**Plan méthodologique pour répondre à la demande de la mairie de Montpellier sur l'utilisation des parkings voitures et vélos**

**Problématique :**

Vous et votre binôme avez été mandatés par la mairie de Montpellier afin d’étudier l’utilisation des parkings de la ville. Plusieurs aspects intéressent Monsieur le Maire, aussi fait-il appel à votre sagacité concernant les points suivants de sa politique de développement de la ville :

*⊲* Le taux d’occupation des parkings voiture *⊲* Le taux d’occupation des parkings vélos *⊲* Le bon fonctionnement du relais voiture / vélo.

1. **Collecte des données :**

* Récupération des données historiques d'occupation des parkings voitures et vélos sur l'année 2023, toutes les 5 minutes, sur le site open data de la mairie de Montpellier
* Récupération des données d'occupation en temps réel des mêmes parkings

1. **Préparation des données :**

* Nettoyage et vérification de la qualité des données collectées
* Formatage dans des tables structurées pour analyse (dates, horaires, taux d'occupation...)

1. **Analyses descriptives :**

* Calcul des taux d'occupation moyens pour chaque parking, toutes les 5 minutes, heures, jours, mois
* Identification des périodes de saturation (taux > 90%)
* Pour les parkings seule la saturation pose problème, pour les vélos on aura : saturation (impossibilité ou difficultés de déposer un vélo), mais aussi pénurie (absence de vélo à emprunter = 0 vélo dispo)
* Comparaison des évolutions selon les types de parkings et leur localisation

1. **Analyses explicatives**

* Corrélation entre taux d'occupation voitures et vélos selon les lieux et périodes
* Évolution des usages le WE / semaine

1. **Synthèse et recommandations**

* Présentation de la méthodologie
* Rédaction d'une synthèse des résultats sous Jupiter Notebook
* Recommandations pour dimensionnement des parkings
* Mise à disposition sur Jupiter Notebook
* Stockage des données, codes et synthèse sur GitHub

