

DOSSIER DE PRESENTATION PROJET

DATA ANALYST & IA

Bienvenue chez Carttrend

Décembre 2024

Par

Laurent BRUNET

Carttrend

Projet final réalisé en collaboration avec

Yanis BIDAK, Ghazi EL AYDI et Isis ZAKY



INTRODUCTION

Ce mémoire est une synthèse des pratiques et des savoirs acquis durant ma formation de trois mois à La Capsule Academy en distanciel cette fin d'année 2024.

Celui-ci se présente comme une description exhaustive d'un projet de Data Analyst Junior dans le cadre d'une entreprise fictive de e-commerce, Carttrend.

Notre projet, commencé le lundi 9 décembre 2024, consiste à comprendre le présent et anticiper l'avenir de cette entreprise grâce à l'analyse de données dans quatre domaines :

- Ventes : Identifier les produits et catégories les plus performants ;
- Satisfaction client : Repérer les facteurs d'insatisfaction ;
- Marketing : Mesurer l'impact des campagnes publicitaires ;
- Logistique : Déetecter les goulots d'étranglement et optimiser la répartition des flux.

Je vais détailler dans chacune des parties, les différentes étapes du pipeline ELT du projet, pipeline qui a permis de traiter les données d'entrée pour pouvoir en extraire des visuels et des synthèses exploitables.

La première partie sera consacrée à notre organisation générale, le choix d'un modèle en flocon et le choix du type de pipeline (ELT par rapport à l'ETL) et la création du projet sur BigQuery.

Une attention sera portée sur notre méthode et notre manière de nous partager les tâches dès le premier jour ainsi que notre planning.

La seconde partie sera consacrée à l'étape de l'extraction via Airflow en code Python vers Big Query. Il s'agira de détailler les étapes Extract et Load de notre pipeline.

La troisième partie sera consacrée à l'étape de transformation en code SQL et le détail de notre modélisation (modèles de staging et modèles analytiques) sur DBT Cloud.

Enfin la quatrième et dernière partie sera consacrée à la visualisation par PowerBI et notre démarche en Machine Learning, démarche cherchant à répondre aux questions posées à l'étape de l'Optimisation Logistique. Puis elle se terminera sur notre tentative de prédire les ventes futures par cette même méthode.

SOMMAIRE

INTRODUCTION.....	02
-------------------	----

PREMIÈRE PARTIE

Prise de connaissance du projet et choix du pipeline ELT.....	06
---	----

1. Prise de connaissance du projet

A. Etude de l'énoncé et définition du planning.....	07
B. Evidence d'un modèle en flocon.....	08

2. Choix du type de pipeline

A. Pipeline ELT.....	09
B. Précisions sur ce qu'aurait pu être un processus ETL.....	10
C. Choix des outils du pipeline.....	11

DEUXIEME PARTIE

Préparation de BigQuery et édition du code Python pour Airflow - Étapes Extract et Load du pipeline ELT -.....	13
--	----

1. Préparation du projet sur BigQuery

A. Création du projet BigQuery.....	14
B. Création du dataset BigQuery et ajout des tables.....	15

2. Extraction sur Airflow – incluant l'étape de pré-transformation via Python

A. Code Python définissant le DAG.....	16
B. Fonctionnement du DAG sur l'interface d'Airflow.....	23
C. Chargement des données sur BigQuery depuis Airflow.....	25

TROISIÈME PARTIE

Configuration de DBT Cloud et édition du code SQL sur cette interface
- Étape Transformation du pipeline ELT -.....27

1. Initialisation de DBT

- A. Intérêt de DBT dans un pipeline ELT.....28
- B. Paramétrage de DBT Cloud.....28
- C. Fonction Python intégrée dans le DAG pour lancer un run DBT.....29

2. L'interface de DBT et création de scripts SQL

- A. Structure et scripts SQL dans DBT.....30
- B. Vues créées par DBT dans BigQuery.....34
- C. Collaboration sur DBT avec GitHub.....34

QUATRIÈME PARTIE

Visualisation des données transformées sur PowerBI – Point sur le Machine Learning.....36

1. Récapitulatif du processus ELT à ce stade avant la visualisation PowerBI

- A. Extraction et chargement des données vers BigQuery.....37
- B. Transformation via DBT Cloud.....37

2. PowerBI et visualisation des données

- A. Choix de PowerBI par rapport à Tableau.....37
- B. Chargement des données transformées sur PowerBI.....38
- C. Étape de visualisation.....38

3. Notre utilisation du Machine Learning.....40

CONCLUSION.....42

ANNEXES.....44

PREMIÈRE PARTIE

Prise de connaissance du projet et choix du
pipeline ELT

1. Prise de connaissance du projet

A. Etude de l'énoncé et définition du planning

Le premier jour du projet, nous nous sommes retrouvés, l'ensemble du groupe, dans les locaux de La Capsule Academy à Paris pour faire un point sur l'organisation et le partage des tâches. Dès le matin, nous nous sommes attachés à étudier l'énoncé, divisé en 4 étapes distinctes :

- L'analyse des ventes
- L'analyse de la satisfaction client
- L'analyse marketing
- Et enfin l'analyse de l'optimisation logistique.

Ces 4 étapes servent à cadrer, repérer et orienter les KPIs de Carttrend.

Un KPI (*Key Performance Indicator*), ou Indicateur Clé de Performance en français, est une mesure quantitative utilisée pour évaluer les performances d'une organisation, d'un processus ou d'un projet par rapport à des objectifs stratégiques définis.

Nous avons projeté un planning dès le premier jour et avons défini la préparation du compte et du projet BigQuery pour le jour 1 ainsi que la définition du schéma de modélisation puis du choix de pipeline (ETL ou ELT).

Les jours suivants et ce jusqu'au jeudi de la semaine 1, nous nous sommes attachés à définir et détailler l'étape de l'extraction et de pré-transformation sur Python avec Airflow.

L'opération de chargement vers BigQuery a été faite le jeudi de la semaine 1 également avec Airflow.

Nous sommes ensuite passé du jeudi de la semaine 1 jusqu'au mardi de la semaine 2 sur dbt sur l'étape de transformation.

L'opération a été plus longue que prévue :

En effet, il était prévu que nous passions à la visualisation PowerBI le lundi de la semaine 2 et au machine learning au plus tard le mardi de la même semaine.

L'étape PowerBI a duré jusqu'au jeudi de la semaine 2 et nous avons donc eu peu de temps pour traiter cette phase de machine learning mais avons amorcé une mise en œuvre pour l'optimisation logistique.

B. Evidence d'un modèle en flocon

Le premier jour, nous avons étudié les différents fichiers reçus et avons constaté l'évidence de l'utilisation d'un datawarehouse tel que BigQuery pour pouvoir stocker nos données. Dans cette optique, nous avons étudié les différents datasets et en avons examiné quel serait le type de modélisation le plus pertinent – soit en étoile, soit en flocon, soit en constellation.

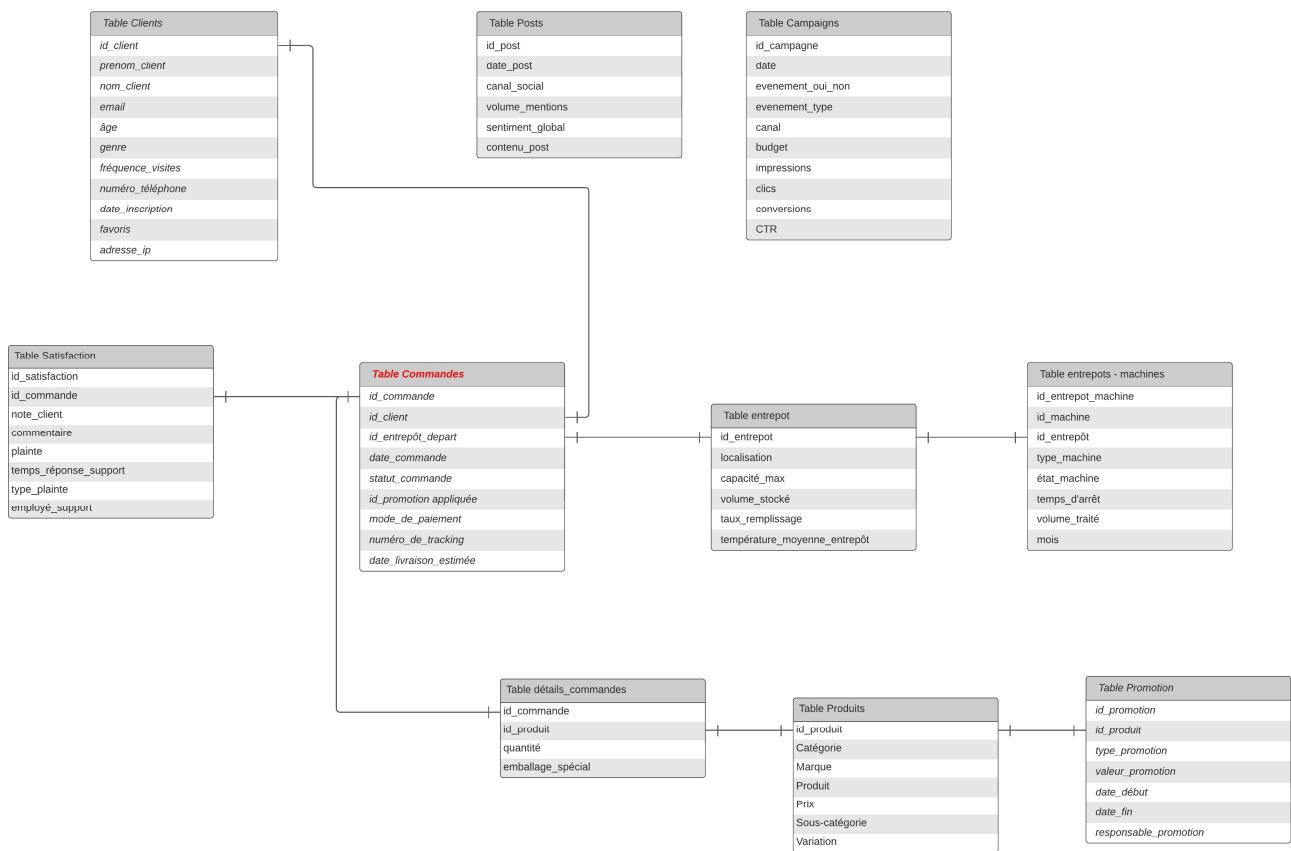
Les modèles en étoile, en flocon ou en constellation sont des schémas de modélisation des bases de données relationnelles utilisés dans les entrepôts de données (*datawarehouses*).

Ils organisent les données pour faciliter l'analyse et les requêtes analytiques.

Le modèle en flocon, une variante du modèle en étoile, dispose d'une table centrale de faits, la table « Commandes » et de tables de dimensions normalisées. Cela signifie que certaines dimensions sont divisées en plusieurs sous-tables pour réduire la redondance des données - ce qui peut rendre les jointures plus complexes mais plus optimisées en termes de stockage.

Nous avons créé la visualisation ci-dessous pour nous aider dans les tâches suivantes, et notamment pour plus de souplesse dans la gestion des jointures des différentes tables.

Modèle en flocon du projet Carttrend ; la table de faits est ici la table « Commandes » :



2. Choix du type de pipeline

A. Pipeline ELT

Un choix s'offrait à nous pour le type de pipeline soit ETL soit ELT.

Dans le workflow d'un Data Analyst, le pipeline est le fil conducteur qui transporte et transforme les données de bout en bout, permettant ainsi d'obtenir, de manière automatique et fiable, des informations prêtes à être analysées ou visualisées. Il en existe deux types, soit ETL - Extract, Transform, Load, soit ELT - Extract, Load, Transform. Son choix est donc crucial.

Le pipeline mis en œuvre durant ce projet final s'inscrit dans le cadre d'un processus ELT car les données sont chargées dans BigQuery avant de les transformer à l'aide de dbt (acronyme de *Data Build Tool*).

Par ailleurs, on peut ajouter que dbt est connu pour surtout être utilisé en tant qu'outil ELT.

Nous avons fait ce choix car l'approche ELT profite pleinement des capacités de calcul et de stockage des datawarehouses modernes, offrant ainsi une plus grande flexibilité, une meilleure scalabilité et une architecture souvent plus simple à maintenir qu'un classique ETL. Le choix d'un ELT fut aussi une évidence car une fonctionnalité nous a été indiquée lors de l'envoi de l'énoncé : en effet, il nous a été précisé à ce moment que les données pouvaient être mises à jour automatiquement.

En résumé, notre pipeline correspond davantage à un ELT qu'à un ETL car :

1. Nous extrayons depuis Google Sheets ;
2. Nous prétraitons (légers nettoyages, conversions de dates, etc.), mais ces opérations restent assez mineures (pas de modélisation complexe, ni d'agrégation en dur) ;
3. Nous chargeons ensuite les données directement (ou presque directement) dans BigQuery, donc la base de données contient encore des tables « raw » ou « staging » qui sont relativement brutes ;
4. Enfin, nous transformons ensuite via dbt, qui exécute ses modèles et requêtes sur BigQuery (c'est-à-dire dans le datawarehouse même).

B. Précisions sur ce qu'aurait pu être un processus ETL

Le modèle ELT surpassé souvent l'ETL dans des environnements modernes grâce à sa flexibilité, sa simplicité et sa capacité à exploiter pleinement les datawarehouses les plus récents.

L'ETL peut donc être écarté dans la plupart des cas, sauf si des contraintes spécifiques liées à la sécurité, aux réglementations ou à des limitations techniques l'imposent.

Cependant, en imaginant le choix d'option de pipeline ETL exclusif, toutes les transformations (incluant celles de dbt) auraient été effectuées avant de charger les données dans BigQuery : dans ce type de modèle, les données sont extraites, transformées hors de l'entrepôt (souvent dans un système intermédiaire ou en mémoire), puis chargées dans leur forme finale dans l'entrepôt. Les transformations incluent typiquement des calculs complexes, des agrégations, et des normalisations effectuées avant le chargement.

C. Choix des outils du pipeline

L'option d'utiliser n8n, un outil à l'ergonomie plus visuelle et plus intuitive, a été brièvement envisagé (en remplacement de dbt) mais la complexité du workflow du projet et le volume important de données - volume pouvant également être mis à jour - nous a rapidement poussé à l'écartier.

Après études, nous avons retenu les outils que sont Airflow, dbt et BigQuery et ceci pour plusieurs raisons, spécifiques à chacun :

Airflow pour commencer :

Airflow est le « chef d'orchestre » du workflow, gérant l'automatisation et la supervision des processus de bout en bout. Ses avantages sont :

1. La planification et l'automatisation : Il déclenche et surveille les pipelines avec une gestion des dépendances ;
2. La flexibilité et la résilience : Compatible avec des workflows simples ou complexes, il offre une interface pour gérer les échecs et relancer les tâches.
3. Enfin, son rôle clé : Airflow coordonne l'extraction des données, les transformations via dbt et le stockage dans BigQuery.

dbt ensuite :

dbt (*data build tool*) se concentre sur les transformations de données directement dans BigQuery. Ses avantages sont :

1. Les transformations en SQL : il crée des modèles de données complexes, accessibles et réutilisables ;
2. La documentation et les tests : il génère automatiquement des documentations interactives et des tests garantissant la qualité des données ;
3. Son rôle clé est de structurer et normaliser les données tout en apportant de la modularité et également de la transparence.

Enfin BigQuery :

BigQuery est un datawarehouse « cloud » conçu pour traiter des données massives ; la liste de ses avantages :

1. Il propose un traitement élastique : il exécute rapidement des requêtes complexes grâce à sa scalabilité (son évolutivité en d'autres termes) automatique ;
2. Il gère des formats modernes : il supporte des données structurées et semi-structurées (JSON, Parquet) ;
3. Son rôle clé est qu'il stocke et traite les données brutes et transformées avec simplicité et efficacité.

Enfin, l'aspect sans doute le plus important dans notre choix est la parfaite complémentarité de ces trois outils entre eux.

- Airflow orchestre les processus.
- dbt gère les transformations et la documentation.
- BigQuery fournit la puissance de calcul et le stockage

DEUXIEME PARTIE

Préparation de BigQuery et édition du code

Python pour Airflow

- Etapes Extract et Load du pipeline ELT -

1. Préparation du projet sur BigQuery

A. Création du projet BigQuery

Le premier jour du projet, nous avons initié en fin d'après-midi, peu après le choix des outils, un compte Google Cloud pour la préparation initiale des données.

Le nom du compte est projet-carttrend@gmail.com.

Nous avons ensuite créé un compte de service en accédant à la console IAM ; nous l'avons nommé CARTTREND-CDS, avec l'ID suivante :

carttrend-cds@projet-carttrend.iam.gserviceaccount.com.

Nous avons ensuite défini 4 autorisations principales liées à 4 fonctionnalités :

- Administrateur BigQuery : pour gérer entièrement les datasets, les tables et les autorisations ;
- Editeur de données BigQuery : pour pouvoir modifier les données ;
- Lecteur de données BigQuery : pour pouvoir consulter les données ;
- Utilisateur BigQuery : pour pouvoir utiliser les données.

Les comptes de service sont des comptes techniques utilisés par des applications ou des scripts, auxquels on attribue des rôles pour leur donner un niveau de privilèges précis.

Les rôles déterminent quelles actions sont autorisées ou interdites sur les ressources BigQuery (ou tout autre service GCP - Google Cloud Platform).

C'est grâce à ces deux mécanismes que GCP assure la sécurité et la gestion granulaires (une gestion avec contrôle fin et précis des autorisations) des accès sur BigQuery et l'ensemble des services Cloud.

Capture d'écran du compte de service sur BigQuery :

Type	Compte principal ↑	Nom	Rôle	Insights sur la sécurité
cartrtrend-cds@projet-carttrend.iam.gserviceaccount.com	CARTTREND-CDS	Administrateur BigQuery		
		Éditeur de données BigQuery		
		Lecteur de données BigQuery		
		Utilisateur BigQuery		

Nous avons ensuite édité une clé de service, tout d'abord en sélectionnant le compte de service créé puis en cliquant ensuite sur « Ajouter une clé ».

Clé ensuite enregistrée et exploitable sous le format JSON.

Ce fichier JSON sera ensuite utilisé pour l'authentification dans les scripts ou les applications.

Enfin, nous avons activé l'API BigQuery directement dans la bibliothèque d'API Google Cloud.

Ceci permet ensuite l'interaction entre les différents programmes.

B. Création du dataset BigQuery et ajout des tables

Sur l'interface de BigQuery, dans la console Google Cloud, nous avons créé et nommé le projet, Projet-Carttrend dans l'onglet du haut de la page :

« Sélectionner un projet » - « Nouveau projet ».

Capture d'écran de la création d'un nouveau projet sur BigQuery :

The screenshot shows the Google Cloud BigQuery interface. The top navigation bar includes 'Google Cloud', 'Projet-Carttrend', a search bar, and various icons. On the left, there's an 'Explorateur' sidebar with sections like 'Notebooks', 'Canvases de données', 'Préparations de données', 'Workflows', and 'Connexions externes'. The main area is titled 'Sélectionner un projet' and contains a search bar and two tabs: 'PROJETS RÉCENTS' and 'TOUS'. Under 'PROJETS RÉCENTS', 'Projet-Carttrend' is selected with a checkmark. Under 'TOUS', 'My First Project' is also listed. At the bottom right of the dialog, there are buttons for 'SUPPRIMER' (Delete), 'EXPORTATION' (Export), 'PROFIL DE DONNÉES' (Data Profile), and 'QUALITÉ DES DONNÉES' (Data Quality). The overall interface is clean and modern, typical of Google's cloud services.

Le dataset data_carttrend ensuite initié, nous avons commencé à éditer les différentes tables. Nous avons choisi de nommer celles-ci avec la même dénomination que chaque fichier d'origine. Notre dataset est ainsi composé de 10 tables :

- Carttrend_Campaigns
- Carttrend_Clients
- Carttrend_Commandes
- Carttrend_Details_Commandes
- Carttrend_Entrepots
- Carttrend_Entrepots_Machines
- Carttrend_Posts
- Carttrend_Produits
- Carttrend_Promotions
- Carttrend_Satisfaction

Pour les noms de colonnes, nous avons là-aussi décidé de rester fidèles au fichier d'origine. Nous avons ensuite défini le type de données de ces colonnes ; 4 types de données furent retenues pour couvrir l'ensemble du dataset, pour couvrir l'ensemble des tables :

- DATETIME pour les dates
- FLOAT pour les nombres à virgules flottantes
- INTEGER pour les nombres entiers
- STRING pour toutes les données en format texte.

2. Extraction sur Airflow – incluant l'étape de pré transformation via Python

A. Code Python définissant le DAG

Ce travail de préparation terminé, nous sommes ensuite passé à Airflow pour l'étape d'extraction et d'orchestration mais aussi de pré transformation des données.

L'étape de l'extraction des données avec Airflow est très souvent la première étape d'un pipeline ELT. Elle consiste à récupérer les données depuis différentes sources (Google Cloud ici) pour les charger dans une destination intermédiaire ou finale.

Airflow, grâce à ses opérateurs, permet d'automatiser et d'orchestrer ce processus.

Dans un premier temps, nous avons initialisé Airflow grâce Ubuntu ; Airflow devient directement accessible depuis le navigateur Microsoft Edge.

Nous avons ensuite rédigé un DAG (*Directed Acyclic Graph*) en Python sur une console Visual Studio Code, enregistré directement dans un dossier Airflow en local sur la machine virtuelle.

Nous avons choisi pour cette tâche l'option Airflow la plus classique, c'est-à-dire le fait de bien utiliser le terminal intégré (ou externe) de VS Code pour lancer les commandes Airflow (webserver, scheduler, trigger, etc.).

