

Mise en situation 1 (E1)

Collecte, stockage et mise à disposition des
données d'un projet IA

Projet ANIMOV : Surveillance et Analyse
Comportementale des Chèvres avec l'IA

Formation Développeur en Intelligence Artificielle
RNCP 37827
Promotion 2023-2024

Manuel CALDEIRA

Table des matières

1. INTRODUCTION.....	3
1.1. CONTEXTE DU PROJET	3
1. AUTOMATISER L'EXTRACTION DE DONNEES	3
1.2. OBJECTIF DE LA COMPETENCE.....	3
1.3. ARCHITECTURE	3
1.3.1. Utilisateur :	3
1.3.2. Streamlit :	3
1.3.3. API Flask :	3
1.3.4. ANIMOV (MySQL) :	4
1.3.5. MongoDB (Record Data) :	4
1.3.6. YOLO (IA) :	4
1.3.7. Site Web :	4
1.3.8. CSV/Files Data ANIMOV :	4
1.3.9. Intercept Data :	4
1.4. UN SERVICE WEB (API) ET UTILISATION DE LA BIBLIOTHEQUE 'REQUESTS'	4
1.5. EXTRACTION DE DONNEES DEPUIS UNE PAGE WEB (WEB SCRAPING)	4
1.6. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES CSV	4
1.7. EXTRACTION DE DONNEES DEPUIS UNE BASE DE DONNEES.....	5
1.8. EXTRACTION DE DONNEES DEPUIS UN SYSTEME BIGDATA.....	5
2. BASE DE DONNEES MYSQL.....	5
2.1. LES TABLES.....	5
2.2. CONFIGURATION DE LA CONNEXION A LA BASE DE DONNEES	6
2.3. PROCEDURES STOCKEES.....	6
2.4. LES VUES.....	6
2.5. REQUETES SQL POUR L'EXTRACTION DE DONNEES	7
3. DEVELOPPER DES REGLES D'AGREGATION DE DONNEES	7
4. DEVELOPPER UNE API REST.....	8
4.1. OBJECTIF DE LA COMPETENCE.....	8
4.2. DOCUMENTATION DE L'APPLICATION.....	8
5. RGPD ET AUTOMATISATION DE L'EXTRACTION DE DONNEES.....	8
6. CONCLUSION.....	8
ANNEXES.....	9
I. UN SERVICE WEB (API) ET UTILISATION DE LA BIBLIOTHEQUE 'REQUESTS'	10
II. EXTRACTION DE DONNEES DEPUIS UNE PAGE WEB (WEB SCRAPING).....	10
III. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES.....	10
IV. EXTRACTION DE DONNEES DEPUIS UNE BASE DE DONNEES.....	11
V. PROCEDURES STOCKEES	11
VI. LES VUES.....	12
VII. DEVELOPPER DES REGLES D'AGREGATION DE DONNEES	13
VIII. API REST	14
IX. REQUÊTES SQL AVEC PROCÉDURES STOCKÉES	15
X. CALCUL DE L'ÉCART-TYPE POUR LES CHEVRES COUCHEES.....	16
XI. CONSOLIDATION DES RESULTATS AVEC UNE PROCEDURE STOCKEE	18
XII. VUE D'AGREGATION DE DONNEES SUR LA DERNIERE JOURNEE.....	19
XIII. UTILISATION DE LA BIBLIOTHEQUE 'REQUESTS' POUR INTERROGER UNE API ET RECUPERER LES DONNEES AU FORMAT JSON.....	20
XIV. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES.....	21
XV. EXTRACTION DE DONNEES DEPUIS UN SYSTEME BIG DATA	22

1. INTRODUCTION

1.1. CONTEXTE DU PROJET

Le projet ANIMOV vise à surveiller et analyser le comportement des chèvres à travers l'automatisation de l'extraction, du traitement et de l'analyse des données provenant de diverses sources.

1. AUTOMATISER L'EXTRACTION DE DONNEES

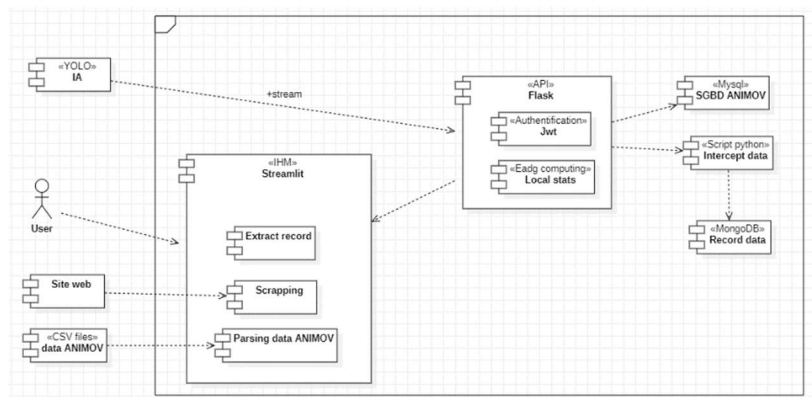
1.2. OBJECTIF DE LA COMPETENCE

Automatiser l'extraction de données depuis divers types de sources.

1.3. ARCHITECTURE

Le projet ANIMOV est conçu comme une plateforme intégrée permettant la collecte, l'analyse et la visualisation de données sur le comportement des chèvres. L'architecture utilise une combinaison de bases de données relationnelles

(SGBD MySQL), NoSQL (MongoDB), d'une API Flask pour la gestion des données, et d'une application Streamlit pour les interactions avec les utilisateurs. L'application utilisateur Streamlit intègre plusieurs outils pour extraire, scraper, et analyser les données.



Ce schéma présente l'architecture du projet ANIMOV, en illustrant les interactions entre les différents composants utilisés pour la collecte, le traitement et l'analyse des données de surveillance comportementale des chèvres. Voici une explication détaillée de chaque composant et des flux de données :

1.3.1. Utilisateur :

L'utilisateur interagit avec l'interface graphique du projet, Streamlit.

1.3.2. Streamlit :

Le module Streamlit dans le projet ANIMOV offre plusieurs fonctionnalités essentielles : il permet aux utilisateurs d'extraire des enregistrements spécifiques depuis des bases de données, des fichiers CSV, par scrapping de site web. Streamlit interagit étroitement avec l'API Flask pour faciliter la récupération et l'envoi des données.

1.3.3. API Flask :

L'API Flask joue un rôle central dans le projet ANIMOV en assurant l'authentification sécurisée des utilisateurs à l'aide de JSON Web Tokens (JWT), garantissant la protection des échanges de données entre le système et

l'utilisateur via Streamlit et la base de données Mysql. Elle intègre également des capacités d'Edge Computing pour effectuer des calculs locaux ou des analyses rapides sur les données avant de les envoyer vers les bases de données.

1.3.4. ANIMOV (MySQL) :

Le SGBD ANIMOV (MySQL) est responsable du stockage des données structurées liées aux chèvres, y compris les enregistrements horodatés de leurs comportements. Ce système de gestion de bases de données interagit étroitement avec l'API Flask pour fournir ou enregistrer ces données.

1.3.5. MongoDB (Record Data) :

MongoDB est utilisée pour stocker des données non structurées, telles que des images ou des enregistrements au format JSON, provenant de la surveillance vidéo.

1.3.6. YOLO (IA) :

YOLO (You Only Look Once) est un modèle d'intelligence artificielle dédié à la détection d'objets en temps réel. Dans le cadre du projet ANIMOV, il est utilisé pour analyser les flux vidéo ou les images des chèvres, permettant de détecter et d'identifier leurs comportements spécifiques. Les résultats de ces analyses sont ensuite envoyés à Flask via un flux de données.

