Analyse des clients

Introduction

Le but de notre application est de créer un environnement permettant aux entreprises d'analyser et de gérer les données de leur clients sous la forme d'un tableau de bord. C'est un outil pour mieux comprendre leurs clients, leur permettant ainsi de cibler plus efficacement des produits ou des campagnes promotionnelles.

Nous avons décidé de réaliser ce projet car il pourrait être un atout sur un CV, notamment parce que les entreprises s'intéressent à ce type d'applications, mais aussi parce que la base de données était simple, intuitive et car l'auteur a bien défini les colonnes.

Les données que nous avons utilisées sont disponibles sur :
https://www.kaggle.com/datasets/imakash3011/customer-personality-analysis?resource=download

Liste des tableaux

Description de Customers

Pour commencer, nous avons la table customers, qui contient les informations détaillées sur les clients. Au début, on avait envisagé de séparer la table en plusieurs sous-tables et les reliées avec des foreign keys de ID. Cela semblait logique, car les données pourraient évidemment être catégorisées en : personnes, produits, promotion, et lieu. Mais, on a changé d'avis car il n'y avait pas de redondances dans la table, donc il était inutile de la diviser. Donc, on l'a finalement créé avec le schéma:

```
CREATE TABLE customers (
        ID INT PRIMARY KEY,
        Year_Birth INT,
        Education VARCHAR(50),
        Marital_Status VARCHAR(50),
        Income FLOAT,
        Kidhome INT,
        Teenhome INT,
        Dt_Customer DATE,
        Recency INT,
        Complain INT,
        MntWines INT,
        MntFruits INT,
        MntMeatProducts INT,
        MntFishProducts INT,
        MntSweetProducts INT,
        MntGoldProds INT,
        NumDealsPurchases INT,
        AcceptedCmp1 INT,
        AcceptedCmp2 INT,
        AcceptedCmp3 INT,
        AcceptedCmp4 INT,
        AcceptedCmp5 INT,
        Response INT,
        NumWebPurchases INT,
        NumCatalogPurchases INT,
        NumStorePurchases INT,
        NumWebVisitsMonth INT
);
```

ID est la seule candidate key de la table customers. L'ensemble des superkeys de customers est l'ensemble des clés qui contiennent **ID**.

ID : Identifiant unique du client.
Year_Birth : Année de naissance du client.
Education : Niveau d'éducation du client.

Marital_Status : Statut marital du client.
Income : Revenu annuel du foyer du client.
Kidhome : Nombre d'enfants dans le foyer du client.
Teenhome : Nombre d'adolescents dans le foyer du client.
Dt_Customer : Date d'inscription du client auprès de l'entreprise.
Recency : Nombre de jours depuis le dernier achat du client.
Complain : Indique si le client a déposé une plainte au cours des 2 dernières années

MntWines : Montant dépensé en vins au cours des 2 dernières années.
MntFruits : Montant dépensé en fruits au cours des 2 dernières années.
MntMeatProducts : Montant dépensé en produits carnés au cours des 2 dernières années.
MntFishProducts : Montant dépensé en poissons au cours des 2 dernières années.
MntSweetProducts : Montant dépensé en produits sucrés au cours des 2 dernières années.
MntGoldProds : Montant dépensé en produits de luxe (or) au cours des 2 dernières années.
NumDealsPurchases : Nombre d'achats effectués avec une remise.

AcceptedCmp1 : Indique si le client a accepté l'offre lors de la 1ère campagne (1 : oui, 0 : non).
AcceptedCmp2 : Indique si le client a accepté l'offre lors de la 2ème campagne (1 : oui, 0 : non).
AcceptedCmp3 : Indique si le client a accepté l'offre lors de la 3ème campagne (1 : oui, 0 : non).
AcceptedCmp4 : Indique si le client a accepté l'offre lors de la 4ème campagne (1 : oui, 0 : non).
AcceptedCmp5 : Indique si le client a accepté l'offre lors de la 5ème campagne (1 : oui, 0 : non).
Response : Indique si le client a accepté l'offre lors de la dernière campagne

NumWebPurchases : Nombre d'achats effectués sur le site web de l'entreprise.
NumCatalogPurchases : Nombre d'achats effectués via un catalogue.
NumStorePurchases : Nombre d'achats effectués directement en magasin.
NumWebVisitsMonth : Nombre de visites sur le site web de l'entreprise au cours du dernier mois.