Le DAG est un concept fondamental pour définir et orchestrer des workflows de données. Il structure et organise les tâches dans un ordre bien précis, garantissant qu'elles sont exécutées suivant la bonne séquence et sans cycles.

En **Annexe 1**, le code Python complet disponible détaillant entre autres l'initialisation du DAG.

Voici une description de ce code bloc par bloc – la partie de code sur dbt, présente dans le même script sera abordée plus précisément par la suite :

Premier bloc - Pour commencer, le code commence par l'import des librairies.



```
 1 import os
 2 import pandas
 3 from google.cloud import bigquery
 4 import datetime
 5 from airflow import DAG, PythonOperator
 6 from requests import post
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
222
```

Deuxième bloc - Par la suite, nous définissons l'ensemble des variables de configuration.



Cette portion de code définit l'ensemble des **variables de configuration** qui seront utilisées dans le reste du script :

- **Identifiants DBT (pour l'étape suivante de transformation):**
 - L'URL de l'API DBT Cloud, l'ID du compte DBT, l'ID du job à exécuter et le jeton d'authentification.
 - Cela sert à déclencher les transformations DBT (dans le code) via l'API.
- **Configuration GCP / BigQuery :**
 - Le nom du projet GCP (project_id) et le dataset (dataset_id) dans lequel les données seront chargées.
 - Le chemin vers les credentials (GOOGLE_APPLICATION_CREDENTIALS) permettant à la bibliothèque google.cloud.bigquery de se connecter à BigQuery.
- **Liste des colonnes à interpréter comme dates** (date_columns) :
 - Indique quelles colonnes seront converties en type datetime lors de la phase de pré-transformation.
- **file_urls** : le dictionnaire associant un nom de table (ex. Cartrend_Campaigns) à l'URL Google Sheets correspondante.
 - Chaque entrée sera utilisée pour télécharger le CSV et charger les données dans BigQuery sous le nom de table correspondant.

En résumé, ce bloc **centralise tous les réglages** (identifiants, chemins, variables) nécessaires au bon fonctionnement du pipeline, avant même d'exécuter l'extraction, la transformation ou le chargement des données.

C'est aussi ici que nous énumérons et collectons les différents clés JSON – dont celle extraite depuis le compte de service BigQuery.

Troisième bloc - La suite du code est l'introduction des fonctions d'extraction, de chargement et de pré transformation.



```
def extract_all_sheets(file_urls):
    """This function takes a list of URLs from Google Sheets and returns a list of DataFrames.
    It uses pandas.read_csv() to read each sheet into a DataFrame and then concatenates them all into one main DataFrame.
    """
    dataframes = []
    for url in file_urls:
        df = pd.read_csv(url)
        dataframes.append(df)
    return dataframes
```

```
def pre_transform(dataframes):
    """This function takes a list of DataFrames and performs some basic cleaning.
    It removes columns with missing values, renames columns to remove spaces and special characters,
    and drops duplicate rows.
    """
    cleaned_dataframes = []
    for df in dataframes:
        df = df.dropna()
        df = df.rename(columns=lambda x: re.sub(r'\s+', '', x))
        df = df.drop_duplicates()
        cleaned_dataframes.append(df)
    return cleaned_dataframes
```

```
def load_to_bigquery(dataframes, file_urls):
    """This function takes a list of DataFrames and a list of URLs, and loads them into BigQuery.
    It uses the write_disposition parameter to truncate existing tables.
    """
    client = bigquery.Client()
    for df, url in zip(dataframes, file_urls):
        table_id = url.replace('https://docs.google.com/spreadsheets/d/', '')
        table_id = table_id.replace('/edit#gid=', '/table/')
        job_config = bigquery.LoadJobConfig(
            write_disposition='WRITE_TRUNCATE',
            schema=table_schema
        )
        job = client.load_table_from_dataframe(df, table_id, job_config=job_config)
        job.result()
```

Cette portion de code en Python définit 3 étapes par 3 fonctions distinctes :

L'extraction, la pré transformation et enfin le chargement.

- **Extraction**

La fonction « extract_all_sheets(file_urls) » convertit les liens Google Sheets en URLs CSV, puis utilise « pandas.read_csv() » pour créer une liste de DataFrames représentant les données brutes (un DataFrame étant la structure de données principale de la bibliothèque pandas en Python.).

- **Pré-transformation**

La fonction « pre_transform(dataframes) » convertit certaines colonnes en format date/heure, nettoie les noms de colonnes (espaces, caractères spéciaux) et supprime les doublons. Résultat : une liste de DataFrames « propres ».

- **Chargement**

La fonction « load_to_bigquery(...) » charge ces DataFrames nettoyés dans BigQuery. Chaque DataFrame est associé à sa table via le dictionnaire « file_urls ».

Avec « write_disposition="WRITE_TRUNCATE" », les tables existantes sont écrasées et recréées à chaque chargement.

- Ainsi, ces fonctions mettent en place un processus « EL » (Extraction + Load) avec un léger pré-traitement. La partie « T » (Transform) plus poussée est gérée après coup (ici, par dbt), directement dans BigQuery.

Note complémentaire sur cette partie : du « code regex », c'est-à-dire du code d'expressions régulières, repérant des chaînes de caractères avec un motif précis, a été rédigé dans la partie pré transformation pour nettoyer et mettre en forme les noms de colonnes pour qu'elles ne causent aucun conflit dans la suite du projet.

Quatrième bloc - Cette partie du code, placée vers la fin du script, décrit la fonction lançant l'exécution d'un RUN DBT, étape de transformation que nous allons voir par la suite.



```
def main():
    # Your Airflow DAG logic here

    # Call the function to run the DBT Cloud job
    run_dbt()

    # Your Airflow DAG logic here

def run_dbt():
    # Check if the DBT Cloud job is available
    if dbt_is_available():
        # Run the DBT Cloud job
        dbt_run()
    else:
        print("DBT Cloud job is not available, skipping execution.")

# Function to check DBT Cloud job availability
def dbt_is_available():
    # Implementation of the function
    pass

# Function to run the DBT Cloud job
def dbt_run():
    # Implementation of the function
    pass
```

Cette fonction **automatise l'exécution** d'un job DBT Cloud via Airflow :

- Elle **vérifie d'abord** la disponibilité du job.
- Puis elle **lance** effectivement l'exécution de ce job (la « run » DBT).

Cinquième et dernier bloc - Cette partie du code, placée à la fin du script, devrait par commodité plutôt être placée au début. Ce bloc de code définit le DAG Airflow – autrement dit, le workflow qui orchestre l'ensemble des tâches.



```
def main():
    # Extract task
    extract_task = PythonOperator(
        task_id='extract_task',
        python_callable=extract_all_sheets,
        op_kwargs={'sheet_ids': sheet_ids, 'output_file': output_file},
        dag=dag
    )

    # Pre-transform task
    pre_transform_task = PythonOperator(
        task_id='pre_transform_task',
        python_callable=pre_transform,
        op_kwargs={'df': df, 'output_file': output_file},
        dag=dag
    )

    # Load task
    load_task = BigQueryOperator(
        task_id='load_task',
        sql='sql/load_data.sql',
        params={'output_file': output_file},
        dag=dag
    )

    # DBT transform task
    dbt_transform_task = PythonOperator(
        task_id='dbt_transform_task',
        python_callable=run_dbt_transform,
        op_kwargs={'output_file': output_file},
        dag=dag
    )

    # Ordonnancement
    extract_task >> pre_transform_task >> load_task >> dbt_transform_task
```

- **with DAG(...)** : Déclare un DAG nommé "cartrend_elt_final", planifié quotidiennement (@daily) et démarrant au 10 décembre 2024.

- **Tâches** :

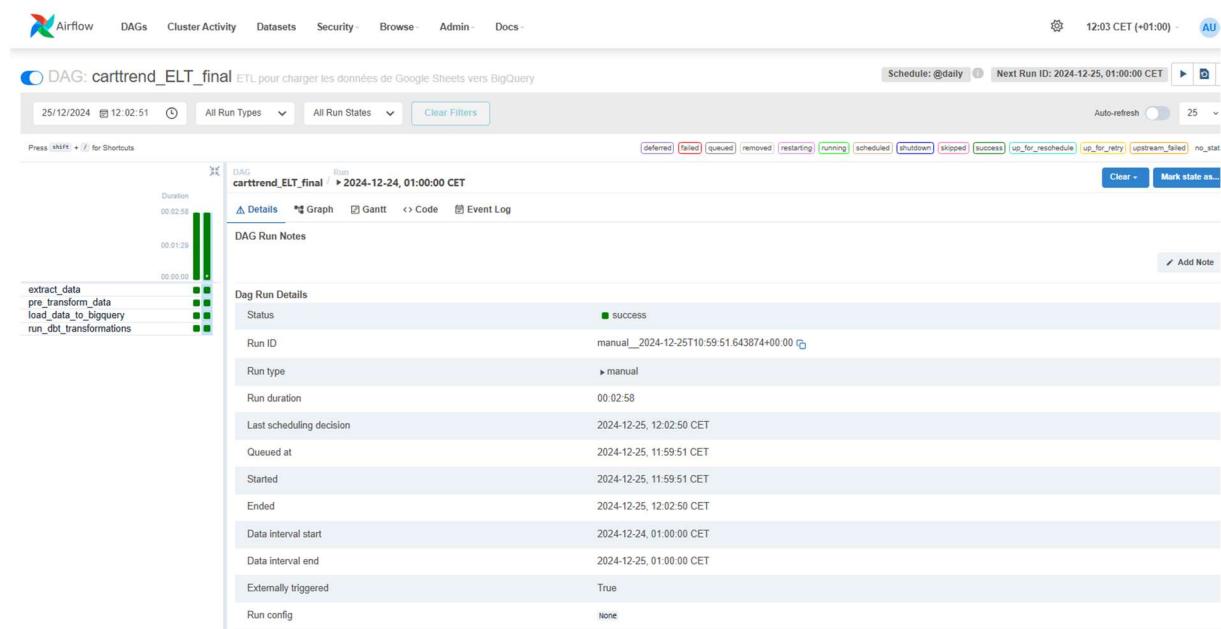
- **extract_task** : Appelle extract_all_sheets pour télécharger les données depuis Google Sheets.
- **pre_transform_task** : Nettoie et convertit les données (colonnes dates, etc.) via pre_transform.
- **load_task** : Charge les DataFrames transformés dans BigQuery grâce à load_to_bigquery.
- **dbt_transform_task** : Lance un job DBT (transformations finales) via run_dbt_transform.
- **Ordonnancement** : extract_task >> pre_transform_task >> load_task >> dbt_transform_task (chaînage séquentiel). Il s'agit de l'ordre auquel s'exécute l'ensemble des tâches précédemment citées.

B. Fonctionnement du DAG sur l'interface d'Airflow

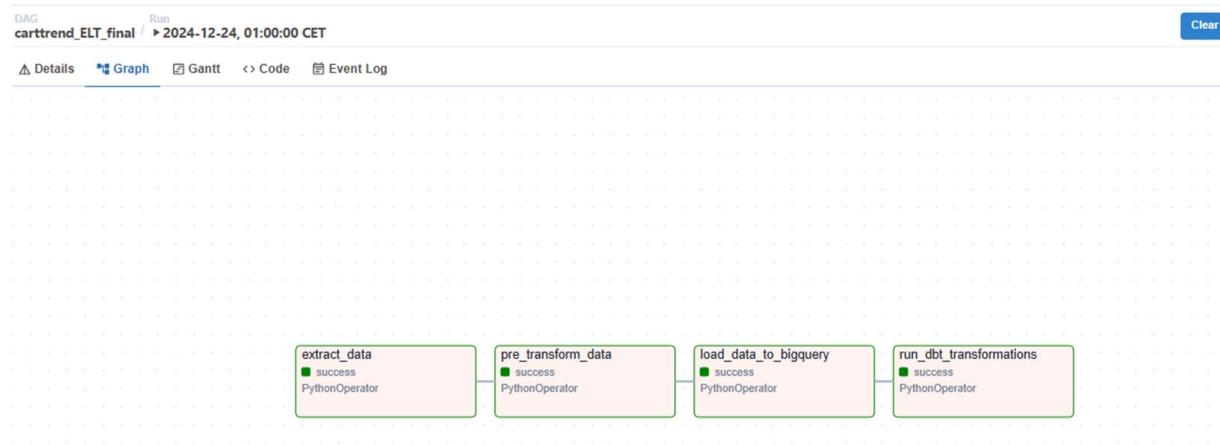
Une fois le code Python du script édité sur Visual Studio Code, le DAG peut être lancé directement sur l'interface Airflow du navigateur Microsoft Edge.

Voici les différentes informations apparaissant sur celle-ci :

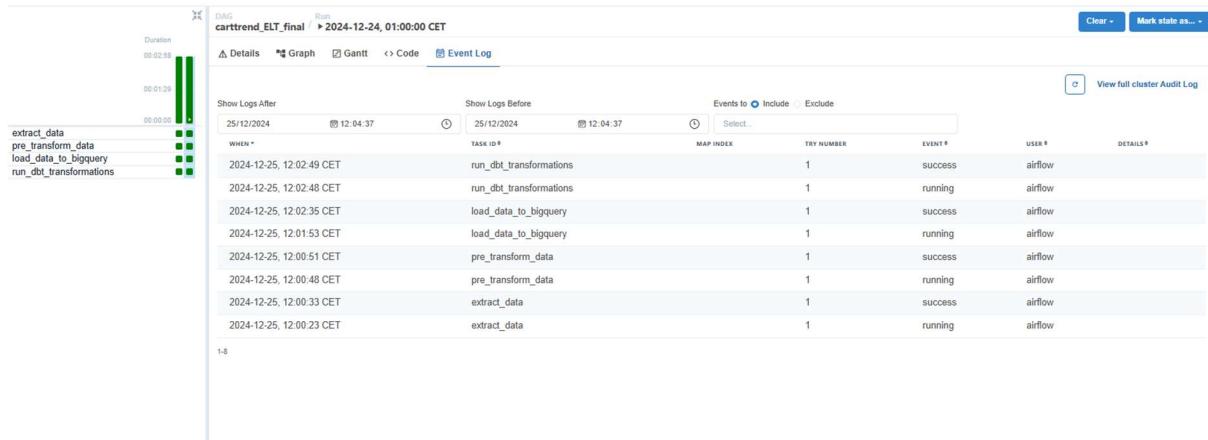
- Première image : Il s'agit de la vue *Details* du DAG carttrend_elt_final dans Airflow. On y voit le statut global de l'exécution (ici *success*), la durée totale (environ 2 minutes 58 secondes), ainsi que les métadonnées liées à la session : la date et l'heure de début, de fin, le type de run (manuel ou programmé), etc. À gauche, un histogramme montre la durée de chaque tâche (`extract_data`, `pre_transform_data`, `load_data_to_bigquery`, `run_dbt_transformations`), toutes en vert (succès).



- Deuxième image : C'est l'onglet *Graph* du même DAG. On y voit visuellement la séquence des tâches :
 - extract_data en premier ;
 - suivi de pre_transform_data ;
 - puis load_data_to_bigquery ;
 - Et enfin run_dbt_transformations. Chaque tâche est représentée par un bloc (vert pour indiquer qu'elle a abouti sans erreur), et les flèches montrent l'ordre d'exécution.



- Troisième image : Il s'agit de l'onglet *Event Log* (ou journal des événements). On y retrouve la liste des tâches, l'heure de lancement ou de fin, le *try number* (tentative d'exécution), ainsi que l'événement associé (ici, *running* puis *success*). Cette vue permet de retracer l'historique exact de l'exécution de chaque tâche, dans l'ordre chronologique.



C. Chargement des données sur BigQuery depuis Airflow

C'est la dernière partie du 3^{ème} bloc du script Python qui permet de charger ces données directement dans BigQuery :

Pour résumer les étapes précédentes, par ce code Python sur la console Visual Studio Code nous avons donc préparé les données :

- Les données sont stockées dans Google Cloud Storage (GCS) avant d'être chargées dans BigQuery.
- L'étape précédente du DAG (*pre_transform_data*) prépare généralement les données.

Une fois les données préparées, nous lançons le DAG depuis cette console :

- L'opérateur se connecte à BigQuery et exécute une tâche de type *load job*.
- Nous avons aussi configuré quelques paramètres clés :
 - Le chemin des données source (dans GCS) ;
 - La table de destination (projet, dataset, table) ;
 - Le format des données source (CSV, JSON, etc.) ;
 - Les options d'écriture (ex. : remplacer ou ajouter des données).

Airflow nous indique ou non la validation du chargement :

- Une fois le chargement terminé, BigQuery retourne un état de réussite ou une erreur que le DAG journalise dans l'onglet *Logs*.
- Une tâche de validation (optionnelle) peut être mise en place pour vérifier que les données sont bien chargées.

Le chargement terminé, nous pouvons par la suite constater que

- Les données sont disponibles dans BigQuery dans la table spécifiée.
- Les étapes suivantes du DAG peuvent accéder aux données pour transformation ou reporting (par DBT ici).

La réussite du processus nous indique que le compte de service utilisé a bien les droits sur GCS et BigQuery (lecture et écriture).

Elle nous indique également la réussite du processus et l'évitement des erreurs suivantes :

- Mauvais format de données ;
- Chemin incorrect dans GCS ;
- Permissions insuffisantes.

TROISIEME PARTIE

Configuration de DBT Cloud et édition du code

SQL sur cette interface

- Etape Transformation du pipeline ELT -

1. Initialisation de DBT

A. Intérêt de DBT dans un pipeline ELT

Ce chargement fait dans BigQuery, nous avons ensuite initialisé et utilisé DBT pour l'étape de transformation.

DBT (*Data Build Tool*) est un outil utilisé pour transformer des données brutes en données prêtes à l'analyse dans BigQuery. DBT ne gère pas l'extraction ou le chargement des données. Il prend les données déjà chargées dans un entrepôt (par des outils comme Airflow ou Fivetran) et applique des transformations en SQL pour les rendre exploitables.

B. Paramétrage de DBT Cloud

Nous nous sommes tout d'abord inscrits sur DBT Cloud via le navigateur internet puis nous avons pu accéder à l'interface de ce service. Nous avons ensuite créé un nouveau projet dans « New Project » pour pouvoir configurer ensuite notre projet.

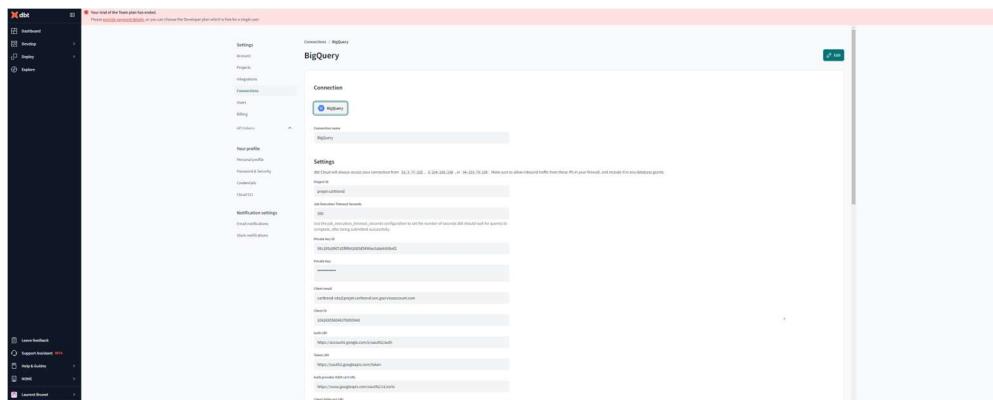
Nous avons bien entendu choisi BigQuery comme entrepôt de données cible dans « Connections ». Lors de cette étape, nous avons chargé la clé de service JSON GCP que nous avions édité auparavant (voir « Deuxième Partie » paragraphe 1.A.)

Nous avons également fourni l'identifiant du projet sur BigQuery et le dataset BigQuery où DBT pourra créer ses tables/vues.

Nous avons ensuite testé la connexion pour nous assurer que tout fonctionne.

En cliquant sur "Develop" (à gauche, dans la capture ci-dessous) dans DBT Cloud, le Cloud IDE teste automatiquement la connexion à BigQuery.

Capture d'écran de l'interface de DBT, étape de la connexion à BigQuery avec insertion de la clé JSON :



C. Fonction Python intégré dans le DAG pour lancer un run DBT

Parallèlement à cette initialisation, nous avons édité une partie de script en Python directement dans le DAG Airflow (que nous avons vu dans la partie précédente) pour pouvoir lancer automatiquement un run DBT.

Intégrée dans un DAG, cela permet de combiner l'orchestration de workflows avec les transformations de DBT.

Capture d'écran du DAG Airflow concernant DBT :

```
138
139 # -----
140 # Fonction pour lancer un RUN DBT
141 #
142
143 def run_dbt_transform():
144     """
145     Lance l'exécution d'un job DBT Cloud (identifiants spécifiés en haut).
146     """
147     headers = {
148         "Authorization": f"Bearer {DBT_CLOUD_API_TOKEN}",
149         "Content-Type": "application/json"
150     }
151
152     # Récupération de la liste de jobs DBT
153     jobs_url = f"https://cloud.getdbt.com/api/v2/accounts/{DBT_CLOUD_ACCOUNT_ID}/jobs/"
154     response = requests.get(jobs_url, headers=headers)
155
156     if response.status_code == 200:
157         jobs = response.json()["data"]
158         if jobs:
159             job_id = jobs[0]["id"]
160             print(f"Job trouvé, ID: {job_id}")
161         else:
162             print("Aucun job trouvé.")
163             return None
164     else:
165         print("Impossible de récupérer la liste des jobs. Erreur : ", response.text)
166         return None
167
168     # On déclenche le job via l'URL de RUN (DBT_CLOUD_API_URL)
169     run_url = DBT_CLOUD_API_URL
170     payload = {
171         "cause": "Airflow triggered run"
172     }
173     run_response = requests.post(run_url, headers=headers, json=payload)
174
175     if run_response.status_code == 200:
176         print("DBT run triggered successfully.")
177     else:
178         print("Failed to trigger DBT run. Error: ", run_response.text)
```

2. L'interface de DBT et création de scripts SQL

A. Structure et scripts SQL dans DBT

Pour rappel, dans notre démarche ELT, DBT se charge de la transformation :

- Extract → Load → Transform : Les données sont d'abord extraites des sources et chargées dans l'entrepôt. Ensuite, DBT transforme les données directement dans l'entrepôt (grâce à la puissance de calcul native).