1.3.7. Site Web :

Le bloc "Site Web" représente un site web qui sert de source pour récupérer des informations externes.

1.3.8. CSV/Files Data ANIMOV :

Les fichiers CSV/Files Data ANIMOV contiennent des données brutes collectées, qui sont ensuite traitées et analysées par le module de parsing dans Streamlit.

1.3.9. Intercept Data :

Ce script d'interception de données, nommé "Intercept Data", est un démon qui interroge les données courantes du flux en continu arrivant à l'API Flask pour récupérer quatre images et les détections associées par heure.

1.4. UN SERVICE WEB (API) ET UTILISATION DE LA BIBLIOTHEQUE `REQUESTS`

Pour interroger une API et récupérer les données au format JSON, nous avons utilisé la bibliothèque ``requests``.

- Exemple de code : Annexe I

1.5. EXTRACTION DE DONNEES DEPUIS UNE PAGE WEB (WEB SCRAPING)

Utilisation de ``BeautifulSoup`` pour extraire des informations spécifiques d'une page web.

- Exemple de code : Annexe II

1.6. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES CSV

Utilisation des fonction ``read_csv`` et ``concat`` de la librairie Pandas pour l'extraction des fichiers CSV de deux types différents.

- Exemple de code : Annexe III

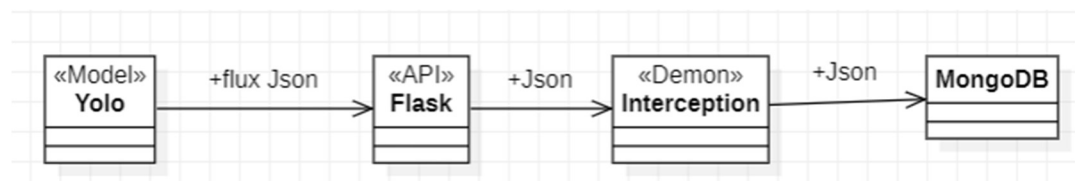
1.7. EXTRACTION DE DONNEES DEPUIS UNE BASE DE DONNEES

Utilisation de SQLAlchemy et pandas pour extraire des données de la base de données MySQL.

- Exemple de code : Annexe VI

1.8. EXTRACTION DE DONNEES DEPUIS UN SYSTEME BIGDATA

Le diagramme représente l'architecture du système dans laquelle le modèle de détection d'objets YOLO génère des données au format JSON après avoir identifié et localisé des animaux dans des images de vidéos. Ces données, images et détection, sont captées par l'API Flask. Un démon nommé "Interception", qui s'exécute toutes les 15 minutes, interroge l'API Flask pour récupérer les images et les détections YOLO. Après avoir intercepté ces données, le démon les transmet tel quel à une base de données MongoDB pour y être stockées. Ce processus assure un flux régulier de données, depuis la détection initiale par YOLO jusqu'au stockage final dans MongoDB.



2. BASE DE DONNEES MYSQL

2.1. LES TABLES.

table_chevres_minute	table_chevres_heures	table_chevres_minute_serveur_v2	ResultatsConsolides
id bigint(20)	id bigint(20)	id int(11)	source int(11)
jour datetime	jour datetime	timestamp bigint(20)	moyenne_total decimal(10,6)
source bigint(20)	source bigint(20)	source int(11)	max_total decimal(10,0)
minutes bigint(20)	heure bigint(20)	total int(11)	min_total decimal(10,0)
heure bigint(20)	brush double	couche int(11)	ecart_type_total decimal(10,6)
brush double	drink double	debout int(11)	moyenne_debout decimal(10,6)
drink double	eat double	max_total int(11)	max_debout decimal(10,0)
eat double	class_0 double	max_couche int(11)	min_debout decimal(10,0)
class_0 double	class_1 double	max_debout int(11)	ecart_type_debout decimal(10,6)
class_1 double		min_total int(11)	moyenne_couche decimal(10,6)
		min_couche int(11)	max_couche decimal(10,0)
		min_debout int(11)	min_couche decimal(10,0)
		std_total decimal(6,3)	ecart_type_couche decimal(10,6)
		std_couche decimal(6,3)	
		std_debout decimal(6,3)	
		Q1_total decimal(6,3)	
		Q1_couche decimal(6,3)	
		Q1_debout decimal(6,3)	
		Q2_total decimal(6,3)	
		Q2_couche decimal(6,3)	
		Q2_debout decimal(6,3)	
		Q3_total decimal(6,3)	
		Q3_couche decimal(6,3)	
		Q3_debout decimal(6,3)	
		mode_total text	
		mode_couche text	
		mode_debout text	
		nb_frames int(11)	

Le diagramme ER représente les différentes tables de la base de données "ANIMOV", utilisées pour gérer et analyser les données d'observation des chèvres à différentes échelles temporelles. La table `ANIMOV.table_chevres_minute` et `ANIMOV.table_chevres_heures` contiennent des enregistrements d'activités des chèvres, collectées respectivement à la minute et à l'heure.

La table `ANIMOV.table_chevres_minute_serveur_v2` joue un rôle clé en stockant les statistiques agrégées calculées à partir des données brutes par edge computing. Pour chaque minute, des statistiques telles que la moyenne, l'écart-type, les quartiles, et les modes des chèvres en position couchée, debout et le total sont calculées puis insérées dans cette table. Cette agrégation permet de résumer le comportement des chèvres sur des intervalles de temps courts et fournit une base pour des analyses plus approfondies.

Enfin, la table `ANIMOV.ResultatsConsolides` est utilisée pour stocker les résultats consolidés via une procédure stockée (`ConsoliderResultatsEcartType()`), qui synthétise et consolide les statistiques à une échelle temporelle plus large, comme les heures ou les jours. Ces résultats consolidés sont ensuite récupérés par le script Python pour des analyses supplémentaires ou pour être affichés dans des rapports ou des interfaces utilisateur.

2.2. CONFIGURATION DE LA CONNEXION A LA BASE DE DONNEES

Voir section 1.7

2.3. PROCEDURES STOCKEES

Une procédure stockée est un ensemble de commandes SQL précompilées et stockées dans la base de données, offrant des avantages significatifs en termes de performance, de sécurité et de maintenance. Étant donné qu'elles sont précompilées, les procédures stockées s'exécutent plus rapidement que les requêtes SQL envoyées individuellement, tout en réduisant le trafic réseau. Elles permettent également de centraliser et de réutiliser la logique métier, tout en limitant l'accès direct aux tables de la base de données, ce qui renforce la sécurité et réduit les risques d'erreurs ou d'injections SQL. De plus, elles simplifient la maintenance, car toute modification peut être effectuée directement dans la procédure sans affecter l'application cliente.