Les données de customers on été insérées via le script: customersparse.py.

La table customers est en BNCF et en 3NF car ID est une super key et la seule dépendance est ID -> Year_Birth, Education, Marital_Status, Income, Kidhome, Teenhome, Dt_Customer, Recency, Complain, MntWines, MntFruits, MntMeatProducts, MntFishProducts, MntSweetProducts, MntGoldProds, NumDealsPurchases, AcceptedCmp1, AcceptedCmp2, AcceptedCmp3, AcceptedCmp4, AcceptedCmp5, Response, NumWebPurchases, NumCatalogPurchases, NumStorePurchases, NumWebVisitsMonth.

Description de Users

Deuxièmement, on a créé la table users qui contient les informations sur les utilisateurs. Cette table commence vide et est remplie au fur et à mesure que les utilisateurs s'inscrivent à l'application. On la crée avec le schéma:

```
CREATE TABLE users(
        ID INT PRIMARY KEY,
        Username VARCHAR(50),
        Password VARCHAR(50),
);
```

ID est la seule candidate key de la table customers. L'ensemble des superkeys de users est l'ensemble des clés qui contiennent **ID**.

ID : Identifiant unique du client.
Username : Pseudonyme de l'utilisateur
Password : Mot de passe de l'utilisateur

Au début, la table users n'était pas en BNCF ou en 3NF car elle contenait aussi la colonne History qui créait beaucoup de redondances. Pour résoudre ce problème, on a divisé les tables pour normaliser la table users. La table users est en BNCF et en 3NF car ID est une super key et la seule dépendance est ID -> Username, Password.

Description de Complaints

Troisièmement, on a la table complaints qui contient les plaintes des clients et les détails de la réponse d'un utilisateur s'il y en a une. On la crée avec le schéma:

```
CREATE TABLE complaints(
        complaint_ID INT,
        complaint VARCHAR(400),
        comment VARCHAR(400),
        resolved BOOLEAN DEFAULT 'f',
        resolvedby INT,
        FOREIGN KEY (Complaint_ID) REFERENCES customers(ID)
        FOREIGN KEY (resolvedby) REFERENCES users(ID)
);
```

complaint_ID et **resolvedby** sont des foreign keys, mais **complaint_ID** est la seule candidate key, donc l'ensemble des superkeys de complaints est l'ensemble des clés qui contiennent **complaint_ID**.

Complaint_ID : Identifiant unique du client qui a déposé la plainte
Complaint : La plainte en forme texte du client
Comment: Le commentaire d'un utilisateur sur la plainte
Resolved : Indique si la plainte a été résolu
Resolvedby : Identifiant unique de l'utilisateur qui a répondu à la plainte

La table complaints est en BNCF et en 3NF car Complaint_ID est une superkey et la seule dépendance est Complaint_ID -> Complaint, Comment, Resolved, Resolvedby.

Description de History

Quatrièmement, on a la table History qui contient l'historique des requêtes SQL exécutées par les utilisateurs. Cette table a été séparée de la table users, car on aurait pu avoir des redondances. Chaque utilisateur pourrait exécuter plusieurs fois la même requête, donc les stocker dans une table distincte est plus efficace. On l'a créée avec le schéma:

```
CREATE TABLE history(
        History_ID INT,
        request VARCHAR(400),
        FOREIGN KEY (History_ID) REFERENCES client(ID)
);
```

History_ID et **request** forment la candidate key, car on peut avoir plusieurs requêtes pour chaque ID et plusieurs ID pour chaque requête, ainsi chaque requête exécutée par un utilisateur sera unique.

History_ID : Identifiant unique de l'utilisateur ayant exécuté la requête.
request : Texte de la requête SQL exécutée par l'utilisateur de longueur maximale de 400.

La table history est en BNCF et en 3NF car la seule dépendance fonctionnelle est History_ID, request → request, ce qui évite toute redondance dans la table.

Les Fonctionnalités Implémentées

1) Connexion utilisateur

Fonctionnalité : Permet à un utilisateur de se connecter en entrant son nom d'utilisateur et un mot de passe.

Requête SQL associée :

SELECT count(*) FROM users WHERE username = '{username}' AND password = '{password}'

Vérifie si les informations d'identification saisies correspondent à un utilisateur existant.