La structure de notre projet est divisée en deux parties principales :

1. Staging Models (stg) : Ces modèles nettoient, standardisent et préparent les données brutes. Chaque fichier SQL (comme stg_clients_data.sql ou stg_commandes_data.sql) se concentre sur un domaine précis (clients, commandes, satisfaction, etc.).
2. Marts : Ces modèles, édités également en SQL, contiennent des analyses avancées organisées par domaines spécifiques (ventes, marketing, logistique, satisfaction client). Ils exploitent les données transformées pour calculer des KPI et produire des tables prêtes à l'analyse.

Le fichier schema.yml assure la qualité des données via des tests automatisés, tandis que source.yml connecte les sources brutes de BigQuery au projet DBT.

Enfin, toutes les étapes de configuration ayant été faites sur DBT Cloud, il n'y a pas besoin de fichier profiles.yml.

Cette structure permet une bonne modularité, des tests intégrés (ici, les fichiers schema.yml assurent la qualité des données grâce à des tests automatisés), de la documentation, et une exploitation efficace du datawarehouse, de BigQuery.

La capture suivante expose la structure de notre projet DBT, composé de modèles analytiques -marts- et de modèles de préparation -staging- ainsi que les fichiers sources et schéma :



Dans les 2 captures ci-dessous, nous retrouvons le fichier source.yml permettant de connecter les sources brutes de Google Cloud à BigQuery puis le fichier schema.yml assurant via des tests la qualité des données :

Capture d'écran du fichier source :

```
source.yml
```

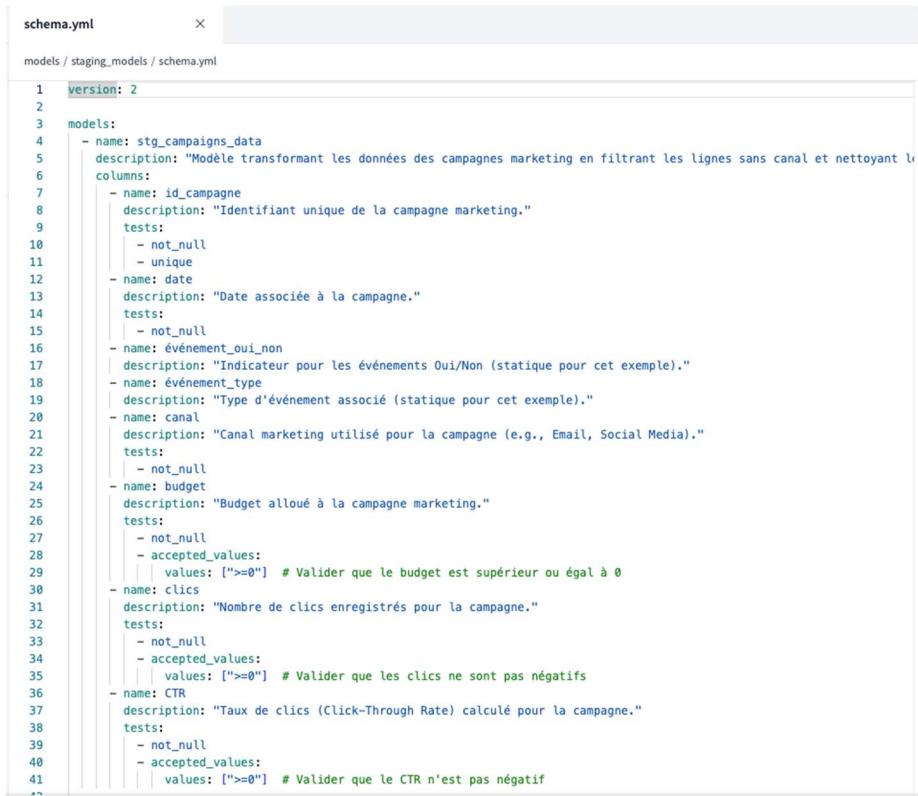
models / source.yml

```

1 version: 2
2
3 sources:
4   - name: data_cartrend
5     schema: data_cartrend
6     description: "Données brutes des campagnes marketing"
7     tables:
8       - name: Cartrend_Campaigns
9         description: "Table contenant les données brutes des campagnes publicitaires"
10        columns:
11          - name: id_campagne
12            description: "Identifiant unique de la campagne"
13            tests:
14              - unique
15              - not_null
16          - name: date
17            description: "Date de la campagne"
18            tests:
19              - not_null
20          - name: événement_oui_non
21            description: "Indique si un événement marketing a eu lieu (ex. Soldes)"
22            tests:
23              - not_null
24          - name: événement_type
25            description: "Type de l'événement marketing (ex. Soldes, Nouveauté)"
26          - name: canal
27            description: "Canal publicitaire utilisé (Google Ads, Facebook, Instagram, etc.)"
28            tests:
29              - not_null
30          - name: budget
31            description: "Budget alloué pour la campagne"
32            tests:
33              - not_null
34          - name: impressions
35            description: "Nombre d'impressions générées par la campagne"
36            tests:
37              - not_null
38          - name: clics
39            description: "Nombre de clics générés par la campagne"
40            tests:
41              - not_null

```

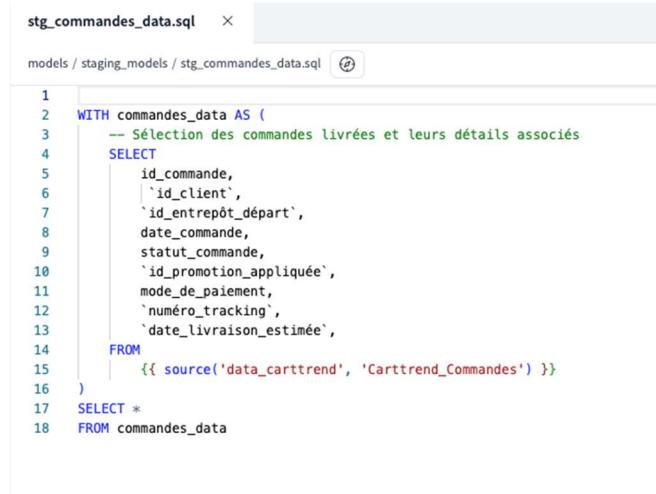
Capture d'écran du fichier schéma :



```
version: 2
models:
- name: stg_campaigns_data
  description: "Modèle transformant les données des campagnes marketing en filtrant les lignes sans canal et nettoyant les données."
  columns:
    - name: id_campagne
      description: "Identifiant unique de la campagne marketing."
      tests:
        - not_null
        - unique
    - name: date
      description: "Date associée à la campagne."
      tests:
        - not_null
    - name: événement_oui_non
      description: "Indicateur pour les événements Oui/Non (statique pour cet exemple)."
    - name: événement_type
      description: "Type d'événement associé (statique pour cet exemple)."
    - name: canal
      description: "Canal marketing utilisé pour la campagne (e.g., Email, Social Media)."
      tests:
        - not_null
    - name: budget
      description: "Budget alloué à la campagne marketing."
      tests:
        - not_null
        - accepted_values:
          values: [">=0"] # Valider que le budget est supérieur ou égal à 0
    - name: clics
      description: "Nombre de clics enregistrés pour la campagne."
      tests:
        - not_null
        - accepted_values:
          values: [">=0"] # Valider que les clics ne sont pas négatifs
    - name: CTR
      description: "Taux de clics (Click-Through Rate) calculé pour la campagne."
      tests:
        - not_null
        - accepted_values:
          values: [">=0"] # Valider que le CTR n'est pas négatif
```

Enfin, voici un exemple de script SQL pour un modèle de staging – préparant et nettoyant les données- et un autre pour un modèle analytique -des modèles d'analyses et de transformations plus avancés que les simples modèles de staging- :

Exemple d'un modèle de staging :



```
stg_commandes_data.sql
```

```
WITH commandes_data AS (
  -- Sélection des commandes livrées et leurs détails associés
  SELECT
    id_commande,
    `id_client`,
    `id_entrepôt_départ`,
    date_commande,
    statut_commande,
    `id_promotion_appliquée`,
    mode_de_paiement,
    `numéro_tracking`,
    `date_livraison_estimée`,
  FROM
    {{ source('data_carttrend', 'Carttrend_Commandes') }}
)
SELECT *
FROM commandes_data
```

Exemple d'un modèle analytique marts, avec les jointures et les fonctions d'agrégations SQL nécessaires pour les transformations :

```

analyse_des_ventes_pro_f... ×
models / marts / analyse_vente / analyse_des_ventes_pro_fr.sql

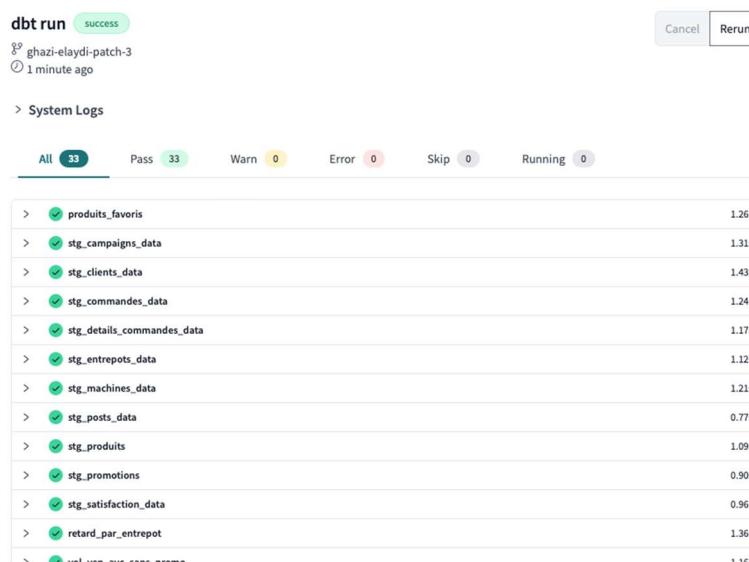
1 WITH Paires_Produits AS (
2     SELECT
3         dc1.id_produit AS produit_1,
4         dc2.id_produit AS produit_2,
5         COUNT(*) AS nombre_occurrences
6     FROM {{ref('stg_commandes_data')}} c1
7     JOIN {{ref('stg_commandes_data')}} c2
8         ON c1.id_client = c2.id_client
9         AND c1.date_commande = c2.date_commande
10    JOIN {{ref('stg_details_commandes_data')}} dc1
11        ON dc1.id_commande = c1.id_commande
12    JOIN {{ref('stg_details_commandes_data')}} dc2
13        ON dc1.id_produit < dc2.id_produit -- Évite les doublons symétriques
14        AND dc2.id_commande = c2.id_commande
15    GROUP BY dc1.id_produit, dc2.id_produit
16 )
17 SELECT
18     produit_1,
19     produit_2,
20     CONCAT(produit_1, ' - ', produit_2) AS combinaison_produits, -- Colonne avec la combinaison des produits
21     p1.Produit AS produit_1_nom, -- Nom du produit 1
22     p2.Produit AS produit_2_nom, -- Nom du produit 2
23     nombre_occurrences
24 FROM Paires_Produits
25 JOIN {{ref('stg_products')}} p1 ON p1.ID = produit_1 -- Jointure pour le nom du produit 1
26 JOIN {{ref('stg_products')}} p2 ON p2.ID = produit_2 -- Jointure pour le nom du produit 2
27 ORDER BY nombre_occurrences DESC
28

```

Preview Compile Build Format Results Code quality Compiled code Lineage

La dernière capture ci-dessous de DBT est une capture après l'exécution d'un DBT RUN via Airflow.

Cette vue confirme que toutes les transformations DBT (33 modèles) ont été exécutées avec succès dans BigQuery, sans erreur ni avertissement. Chaque modèle a été traité rapidement (moins de 2 secondes chacun). L'interface DBT Cloud fournit ainsi des informations claires pour analyser les performances ou relancer une exécution si nécessaire :



B. Vues créées par DBT dans BigQuery

A la suite de cette exécution du DBT RUN, des vues apparaissent sur BigQuery pour les modèles staging et marts (cf capture d'écran ci-dessous).

Cependant il faut noter qu'une meilleure pratique est de créer des vues pour les modèles staging et des tables pour les modèles analytiques.

Vue du dataset de notre projet dans BigQuery avec les tables (logo sombre) et les vues (logo pointillé) après l'exécution d'un DBT RUN :

The screenshot shows the Google Cloud BigQuery interface. On the left, there's a sidebar with various services: BigQuery Studio, Transferts de données, Requêtes programmées, Analytics Hub, Dataform, Centre des partenaires, Orchestrator (BETA), Migration, Évaluation, Traduction SQL, Administration, Surveillance, Explorateur de jobs, Gestion de la capacité, BI Engine, Paramètres (BETA), and Notes de version. The main area shows the 'data_carttrend' dataset details. It includes sections for 'Informations sur l'ensemble de données' (with fields like ID de l'ensemble de données: projet-carttrend.data_carttrend, Crédit: 9 déc. 2024, 16:42:30 UTC+1, etc.) and 'Informations sur les instances répliquées de l'ensemble de données' (with Emplacement principal: EU). Below these are sections for 'Historique du job' and 'RÉSUMÉ'.

C. Collaboration sur DBT avec GitHub

En termes de bonnes pratiques et d'axes d'amélioration dans notre usage de DBT Cloud, et encore plus dans le cadre d'un projet collaboratif, il est fortement recommandé d'héberger ses projets DBT dans un dépôt Git, et GitHub est un choix très courant. DBT Cloud propose une intégration native avec cette plateforme : nous avons d'ailleurs créé un compte GitHub lors du paramétrage initial de DBT.

GitHub est une plateforme de collaboration et de « suivi de versions » de code particulièrement adaptée pour gérer un projet ELT avec 4 étapes parallèles comme celui de Carttrend.

Elle permet d'avoir une branche principale (*master* ou *main*) pour la version stable du code. Elle permet également de créer des branches dédiées (une par étape, par exemple) pour travailler en parallèle. La fusion (*merge*) des modifications de ces branches vers la branche principale peut être exécutée une fois celles-ci testées et validées.

Des *commits* permettent de sauvegarder des modifications. Chaque commit est une “sauvegarde” des modifications, accompagné d'un message décrivant les changements. Ils permettent de suivre l'historique, de comparer les versions et d'effectuer d'éventuels retours en arrière (*rollback*).

En termes de collaboration, les *pull requests* centralisent les revues de code et garantissent la qualité avant la fusion. La *pull request* est donc le moyen de collaborer, de valider la qualité du code et de sécuriser l'intégration des nouveautés.

Ainsi GitHub procure un cadre complet pour organiser, développer et valider un pipeline ELT composé de plusieurs phases qui évoluent en parallèle comme notre projet.

Cependant, notre organisation (nous avons développé tous ensemble chaque étape du projet) ne nous a pas donné l'opportunité de profiter du potentiel complet de cette plateforme.

QUATRIEME PARTIE

Visualisation des données transformées sur
PowerBI – Point sur le Machine Learning

1. Récapitulatif du processus ELT à ce stade avant la visualisation PowerBI

A. Extraction et chargement des données vers BigQuery

Les données brutes provenant de Google Cloud Platform sont extraites puis chargées dans BigQuery, via Airflow.

B. Transformation via DBT Cloud

DBT Cloud effectue les transformations directement dans BigQuery.

Ainsi, des vues optimisées depuis les modèles staging et analytiques -marts- apparaissent dans BigQuery.

2. PowerBI et visualisation des données

A. Choix de PowerBI par rapport à Tableau

PowerBI et Tableau sont deux outils de visualisation de données performants que nous avons pu étudier durant le batch. Notre choix s'est cependant porté sur PowerBI pour différentes raisons :

- Sa popularité :
En raison de son implantation dans l'environnement Microsoft, PowerBI nous a semblé être un outil plus populaire que Tableau.
- Son budget et sa présence sur notre machine virtuelle :
Contrairement à Tableau qui fait appel à un abonnement pour libérer toutes ses fonctionnalités, il nous a semblé que l'usage de PowerBI était moins contraint, plus souple dès lors que celui-ci est déjà présent sur un poste ; PowerBI était en effet déjà installé sur nos machines virtuelles.
- L'opportunité de pouvoir faire quelques transformations :
PowerBI nous semblait plus adapté pour faire quelques transformations au besoin. Soit légères avec les DAX (*Data Analysis Expressions*) pour faire des calculs, de nouvelles

colonnes ou mesures soit lourdes avec l'outil intégré PowerQuery pour faire des fusions ou des nettoyages.

Cependant, notre choix n'a pas été si évident car nous avions trouvé l'interface de Tableau plus ergonomique et plus visuelle.

B. Chargement des données transformées sur PowerBI

Une fois les données transformées par DBT et visualisables sur BigQuery sous forme de vues ou de tables, nous nous sommes connectés à PowerBI.

Lors de la connexion, nous devions choisir la source de données -qui est donc BigQuery ici- et le choix du type de connexion.

Nous avions le choix entre le mode Import, plus performant et plus flexible pour la plupart des cas, mais qui nécessite des actualisations régulières et le mode DirectQuery, idéal pour des rapports nécessitant des données en temps réel, mais avec des limitations sur les calculs et transformations.

Notre choix s'est donc fait en tenant compte du fait que nos données pouvaient être mises à jour automatiquement en temps réel.

Nous avons donc choisi l'option DirectQuery.

En revanche lors de notre visualisation, nous avons été confrontés à des limitations provenant de ce choix de source : les fonctionnalités DAX étaient restreintes et PowerQuery était inopérant, nous empêchant d'effectuer des transformations aussi facilement que nous le souhaitions.

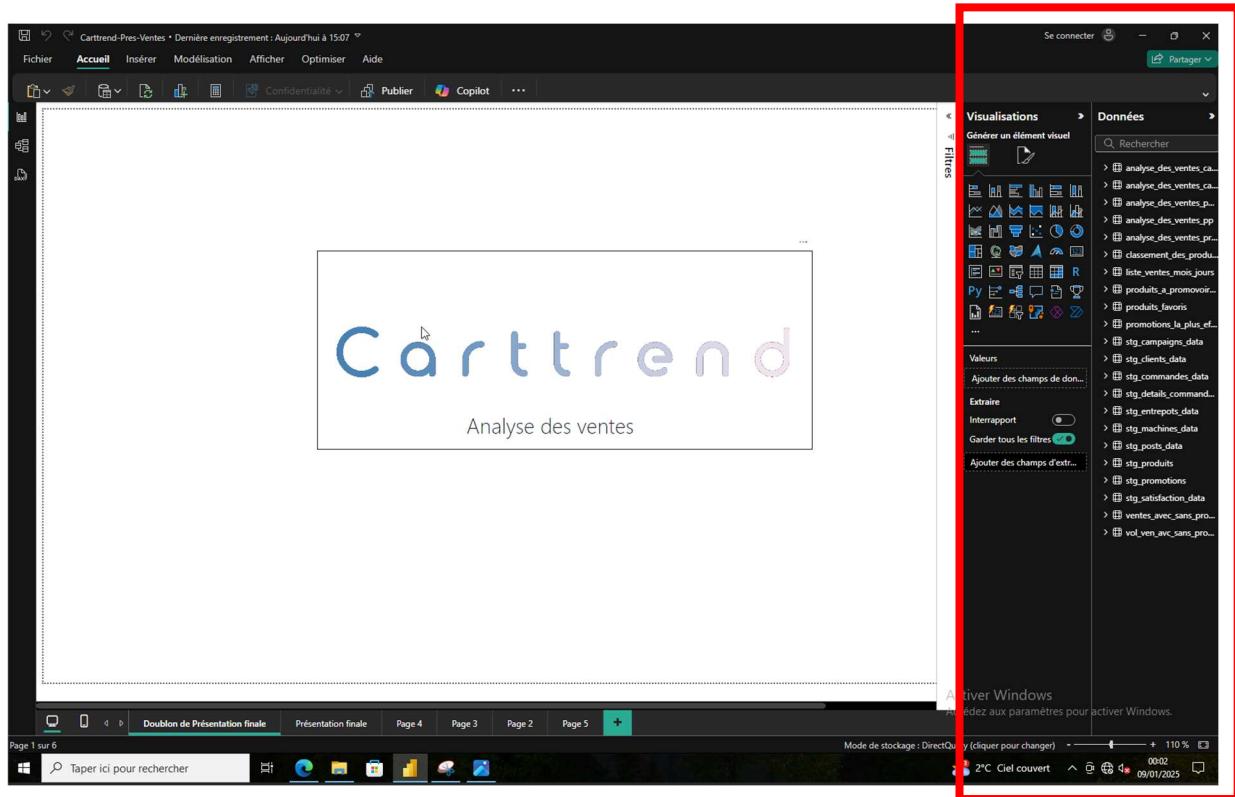
C. Etape de visualisation

Une fois ces étapes passées, nous avons créé des visuels en choisissant les types de visualisation. Parmi ces différents types, nous avons fréquemment choisi des graphiques : à barres, en colonnes, en secteurs, en courbes.