- Exemple de procédure stockée : Annexe V

2.4. LES VUES

ANIMOV.vue_chevres_serier_heure	ANIMOV.vue_chevres_serier_jour	ANIMOV.vue_chevre_derniere_heure_last	ANIMOV.vue_chevres_derniere_minute	ANIMOV.vue_chevre_derniere_jour_last
o id int(11)	A timestamp varchar(18)	o id int(11)	o timestamp bigint(11)	A timestamp varchar(21)
o timestamp bigint(20)	o source int(11)	o timestamp bigint(20)	o source int(11)	o source int(11)
o source int(11)	o total decimal(14,4)	o source int(11)	o total decimal(36,4)	o total decimal(35,3)
o total int(11)	o couche decimal(14,4)	o total int(11)	o couche decimal(36,4)	o couche decimal(35,3)
o couche int(11)	o debout decimal(14,4)	o couche int(11)	o debout decimal(36,4)	o debout decimal(35,3)
o debout int(11)	o max_total int(11)	o debout int(11)	o std_total double	o max_total int(11)
o max_total int(11)	o max_couche int(11)	o max_total int(11)	o std_couche double	o max_couche int(11)
o max_couche int(11)	o max_debout int(11)	o max_couche int(11)	o std_debout double	o max_debout int(11)
o max_debout int(11)	o min_total int(11)	o max_debout int(11)	o max_total decimal(11,0)	o min_total int(11)
o min_total int(11)	o min_couche int(11)	o min_total int(11)	o max_couche decimal(11,0)	o min_couche int(11)
o min_couche int(11)	o min_debout int(11)	o min_couche int(11)	o max_debout decimal(11,0)	o min_debout int(11)
o min_debout int(11)	o nb_frames decimal(14,4)	o min_debout int(11)	o min_total decimal(11,0)	o nb_frames decimal(32,0)
o std_total decimal(6,3)		o std_total decimal(6,3)	o min_couche decimal(11,0)	
o std_couche decimal(6,3)		o std_couche decimal(6,3)	o min_debout decimal(11,0)	
o std_debout decimal(6,3)		o std_debout decimal(6,3)		
o Q1_total decimal(6,3)		o Q1_total decimal(6,3)		
o Q1_couche decimal(6,3)		o Q1_couche decimal(6,3)		
o Q1_debout decimal(6,3)		o Q1_debout decimal(6,3)		
o Q2_total decimal(6,3)		o Q2_total decimal(6,3)		
o Q2_couche decimal(6,3)		o Q2_couche decimal(6,3)		
o Q2_debout decimal(6,3)		o Q2_debout decimal(6,3)		
o Q3_total decimal(6,3)		o Q3_total decimal(6,3)		
o Q3_couche decimal(6,3)		o Q3_couche decimal(6,3)		
o Q3_debout decimal(6,3)		o Q3_debout decimal(6,3)		
A mode_total text		A mode_total text		
A mode_couche text		A mode_couche text		
A mode_debout text		A mode_debout text		
o nb_frames int(11)		o nb_frames int(11)		

Les vues dans un SGBD simplifient l'accès aux données en masquant la complexité des requêtes SQL. Elles offrent une sécurité renforcée en limitant l'accès aux informations sensibles, permettent une meilleure maintenabilité en centralisant des requêtes réutilisables, et isolent les utilisateurs des détails techniques de la structure de la base de données.

Le diagramme présente les vues au sein de la base de données Mysql, elles sont toutes liées à la surveillance et au suivi des comportements des chèvres. Chaque vue est conçue pour agréger les données sur une échelle de temps spécifique (minutes, horaire, journalier), ce qui permet une analyse des activités des chèvres à différentes résolutions temporelles.

Toutes les vues partagent un ensemble de colonnes communes, telles que `total`, `couche`, `debout`, ainsi que des statistiques comme les valeurs maximales (`max_total`), minimales (`min_total`), et l'écart type (`std_total`). Ces colonnes sont utilisées pour capturer des mesures clés, telles que le nombre total de chèvres, le nombre de chèvres couchées ou debout, et les variations de ces mesures. L'inclusion de statistiques avancées, telles que les quartiles (Q1, Q2, Q3) et les modes (`mode_total`, `mode_couche`, `mode_debout`), permet une analyse plus fine de la répartition et des variations de ces comportements.

La structure des vues est très similaire d'une période à l'autre, car les mêmes types d'analyses sont appliqués indépendamment de la granularité temporelle. Cette uniformité dans la conception des vues permet une approche cohérente pour l'analyse des données, permettant ainsi une comparaison facile entre différentes périodes.

Enfin, la présence de la colonne `nb_frames` dans chaque vue indique le nombre de trames (ou images) analysées pour calculer les statistiques agrégées.

- Exemple de code : Annexe VI

2.5. REQUETES SQL POUR L'EXTRACTION DE DONNEES

Exemples de requêtes SQL pour extraire des données.

- Extraction des Sources de Données :

```
SELECT source
FROM {database}.table_chevres_heures
GROUP BY source
```

3. DEVELOPPER DES REGLES D'AGREGATION DE DONNEES

L'agrégation de données est un processus consistant à regrouper et résumer plusieurs valeurs ou enregistrements en un seul ensemble représentatif. Cela permet de condenser des données brutes en informations plus synthétiques, facilitant ainsi l'analyse.

- Exemple de code : Annexe VII

4. DEVELOPPER UNE API REST

4.1. OBJECTIF DE LA COMPETENCE

L'API REST permet un accès structuré et sécurisé aux données stockées dans le système ANIMOV. Cette API servira de point d'accès centralisé pour interagir avec les différentes bases de données du projet, facilitant ainsi la récupération, la mise à jour, et l'analyse des données par des utilisateurs ou systèmes externes.

4.2. DOCUMENTATION DE L'APPLICATION

Pour faciliter la documentation et l'interaction avec l'API REST développée pour le projet ANIMOV, nous utilisons Swagger, un outil puissant qui permet de générer et de visualiser la documentation API de manière interactive.

5. RGPD ET AUTOMATISATION DE L'EXTRACTION DE DONNEES

Bien que le RGPD impose des obligations strictes en matière de protection des données personnelles, il est important de noter que ces réglementations s'appliquent exclusivement aux données concernant les personnes physiques. Dans le cadre de ce projet, qui concerne exclusivement la surveillance vidéo automatisée de l'enclos de chèvres, le RGPD ne s'applique pas, car il s'agit d'animaux et non de personnes. Par conséquent, les obligations relatives à la protection des données personnelles, telles que l'anonymisation ou le floutage des visages, ne sont pas requises dans ce contexte spécifique. Toutefois, il reste essentiel de maintenir de bonnes pratiques en matière de sécurité des données pour protéger toute autre information sensible qui pourrait être collectée ou traitée.

6. CONCLUSION

Les compétences en extraction, manipulation, stockage et exposition des données ont été démontrées avec succès à travers les différentes tâches réalisées. Ce rapport atteste de la capacité à automatiser les processus de gestion des données et à développer des solutions efficaces et conformes aux standards professionnels.

ANNEXES

I. UN SERVICE WEB (API) ET UTILISATION DE LA BIBLIOTHEQUE `REQUESTS`

- Exemple de code :

```
import requests

def query_get_sources(self):
    url = f'http://{END_POINT}/sources'
    response = requests.get(url)
    try:
        response.raise_for_status()
        data = response.json()
        df = pd.DataFrame(data)
        return df
    except requests.exceptions.HTTPError as http_err:
        print(f"Erreur HTTP : {http_err} URL : {url}")
    except requests.exceptions.RequestException as err:
        print(f"Erreur de requête : {err} URL : {url}")
    except requests.exceptions.JSONDecodeError as json_err:
        print(f"Erreur de décodage JSON : {json_err} Réponse : {response.text}")
```

II. EXTRACTION DE DONNEES DEPUIS UNE PAGE WEB (WEB SCRAPING)

- Exemple de code :

```
import requests
from bs4 import BeautifulSoup

# Envoyer une requête HTTP à l'URL
response = requests.get(url)

# Analyser le contenu HTML de la page avec BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Rechercher tous les éléments avec la classe 'sta5'
elements_sta5 = soup.find_all(class_='sta5')
# Extraire les informations nécessaires de chaque élément et les ajouter à la liste
for element in elements_sta5:
    product_name = element.find('a').text.strip()
    product_price = element.find('strong').text.strip()
    product_variation = element.find_all('td')[0].text.strip()

# Convertir le prix en valeur numérique si possible, sinon ignorer cet élément
product_price = float(product_price.replace(',', '.'))

# Ajouter un dictionnaire pour chaque produit dans la liste
data.append({
    "Produit": product_name,
    "Prix": product_price,
    "Variation": product_variation
})
```

III. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES

- Exemple de code :

```
import glob

# Fonction pour le traitement et le transfert des données
def data_traitement(chemin_dossier, base_de_donnees_url, source):
    engine = create_engine(base_de_donnees_url)
    resultats = {}

    # Pour les types de fichiers -1.csv et -2.csv
    for type_fichier in ['-1.csv', '-2.csv']:
        fichiers_csv = glob.glob(os.path.join(chemin_dossier, f'*{type_fichier}'))
        dataframes = [pd.read_csv(fichier) for fichier in fichiers_csv]
        if dataframes:
            df_final = pd.concat(dataframes, ignore_index=True)
            df_final["Source"] = source
```

```

# Nom de la table en fonction du type de fichier
table_name = 'final' if '-1' in type_fichier else 'final_2'
df_final.to_sql(table_name, engine, if_exists='append', index=False)
resultats[type_fichier] = df_final
else:
    resultats[type_fichier] = pd.DataFrame()

return resultats

```

IV. EXTRACTION DE DONNEES DEPUIS UNE BASE DE DONNEES

- Exemple de code :

```

from sqlalchemy import create_engine

class DatabaseExtractor:
    def __init__(self):
        # Lors de l'initialisation de l'objet, une instance de l'engine SQLAlchemy est créée
        # en utilisant la méthode connection pour récupérer l'URL de connexion.
        self.engine = sqlalchemy.create_engine(self.connection())

    def connection(self):
        # Cette méthode lit le fichier config.yaml pour obtenir les informations de configuration de la base de données.
        with open('config.yaml', 'r') as file:
            config = yaml.safe_load(file)

        # Les informations de la base de données sont extraites du fichier de configuration.
        db_info = config['database']

        # L'URL de connexion est créée à l'aide des informations récupérées.
        connection_url = URL.create(
            drivename=f"{db_info['type']}+{db_info['driver']}",
            username=db_info['username'],
            password=db_info['password'],
            host=db_info['host'],
            port=db_info['port'],
            database=db_info['database_name']
        )
        return connection_url

    def query_get_sources(self):
        # Cette méthode crée une requête SQL pour récupérer les sources dans la table 'table_chevres_heures',
        # en regroupant les résultats par source.
        sql_query = f"SELECT source FROM {db_info['database_name']}.table_chevres_heures GROUP BY source"

        # La requête est exécutée en utilisant une connexion à la base de données,
        # et le résultat est stocké dans un DataFrame Pandas.
        with self.engine.connect() as connection:
            df = pd.read_sql(sql_query, connection)

        return df

```

V. PROCEDURES STOCKEES

La procédure stockée supprime d'abord toute table temporaire existante pour éviter les conflits, puis en crée une nouvelle pour stocker les statistiques calculées (moyenne, maximum, minimum et écart-type) des écart-type du taux de chèvres couchées. Les statistiques sont calculées pour chaque source de données en utilisant des sous-requêtes qui agrègent les données de la dernière heure. Les résultats sont ensuite insérés dans la table temporaire, permettant d'analyser les variations de chèvres couchés par source pas heure.

- Exemple de procédure stockée :

```

CREATE DEFINER=`animov`@`%` PROCEDURE `ANIMOV`.`CalculerEcartTypeHeureCouche`()
BEGIN
    -- Suppression de la table temporaire si elle existe déjà pour éviter les conflits
    DROP TEMPORARY TABLE IF EXISTS ANIMOV.ResultatsEcartTypeCouche;

    -- Création d'une nouvelle table temporaire pour stocker les résultats
    CREATE TEMPORARY TABLE ANIMOV.ResultatsEcartTypeCouche (

```

```

source INT, -- Source de données
moyenne_couche DECIMAL(10, 6), -- Moyenne des heures de coucher
max_couche DECIMAL(10, 0), -- Heure de coucher maximale
min_couche DECIMAL(10, 0), -- Heure de coucher minimale
ecart_type_couche DECIMAL(10, 6) -- Écart-type des heures de coucher
);

-- Insertion des données calculées dans la table temporaire
INSERT INTO ANIMOV.ResultatsEcartTypeCouche (source, moyenne_couche, max_couche, min_couche, ecart_type_couche)
SELECT
    t.source, -- Source de données
    MIN(t.moyenne_heure) as moyenne_couche, -- Moyenne des heures de coucher par source
    MIN(t.max_heure) as max_couche, -- Heure de coucher maximale par source
    MIN(t.min_heure) as min_couche, -- Heure de coucher minimale par source
    -- Calcul de l'écart-type en utilisant la formule statistique adaptée à un échantillon
    ROUND(SQRT(
        (SUM((t.N - 1) * t.sigma * t.sigma) + SUM(t.N * t.termes_ecart_moyenne)) / (SUM(t.N) - COUNT(*))
    ), 3) AS ecart_type_couche
FROM (
    -- Sous-requête pour obtenir les statistiques nécessaires pour chaque source
    SELECT
        a.source, -- Source de données
        m.moyenne_globale as moyenne_heure, -- Moyenne globale des heures de coucher
        m.max_global as max_heure, -- Heure maximale globale de coucher
        m.min_global as min_heure, -- Heure minimale globale de coucher
        a.nb_frames AS N, -- Nombre de frames, représentant le nombre d'observations
        a.std_couche AS sigma, -- Écart-type de l'heure de coucher pour la source
        -- Calcul du terme d'écart par rapport à la moyenne globale pour chaque source
        POW((a.couche / NULLIF(a.nb_frames, 0)) - m.moyenne_globale, 2) AS termes_ecart_moyenne
    FROM
        ANIMOV.table_chevres_minute_serveur_v2 a
    INNER JOIN (
        -- Sous-requête pour calculer les moyennes globales, maximum et minimum par source
        SELECT
            source,
            AVG(couche / NULLIF(nb_frames, 0)) AS moyenne_globale, -- Moyenne globale des heures de coucher
            max(couche / NULLIF(nb_frames, 0)) AS max_global, -- Maximum global des heures de coucher
            min(couche / NULLIF(nb_frames, 0)) AS min_global -- Minimum global des heures de coucher
        FROM
            table_chevres_minute_serveur_v2
    ) m ON a.source = m.source
) as t
WHERE
    `timestamp` >= UNIX_TIMESTAMP(NOW()) - INTERVAL 1 hour) -- Filtrer les données pour la dernière heure
GROUP BY
    source -- Grouper par source
) as m ON a.source = m.source -- Joindre les données calculées avec les données sources
WHERE
    a.`timestamp` >= UNIX_TIMESTAMP(NOW()) - INTERVAL 1 hour) -- Filtrer les données pour la dernière heure
) as t
GROUP BY
    t.source; -- Grouper les résultats finaux par source
END

```

VI. LES VUES

- **Vue de surveillance des statistiques de chèvres par minute :**

```

CREATE OR REPLACE
ALGORITHM = UNDEFINED VIEW `vue_chevres_derniere_minute_v2` AS
select
    *
from
    `table_chevres_minute_serveur_v2`
where
    (`table_chevres_minute_serveur_v2`.`timestamp` >= unix_timestamp((now() - interval 2 minute)))
order by
    `table_chevres_minute_serveur_v2`.`timestamp`,
    `table_chevres_minute_serveur_v2`.`source` desc
limit 4;

