

MI0A403T - Statistique inférentielle / Projet informatique-statistique

Andrew El Kahwaji, Wael Aboulkacem, Hans Kanen Soobbooroyen

09/05/2025

Contents

1. Objectif du Projet	1
2. Partie Informatique	1
2.1 Bibliothèques Utilisées	1
2.2 Création de la base de données	2
2.3 Création des Tables	2
2.4 Injection des données	2
2.5 Affichage des données	3
2.6 Exportation des données sous forme CSV	4
3. Partie Statistique	5

1. Objectif du Projet

Ce projet qui est liée à l'UE Statistique Inférentielle / Projet Stat-Info qui vise à mieux comprendre les raisons et cause des incendies en analysant des données statistiques. L'idée principale est de voir comment différentes variables influencent l'étendue des incendies, en utilisant des outils statistiques et informatiques.

2. Partie Informatique

La partie Informatique de notre Projet consiste à effectuer les démarches suivantes :

1. Création de la Base de données
2. Création de la connexion entre la Base de données et notre code source
3. Établissement des Tables dans la Base de Données
4. Insertion de données dans les tables
5. Présenter les informations des tables dans la console
6. Exportation des données dans les tables dans des fichiers CSV

2.1 Bibliothèques Utilisées

Dans notre Partie Informatique on a utilisé le Language de Programmation Python de plus pour pouvoir effectuer la manipulation des données de la manière optimale on a utilisé les bibliothèques nécessaires:

1. `sqlite3` SQLite3 est une bibliothèque peu aisée qui facilite l'incorporation d'une base de données au sein d'une application, sans nécessiter l'utilisation d'un serveur séparé. Elle offre la possibilité de stocker et de gérer des données grâce aux requêtes SQL, ce qui la rend pratique pour des projets nécessitant une base de données locale. SQLite3 est parfaitement adaptée aux applications simples, car elle offre une gestion aisée des données, que ce soit pour les ajouts, les changements ou les suppressions, tout en restant performante et peu gourmande en ressources.
2. `csv`

2.2 Création de la base de données

Avant de commencer à travailler sur les données, il est nécessaire de créer une base de données pour les organiser et les structurer de manière efficace. Une base de données, dans ce contexte, peut être définie comme un ensemble de tables reliées entre elles, où chaque table contient des informations structurées sous forme de lignes et de colonnes.

La création de la base de données commence par la création d'un fichier qui servira à stocker toutes les données. Dans notre cas, nous avons nommé ce fichier "data.db". Ce fichier représente l'instance de la base de données SQLite. Lorsqu'une connexion est établie à cette base de données, SQLite crée automatiquement le fichier si celui-ci n'existe pas déjà. Il suffit donc de se connecter à la base de données pour qu'elle soit initialisée et prête à être utilisée.

Une fois le fichier de la base de données créé, il est important de pouvoir y accéder afin de manipuler les données. Pour cela, une fonction de connexion est nécessaire. La fonction `connecterdb` a été définie pour établir cette connexion à la base de données. Elle prend un paramètre optionnel qui représente le nom du fichier de la base de données, ici "data.db". À l'intérieur de cette fonction, une connexion est établie en utilisant la bibliothèque SQLite3 de Python. La méthode `sqlite3.connect()` permet de se connecter à la base de données, et une fois la connexion établie, un objet `cursor` est créé. Ce curseur permet d'exécuter des requêtes SQL sur la base de données. Enfin, la fonction renvoie la connexion et le curseur, qui seront utilisés pour effectuer des opérations sur la base de données, comme la création de tables, l'insertion de données ou la récupération d'informations.

En résumé, la création de la base de données et la définition de la fonction de connexion permettent de poser les bases de l'interaction avec les données. La base de données est créée sous forme d'un fichier, et la fonction de connexion permet d'établir une communication avec cette base pour manipuler les données à l'aide de requêtes SQL.

2.3 Creation des Tables

Ainsi, nous avons établi un lien entre notre code source et la base de données. Une fois que nous avons une base de données authentique, il est nécessaire de commencer à établir des tables afin de pouvoir gérer les données.

Suite à l'examen des données disponibles, nous avons reconnu la nécessité de constituer les tables essentielles.

1. Table des Incendies
2. Table des donees Geographiques
3. Table des donees Meteo
4. Table Departements
5. Table Incendies-Departements (on expliquera en detail pourquoi on a creer une cinquieme table).