Ou, pour des données détaillées, des tableaux et des matrices.

Enfin, nous avons également personnalisé nos visuels en modifiant les couleurs, polices, axes ou étiquettes.

Capture d'écran de l'interface de PowerBI avec les onglets de visualisations et de données à droite :



Nous avons créé 4 rapports différents, une pour chaque étape, et les avons ensuite exportés en PDF.

Nous nous sommes attachés à respecter une mise en page et une charte graphique commune pour un exposé cohérent des données.

En **Annexe 2**, la présentation finale de Carttrend.

3. Notre utilisation du Machine Learning

Notre premier fichier édité sur Jupyter Notebook en Python n'incluait pas de tentative d'entraînement de modèles prédictifs. Il se limitait en fait à l'exploration des données et aux tests statistiques.

Notre fichier ne précisait pas clairement si l'objectif était de prédire les pannes, d'identifier des relations ou simplement de visualiser les données.

De cette expérience, nous en avons conclu qu'un modèle prédictif était davantage utile à la catégorie « Analyse des ventes ». Nous avons ainsi retenté un projet alternatif de Machine Learning (**Annexe 3**).

Ce projet de Machine Learning lié aux ventes suit ainsi une démarche structurée en plusieurs étapes essentielles.

Tout d'abord, les données brutes ont été consolidées grâce à des jointures entre différentes tables (commandes, détails commandes, clients, promotions), enrichissant ainsi le dataset avec des informations clés comme les dates ou les promotions.

Des variables supplémentaires ont ensuite été créées, telles que le revenu calculé par commande (quantité * prix), des indicateurs temporels (mois, année) et des métriques spécifiques comme l'effet des promotions sur les revenus, renforçant ainsi la pertinence des données pour l'analyse. Une exploration visuelle des données a permis de dégager des tendances importantes, notamment l'évolution des ventes mensuelles et l'impact des promotions, grâce à des graphiques clairs et informatifs.

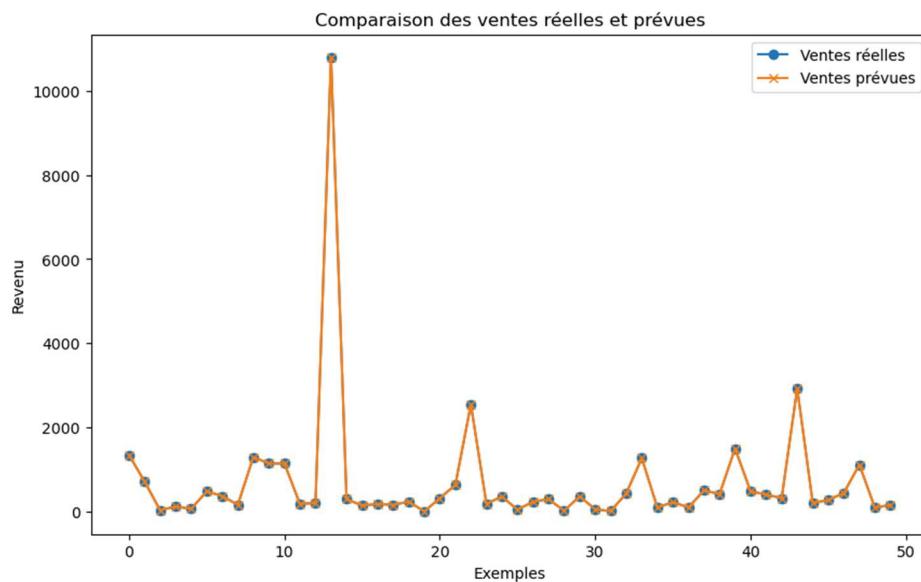
La modélisation a été effectuée à l'aide d'un Random Forest Regressor, un algorithme adapté aux relations non linéaires des données, avec une séparation des ensembles d'entraînement et de test pour garantir une évaluation robuste. Un Random Forest Regressor repose sur le concept des forêts aléatoires, qui combinent plusieurs arbres de décision pour obtenir des prédictions robustes et précises.

Les performances du modèle ont été évaluées avec des métriques comme le MAE et le RMSE, indiquant un modèle performant.

Enfin, l'analyse de l'importance des caractéristiques a mis en lumière les variables clés comme le prix et la quantité, confirmant leur rôle central dans la prédiction des revenus.

Voici la visualisation de la prédiction des ventes futures grâce à notre modélisation :

```
# Comparaison des valeurs réelles et prévues
plt.figure(figsize=(10, 6))
plt.plot(y_test.values[:50], label='Ventes réelles', marker='o')
plt.plot(y_pred[:50], label='Ventes prévues', marker='x')
plt.title("Comparaison des ventes réelles et prévues")
plt.xlabel("Exemples")
plt.ylabel("Revenu")
plt.legend()
plt.show()
```



CONCLUSION

A la suite de notre mission et du pipeline ELT mis en place, nous sommes donc en mesure de préconiser un certain nombre de recommandations auprès des dirigeants de Carttrend pour guider leurs décisions et mieux cibler leurs investissements :

- **Sur l'axe Ventes (analyse des ventes)**

- Les produits « stars » sont incontestablement les produits « Electronique » et « Mode » (voir graphe *Analyse des Ventes*).
- Les « Jouets » et l'« Alimentation » ne génèrent quasiment pas de Chiffre d'Affaire pour Carttrend.
- Une difficulté est la forte sensibilité des produits les plus demandés à la saisonnalité (probablement en relation avec les campagnes promotionnelles pour l'Electronique et la saison pour la Mode (on constate deux pics de vente en été et en hiver)).
- On remarque que les catégories les plus populaires croissent régulièrement selon l'âge des consommateurs. Ceci est cohérent avec un pouvoir d'achat qui augmente avec l'âge (voir graphe *Catégories les plus populaires par âge*).
- La mise en relation des graphiques « Revenus par mois et par catégories » et « Revenus générés par mois par réponses aux promotions » montrent une plus forte présence des promotions pendant les périodes les plus creuses et des revenus générés par celles-ci, sans doute pour stimuler davantage la consommation.

- **Sur l'axe Relation Client (analyse de la satisfaction Client)**

- La satisfaction client est difficile à déterminer sur la base du type de produit, notamment du fait de la diversité des références. On peut en conclure que la satisfaction dépend fortement de ce que le client attend du produit qu'il a sélectionné et que son choix lui est propre (voir graphe *Analyse de la satisfaction client – Impact du type de catégorie sur les notes*). Cependant on

peut remarquer que la ponctualité ou le retard d'une commande ne semble pas avoir d'impact significatif sur l'appréciation globale des clients.

- Par contre, l'analyse des retours enquêtes montre que la satisfaction client est liée à la notion de service : les qualificatifs qui reviennent le plus souvent sont « *Perfect experience* », « *Fast delivery* » pour les commentaires positifs et « *Customer Service* », « *Terrible experience* » pour les commentaires négatifs (voir figure *Nuages de Mots Récurrents dans les commentaires*).
- L'action doit par conséquent se concentrer sur le Service Client.

- **Sur l'axe Efficacité Marketing (impact des campagnes publicitaires)**

- Les réseaux sociaux restent traditionnels (E-mailing, Google Ads, Facebook et Instagram) et les Clics générés par canal sont en faveur d'une diffusion ciblée (E-mailing). Il faut cependant rester attentif à l'évolution des canaux qui peut varier très vite (une veille technologique et médias semble nécessaire).

- **Sur le Processus Logistique (gestion des entrepôts et entretien des machines)**

- En termes de volume, l'activité des entrepôts est relativement homogène (voir graphe *Charge de travail par entrepôt*), sauf l'entrepôt E009 qui semble sous-exploité.
- Une analyse fine des processus logistique de l'entrepôt E009 est à mener pour le ramener au niveau des autres.
- Les analyses pour déterminer les autres aspects, notamment les pannes machines et la charge de travail entre les entrepôts, doivent être affinées. Notre expérience nous a montré que celles-ci ne semblent pas spécialement pertinentes pour être l'objet d'une analyse prédictive par Machine Learning.

Enfin, un point d'amélioration consisterait à visualiser les prédictions de vente des années à venir par l'utilisation du Machine Learning.

ANNEXES

Annexe 1 : Code Python intégral définissant entre autres l'initialisation du DAG

Premier et deuxième blocs :

```
1 import os
2 import pandas as pd
3 from google.cloud import bigquery
4 from datetime import datetime
5 from airflow import DAG
6 from airflow.operators.python import PythonOperator
7 import requests
8 # from airflow.providers.google.cloud.transfers.gcs_to_bigquery import GCSToBigQueryOperator # Import si nécessaire
9
10 # -----
11 # Variables de configuration
12 # -----
13
14 # Identifiants DBT
15 DBT_CLOUD_API_URL = "https://cloud.getdbt.com/api/v2/accounts/70471823406401/jobs/70471823406468/run/"
16 DBT_CLOUD_ACCOUNT_ID = "70471823406401"
17 DBT_CLOUD_JOB_ID = "70471823406468"
18 DBT_CLOUD_API_TOKEN = "dbtc_gF3PHUnbext-uH4bsDWNTWuyST_d2IzI9TgCMsVzwaEKYQv8"
19
20 # Identifiants & configuration GCP / BigQuery
21 project_id = "projet-carttrend"
22 dataset_id = "data_carttrend"
23 os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "/home/analyst/airflow/airflow-env/bin/cle_jason.json"
24
25 # Liste des colonnes à interpréter comme dates
26 date_columns = [
27     "date",
28     "date_inscription",
29     "date_commande",
30     "date_livraison_estimée",
31     "date_début",
32     "date_fin",
33     "date_post"
34 ]
35
36 # Mapping {nom_table: url Google Sheets}
37 file_urls = {
38     "Carttrend_Campaigns": "https://docs.google.com/spreadsheets/d/1khfSdwGGCnreMgSWF9nfpU-Ye_RnC1XIsSubnj6p9gs/edit",
39     "Carttrend_Clients": "https://docs.google.com/spreadsheets/d/1xPksl1ne020jIhBx8zdQ_bI_wZ9MenryYMyGxRXBic/edit",
40     "Carttrend_Commandes": "https://docs.google.com/spreadsheets/d/10VXnfhP0s20pVbvU00Qc1kgdNeWclockCr1hgAi/edit",
41     "Carttrend_Details_Commandes": "https://docs.google.com/spreadsheets/d/1khaO2D-L1vBLST2s2Reu3JP9NMudwXVt6de1_4hcyw/edit",
42     "Carttrend_Entrepots_Machines": "https://docs.google.com/spreadsheets/d/1sR9RGeJIPccWz_0P2TR4X3VKfnbBAKTxn/edit",
43     "Carttrend_Entrepots": "https://docs.google.com/spreadsheets/d/1N8d1d3Gz9PW3Hlin1n8pp93Euh211zBqio04kh1X0k/edit",
44     "Carttrend_Posts": "https://docs.google.com/spreadsheets/d/1D2hi8rZG2YSH1NBz9EE70Eoc-v1kbVHOp6GrvNhA4/edit",
45     "Carttrend_Produits": "https://docs.google.com/spreadsheets/d/12p0-Zgnhmcf0xmlbkL7rx9qKiwg65RcFgyVwb4cyS4/edit",
46     "Carttrend_Promotions": "https://docs.google.com/spreadsheets/d/1G7rS7T8z_zcewX9CUurIVqT5kFWCU_i6Z39P8edZw/edit",
47     "Carttrend_Satisfaction": "https://docs.google.com/spreadsheets/d/1G7rS7T8z_zcewX9CUurIVqT5kFWCU_i6Z39P8edZw/edit"
48 }
```

Troisième bloc – première partie :

```
50  # -----
51  # Fonctions d'extraction, de pré-transformation et de chargement
52  # -----
53
54  def extract_all_sheets(file_urls):
55      """
56          Récupère les données CSV issues des liens Google Sheets et renvoie
57          la liste de DataFrames correspondants.
58      """
59      all_dataframes = []
60      for key, value in file_urls.items():
61          # Convertir le lien Google Sheets en lien CSV (réglage du paramètre "export?format=csv")
62          csv_url = value.replace("/edit", "/export?format=csv")
63          try:
64              df = pd.read_csv(csv_url)
65              if df is not None:
66                  all_dataframes.append(df)
67          except Exception as e:
68              print(f"Erreur lors du téléchargement de {key}: {e}")
69      return all_dataframes
70
71
72  def pre_transform(dataframes):
73      """
74          Effectue des transformations préliminaires (conversion date,
75          nettoyage de noms de colonnes, suppression de doublons) sur une
76          liste de DataFrames.
77      """
78      dataframes_cleaned = []
79      for df in dataframes:
80          # Conversion en datetime pour toutes les colonnes définies dans date_columns
81          for col in date_columns:
82              if col in df.columns:
83                  try:
84                      df[col] = pd.to_datetime(df[col], errors='coerce')
85                  except Exception as e:
86                      print(f"Erreur lors de la conversion de la colonne {col} en date: {e}")
87
88          # Nettoyage des noms de colonnes :
89          # - on supprime les espaces, on remplace par underscore,
90          # - on supprime les caractères spéciaux
91          df.columns = (
92              df.columns
93              .str.strip()
94              .str.replace(' ', '_')
95              .str.replace(r'^[\w]', '', regex=True)
96          )
97
98          # Suppression des doublons
99          df = df.drop_duplicates()
100
101          dataframes_cleaned.append(df)
102
103      return dataframes_cleaned
104
105
```

Troisième bloc - deuxième partie :

```
106     def load_to_bigquery(dataframes_cleaned, project_id, dataset_id, file_urls):
107         """
108             Charge chaque DataFrame dans BigQuery, dans une table nommée
109             d'après sa clé (dans file_urls).
110             """
111         client = bigquery.Client(project=project_id)
112         table_keys = list(file_urls.keys())
113
114         for i, df in enumerate(dataframes_cleaned):
115             table_name = table_keys[i]
116             table_id = f"{project_id}.{dataset_id}.{table_name}"
117
118             job_config = bigquery.LoadJobConfig(
119                 write_disposition="WRITE_TRUNCATE", # "WRITE_APPEND" si vous préférez accumuler
120                 autodetect=True
121             )
122
123             try:
124                 load_job = client.load_table_from_dataframe(
125                     df,
126                     table_id,
127                     job_config=job_config
128                 )
129                 # On attend la fin du chargement
130                 load_job.result()
131                 print(f"La table {table_id} a été chargée avec succès.")
132             except Exception as e:
133                 print(f"Erreur lors du chargement de la table {table_id}: {e}")
134                 print("Aperçu des données problématiques :")
135                 print(df.head(10))
136             raise
137
138
```

Quatrième bloc :

```
137
138 # -----
139 # Fonction pour lancer un RUN DBT
140 # -----
141
142
143 def run_dbt_transform():
144     """
145         Lance l'exécution d'un job DBT Cloud (identifiants spécifiés en haut).
146     """
147     headers = {
148         "Authorization": f"Bearer {DBT_CLOUD_API_TOKEN}",
149         "Content-Type": "application/json"
150     }
151
152     # Récupération de la liste de jobs DBT
153     jobs_url = f"https://cloud.getdbt.com/api/v2/accounts/{DBT_CLOUD_ACCOUNT_ID}/jobs/"
154     response = requests.get(jobs_url, headers=headers)
155
156     if response.status_code == 200:
157         jobs = response.json().get("data", [])
158         if jobs:
159             job_id = jobs[0]["id"]
160             print(f"Job trouvé, ID: {job_id}")
161         else:
162             print("Aucun job trouvé.")
163             return None
164     else:
165         print("Impossible de récupérer la liste des jobs. Erreur : ", response.text)
166         return None
167
168     # On déclenche le job via l'URL de RUN (DBT_CLOUD_API_URL)
169     run_url = DBT_CLOUD_API_URL
170     payload = {
171         "cause": "Airflow triggered run"
172     }
173     run_response = requests.post(run_url, headers=headers, json=payload)
174
175     if run_response.status_code == 200:
176         print("DBT run triggered successfully.")
177     else:
178         print("Failed to trigger DBT run. Error:", run_response.text)
179
180
```

Cinquième et dernier bloc :

```

181 # -----
182 # Définition du DAG Airflow
183 # -----
184
185 with DAG(
186     "carttrend_elt_final",
187     description="ETL pour charger les données de Google Sheets vers BigQuery",
188     schedule_interval="@daily",      # s'exécute 1 fois par jour
189     start_date=datetime(2024, 12, 10),
190     catchup=False,
191     tags=["carttrend", "ETL", "bigquery"]
192 ) as dag:
193
194     # 1) Extraire les données des fichiers Google Sheets
195     extract_task = PythonOperator(
196         task_id="extract_data",
197         python_callable=extract_all_sheets,
198         op_args=[file_urls]
199     )
200
201     # 2) Pré-transformer ces données
202     pre_transform_task = PythonOperator(
203         task_id="pre_transform_data",
204         python_callable=lambda ti: pre_transform(ti.xcom_pull(task_ids="extract_data")),
205         provide_context=True
206     )
207
208     # 3) Charger les données dans BigQuery
209     load_task = PythonOperator(
210         task_id="load_data_to_bigquery",
211         python_callable=lambda ti: load_to_bigquery(
212             ti.xcom_pull(task_ids="pre_transform_data"),
213             project_id,
214             dataset_id,
215             file_urls
216         ),
217         provide_context=True
218     )
219
220     # 4) Lancer un job DBT à la fin
221     dbt_transform_task = PythonOperator(
222         task_id="run_dbt_transformations",
223         python_callable=run_dbt_transform
224     )
225
226     # Ordonnancement
227     extract_task >> pre_transform_task >> load_task >> dbt_transform_task
228