```

- **Vue d'agrégation de données sur la dernière journée :**

```
CREATE OR REPLACE
ALGORITHM = UNDEFINED VIEW `vue_chevre_derniere_jour_last` AS
-- Création ou remplacement d'une vue nommée `vue_chevre_derniere_jour_last`
-- La vue agrège les données de la table `table_chevres_minute_serveur_v2` sur les 24 dernières heures (1 jour)

SELECT
-- Formate le timestamp en segment de 5 minutes
date_format(from_unixtime(floor((unix_timestamp(from_unixtime('t1'.`timestamp`)) / 300)) * 300)), '%Y-%m-%d %H:%i') AS
`timestamp`,
`t1`.`source` AS `source`, -- Identifiant de la source
round(sum(`t1`.`total`), 3) AS `total`, -- Somme totale des valeurs `total`, arrondie à 3 décimales
round(sum(`t1`.`couche`), 3) AS `couche`, -- Somme des valeurs `couche`, arrondie à 3 décimales
round(sum(`t1`.`debout`), 3) AS `debout`, -- Somme des valeurs `debout`, arrondie à 3 décimales
max(`t1`.`max_total`) AS `max_total`, -- Valeur maximale de `max_total`
max(`t1`.`max_couche`) AS `max_couche`, -- Valeur maximale de `max_couche`
max(`t1`.`max_debout`) AS `max_debout`, -- Valeur maximale de `max_debout`
min(`t1`.`min_total`) AS `min_total`, -- Valeur minimale de `min_total`
min(`t1`.`min_couche`) AS `min_couche`, -- Valeur minimale de `min_couche`
min(`t1`.`min_debout`) AS `min_debout`, -- Valeur minimale de `min_debout`
sum(`t1`.`nb_frames`) AS `nb_frames` -- Somme du nombre de frames
FROM
(`ANIMOV`.`table_chevres_minute_serveur_v2` `t1`
JOIN (
-- Sous-requête pour obtenir le dernier timestamp par source sur 24 heures
SELECT
`ANIMOV`.`table_chevres_minute_serveur_v2`.`source` AS `source`,
max(`ANIMOV`.`table_chevres_minute_serveur_v2`.`timestamp`) AS `max_timestamp`
FROM
`ANIMOV`.`table_chevres_minute_serveur_v2`
GROUP BY
`ANIMOV`.`table_chevres_minute_serveur_v2`.`source`
) `t2`
ON ((`t1`.`source` = `t2`.`source`) -- Jointure sur la source
AND (`t1`.`timestamp` > (`t2`.`max_timestamp` - (86400 * 1)))) -- Filtrage sur les 24 dernières heures
GROUP BY
`t1`.`source`, -- Agrégation par source
date_format(from_unixtime(floor((unix_timestamp(from_unixtime('t1'.`timestamp`)) / 300)) * 300)), '%Y-%m-%d %H:%i'); --
Agrégation par segment de 5 minutes
```

Cette vue agrège les données par source et par intervalles de 5 minutes, en calculant les sommes, les minimums, les maximums, et les totaux sur les 24 dernières heures. Une sous-requête détermine le dernier timestamp pour chaque source, permettant de filtrer les enregistrements pertinents sur cette période. Les données sont ensuite regroupées par source et par tranche de 5 minutes, ce qui permet de fournir des résumés précis sur les périodes spécifiées.

VII. DEVELOPPER DES REGLES D'AGREGATION DE DONNEES

- **Exemple de code :**

```
def load_clean_data(df):
    # Convertit la colonne 'date' en format datetime, en remplaçant les erreurs par NaT
    df['date'] = pd.to_datetime(df['date'], errors='coerce')
    # Supprime les lignes où la conversion de 'date' a échoué (c'est-à-dire NaT)
    df = df.dropna(subset=['date'])
    # Crée une nouvelle colonne 'jour' contenant uniquement la date (sans l'heure)
    df['jour'] = df['date'].dt.date
    # Crée une colonne 'quart_heure' en arrondissant les heures au quart d'heure le plus proche
    df['quart_heure'] = df['date'].dt.floor('15T').dt.time
    # Groupe les données par 'jour' et 'quart_heure'
    grouped = df.groupby(['jour', 'quart_heure'])
    # Calcule la médiane des colonnes 'Effectif couche' et 'Effectif debout' pour chaque groupe
    df_aggregated = grouped[['Effectif couche', 'Effectif debout']].median().reset_index()
    # Crée une nouvelle colonne 'ratio_debout' qui représente le pourcentage de l'effectif debout par rapport au total
    df_aggregated['ratio_debout'] = (df_aggregated['Effectif debout'] / (df_aggregated['Effectif debout'] + df_aggregated['Effectif
couche'])) * 100
    # Combine les colonnes 'jour' et 'quart_heure' pour créer une nouvelle colonne datetime
```

```
df_aggregated['datetime'] = pd.to_datetime(df_aggregated['jour'].astype(str) + ' ' + df_aggregated['quart_heure'].astype(str))  
# Retourne le DataFrame agrégé et nettoyé  
return df_aggregated
```

La fonction de nettoyage et d'agrégation de données ci-dessus permet de structurer des données en les regroupant par tranches de 15 minutes. Elle commence par convertir la colonne 'date' en un format datetime, tout en gérant les erreurs de conversion. Les lignes pour lesquelles la conversion échoue sont ensuite supprimées pour garantir l'intégrité des données. La fonction crée ensuite une colonne 'jour' pour isoler la date sans l'heure, et une autre colonne 'quart_heure' qui arrondit les heures au quart d'heure le plus proche. Les données sont ensuite groupées par 'jour' et 'quart_heure', et la médiane des effectifs couchés et debout est calculée pour chaque groupe. Un ratio est également calculé pour indiquer la proportion de l'effectif debout par rapport au total. Enfin, une colonne datetime combinant 'jour' et 'quart_heure' est créée pour une meilleure traçabilité temporelle. Le résultat est un DataFrame propre et agrégé, prêt pour une analyse plus approfondie.

VIII. API REST

- **/receive_data_animov (POST)** : Ce endpoint reçoit des données au format JSON, permettant d'ajouter ou de mettre à jour les enregistrements dans la base de données ANIMOV.
- **/get_data_animov_ch (GET)** : Ce endpoint permet de récupérer des données filtrées selon certains critères spécifiques, facilitant ainsi l'accès aux informations pertinentes.
- **/get_data_animov_ch_minutes (GET)** : Utilisé pour récupérer des données agrégées ou enregistrées par minute, ce qui est crucial pour des analyses fines et en temps réel.
- **/get_data_animov (GET)** : Endpoint général pour récupérer l'ensemble des données disponibles dans le système, offrant une vue globale des informations stockées.
- **/chevres_heures (GET)** : Accède aux données agrégées par heure, permettant des analyses temporelles sur le comportement des chèvres.
- **/chevres_minutes (GET)** : Similaire à /chevres_heures, mais pour des données agrégées par minute, offrant une granularité plus fine.
- **/sources (GET)** : Ce endpoint retourne la liste des sources de données disponibles, ce qui aide à identifier l'origine des données collectées.
- **/dates (GET)** : Permet d'obtenir la liste des dates pour lesquelles des données sont disponibles, facilitant la navigation dans les archives du système.
- **/stats_minute (GET)** : Récupère des statistiques calculées pour chaque minute, utile pour une analyse détaillée et en temps réel.
- **/stats_heure (GET)** : Récupère des statistiques calculées pour chaque heure, permettant une vision plus agrégée des comportements.
- **/get_serie_heure (GET)** : Fournit des séries temporelles agrégées pour la dernière heure, offrant une analyse immédiate des dernières données collectées.
- **/get_serie_jour (GET)** : Fournit des séries temporelles agrégées pour la dernière journée, utile pour des récapitulatifs quotidiens.
- **/get_serie_last_heure (GET)** : Donne accès aux données de la dernière heure enregistrée, permettant un suivi en temps réel.