Utilisation : Si un utilisateur est trouvé, l'application permet la connexion.

2) Inscription (Sign Up)

Fonctionnalité : Ajoute un nouvel utilisateur dans la base de données avec un identifiant, un nom d'utilisateur, et un mot de passe

Requête SQL associée :

I**NSERT INTO users(id, username, password) VALUES ({id}, '{username}', '{password}')**

Ajoute un nouvel utilisateur avec un identifiant unique généré automatiquement, égale au nombre d'utilisateurs + 1.

3) Affichage des clients (menu principal)

Fonctionnalité : Affiche les informations des clients dans une table.

Requête SQL associée :

SELECT * FROM customers ORDER BY id

Charge et affiche toutes les données de la table customers, triées par l'identifiant.

4) Recherche des clients

Fonctionnalité : Permet aux utilisateurs de rechercher des clients par divers critères (ID, statut marital, éducation, etc.).

Requête SQL associée :

SELECT * FROM customers

WHERE CAST(id AS TEXT) LIKE '%{search_query}%'

OR LOWER(education) LIKE LOWER('%{search_query}%')

OR LOWER(marital_status) LIKE LOWER('%{search_query}%')

OR CAST(kidhome AS TEXT) LIKE '%{search_query}%'

Recherche dans plusieurs colonnes en fonction de la saisie de l'utilisateur.

5) Tri des clients

Fonctionnalité : Trie les clients affichés dans la table selon une colonne sélectionnée.

Requête SQL associée :

SELECT * FROM customers ORDER BY {column_name}

Trie les données de la table customers selon une colonne choisie (par exemple, id, name).

6) Gestion des plaintes

Fonctionnalité : Affiche les plaintes des clients dans une boîte de sélection.

Permet de répondre aux plaintes et de les marquer comme résolues.

Requêtes SQL associées :

1. Chargement des plaintes :

SELECT * FROM complaints ORDER BY complaint_ID

2. Afficher les détails d'une plainte :

SELECT * FROM complaints WHERE complaint = '{complaint_text}'

3. Mettre à jour une plainte avec une réponse :

UPDATE complaints

SET comment = '{response}', resolved = {resolved}, resolvedby = {user_id}

WHERE complaint = '{complaint_text}'

Charge toutes les plaintes pour affichage.

Affiche les détails d'une plainte sélectionnée.

Permet à un utilisateur d'ajouter une réponse et de marquer la plainte comme résolue.

7) Gestion de l'historique

Fonctionnalité : Enregistre les requêtes SQL exécutées de la dernière session de l'utilisateur dans l'historique.

Permet de réexécuter les requêtes depuis l'historique.

Requêtes SQL associées :

1. Enregistrer une requête dans l'historique :

INSERT INTO history(history_ID, request) VALUES ({user_id}, '{sql_query}')

2. Charger l'historique de l'utilisateur :

SELECT DISTINCT * FROM history WHERE history_ID = {user_id}

3. Exécuter une requête depuis l'historique :

- Selon la requête enregistrée, par exemple :

SELECT * FROM customers ORDER BY id

Enregistre les actions importantes et permet de réexécuter facilement des requêtes.

8) Catégorisation des clients

Fonctionnalité : Divise les clients en 5 catégories en fonction de leur comportement d'achat puis affiche leur pourcentage en forme d'un graphe circulaire.

Permet à l'utilisateur de voir combien de clients appartiennent à chaque type afin de mieux cibler leurs campagnes promotionnelles.

Requêtes SQL associées :