```

Annexe 2 : Présentation finale de Carttrend. Exposé des visualisations

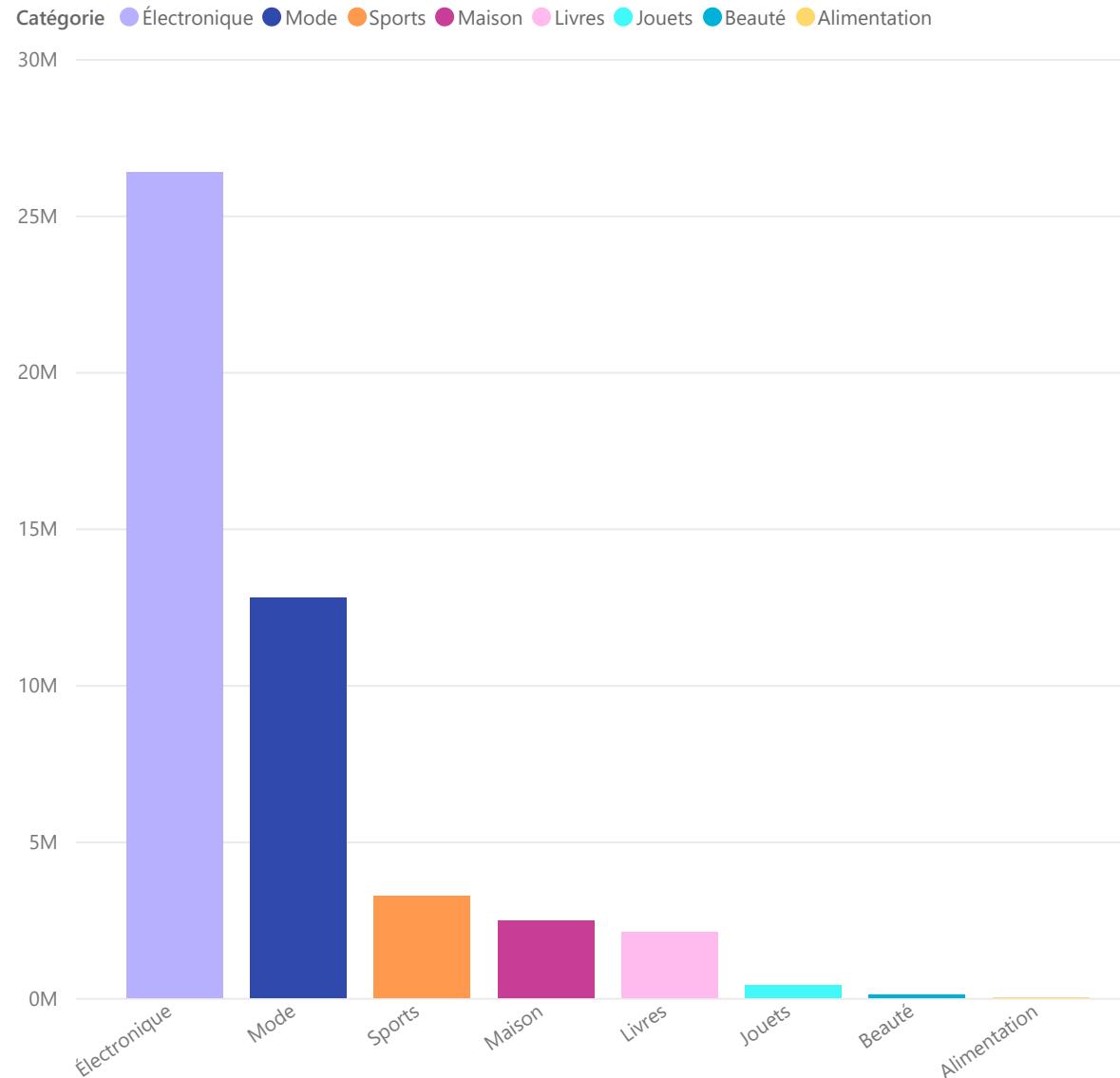
Cartrend

Analyse des ventes

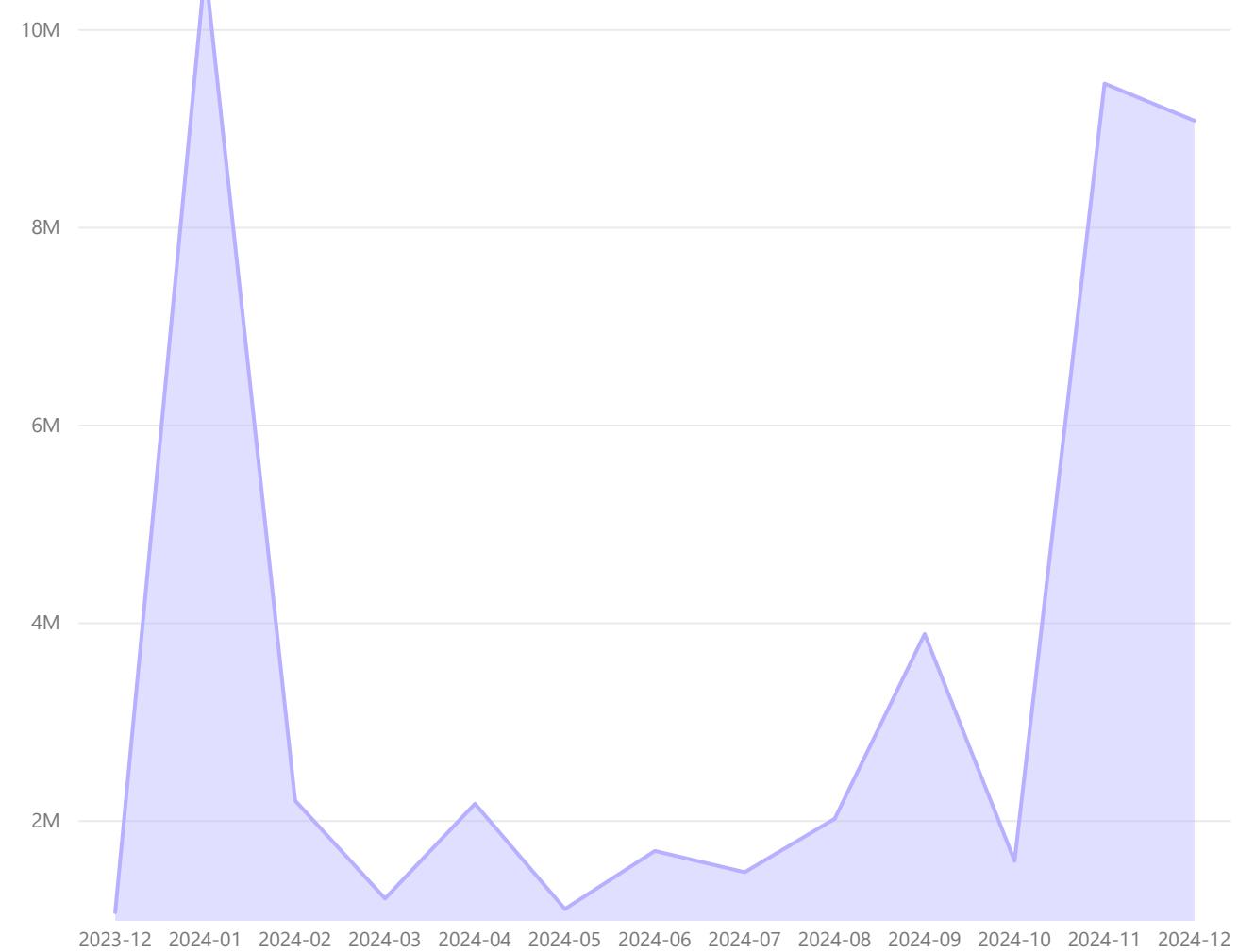
Analyse des ventes - Revenus totaux

Carttrend

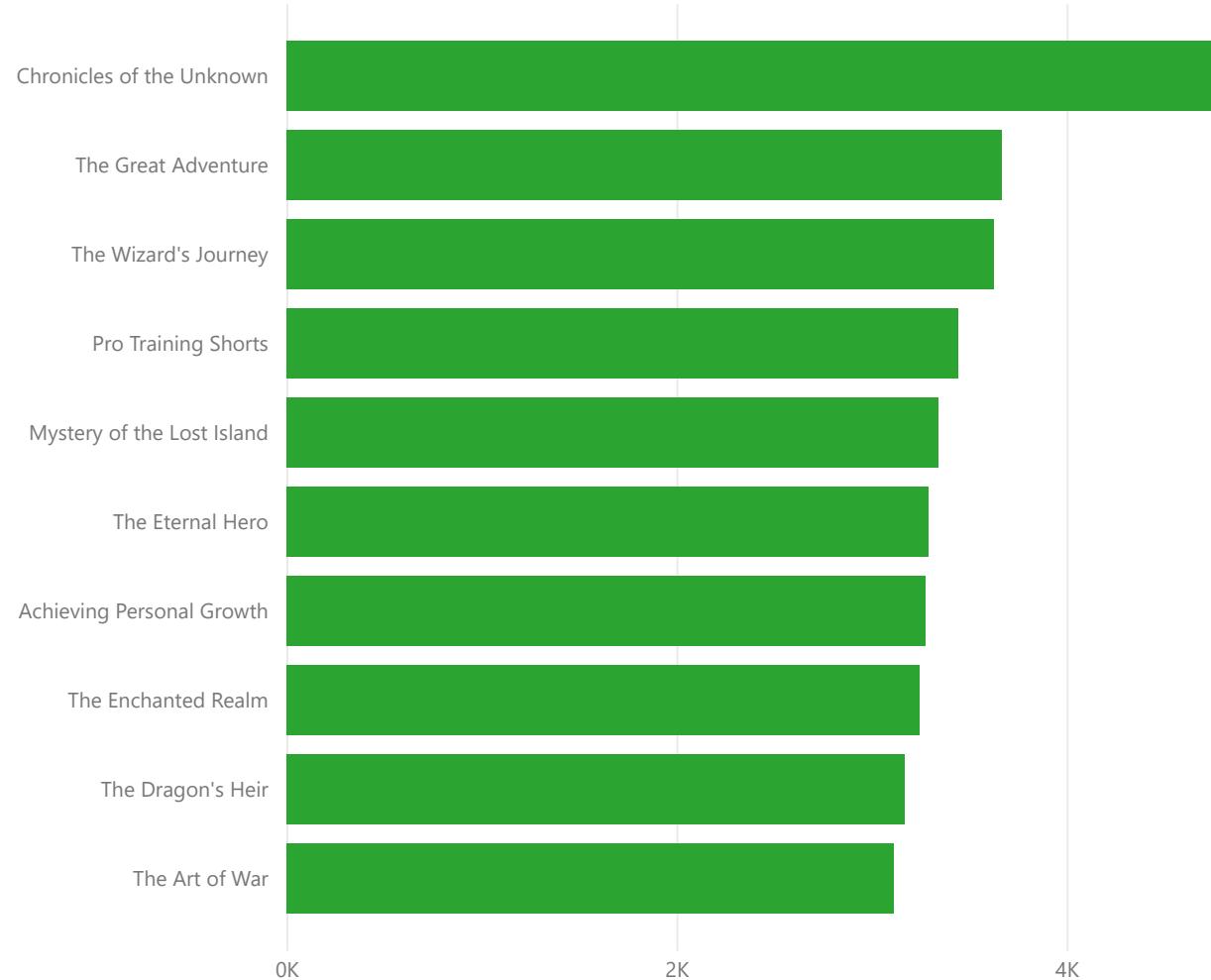
Catégories ayant générées le plus de revenus



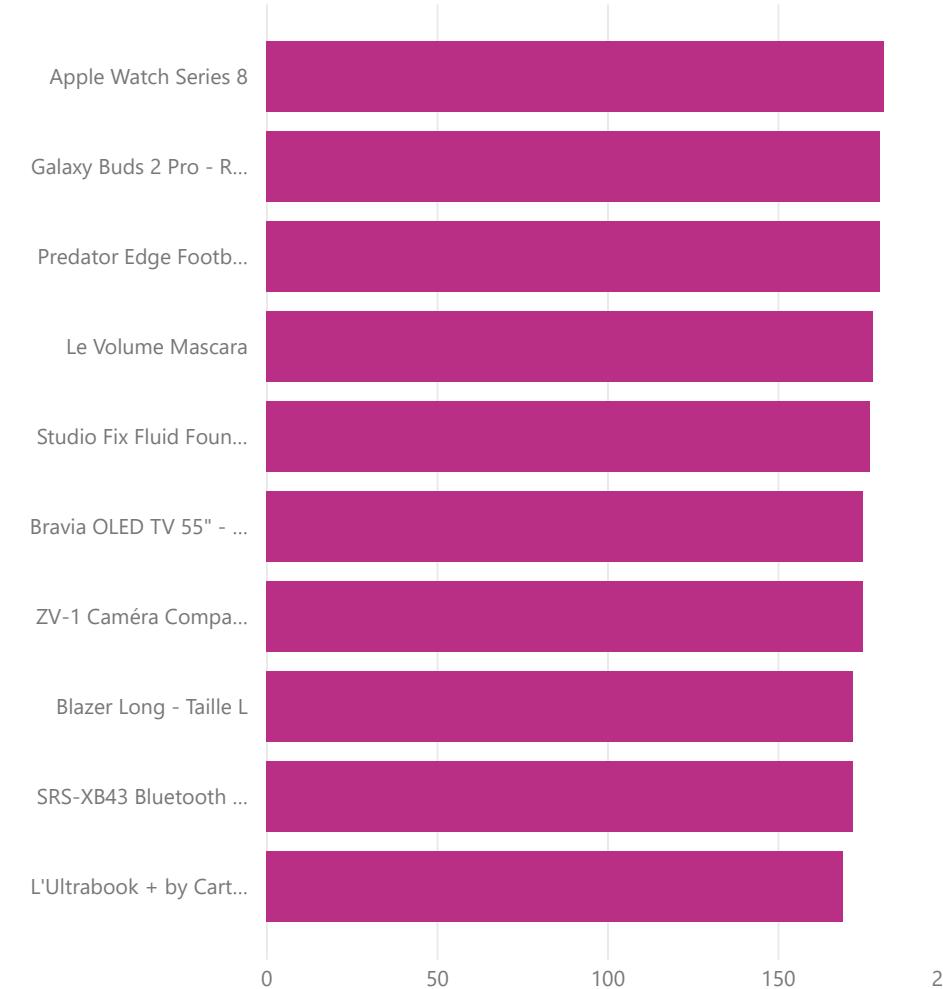
Revenus totaux de Carttrend par mois



Les 10 produits les plus populaires



Les 10 produits les moins populaires



Analyse des ventes par groupes d'âge

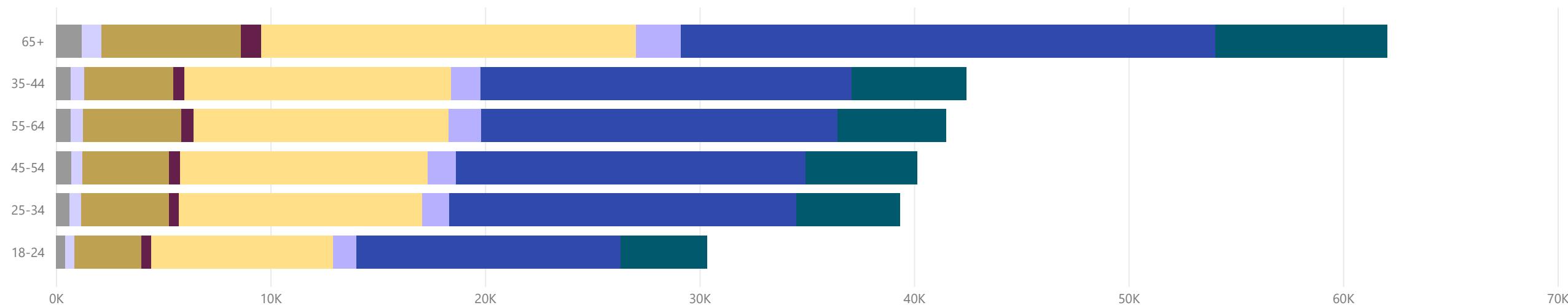
Carttrend

Produits les plus populaires par âge



Catégories les plus populaires par âge

Catégorie ● Alimentation ● Beauté ● Électronique ● Jouets ● Livres ● Maison ● Mode ● Sports

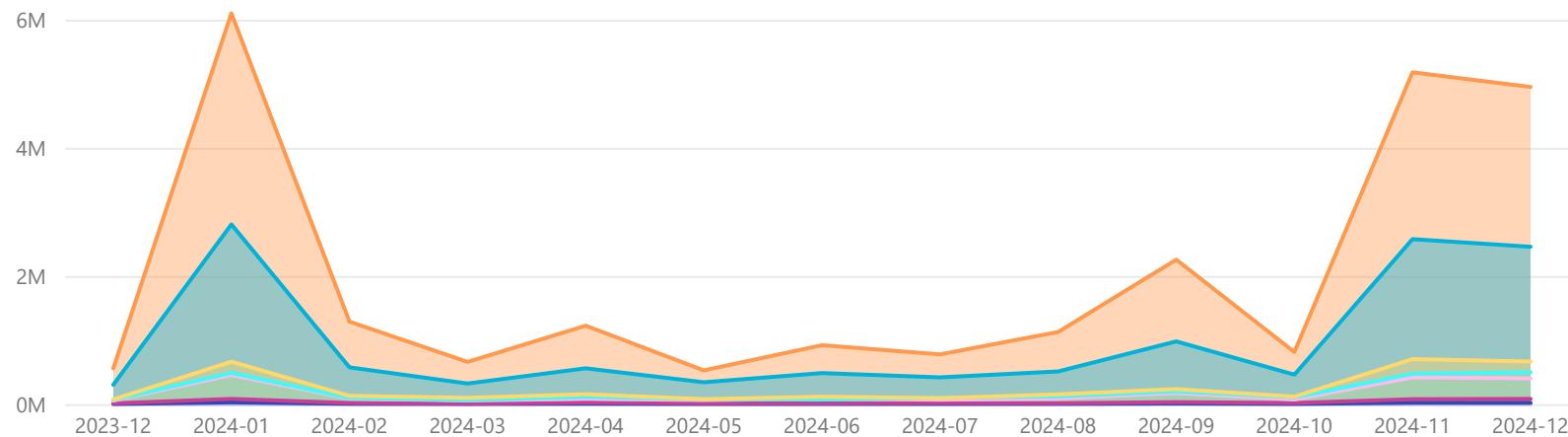


Analyse des ventes - Revenus générés par catégories et impact des promotions

Cartrend

Revenus par mois et par catégories

Catégorie ● Alimentation ● Beauté ● Électronique ● Jouets ● Livres ● Maison ● Mode ● Sports

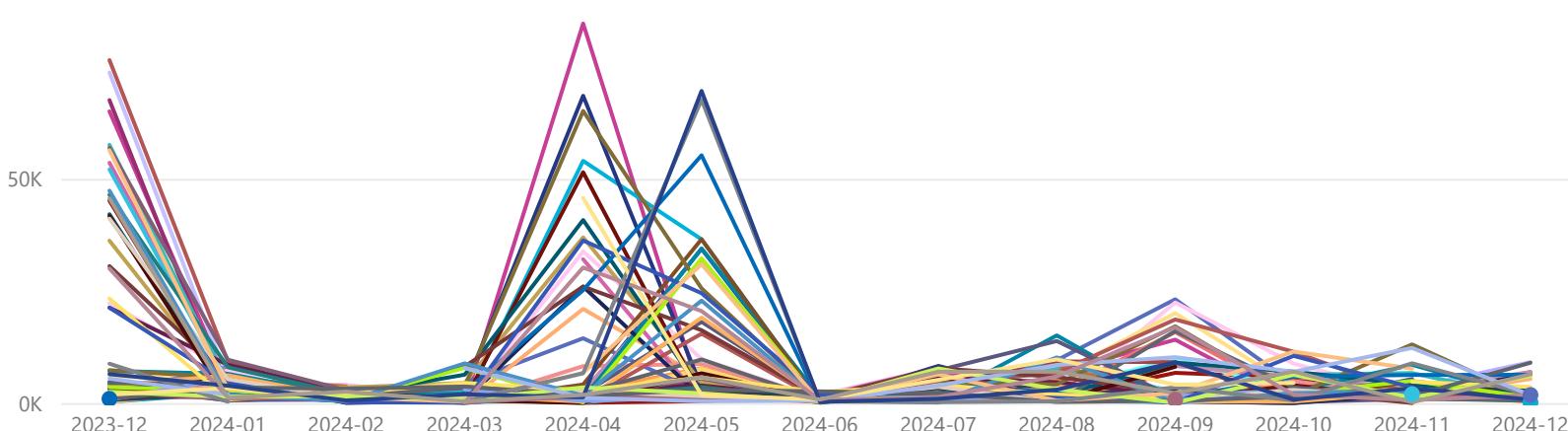


Revenus générés par mois par réponses aux promotions

Promotion appliquée ● P001 ● P002 ● P003 ● P004 ● P005 ● P006 ● P007 ● P008 ● P009 ● P010 ● P011 ● P012 ● P013 ● P014 ● P015 ● P016



100K

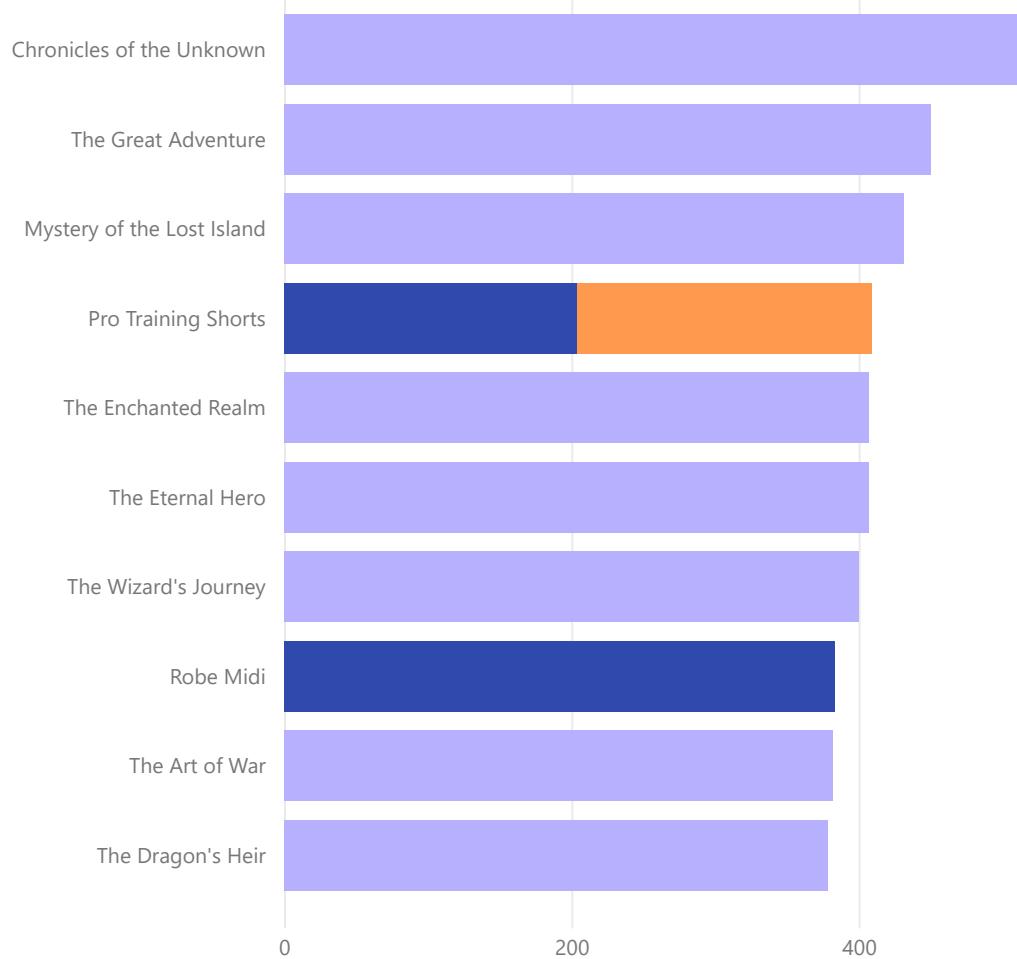


Analyse des ventes - Produits plébiscités dans les favoris - Produits achetés ensemble

Carttrend

Produits les plus ajoutés dans les favoris et leur catégorie

Catégorie ● Livres ● Mode ● Sports



Produits le plus souvent achetés ensemble (combinaisons)