- **/get_serie_last_jour (GET)** : Ce endpoint permet d'accéder aux données du dernier jour enregistré, offrant ainsi un aperçu complet des dernières 24 heures de surveillance.

IX. REQUÊTES SQL AVEC PROCÉDURES STOCKÉES

L'utilisation de procédures stockées permet d'exécuter des traitements complexes directement dans la base de données, améliorant ainsi l'efficacité et la réutilisabilité du code. Voici un exemple de requête utilisant une procédure stockée pour consolider des résultats.

```
def query_get_stats_heure(self):
    # Appel de la procédure stockée
    sql_query_procedure = text("CALL `ANIMOV`.`ConsoliderResultatsEcartType`();")
    try:
        # Utilisation de la connexion pour exécuter directement la procédure
        with self.engine.begin() as conn:
            conn.execute(sql_query_procedure)
    except Exception as e:
        print(f"Erreur lors de l'exécution de la procédure stockée: {e}")
        return pd.DataFrame() # Retourner un DataFrame vide en cas d'erreur

    # Récupération des résultats dans un DataFrame
    sql_query_results = "SELECT * FROM `ANIMOV`.`ResultatsConsolides`;"
    df = pd.read_sql(sql_query_results, self.engine)

    return df
```

1. Appel de la procédure stockée :

```
sql_query_procedure = text("CALL `ANIMOV`.`ConsoliderResultatsEcartType`();")
```

Cette ligne prépare une requête SQL pour appeler la procédure stockée `ConsoliderResultatsEcartType` de la base de données `ANIMOV`.

2. Exécution de la procédure stockée :

```
try:
    # Utilisation de la connexion pour exécuter directement la procédure
    with self.engine.begin() as conn:
        conn.execute(sql_query_procedure)
except Exception as e:
    print(f"Erreur lors de l'exécution de la procédure stockée: {e}")
    return pd.DataFrame() # Retourner un DataFrame vide en cas d'erreur
```

Cette section utilise un bloc try-except pour gérer les exceptions éventuelles lors de l'exécution de la procédure stockée. La méthode `self.engine.begin()` initialise une connexion à la base de données. Si l'exécution échoue, une erreur est imprimée et la fonction retourne un `DataFrame` vide.

3. Récupération des résultats :

```
sql_query_results = "SELECT * FROM 'ANIMOV'.`ResultatsConsolides`;"  
df = pd.read_sql(sql_query_results, self.engine)
```

Une fois la procédure stockée exécutée avec succès, cette partie du code exécute une requête SQL pour sélectionner tous les enregistrements de la table ResultatsConsolides. Les résultats de cette requête sont ensuite chargés dans un DataFrame pandas à l'aide de la fonction pd.read_sql.

4. Retour du DataFrame :

```
return df
```

ANIMOV.ResultatsConsolides	
123 source	int(11)
123 moyenne_total	decimal(10,6)
123 max_total	decimal(10,0)
123 min_total	decimal(10,0)
123 ecart_type_total	decimal(10,6)
123 moyenne_debout	decimal(10,6)
123 max_debout	decimal(10,0)
123 min_debout	decimal(10,0)
123 ecart_type_debout	decimal(10,6)
123 moyenne_couche	decimal(10,6)
123 max_couche	decimal(10,0)
123 min_couche	decimal(10,0)
123 ecart_type_couche	decimal(10,6)

X. CALCUL DE L'ÉCART-TYPE POUR LES CHEVRES COUCHEES

Pour calculer l'écart type d'un regroupement de distribution :

1. Calcul de la moyenne pondérée des distributions

$$\mu_{total} = \frac{\sum_{i=1}^n N_i \cdot \mu_i}{\sum_{i=1}^n N_i}$$

2. Calcul de la variance totale

$$\sigma_{total}^2 = \frac{\sum_{i=1}^n N_i \cdot (\sigma_i^2 + (\mu_i - \mu_{total})^2)}{\sum_{i=1}^n N_i}$$

3. Calcul de l'écart type

$$\sigma_{total} = \sqrt{\sigma_{total}^2}$$

Cette procédure stockée effectue des calculs statistiques complexes pour déterminer l'écart-type des chèvres couchées par source sur une période d'une heure. Les résultats sont organisés et stockés dans une table temporaire, facilitant ainsi leur accès et leur utilisation pour des analyses ultérieures.

```
CREATE DEFINER='animov'@'%' PROCEDURE `ANIMOV`.`CalculerEcartTypeHeureCouche`()  
BEGIN  
  -- Supprimer la table temporaire si elle existe déjà  
  DROP TEMPORARY TABLE IF EXISTS ANIMOV.ResultatsEcartTypeCouche;  
  
  -- Créer une nouvelle table temporaire pour stocker les résultats  
  CREATE TEMPORARY TABLE ANIMOV.ResultatsEcartTypeCouche (  
    source INT,  
    moyenne_couche DECIMAL(10, 6),  
    max_couche DECIMAL(10, 0),  
    min_couche DECIMAL(10, 0),  
    ecart_type_couche DECIMAL(10, 6)  
  );
```

```

-- Calculer et insérer les résultats dans la table temporaire
INSERT INTO ANIMOV.ResultatsEcartTypeCouche (source, moyenne_couche, max_couche, min_couche,
ecart_type_couche)
SELECT
t.source,
MIN(t.moyenne_heure) as moyenne_couche,
MIN(t.max_heure) as max_couche,
MIN(t.min_heure) as min_couche,
ROUND(SQRT(
(SUM((t.N - 1) * t.sigma * t.sigma) + SUM(t.N * t.terme_ecart_moyenne)) / (SUM(t.N) - COUNT(*))
), 3) AS ecart_type_couche
FROM (
SELECT
a.source,
m.moyenne_globale as moyenne_heure,
m.max_global as max_heure,
m.min_global as min_heure,
a.nb_frames AS N,
a.std_couche AS sigma,
POW((a.couche / NULLIF(a.nb_frames, 0)) - m.moyenne_globale, 2) AS terme_ecart_moyenne

FROM
ANIMOV.table_chevres_minute_serveur_v2 a
INNER JOIN (
-- Sous-requête pour calculer la moyenne globale par source
SELECT
source,
AVG(couche / NULLIF(nb_frames, 0)) AS moyenne_globale,
max(couche / NULLIF(nb_frames, 0)) AS max_global,
min(couche / NULLIF(nb_frames, 0)) AS min_global
FROM
table_chevres_minute_serveur_v2
WHERE
`timestamp` >= UNIX_TIMESTAMP(NOW()) - INTERVAL 1 hour)
GROUP BY
source
) as m ON a.source = m.source
WHERE
a.`timestamp` >= UNIX_TIMESTAMP(NOW()) - INTERVAL 1 hour)
) as t
GROUP BY
t.source;
END

```

- **Explication de la Procédure Stockée**

- **`CalculerEcartTypeHeureCouche`**

1. Suppression de la Table Temporaire Existant
2. Création de la Nouvelle Table Temporaire
3. Insertion des Résultats dans la Table Temporaire

- a. Sélection des Données de la Table Source

Les données sont extraites de la table `table_chevres_minute_serveur_v2` en se basant sur les enregistrements de la dernière heure (`WHERE a.timestamp >= UNIX_TIMESTAMP(NOW()) - INTERVAL 1 hour`).