Calcul et chargement des catégories et des comptes des clients :

```
WITH CategorizedCustomers AS ( SELECT ID,
                                    CASE
                                        WHEN NumWebVisitsMonth > 0 AND
((MntFishProducts + MntFruits + MntGoldProds + MntMeatProducts +
MntSweetProducts + MntWines) * NumCatalogPurchases) /
NULLIF(NumWebVisitsMonth, 0) > 25000.00 THEN 'Loyal Customer'
                                        WHEN NumDealsPurchases > 0 AND
(NumCatalogPurchases + NumStorePurchases + NumWebPurchases) /
NULLIF(NumDealsPurchases, 0) > 8.00 THEN 'Discount Buyer'
                                        WHEN (NumCatalogPurchases +
NumStorePurchases + NumWebPurchases) > 0 AND (MntFishProducts + MntFruits +
MntGoldProds + MntMeatProducts + MntSweetProducts + MntWines) /
NULLIF((NumCatalogPurchases + NumStorePurchases + NumWebPurchases), 0) >
10000 THEN 'Luxury Buyer'
                                        WHEN NumWebVisitsMonth > 0 AND
NumWebPurchases / NULLIF(NumWebVisitsMonth, 0) < 0.75 THEN 'Window Shopper'
                                        WHEN NumWebVisitsMonth > 0 AND
NumWebPurchases / NULLIF(NumWebVisitsMonth, 0) >= 1.5 THEN 'Impulse Buyer'
                                        ELSE 'Uncategorized'
                                    END AS Category
        FROM customers ),
CategoryCounts AS (SELECT  Category, COUNT(*) AS CustomerCount
        FROM CategorizedCustomers
        GROUP BY Category)
SELECT Category, CustomerCount
FROM CategoryCounts
ORDER BY CustomerCount DESC;
```

Charge toutes les catégories des clients.

Affiche les données dans un graphique circulaire interactif avec des couleurs et des labels distincts.

9) Classement de la fidélité de chaque client

Fonctionnalité : Classe les clients en fonction de leur fidélité à l'aide d’un score calculé.

Permet à l'utilisateur de visualiser les clients les plus fidèles pour optimiser les initiatives de fidélisation.

Requêtes SQL associées :

Calcul du score de fidélité et classement des clients

```
WITH CustomerScores AS (SELECT ID,
                        CASE
                            WHEN NumWebVisitsMonth > 0
                            THEN (MntFishProducts + MntFruits +
MntGoldProds + MntMeatProducts + MntSweetProducts + MntWines) *
NumCatalogPurchases / (NumWebVisitsMonth * 10000)
                            ELSE 0
                        END AS Score
FROM customers)

SELECT c.ID, c.Score, (SELECT COUNT(*)
                       FROM CustomerScores c2
                       WHERE c2.Score > c.Score)
                       + 1 AS Rank
FROM CustomerScores c;
```

Le score est calculé en fonction des montants dépensés sur plusieurs types de produits, multipliés par les achats par catalogue, puis normalisé par le nombre de visites sur le site web.

Si un client n'a pas de visites sur le web, le score est défini à 0 pour éviter les divisions par zéro.

Les difficultés rencontrées

Connexion avec psycopg2

Au début, on voulait utiliser le module psycopg2 pour gérer les comptes des utilisateurs. Cependant, au fur et à mesure du projet, on a réalisé qu'on devait utiliser ce module pour intégrer la base de données. Donc, on a décidé de ne pas utiliser psycopg2 pour stocker les détails des utilisateurs avec PostgreSQL car:

- On aurait dû l'intégrer deux fois pour des objectifs différents.
- Le processus de création d'un utilisateur était compliqué et a causé de nombreux bugs dans le code.
- L'apprentissage de PostgreSQL a pris du temps et a réduit celui qu'on aurait pu consacrer aux autres fonctionnalités du projet.

En fin de compte, on a décidé de créer un système simple pour gérer les comptes des utilisateurs, en utilisant une table séparée, qui crée un identifiant pour chaque utilisateur et stocke leur nom d'utilisateur et mot de passe. Cela signifie que tout le programme sera

maintenant lié à une seule base de données, mais ça ne pose pas de problème. Tout ce qu'on a à faire, c'est de changer la portée du projet pour qu'il concerne les clients d'une seule entreprise, et les utilisateurs seront les employés de cette entreprise. Ainsi, les détails des clients seront les mêmes pour tous les utilisateurs.

Chargement des tables

On a eu un petit problème pour comprendre comment charger les en-têtes des tables, car on ne l'avait jamais fait en TP. Cependant, c'est une fonctionnalité essentielle pour une application de tableau de bord, car il est important de savoir ce que chaque colonne représente.

Finalement, on a compris qu'on pouvait utiliser cursor.description pour extraire les en-têtes en récupérant le premier élément des listes.

