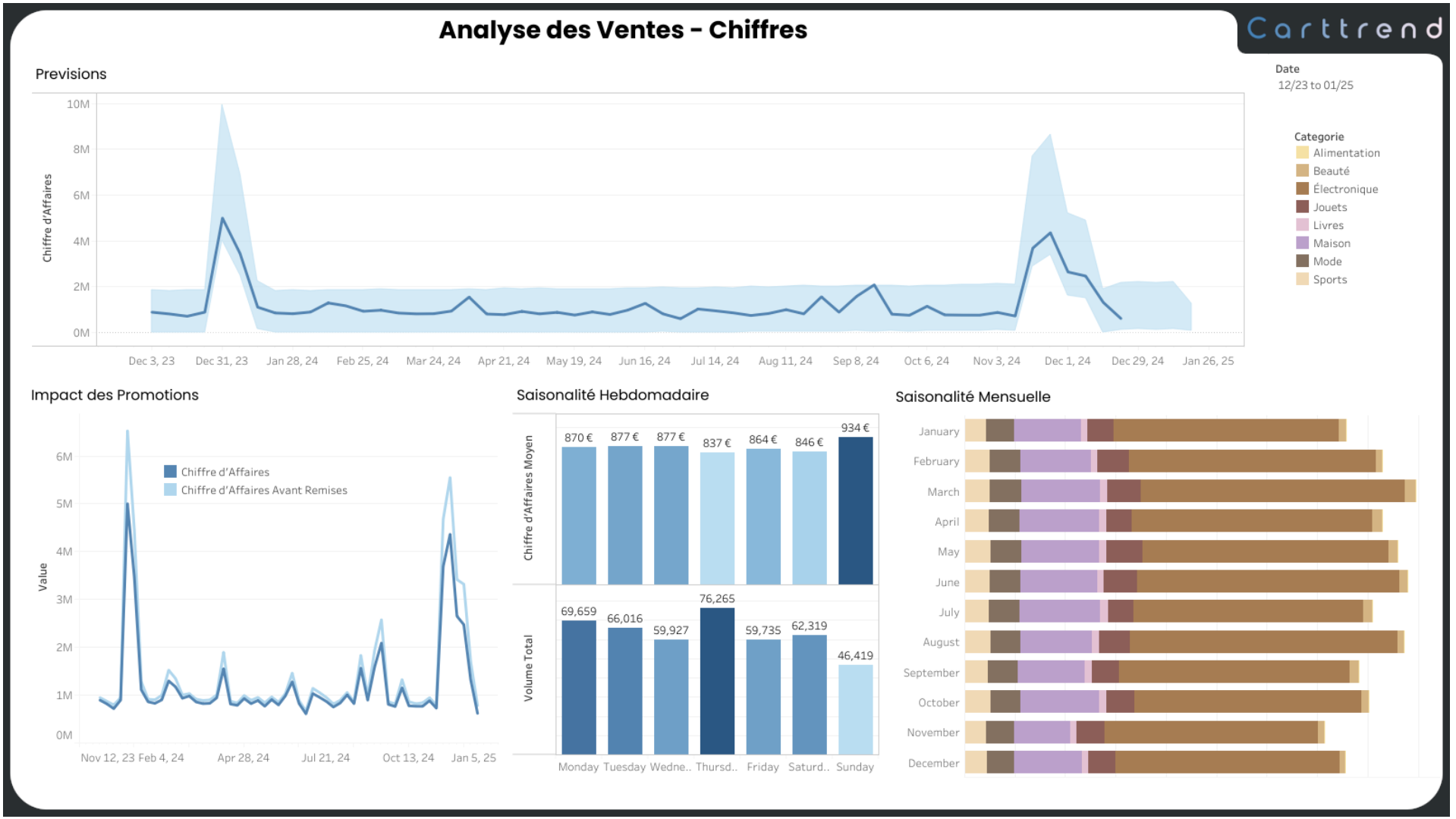


FIGURE D.4 – Ventes - Chiffres