ANIMOV.table_chevres_minute_serveur_v2	
123 id	int(11)
123 timestamp	bigint(20)
123 source	int(11)
123 total	int(11)
123 couche	int(11)
123 debout	int(11)
123 max_total	int(11)
123 max_couche	int(11)
123 max_debout	int(11)
123 min_total	int(11)
123 min_couche	int(11)
123 min_debout	int(11)
123 std_total	decimal(6,3)
123 std_couche	decimal(6,3)
123 std_debout	decimal(6,3)
123 Q1_total	decimal(6,3)
123 Q1_couche	decimal(6,3)
123 Q1_debout	decimal(6,3)
123 Q2_total	decimal(6,3)
123 Q2_couche	decimal(6,3)
123 Q2_debout	decimal(6,3)
123 Q3_total	decimal(6,3)
123 Q3_couche	decimal(6,3)
123 Q3_debout	decimal(6,3)
ABC mode_total	text
ABC mode_couche	text
ABC mode_debout	text
123 nb_frames	int(11)

b. Calcul des Statistiques Globales

Une sous-requête calcule les statistiques globales par source, incluant la moyenne, le maximum et le minimum des heures de coucher pour chaque source sur la dernière heure.

Nom	Type / Taille	Algorithme
PRIMARY KEY	PRIMARY	BTREE
id		
idx_source	KEY	BTREE
source		
idx_timestamp	KEY	BTREE
timestamp		

c. Calcul des Terme d'Écart à la Moyenne

Pour chaque enregistrement, les termes nécessaires au calcul de l'écart-type sont déterminés, notamment le nombre de frames ('nb_frames'), l'écart-type individuel ('std_couche'), et le terme d'écart à la moyenne.

d. Calcul de l'Écart-Type

L'écart-type global est calculé en utilisant la formule adaptée pour la somme des variances pondérées par le nombre d'observations moins un.

e. Insertion dans la Table Temporaire

Les résultats sont ensuite insérés dans la table temporaire 'ResultatsEcartTypeCouche', regroupés par source.

XI. CONSOLIDATION DES RESULTATS AVEC UNE PROCEDURE STOCKEE

```
CREATE DEFINER='animov'@'%' PROCEDURE `ANIMOV`.`ConsoliderResultatsEcartType`()
BEGIN
    -- Appeler les procédures existantes pour calculer les statistiques
    CALL `ANIMOV`.`CalculerEcartTypeHeureTotal`();
    CALL `ANIMOV`.`CalculerEcartTypeHeureDebout`();
    CALL `ANIMOV`.`CalculerEcartTypeHeureCouche`();

    -- Supprimer la table finale si elle existe déjà
    DROP TABLE IF EXISTS `ANIMOV`.`ResultatsConsolides`;

    -- Créer une nouvelle table pour stocker les résultats consolidés
    CREATE TABLE `ANIMOV`.`ResultatsConsolides` AS
    SELECT
        T.source,
        T.moyenne_total, T.max_total, T.min_total, T.ecart_type_total,
        D.moyenne_debout, D.max_debout, D.min_debout, D.ecart_type_debout,
        C.moyenne_couche, C.max_couche, C.min_couche, C.ecart_type_couche
    FROM
        `ANIMOV`.`ResultatsEcartTypeTotal` T
    JOIN
        `ANIMOV`.`ResultatsEcartTypeDebout` D ON T.source = D.source
    JOIN
        `ANIMOV`.`ResultatsEcartTypeCouche` C ON D.source = C.source;
END
```

1. Appel des Procédures de Calcul de Statistiques

2. Suppression de la Table Finale Existant

3. Création de la Nouvelle Table pour les Résultats Consolidés

a. Création de la Table ResultatsConsolides

b. Sélection et Jointure des Tables Temporaires

Les données sont sélectionnées et combinées à partir des tables temporaires ResultatsEcartTypeTotal, ResultatsEcartTypeDebout, et ResultatsEcartTypeCouche. Les jointures sont effectuées sur la colonne source pour rassembler les statistiques correspondantes de chaque source.

XII. VUE D'AGREGATION DE DONNEES SUR LA DERNIERE JOURNEE.

```
CREATE OR REPLACE
ALGORITHM = UNDEFINED VIEW `vue_chevre_derniere_jour_last` AS
select
  date_format(from_unixtime((floor((unix_timestamp(from_unixtime('t1'.`timestamp`)) / 300)) * 300)), '%Y-%m-%d %H:%i')
  AS `timestamp`,
  `t1`.`source` AS `source`,
  round(sum('t1`.`total`), 3) AS `total`,
  round(sum('t1`.`couche`), 3) AS `couche`,
  round(sum('t1`.`debout`), 3) AS `debout`,
  max('t1`.`max_total`) AS `max_total`,
  max('t1`.`max_couche`) AS `max_couche`,
  max('t1`.`max_debout`) AS `max_debout`,
  min('t1`.`min_total`) AS `min_total`,
  min('t1`.`min_couche`) AS `min_couche`,
  min('t1`.`min_debout`) AS `min_debout`,
  sum('t1`.`nb_frames`) AS `nb_frames`
from
  (`ANIMOV`.`table_chevres_minute_serveur_v2` `t1`
join (
  select
    `ANIMOV`.`table_chevres_minute_serveur_v2`.`source` AS `source`,
    max(`ANIMOV`.`table_chevres_minute_serveur_v2`.`timestamp`) AS `max_timestamp`
  from
    `ANIMOV`.`table_chevres_minute_serveur_v2`
  group by
    `ANIMOV`.`table_chevres_minute_serveur_v2`.`source`) `t2` on
  (((`t1`.`source` = `t2`.`source`)
  and (`t1`.`timestamp` > (`t2`.`max_timestamp` - (86400 * 1))))))
group by
  `t1`.`source`,
  date_format(from_unixtime((floor((unix_timestamp(from_unixtime('t1'.`timestamp`)) / 300)) * 300)), '%Y-%m-%d %H:%i');
```

1. Création de la vue :

2. Sélection des colonnes et agrégation :

Cette section spécifie les colonnes sélectionnées et les fonctions d'agrégation utilisées. Les données sont arrondies à trois décimales et agrégées pour obtenir les sommes, les maximums, les minimums et les nombres de frames.

3. Jointure interne :

```
from
  (`ANIMOV`.`table_chevres_minute_serveur_v2` `t1`
join (
  select
    `ANIMOV`.`table_chevres_minute_serveur_v2`.`source` AS `source`,
    max(`ANIMOV`.`table_chevres_minute_serveur_v2`.`timestamp`) AS
  `max_timestamp`
  from
    `ANIMOV`.`table_chevres_minute_serveur_v2`
  group by
    `ANIMOV`.`table_chevres_minute_serveur_v2`.`source`) `t2` on
  (((`t1`.`source` = `t2`.`source`)
  and (`t1`.`timestamp` > (`t2`.`max_timestamp` - (86400 * 1))))))
```

Cette partie effectue une jointure entre la table principale `table_chevres_minute_serveur_v2` et une sous-requête qui sélectionne la source et le maximum des timestamps. La jointure se fait sur la source et le timestamp supérieur au maximum des timestamps moins une journée (86400 secondes).

4. Regroupement des résultats :

Les résultats sont regroupés par source et par timestamp, arrondis à l'intervalle de 5 minutes.

XIII. UTILISATION DE LA BIBLIOTHEQUE `REQUESTS` POUR INTERROGER UNE API ET RECUPERER LES DONNEES AU FORMAT JSON

L'extraction de données depuis un service web (API) est une fonctionnalité essentielle de notre application de visualisation et d'analyse des comportements des chèvres. Nous utilisons la bibliothèque ``requests`` pour interroger une API et récupérer les données au format JSON. Voici une description détaillée de la manière dont cette extraction est mise en œuvre dans notre code.