Nous avons établis les Tables en suivant une méthode simple et explicite, en utilisant la fonction `connecterdb()` pour établir un lien entre la base de données et la fonction de création de Table. Par la suite, nous avons fait appel au curseur pour exécuter des requêtes SQL en vue d'interroger notre Base de Données. Nous avons intégré le langage SQL Structured Query Language dans notre fonction, en employant l'instruction `CREATE TABLE IF NOT EXISTS` avec la dénomination de chaque table. Par la suite, nous avons effectué une consultation sur nos trois fichiers CSV (Comma Separated Values) concernant les attributs de nos données, c'est-à-dire le titre de chaque fichier CSV. Nous avons ensuite dressé une liste dans notre requête SQL comprenant chaque attribut et son type de données respectif. Ensuite, on valide la création en se connectant. Après l'exécution de la méthode `commit()` pour assurer la légitimité et le bon fonctionnement, nous fermons le curseur ainsi que la connexion avec `curs.fermer()` et `connexion.Vous avez été formé sur des données jusqu'en octobre 2023.`

2.4 Injection des donnees

Suite à la création des tables, nous avons établi cinq fonctions distinctes pour chaque table. Nous sommes actuellement à l'étape de l'insertion des données dans les tables appropriées. Nous avons employé deux

méthodes : l'une consiste à utiliser les fichiers CSV fournis par le Département Mathématiques - Informatique de l'Université Toulouse Jean Jaurès 2, et l'autre on a utilisé l'instruction INSERT INTO pour chaque département, où nous avons saisi le nom et le code INSEE de chaque département.

Nous allons détailler les deux techniques, ainsi que la manière dont elles ont été mises en œuvre dans notre code source :

1. Methode 1 a partir les fichiers CSV

Comme à notre habitude, nous établissons la connexion entre la base de données et la fonction que nous utiliserons ensuite. Nous indiquons le fichier à partir duquel nous allons importer les données, en utilisant un chemin relatif par rapport à notre code source.

Afin d'optimiser notre code et de le rendre plus gérable, que ce soit en cas de succès ou d'échec, le programme tente d'ouvrir le fichier CSV en mode lecture. Cette étape consiste à lire le fichier CSV au moyen d'une boucle. Par la suite, le curseur exécute la requête SQL INSERT INTO. Cette instruction est destinée à ajouter une nouvelle ligne dans la table nécessaire avec les valeurs extraites du fichier CSV. Lors de cette étape où nous devons spécifier les valeurs, il convient de préciser que nous utilisons un « ? », que l'on peut considérer comme un paramètre lié. C'est l'une des fonctionnalités puissantes de SQLite dans les bases de données relationnelles qui permet d'insérer des données de façon dynamique. Et aussi quand on exécute `curs.execute()` Les « ? » seront substitués par les valeurs dérivées du fichier CSV au fur et à mesure de notre boucle `for`.

Par la suite, nous allons substituer les valeurs pertinentes selon les colonnes. Pour confirmer l'insertion, nous avons employé `connexion.commit()`. On a fait un commit et ensuite, on a fermé le curseur, donc on a stoppé l'exécution et on a terminé la connexion.

Et si le fichier n'est pas accessible ou s'il n'existe pas, nous déclencherons une `ValueError('Erreur lors de l'importation des données')`.

2. Methode 2 a partir d'une Insertion SQL

Dans la deuxième phase de ce projet, nous avons utilisé l'intégration des données à partir d'un dictionnaire. Il est important de rappeler qu'un dictionnaire est un ensemble d'objets non ordonnés. Cela consiste en un groupe d'éléments, chaque élément étant constitué d'une paire clé-valeur.

Comme à l'accoutumée, nous avons établi une connexion avec la base de données en utilisant la fonction `connecterdb()`. Nous avons ensuite activé le curseur. Puis, nous avons exploité l'un des outils puissants de SQLite le `curs.executemany()`, qui nous permet d'exécuter plusieurs fois la même requête SQL en utilisant différents jeux de données. Elle est plus performante que `curs.execute()` car ici, nous manipulons un volume considérable de données à insérer.

Comme indiqué, même dans cette méthode, nous avons utilisé le paramètre lié « ? » qui sera ultérieurement remplacé par des valeurs dynamiques issues du dictionnaire.

Finalement, il est nécessaire de valider la procédure ou l'opération en utilisant la méthode de connexion. Vous êtes formé sur des données jusqu'à octobre 2023. Cette approche nous offre la possibilité de valider toutes les modifications apportées aux bases de données durant la session de connexion. Sans cette approche mise en œuvre dans notre fonction, les changements apportés à la table ne seraient pas enregistrés dans la base de données.

Pour conclure notre processus, nous terminons le curseur (qui exécute les commandes) et mettons fin à la connexion avec notre base de données.