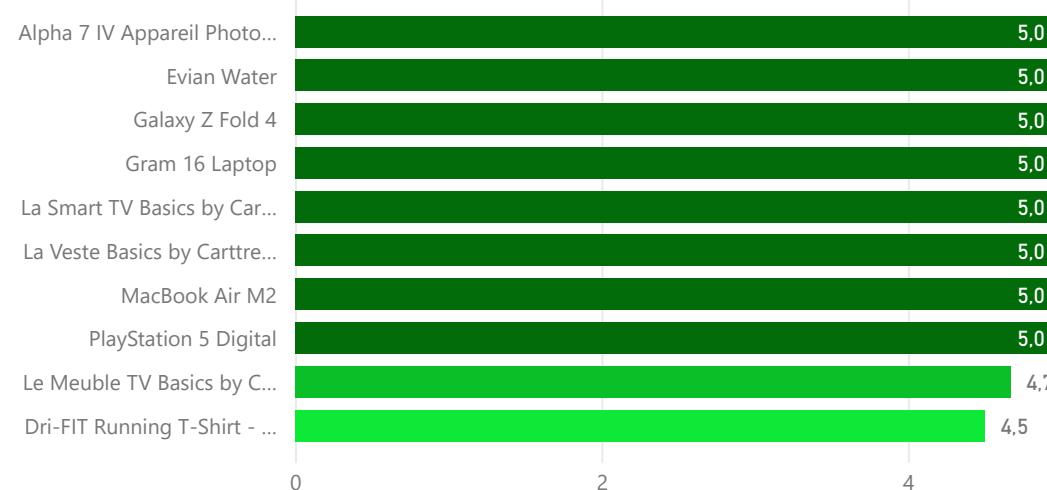
Nombre de combinaisons	Produit 1	Produit 2
4	Achieving Personal Growth	The Enchanted Realm
4	La Chaussure de Sport + by Carttrend	Pro Training Shorts
4	La Sneaker + by Carttrend	The History of Technology
4	Pro Training Shorts	The Power of Habits
4	The Eternal Hero	Tiro 21 Training Pants
3	Adicolor Classics Tee	NMD_R1 Sneakers
3	Chronicles of the Unknown	La Sneaker Basics by Carttrend
3	Chronicles of the Unknown	The Eternal Hero
3	Chronicles of the Unknown	The Story of Civilization
3	Famous Historical Figures	Predator Edge.1
3	La Chaussure de Sport + by Carttrend	La Sandale + by Carttrend
3	La Chaussure de Sport Basics by Carttrend	NMD_R1 Primeblue
3	La Sandale Pro by Carttrend	Zoom Pegasus 39
3	Le T-shirt Basics by Carttrend	The Art of War
3	NMD_R1 Primeblue	Pegasus Running Shoes
3	NMD_R1 Sneakers	The Enchanted Realm
3	Pull en Cachemire	The Wizard's Journey
3	Quantum Mechanics Simplified	The History of Technology
3	The Eternal Hero	Training Shorts
3	The Great Adventure	The Power of Habits
3	The Great Adventure	The Wonders of Biology
2	Achieving Personal Growth	Chemisier Satiné
2	Achieving Personal Growth	Chronicles of the Unknown
2	Achieving Personal Growth	La Chaussure de Sport Basics by Carttrend
2	Achieving Personal Growth	La Sneaker + by Carttrend
2	Achieving Personal Growth	La Sneaker Pro by Carttrend

2002

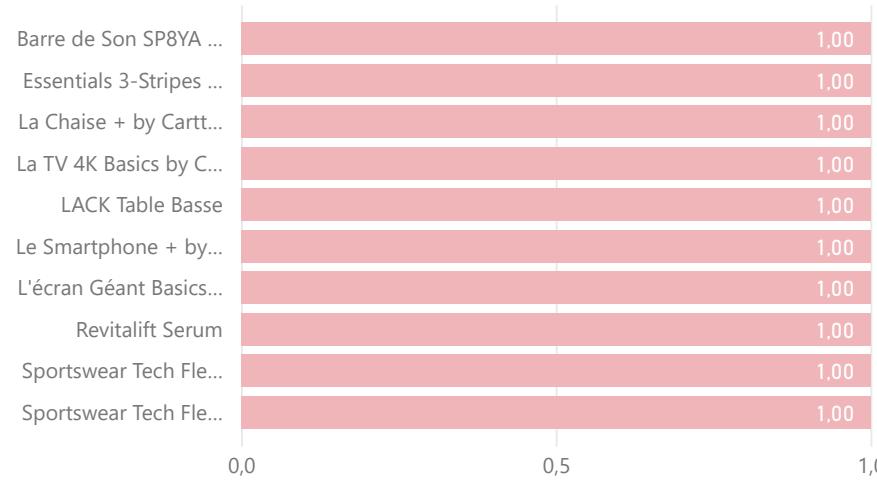
Cartrend

Analyse de la satisfaction client

Les produits les mieux notés

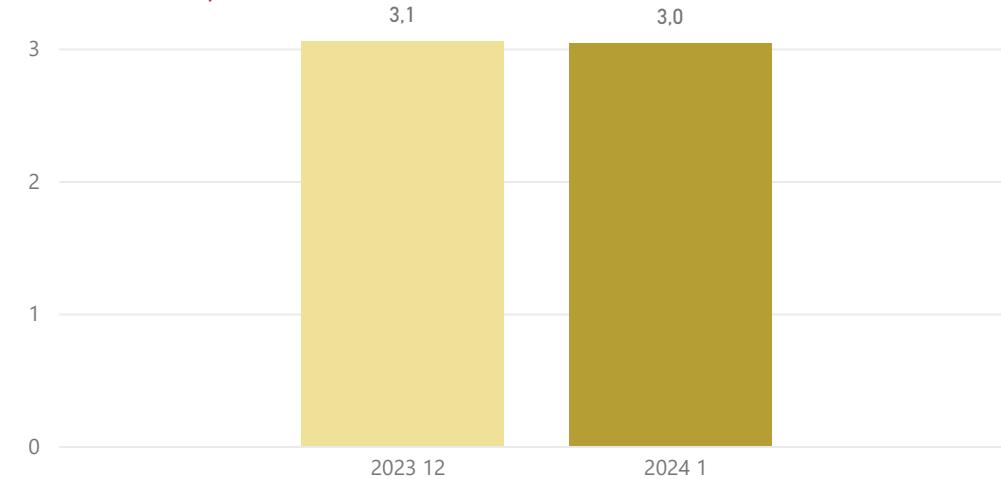


Les produits les moins bien notés

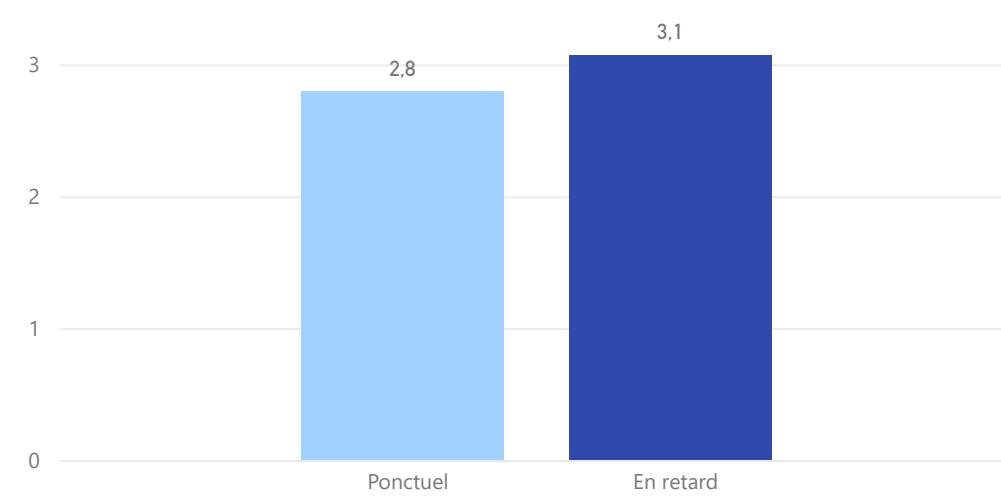


Evolution de la note moyenne

• Pas de notes présentes sur les autres mois

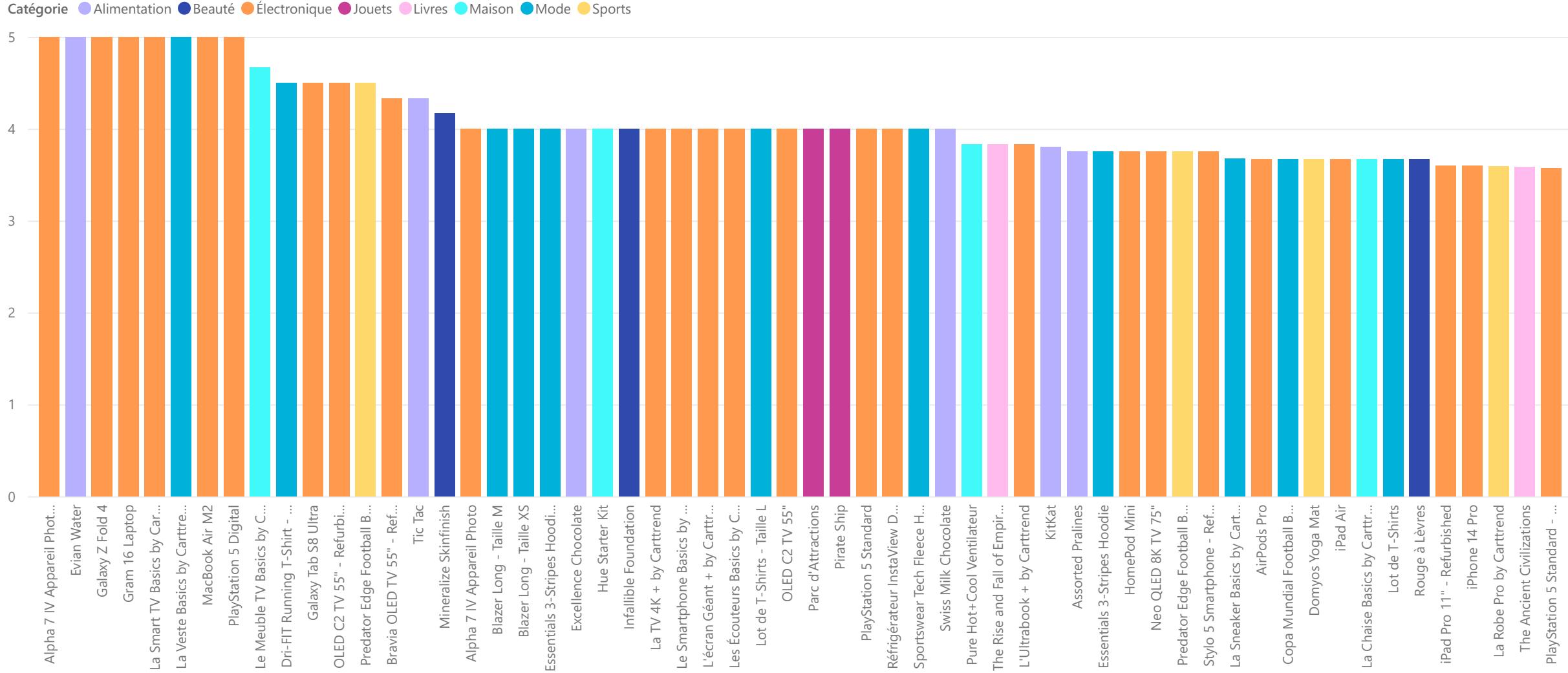


Satisfaction Client selon le Statut des Commandes



Analyse de la satisfaction client - Impact du type de catégorie sur les notes

Cartrend



Nuage de Mots Récurrents dans les Commentaires :

Nuage de mots - Commentaires Positifs:



Nuage de mots - Commentaires Négatifs:



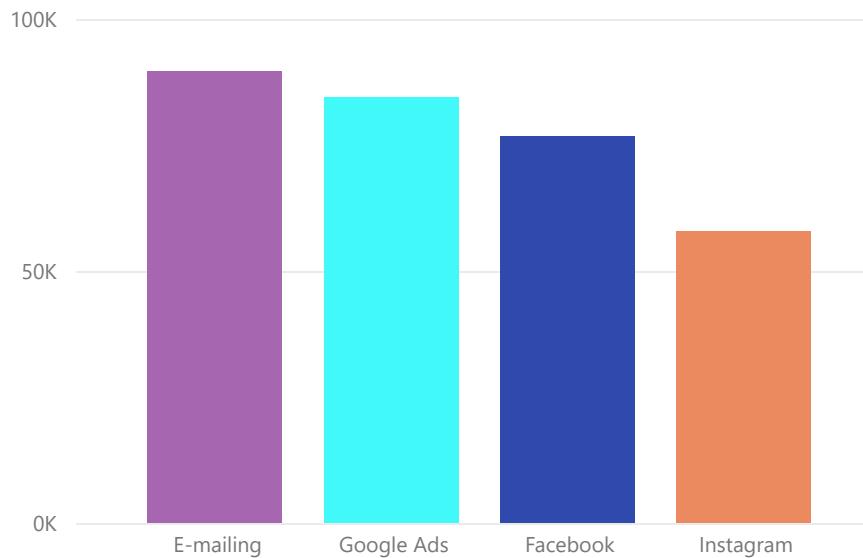
Cartrend

Analyse marketing

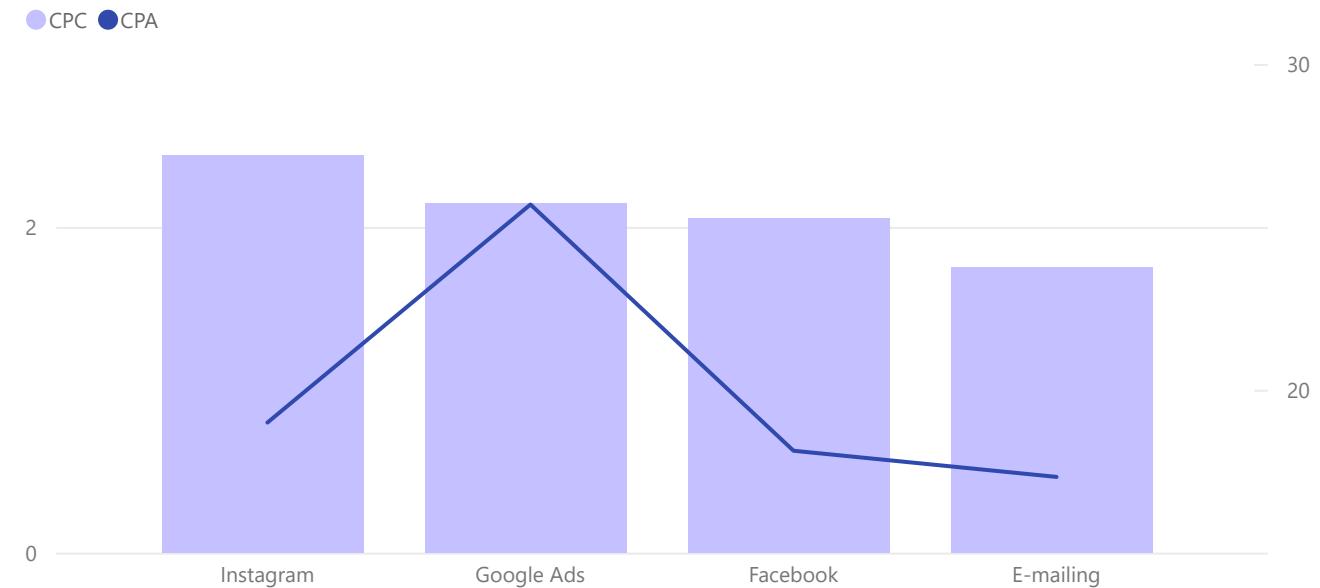
Analyse marketing - Identification des canaux publicitaires les plus efficaces

Cartrend

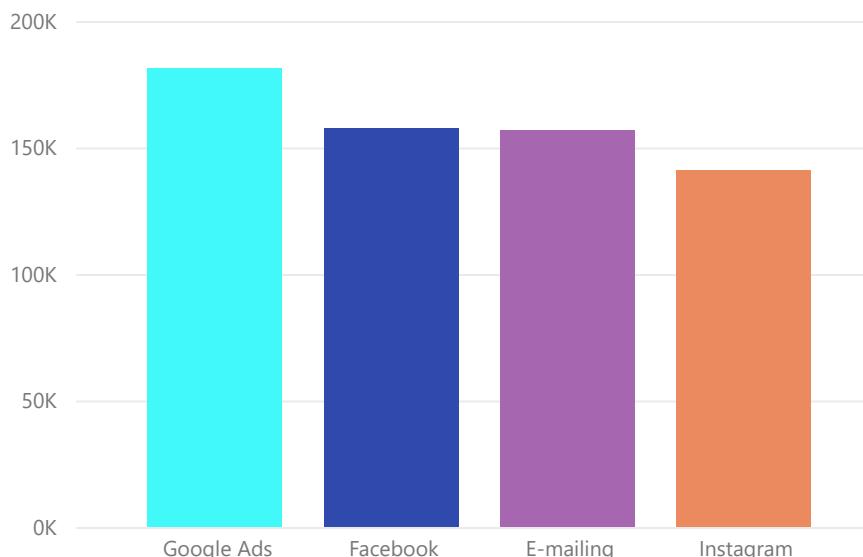
Clics générés par type de canal



Coût en euros par clic CPC et par conversion CPA pour chaque canal



Budget par type de canal



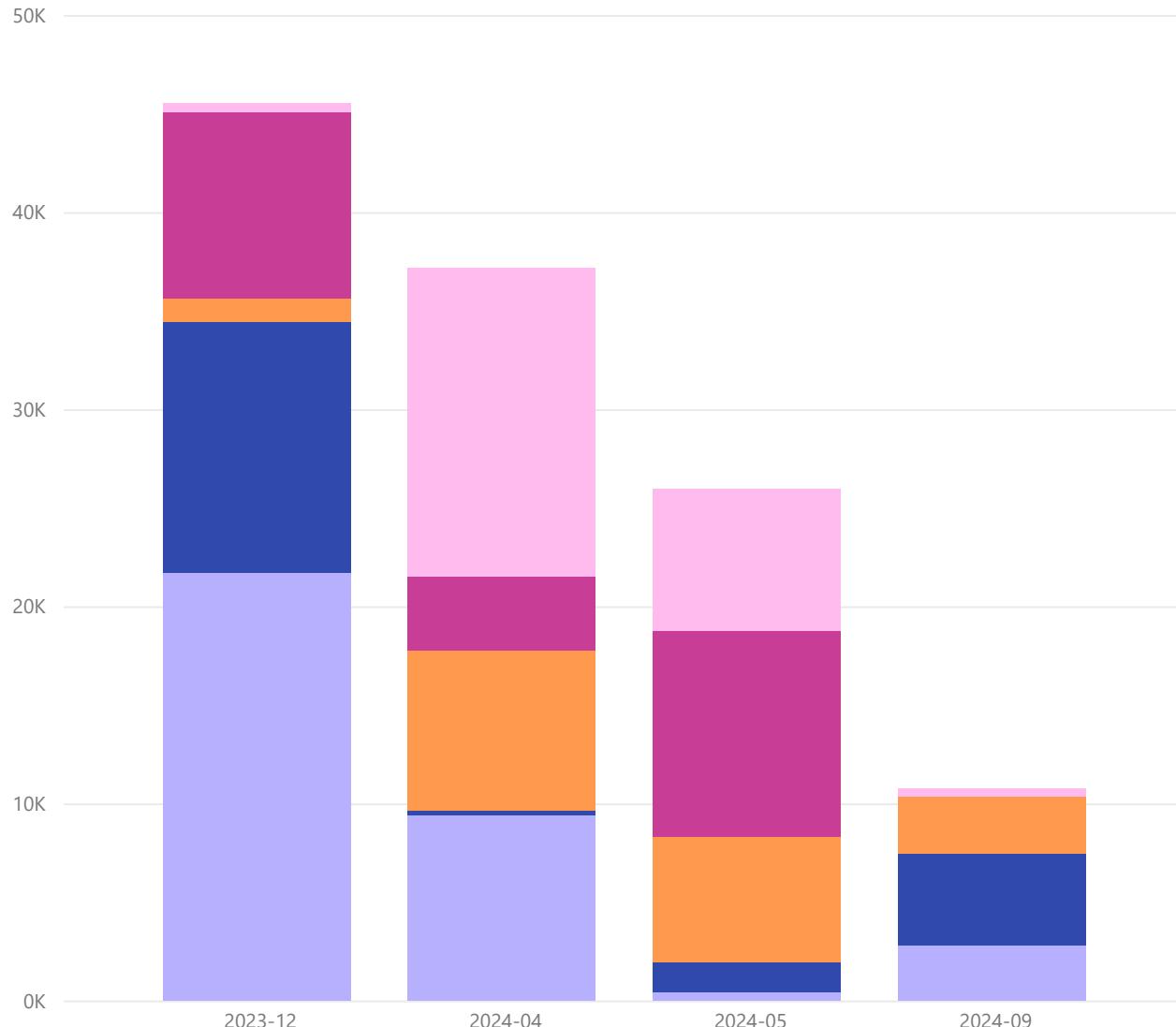
Canal	CPA (Coût par conversion)	CPC (Coût par clic)	Budget cumulé	Total de clics	Total de conversions
E-mailing	17,33	1,75	156896	89493	9056
Facebook	18,13	2,06	157538	76625	8690
Google Ads	25,69	2,15	181197	84406	7054
Instagram	18,99	2,44	140738	57741	7410
Total	80,13	8,39	636369	308265	32210