1. Déclaration de l'URL de l'API

Nous avons défini un point de terminaison API (``END_POINT``) à partir duquel nous récupérerons les données. Le point de terminaison est stocké dans une variable globale afin de faciliter les appels aux différentes URL de l'API.

```
END_POINT =  
XXX.XXX.XXX.XXX:5000'
```

2. Envoi de requêtes HTTP GET avec ``requests``

Pour interagir avec l'API, nous utilisons la méthode ``get`` de la bibliothèque ``requests``. Cette méthode envoie une requête GET à l'URL spécifiée et retourne la réponse du serveur. Voici un exemple de méthode pour interroger une API et récupérer des données JSON :

```
def query_get_sources(self):  
    url = f'http://{END_POINT}/sources'  
    response = requests.get(url)
```

3. Vérification du statut de la réponse et gestion des erreurs

Après avoir envoyé la requête GET, nous vérifions si la réponse est correcte (code de statut 200). En cas d'erreur, nous gérons les exceptions pour éviter que l'application ne plante et fournir des messages d'erreur explicites.

```
try:  
    response.raise_for_status() # Vérifiez si le statut de la réponse est OK  
    (200)  
    data = response.json() # Essayez de décoder la réponse JSON  
    df = pd.DataFrame(data)  
    return df  
  
except requests.exceptions.HTTPError as http_err:  
    print(f"Erreur HTTP : {http_err}, URL : {url}")
```

```
except requests.exceptions.RequestException as err:
    print(f"Erreur de requête : {err}, URL : {url}")
except requests.exceptions.JSONDecodeError as json_err:
    print(f"Erreur de décodage JSON : {json_err}, Réponse : {response.text}")
```

4. Décodage de la réponse JSON et transformation en DataFrame Pandas

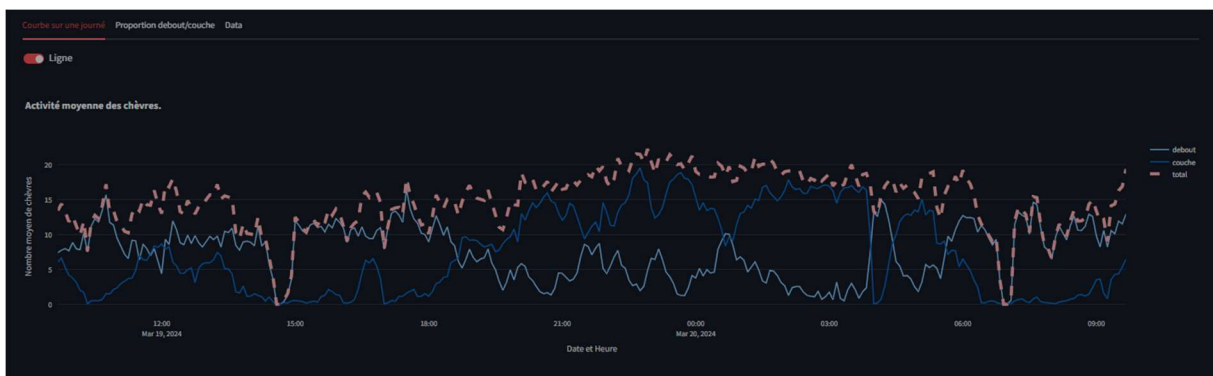
Si la requête est réussie, nous décodons la réponse JSON en utilisant la méthode `json` de l'objet réponse. Cette méthode convertit le JSON en un dictionnaire Python. Pour une manipulation et une analyse aisées des données, nous convertissons ce dictionnaire en un DataFrame Pandas.

```
data = response.json() # Essayez de décoder la réponse JSON
df = pd.DataFrame(data)
```

5. Utilisation des données JSON dans l'application

Les données récupérées et transformées en DataFrame Pandas sont ensuite utilisées dans l'application pour diverses analyses et visualisations. Voici quelques exemples :

- Création de graphiques



- Filtrage et sélection des données

XIV. EXTRACTION DE DONNEES DEPUIS UN FICHIER DE DONNEES

Ce script traite des fichiers CSV de deux types différents, les concatène, ajoute une colonne de source, les enregistre dans une base de données et retourne les DataFrames résultants.

1. Import des modules et création du moteur de base de données :

```
engine = create_engine(base_de_donnees_url)
resultats = {}
```

2. Traitement des fichiers CSV :

```
for type_fichier in ['-1.csv', '-2.csv']:
    fichiers_csv = glob.glob(os.path.join(chemin_dossier, f*{type_fichier}'))
    dataframes = [pd.read_csv(fichier) for fichier in fichiers_csv]
```

La boucle `for` itère sur deux types de fichiers CSV : ceux qui se terminent par `-1.csv` et ceux qui se terminent par `-2.csv`. Pour chaque type de fichier, la fonction

utilise `glob` pour trouver tous les fichiers correspondants dans le dossier spécifié (`chemin_dossier`). Les fichiers trouvés sont ensuite lus et convertis en DataFrames pandas.

3. Concaténation et ajout d'une colonne source :

```
if dataframes:
    df_final = pd.concat(dataframes, ignore_index=True)
    df_final['Source'] = source
```

Si des fichiers CSV ont été trouvés, les DataFrames sont concaténés en un seul DataFrame (`df_final`). Une nouvelle colonne `Source` est ajoutée au DataFrame avec la valeur fournie par l'argument `source`.

4. Enregistrement dans la base de données :

```
table_name = 'final' if '-1' in type_fichier else 'final_2'
df_final.to_sql(table_name, engine, if_exists='append',
index=False)
resultats[type_fichier] = df_final
```

En fonction du type de fichier, la fonction détermine le nom de la table dans laquelle les données seront stockées : `final` pour les fichiers `-1.csv` et `final_2` pour les fichiers `-2.csv`. Les données du DataFrame sont ensuite insérées dans la table correspondante de la base de données. Le DataFrame final est également stocké dans le dictionnaire `resultats`.

5. Gestion des cas où aucun fichier n'est trouvé :

```
else:
    resultats[type_fichier] = pd.DataFrame()
```

Si aucun fichier CSV correspondant n'est trouvé, un DataFrame vide est ajouté au dictionnaire `resultats` pour le type de fichier concerné.

6. Retour des résultats :

```
return resultats
```

XV. EXTRACTION DE DONNEES DEPUIS UN SYSTEME BIG DATA

Notre code permet d'afficher des données et des images stockées dans une base de données MongoDB via une interface web interactive créée avec Streamlit. Voici un résumé du fonctionnement du code :

1. Connexion à la Base de Données MongoDB :

```
client =
MongoClient('mongodb://xxxx:xxxxxxx@XXX.XXX.XXX.XXX:27017/')
db = client['ANIMOV']
collection = db['chevres']
```

2. Calcul du Nombre Total d'Enregistrements :

```
total_enregistrements = collection.count_documents({})
```

3. Extraction des données


```
n_ieme_enregistrement = collection.find().skip(n).limit(1).next()
```

Le code extrait le n-ième enregistrement de la collection en fonction de l'indice n.

4. Vérification et Extraction des Données

```
if 'data' in n_ieme_enregistrement and n_ieme_enregistrement['data']:  
    detect_data = n_ieme_enregistrement['data'][selected_option].get('detect', None)
```

Le code vérifie si le champ data existe et n'est pas vide, puis extrait les données.

5. Affichage des Données (Streamlit)

```
frame_data = n_ieme_enregistrement['data'][selected_option].get('frame', None)  
if frame_data:  
    image_data = base64.b64decode(frame_data)  
    image = Image.open(io.BytesIO(image_data))  
    st.write("Frame :")  
    st.image(image, caption=f'Frame {n}.')
```

6. Gestion des Erreurs

```
else:  
    st.error("Le champ 'detect' est absent du premier élément de 'data'.")
```

Si les données de détection ou l'image sont absentes, un message d'erreur est affiché pour informer l'utilisateur.