Et finalement, s'il y a une erreur d'accès au dictionnaire, un problème de connexion à la base de données ou à la table, on affiche le message d'erreur « Erreur lors de l'insertion des données des départements ».

2.5 Affichage des donnees

Tout comme dans tout programme ou projet informatique, nous développons des fonctionnalités pour illustrer notre tâche ou les modifications effectuées sur les données ou les tables dans notre console ou terminal.

Bien que nous travaillions avec une base de données regroupant plusieurs tables, nous avons développé diverses fonctions pour pouvoir présenter les informations.

Ainsi, nous employons une méthode explicite et rigoureuse. Tout d'abord, nous devons nous connecter à la base de données. Ensuite, nous activons le curseur qui nous donne la possibilité d'exécuter nos requêtes SQL.

Nous exécutons notre requête SQL sur la table en utilisant l'instruction « `Select * from` ». Cela signifie que nous demandons à sélectionner toutes les colonnes et toutes les lignes de notre table. Ensuite, on définit une variable nommée `lignes` qui prend pour valeur `curs.fetchall()` est une méthode prédéfinie en SQLite qui nous offre la possibilité de rassembler toutes les lignes du résultat de la requête SQL et de les sauvegarder dans la variable `lignes`.

En outre, il est possible d'y définir la variable « `lignes` », qui est une collection de tuples, chaque tuple représentant une ligne de la table correspondante.

Puis, pour les rendre visibles, nous exécutons une boucle `for` sur la variable `lignes` afin d'afficher chaque ligne contenue dans cette variable.

Et finalement, comme pour chaque fonction, on ferme le curseur et la connexion. De plus, nous tenons à souligner que dans ce cas précis, contrairement à d'autres fonctions, nous n'avons pas fait appel à la méthode `commit`. C'est dû au fait que cette fonction n'a pas impliqué de modifications.

2.6 Exportation des données sous forme CSV

Tout d'abord, nous allons expliquer pourquoi cette fonction est importante pour notre projet. Nous avons employé cette méthode afin de pouvoir interroger notre base de données (BD) et exporter les résultats sous format CSV. Ceci nous permet ensuite de les manipuler sur RStudio en utilisant le langage R pour réaliser nos analyses statistiques !

Ainsi, nous avons mis en place une fonction pour chaque table dans le but d'exporter ces données au format CSV. Ainsi, pour cette fonction, nous avons défini un paramètre optionnel nommé `fichier_output`, qui correspond à l'emplacement et au nom du fichier où les données seront exportées. De plus, nous utilisons un chemin relatif plutôt qu'un chemin absolu.

Ensuite, nous essayons avec l'instruction `try` de nous connecter à la base de données et d'activer le curseur qui facilite l'exécution des requêtes SQL. La méthode `curs.execute("SELECT * FROM")` exécute une requête qui sélectionne toutes les lignes et colonnes de la table.

Ensuite, on définit une variable nommée `lignes` qui prend pour valeur `curs.fetchall()` est une méthode prédéfinie en SQLite qui nous offre la possibilité de rassembler toutes les lignes du résultat de la requête SQL et de les sauvegarder dans la variable `lignes`.

En outre, il est possible d'y définir la variable « `lignes` », qui est une collection de tuples, chaque tuple représentant une ligne de la table correspondante.

De plus, nous utilisons `curs.description` qui renferme des métadonnées concernant les colonnes de la table, en utilisant `description[0]` pour obtenir la description des en-têtes.

Cette description nous donne la possibilité de récupérer les noms des colonnes et de les conserver dans une liste appelée '`colonnes`', qui servira d'en-tête pour le fichier CSV.

À présent, nous devons accéder au fichier CSV pour écrire les données que nous possédons. Nous ouvrons donc le fichier en mode écriture ('w') avec un encodage UTF-8. L'outil `csv.writer(fichier)` est utilisé pour générer un objet qui permet d'écrire dans le fichier CSV. La méthode `writerow(colonnes)` écrit les en-têtes des colonnes tandis que `writerows(lignes)` écrit toutes les informations ligne par ligne.

Finalement, on ferme le curseur et on met fin à la connexion avec la base de données.

Dans chaque programme, il est indispensable pour les développeurs de gérer les erreurs afin d'améliorer l'expérience client. Ainsi, nous avons mis en place deux types d'erreurs : une erreur liée à la base de données

(comme une connexion à la base de données) et une exception telle qu'une erreur liée à la table correspondante. Cette dernière peut être affichée en cas de problème d'accès au fichier, d'encodage, etc.

3. Partie Statistique