Analyse marketing - Impact des promotions sur les ventes et les marges

Cartrend

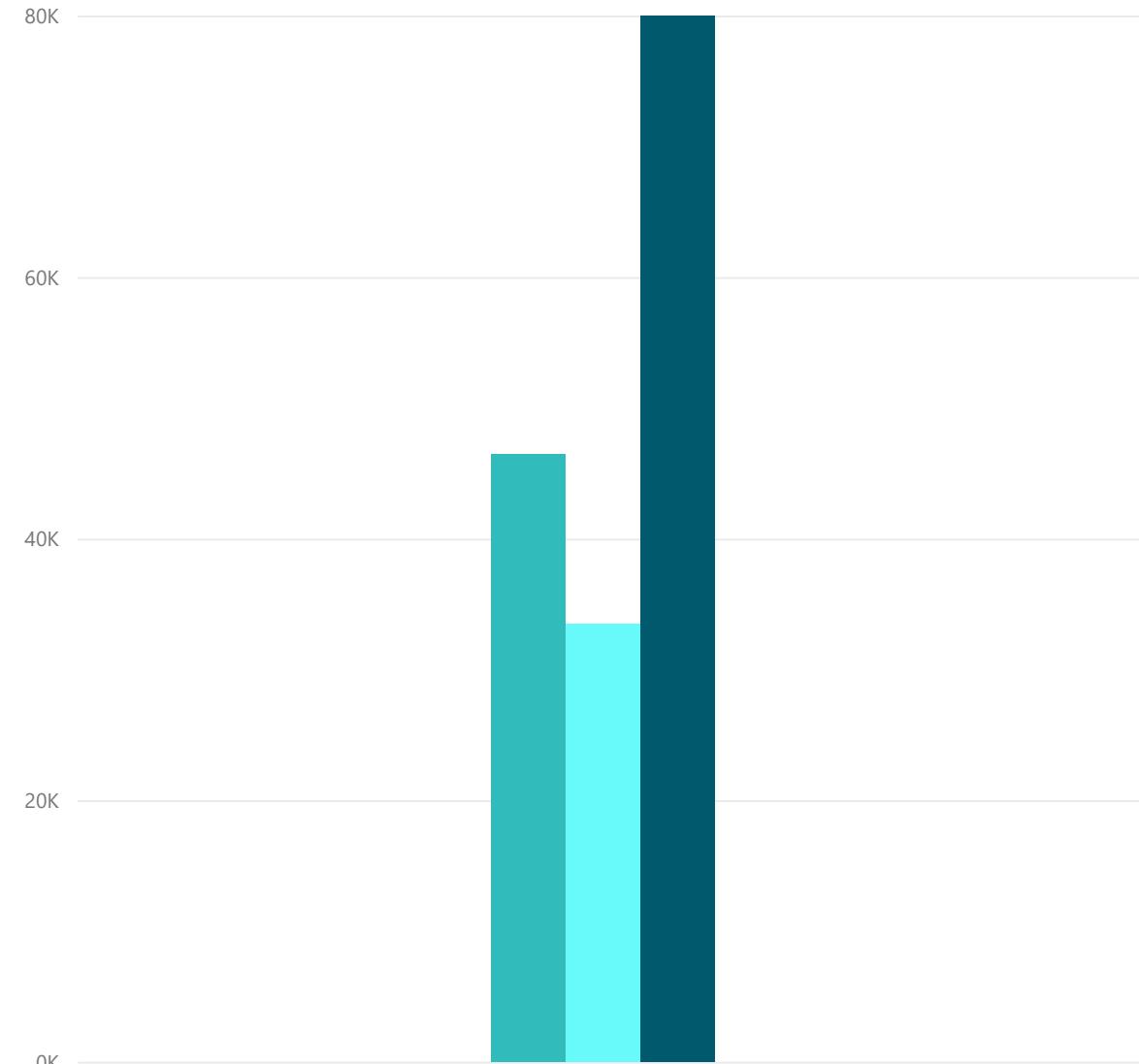
Promotions les plus efficaces par mois les plus performants

Type de promotion P004 P029 P038 P092 P096

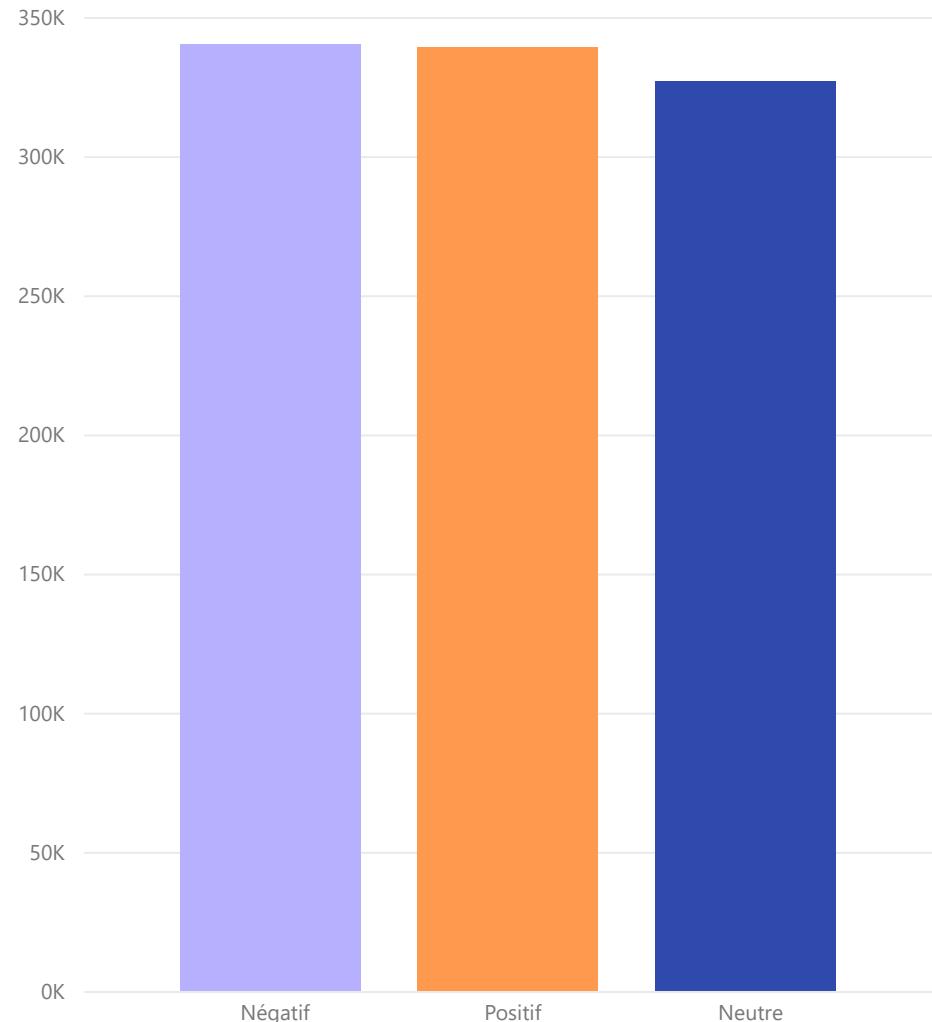


Ventes avec ou sans promotions par rapport au volume total

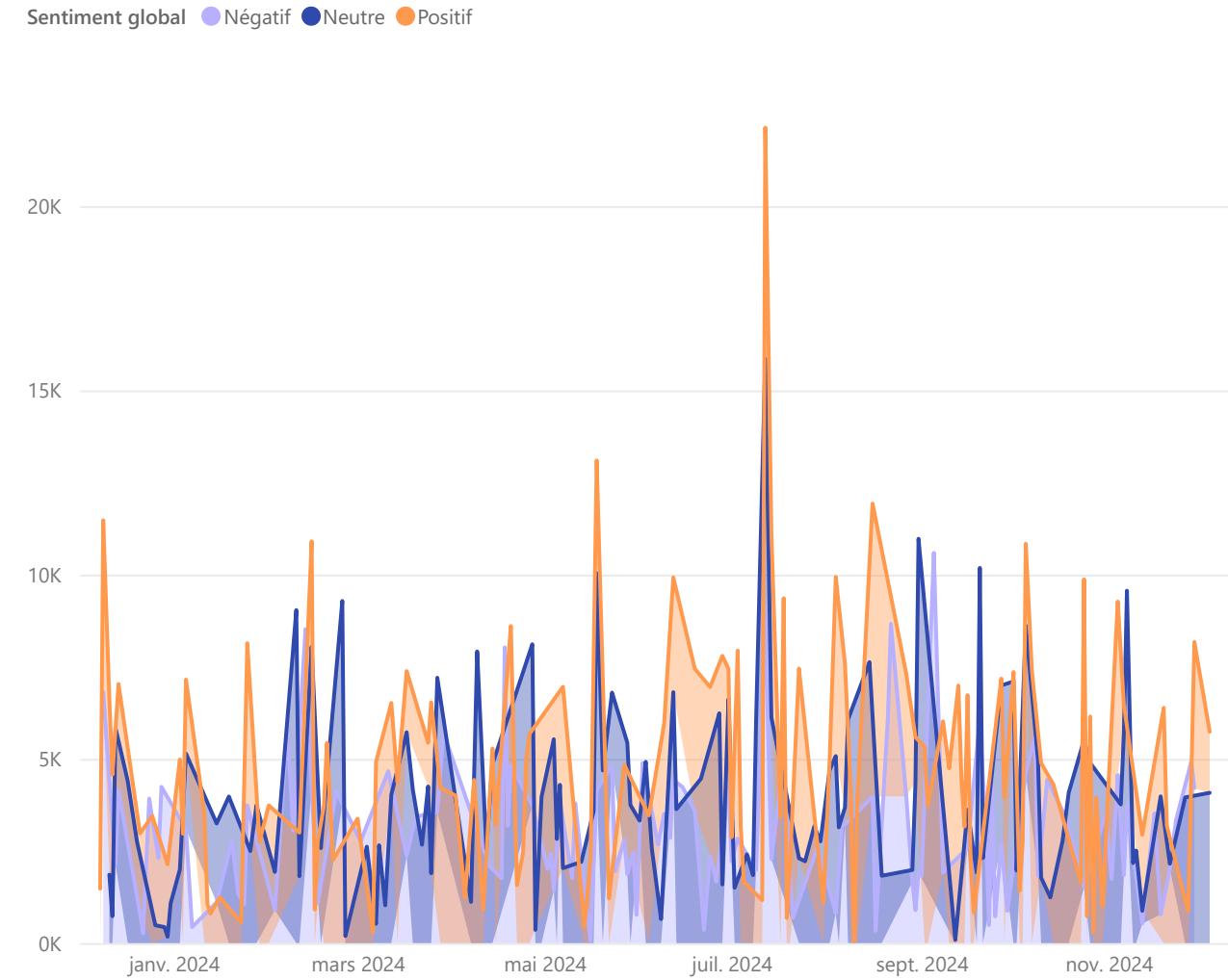
Ventes avec promotions Ventes sans promotion Volume de ventes



Analyse des volumes de mentions et sentiment global



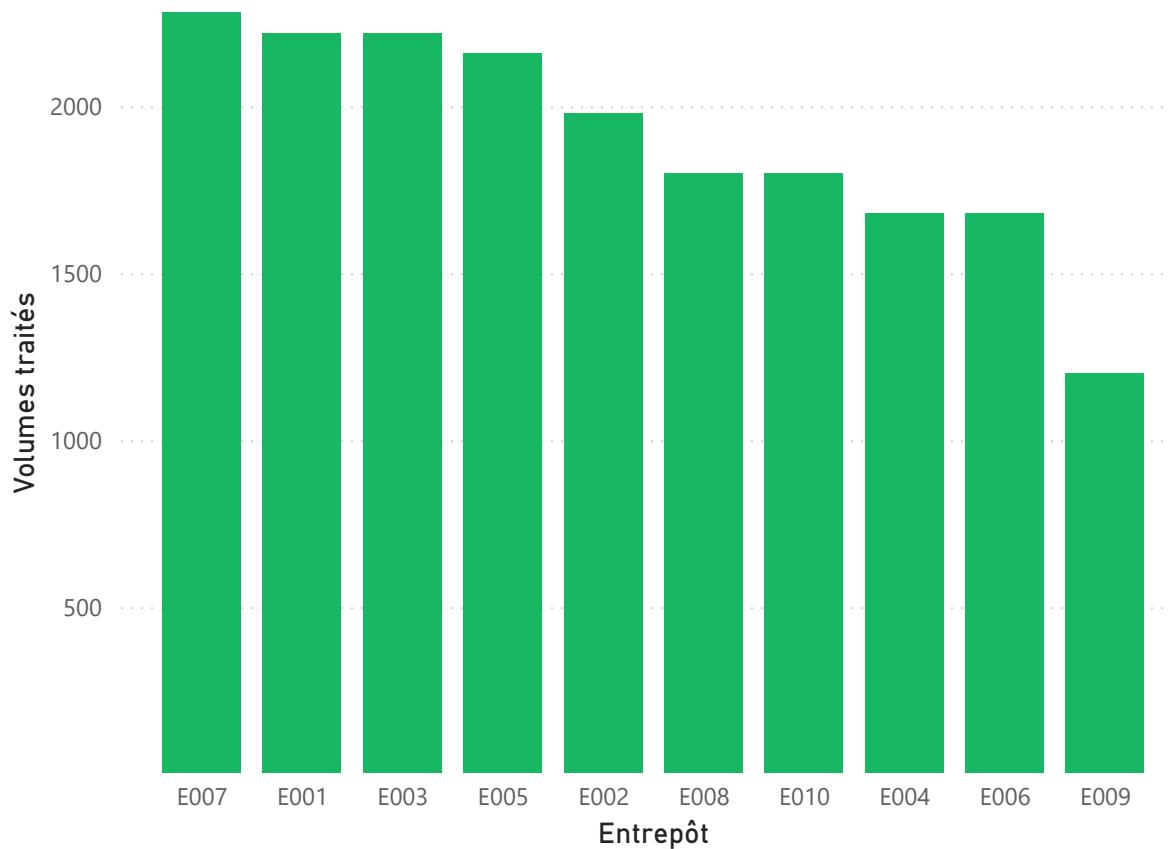
Les pics de mentions de satisfaction selon les ventes



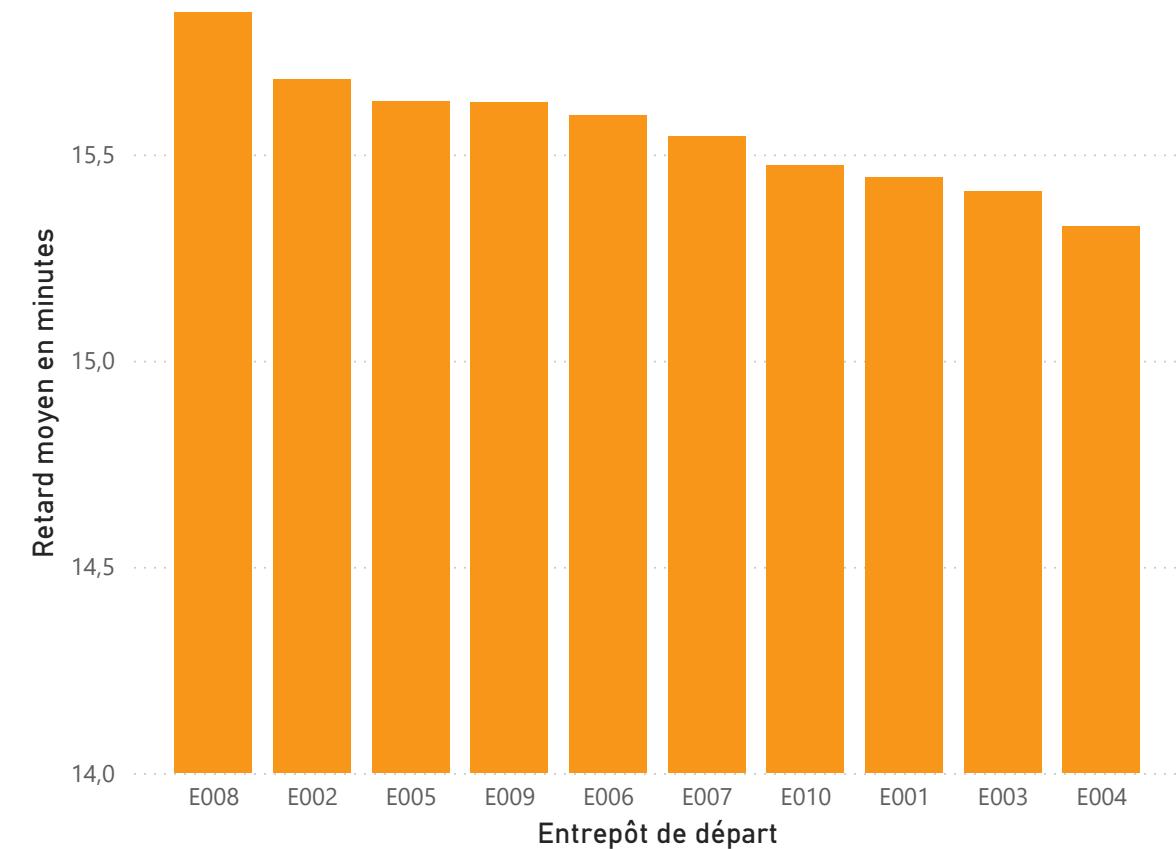
C a r t r e n d

Optimisation logistique

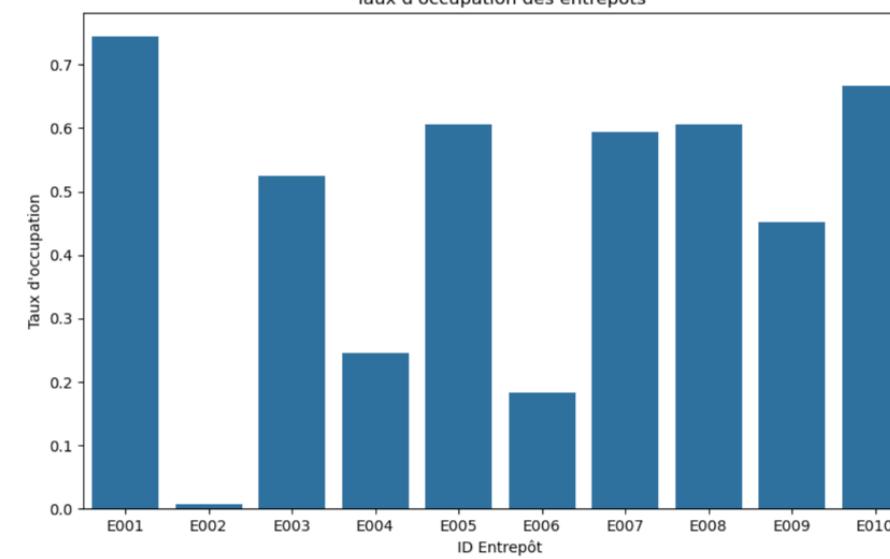
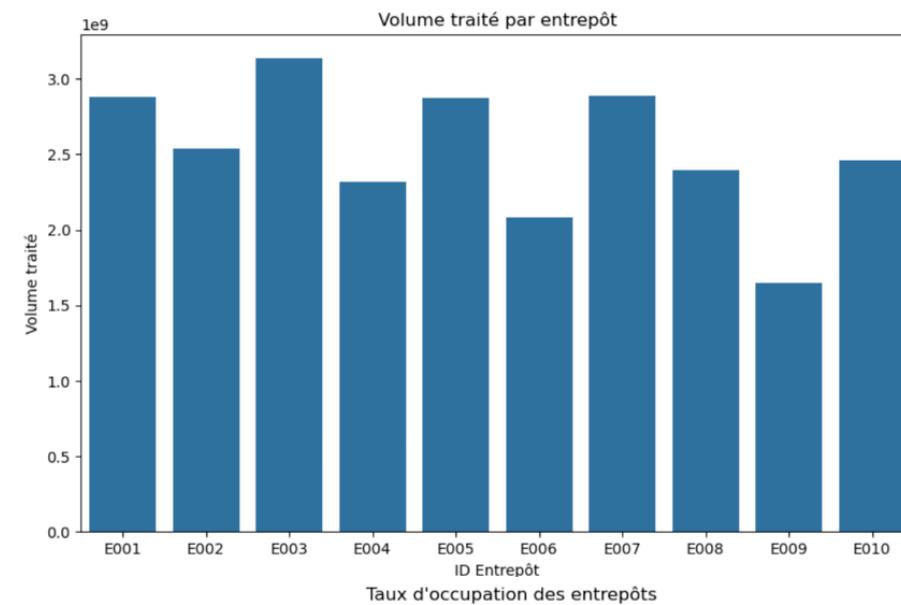
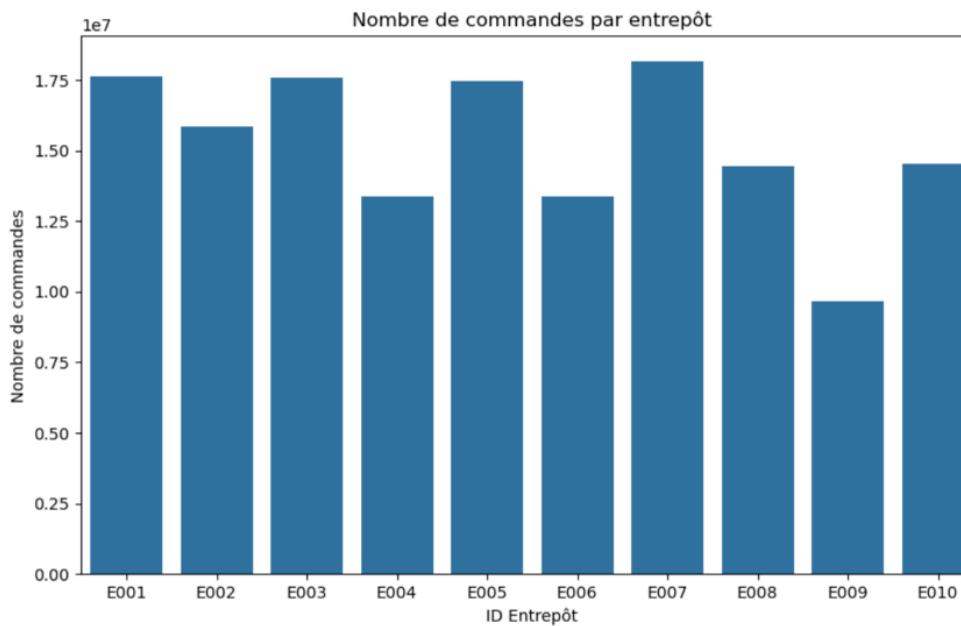
Entrepôts qui gèrent les plus gros volumes