Tri des clients

Ensuite, une fonctionnalité qui semblait facile, le tri de la table, nous a pris beaucoup de temps à bien réaliser. La raison principale est que cela faisait souvent planter notre programme. Au début, on utilisait Qt5 pour gérer le tri, mais on a trouvé plus pertinent de développer cette fonctionnalité avec SQL.

La première étape consistait à vider la table, pour pouvoir la recharger entièrement en utilisant la même commande avec un SORT BY ajouté. La deuxième étape était de remettre correctement le numéro des lignes avec les en-têtes. La troisième étape, celle qui a vraiment corrigé le problème, était de vérifier que la variable utilisée pour le tri appartenait bien à la table, même si on était sûrs que c'était le cas. Une fois cette vérification faite, on pouvait exécuter la requête SQL et réinsérer les résultats dans la table.

Calcul des marges pour chaque type de client

On voulait créer une fonction pour classer les clients en catégories, mais la base de données donnée par l'auteur n'avait pas beaucoup de colonnes intéressantes à cet effet. Donc notre plus gros problème était de savoir comment donner une catégorie à chaque client : est-ce qu'on devait utiliser une seule formule ou prendre en compte plusieurs facteurs ?

Après avoir passé du temps à examiner les données et les colonnes dans la table, on a choisi 5 catégories, chacune avec une formule spécifique, qu'on peut attribuer à un client. Nous choisissons aussi le seuil pour chaque en prenant les valeurs au-dessus du troisième quartile pour "haut" et celles en dessous pour "bas".

- **Loyal Customer** si (MntFishProducts + MntFruits + MntGoldProds + MntMeatProducts + MntSweetProducts + MntWines) * NumCatalogPurchases / (NumWebVisitsMonth*10000) = Haut **> 25.00**
- **Discount Buyer** si (NumCatalogPurchases + NumStorePurchases + NumWebPurchases) / NumDealsPurchases = Haut **> 8.00**

- **Luxury Buyer** si (MntFishProducts + MntFruits + MntGoldProds + MntMeatProducts + MntSweetProducts + MntWines) / (NumCatalogPurchases + NumStorePurchases + NumWebPurchases) = Haut **> 10000**
- **Window Shopper** si NumWebPurchases / NumWebVisitsMonth = Bas **< 0.75**
- **Impulse Buyer** si NumWebPurchases / NumWebVisitsMonth = Haut **> 1.5**

Contributions de DIA Aliou

Ma contribution au projet se concentre sur l'amélioration de l'interactivité et de l'expérience utilisateur grâce à l'implémentation de fonctionnalités clés.

J'ai développé une barre de recherche intuitive qui permet aux utilisateurs d'effectuer des recherches dynamiques sur plusieurs critères, tels que l'identifiant ou l'éducation des clients.

J'ai également intégré un bouton associé pour déclencher ces recherches, en veillant à ce que les résultats soient affichés de manière fluide dans une table interactive.

En complément, on a mis en place un système d'historique des recherches qui enregistre et affiche les requêtes effectuées, permettant ainsi aux utilisateurs de consulter ou réutiliser facilement leurs précédentes recherches. J'ai contribué à l'agencement de l'interface en optimisant la navigation et l'esthétique de l'application.

Contributions de JENY JEYARAJ John-Michael

Pour commencer, j'ai "parsé" les données du fichier 'marketing_campaign.csv' pour les intégrer dans la base de données. Puis, j'ai positionné chaque élément de l'interface à son emplacement précis. Ensuite, j'ai développé le système de gestion des comptes utilisateurs avec une table dédiée. Ceci permet la création de comptes utilisateurs et leur connexion à l'application.

Après avoir connecté la base de données dans le menu, je l'ai intégré au tableau contenant les données des clients. De plus, j'ai également implémenté la fonction de tri sur ce tableau.

J'ai conçu le système de gestion de plaintes des clients, ce qui permet à l'utilisateur de consulter, traiter et répondre efficacement à chaque plainte.

Pour mieux analyser les clients, je les ai catégorisés selon leur comportement : Loyal Customer, Discount Buyer, Luxury Buyer, Impulsive Buyer, et Window Shopper. J'ai représenté ces catégories sous forme d'un diagramme circulaire (pie chart), et j'ai classé les clients en fonction de leur fidélité à gauche du diagramme.

Finalement, j'ai structuré l'historique des utilisateurs, et intégré ce-dernier dans la fonction de tri et la gestion de plaintes.