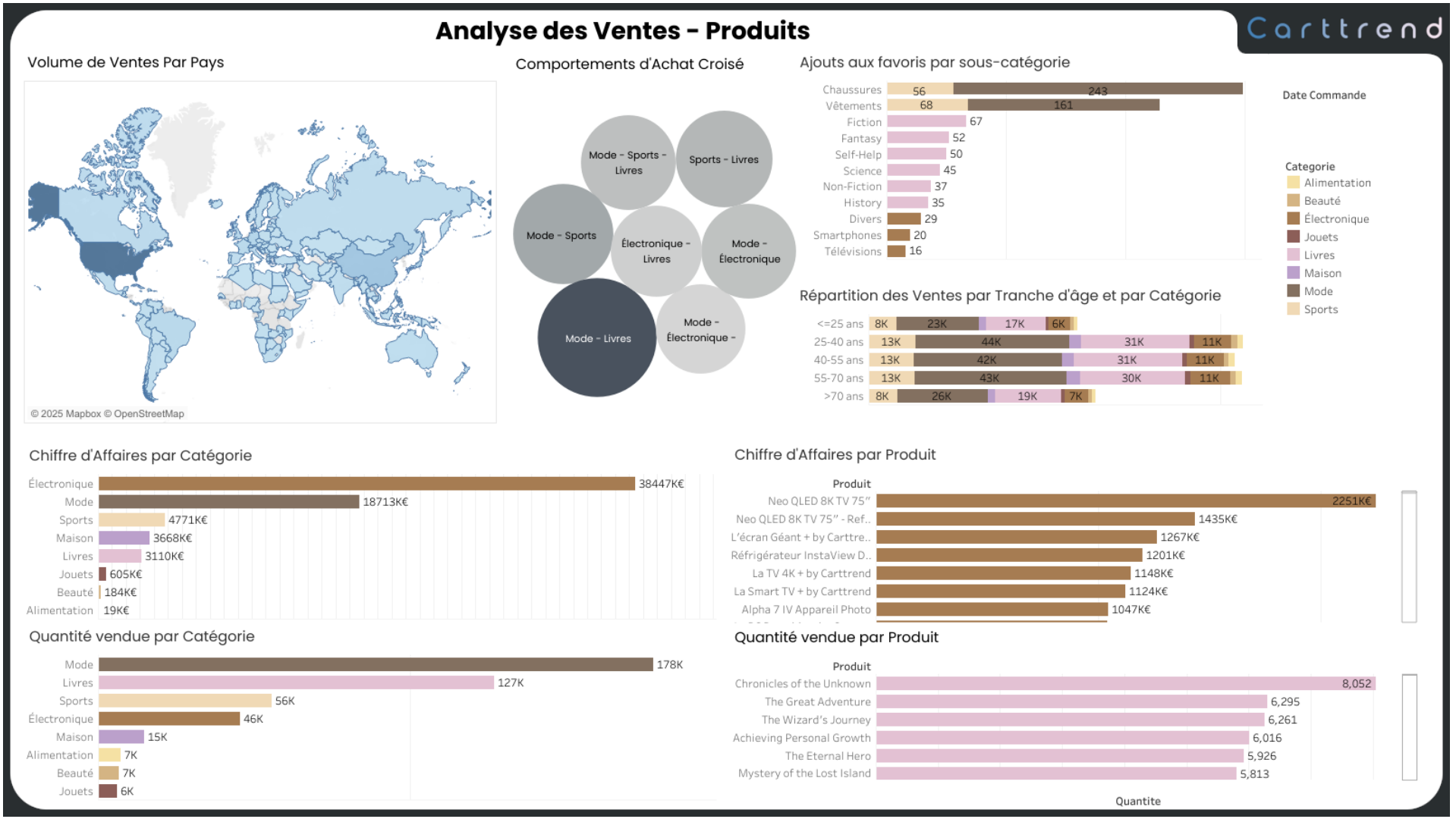


FIGURE D.5 – Ventes - Produits