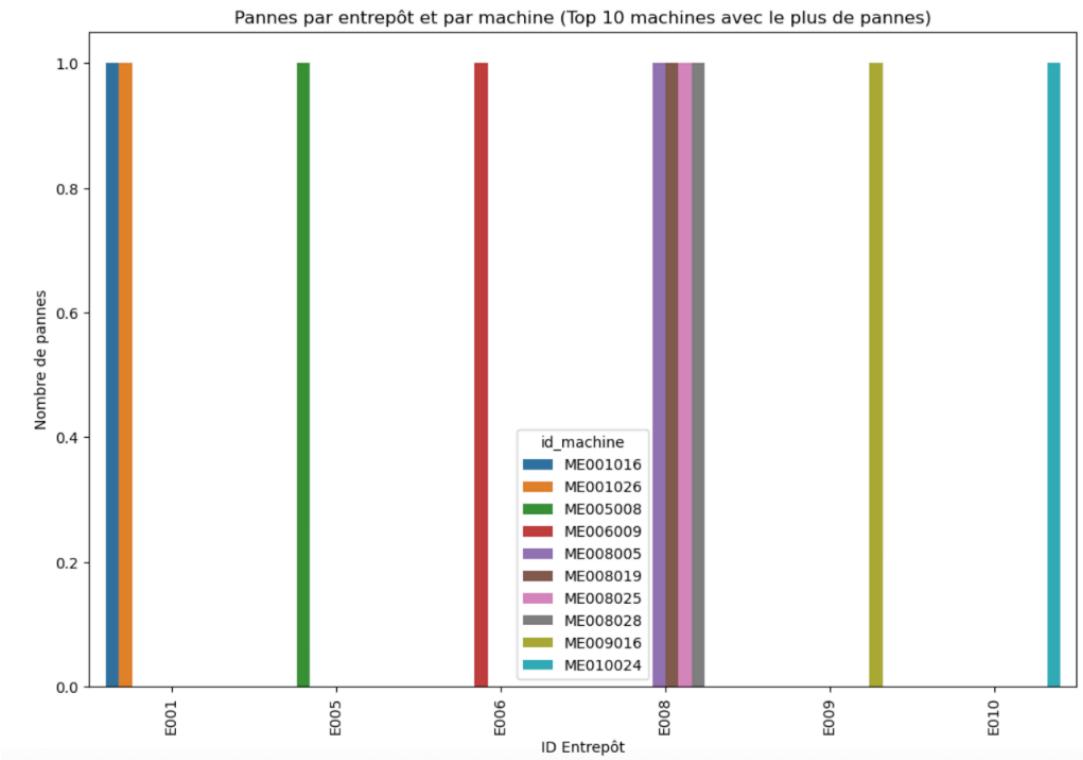
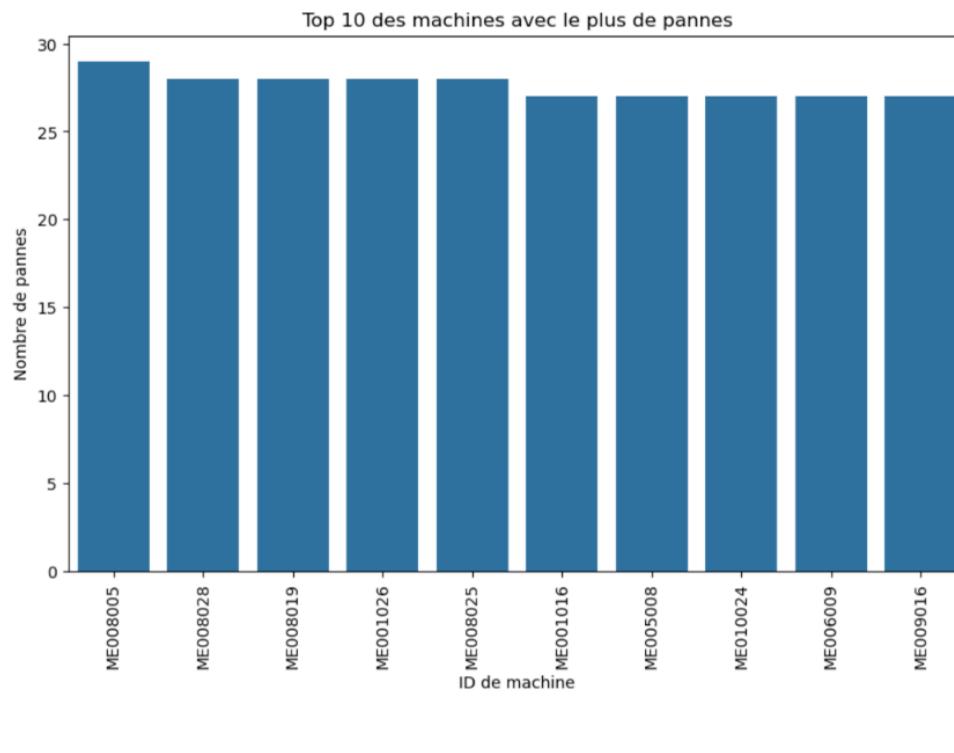
Entrepôts qui causent le plus de retards



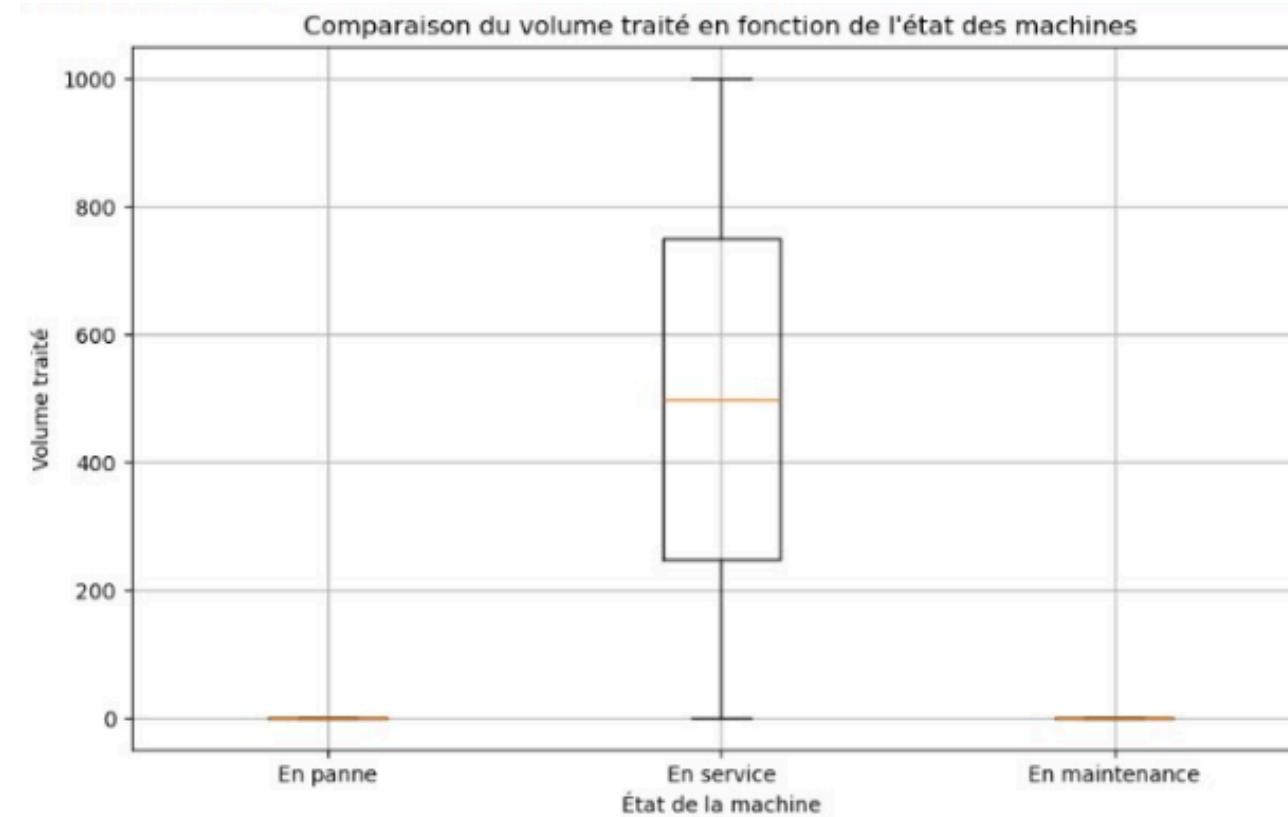
La charge de travail est-elle répartie équitablement entre les entrepôts ?



Certaines machines tombent-elles en panne plus fréquemment que d'autres ?



Quel est l'impact des pannes de machines sur le volume de commande traité ?



Corrélation entre les pannes machines et les volumes traités

id_entrepôt	Somme de nb_machines_en_maintenance	Somme de nb_machines_en_panne	Somme de nb_machines_en_service	Somme de total_commandes_traitées
E001	734	753	733	362624
E002	684	662	634	316528
E003	693	733	794	395630
E004	535	557	588	291102
E005	741	707	712	355100
E006	573	576	531	261527
E007	820	732	728	362529
E008	608	606	586	297977
E009	380	406	414	204967
E010	603	586	611	304821
Total	6371	6318	6331	3152805

Annexe 3 : Prédictions des ventes futures via Machine Learning

```
# Fusionner Commandes et Details_Commandes (relation via id_commande)
data = commandes.merge(details_commandes, on='id_commande', how='inner')

# Ajouter les informations sur les produits (via id_produit)
data = data.merge(produits, left_on='id_produit', right_on='ID', how='left')

# Ajouter les informations sur les clients (via id_client)
data = data.merge(clients, on='id_client', how='left')

# Ajouter les informations sur les promotions (via id_promotion_appliquée et id_produit)
data = data.merge(promotions, left_on='id_promotion_appliquée', right_on='id_promotion', how='left')

# Vérifier la table fusionnée
print(data.head())

id_commande id_client id_entrepôt_départ date_commande statut_commande \
0 CMD00020 C06919 E001 2023-12-03 Livrée
1 CMD00031 C02041 E001 2023-12-03 Livrée
2 CMD00161 C02419 E001 2023-12-04 Livrée
3 CMD00398 C08990 E001 2023-12-06 Livrée
4 CMD00648 C01596 E001 2023-12-07 Livrée

id_promotion_appliquée mode_de_paiement \
0 P065 PayPal
1 P097 Carte de crédit
2 P090 Virement bancaire
3 P028 Carte de crédit
4 P043 PayPal

numéro_tracking date_livraison_estimée id_produit_x \
0 8dd139f6-79a7-42ca-bb0b-aa4767a7c1e8 2023-12-16 P00075
1 426338cd-9fc8-40a6-a533-fb44e5e8ede4 2023-12-12 P00662
2 f981ee2b-69a5-4915-82fa-e095a6e8411d 2023-12-10 P00801
3 8dd20263-3d9d-46f8-a584-6d3cffd3679 2023-12-11 P00584
4 add7ac62-3f32-41d3-b817-379fd3b4fb5 2023-12-18 P00357

... date_inscription favoris id_promotion id_produit_y \
0 ... 2023-01-17 P493,P039,P215,P999 NaN NaN
1 ... 2024-06-30 P505,P756 NaN NaN
2 ... 2022-08-23 P793,P789,P598,P048 NaN NaN
3 ... 2022-12-21 P630 NaN NaN
4 ... 2024-10-31 P240 NaN NaN

type_promotion valeur_promotion valeur_promotion_float date_début date_fin \

```

```
# Ajouter les campagnes en fonction des dates (approximation)
data['date_commande'] = pd.to_datetime(data['date_commande'])
campaigns['date'] = pd.to_datetime(campaigns['date'])

# Fusionner les campagnes sur des dates proches
data = data.merge(campaigns, left_on='date_commande', right_on='date', how='left')

print(data.head())

id_commande id_client id_entrepôt_départ date_commande statut_commande \
0 CMD00020 C06919 E001 2023-12-03 Livrée
1 CMD00031 C02041 E001 2023-12-03 Livrée
2 CMD00161 C02419 E001 2023-12-04 Livrée
3 CMD00398 C08990 E001 2023-12-06 Livrée
4 CMD00648 C01596 E001 2023-12-07 Livrée

id_promotion_appliquée mode_de_paiement \
0 P065 PayPal
1 P097 Carte de crédit
2 P090 Virement bancaire
3 P028 Carte de crédit
4 P043 PayPal

numéro_tracking date_livraison_estimée id_produit_x \
0 8dd139f6-79a7-42ca-bb0b-aa4767a7c1e8 2023-12-16 P00075
1 426338cd-9fc8-40a6-a533-fb44e5e8ede4 2023-12-12 P00662
2 f981ee2b-69a5-4915-82fa-e095a6e8411d 2023-12-10 P00801
3 8dd20263-3d9d-46f8-a584-6d3cffd3679 2023-12-11 P00584
4 add7ac62-3f32-41d3-b817-379fd3b4fb5 2023-12-18 P00357

... id_campagne date événement_oui_non événement_type canal budget \
0 ... NaN NaT NaN NaN NaN <NA>
1 ... NaN NaT NaN NaN NaN <NA>
2 ... NaN NaT NaN NaN NaN <NA>
3 ... NaN NaT NaN NaN NaN <NA>
4 ... NaN NaT NaN NaN NaN <NA>

impressions clics conversions CTR
0 <NA> <NA> <NA> NaN
1 <NA> <NA> <NA> NaN
2 <NA> <NA> <NA> NaN
3 <NA> <NA> <NA> NaN
4 <NA> <NA> <NA> NaN
```

```

# Calcul du revenu par commande (quantité * prix)
data['revenu'] = data['quantité'] * data['Prix']

# Ajouter des informations temporelles
data['mois_commande'] = pd.to_datetime(data['date_commande']).dt.month
data['année_commande'] = pd.to_datetime(data['date_commande']).dt.year

# Indiquer si une commande a bénéficié d'une promotion
data['promotion_appliquée'] = data['id_promotion_appliquée'].notnull().astype(int)

# Ajouter une variable d'interaction (effet du type de promotion sur le revenu)
data['effet_promotion'] = data['valeur_promotion'].fillna(0) * data['quantité']

# Aperçu des nouvelles colonnes
print(data[['revenu', 'mois_commande', 'année_commande', 'promotion_appliquée', 'effet_promotion']].head())

```

	revenu	mois_commande	année_commande	promotion_appliquée	effet_promotion
0	106.2	12	2023	1	0
1	380.0	12	2023	1	0
2	376.9	12	2023	1	0
3	1350.0	12	2023	1	0
4	836.838	12	2023	1	0

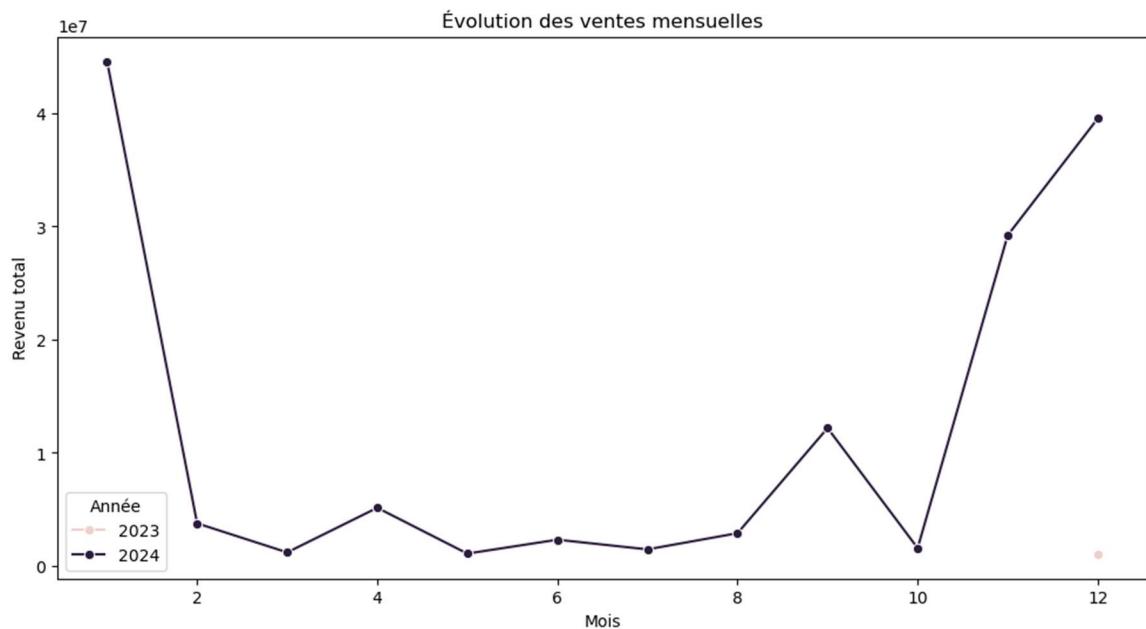
```

import seaborn as sns
import matplotlib.pyplot as plt

# Agrégation des revenus par mois et année
ventes_temps = data.groupby(['année_commande', 'mois_commande']).agg({'revenu': 'sum'}).reset_index()

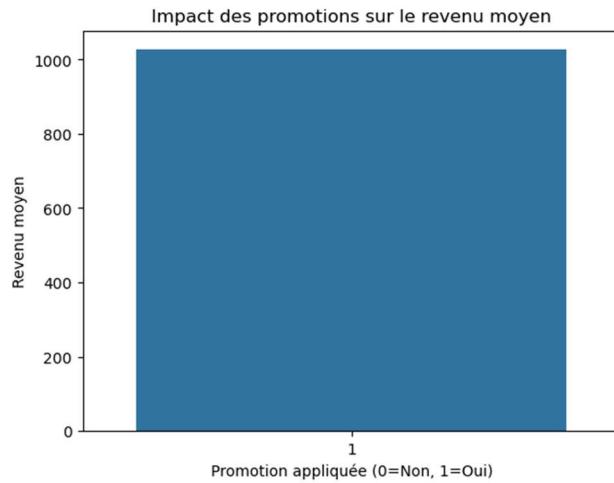
# Visualisation
plt.figure(figsize=(12, 6))
sns.lineplot(data=ventes_temps, x='mois_commande', y='revenu', hue='année_commande', marker='o')
plt.title("Évolution des ventes mensuelles")
plt.xlabel("Mois")
plt.ylabel("Revenu total")
plt.legend(title='Année')
plt.show()

```



```
# Calcul de l'impact des promotions sur le revenu
impact_promotion = data.groupby('promotion_appliquée').agg({'revenu': 'mean'}).reset_index()

# Visualisation
sns.barplot(data=impact_promotion, x='promotion_appliquée', y='revenu')
plt.title("Impact des promotions sur le revenu moyen")
plt.xlabel("Promotion appliquée (0=Non, 1=Oui)")
plt.ylabel("Revenu moyen")
plt.show()
```



```
# Variables explicatives et cible
features = ['Prix', 'quantité', 'mois_commande', 'année_commande', 'promotion_appliquée', 'effet_promotion']
target = 'revenu'

# Créer les ensembles d'entraînement et de test
from sklearn.model_selection import train_test_split

X = data[features]
y = data[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Initialiser le modèle
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Entrainer le modèle
model.fit(X_train, y_train)

# Prédire les valeurs sur l'ensemble de test
y_pred = model.predict(X_test)

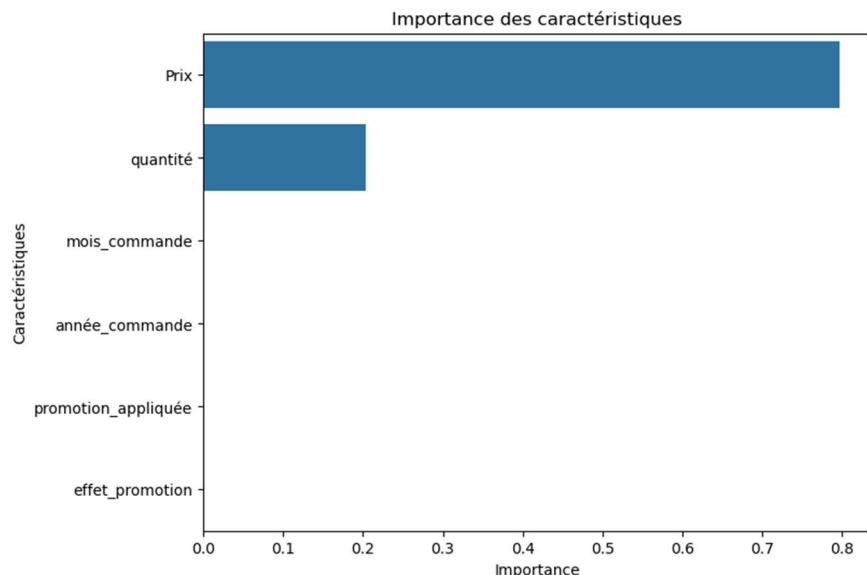
# Évaluer les performances
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f"MAE: {mae}")
print(f"RMSE: {rmse}")

MAE: 0.08101068343021665
RMSE: 4.057429730599401
```

```
# Importance des caractéristiques
importances = pd.Series(model.feature_importances_, index=features).sort_values(ascending=False)

# Visualisation
plt.figure(figsize=(8, 6))
sns.barplot(x=importances, y=importances.index)
plt.title("Importance des caractéristiques")
plt.xlabel("Importance")
plt.ylabel("Caractéristiques")
plt.show()
```



```
# Comparaison des valeurs réelles et prévues
plt.figure(figsize=(10, 6))
plt.plot(y_test.values[:50], label='Ventes réelles', marker='o')
plt.plot(y_pred[:50], label='Ventes prévues', marker='x')
plt.title("Comparaison des ventes réelles et prévues")
plt.xlabel("Exemples")
plt.ylabel("Revenu")
plt.legend()
plt.show()
```