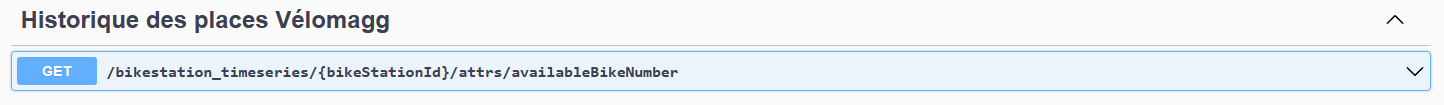
1. **Collecte des données :**

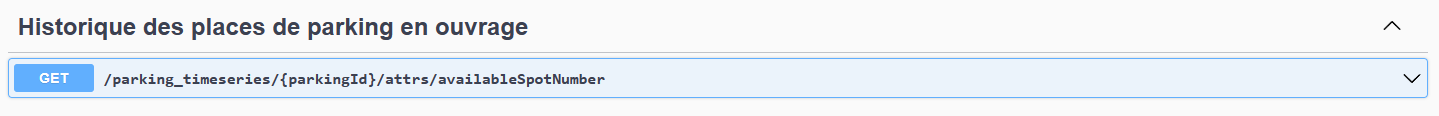
* **Sources :**

**Les donnée sont disponibles sur le site : Open Data Montpellier Méditerranée Métropole**

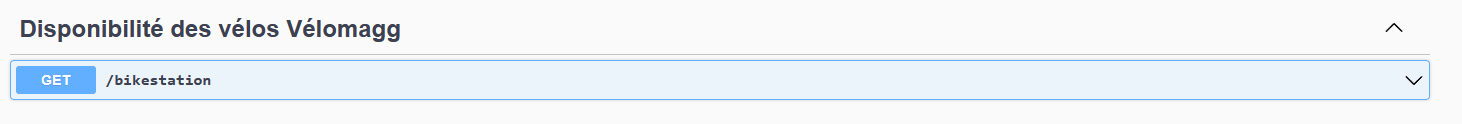
**Plusieurs API sont disponibles :**

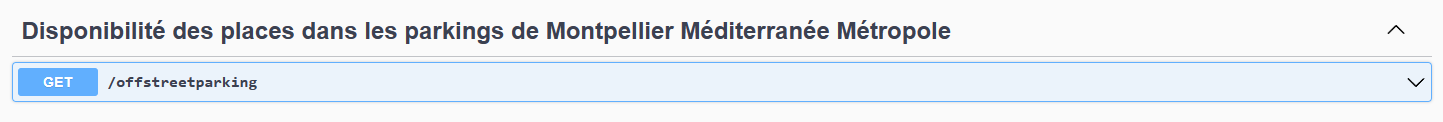
* Récupération des données historiques d'occupation des parkings voitures et vélos sur l'année 2023, toutes les 5 minutes, sur le site open data de la mairie de Montpellier





* Récupération des données d'occupation en temps réel des mêmes parkings

****

****

Compte tenu de l’objectif poursuivi : Analyse des taux d’occupation, les bases qui nous intéressent sont les données d’historique.

Les données en temps réels n’apportent pas d’information pertinente (en dehors de la valeur à un instant ‘t’) sur le taux d’occupation ou la pénurie des vélos au sein de la métropole de Montpellier.

Nous avons décidé de récupérer tous les ‘data’ des Parkings et Parcs à Vélo présents dans les bases de données de ’Open Data Montpellier Méditerranée Métropole’ pour l’année 2023 de façon à avoir une année complète.

Les bases de données ne disposent pas d’historique en deçà de 2023 (First recorded date at 2023-01-04).

On ne pourra donc pas établir des statistiques sur plusieurs années, ni faire une analyse des séries temporelles (étude saisonnalité et tendances).

* **bases Parkings :**

Les bases de données concernent 23 parkings, dont 19 sur la commune de Montpellier et 4 localisés sur les communes limitrophes (St Jean de Vedas (1) et Castelnau le Lez (3).

Sur les 23 parkings, 2 ne contiennent aucun historique sur la période étudiée (année 2023) et ne peuvent être traités :

* Gaumont Est
* Gaumont Ouest
* **bases Vélos :**

Les bases de données concernent 57 parcs à vélo, dont 56 sur la commune de Montpellier et 1 localisé sur une autre commune (Pérols).

* **Format des fichiers :**

Les données stockées sont au format JSON.

JSON est un format qui permet de stocker des informations structurées :

* Les données sont présentées sous forme de paires clé/valeur.
* ':' est placé entre chaque clé et chaque valeur
* Les éléments de données sont séparés par des virgules.
* Les crochets {} désignent les objets.
* Les crochets [] désignent des tableaux.
* {“key”:“value”,“key”:“value”,“key”:“value”.}
* **Structure des fichiers :**

**Chaque parking voiture ou parc vélo dispose d’un code unique, qui se présente de la façon suivante : « urn:ngsi-ld:parking:001 »**

**Chaque fichier parking contient les champs suivants :**

* attrName, str, 19 (nom de variable : availableSpotNumber, nombre de places disponibles)
* entityId, str, 23 (code parking)
* entityType, str, 16 (type : OffStreetParking)
* index, list, 8526 (timestamp)
* values, list, 8526 (nbr de places disponibles)

Timestamps constitue l’horodatage de la donnée (values), cette variable permet d’identifier la date et l’heure, elle se présente de la façon suivante : « 2023-01-04T10:50:51.000+00:00 »

# 1. Extraction des données :

Collecte des métadonnées (description des variables, unités, période couverte, etc) :

L’extraction et le traitement des données a été réalisée à partir de programmes spécifiques développés sous Python.

* **Base Parking :**
* nombre d'enregistrements attendus : 365 jours x 24 heures x 60 mn / 5 mn (temps de réponse entre 2 enregistrements) = 105.120 enregistrements
* les limites d'extraction sont de 10.000 lignes par extraction, il faut donc au moins 11 extractions pour récupérer une base complète
* le plus simple c'est de générer une boucle avec une extraction par mois, avec un cumul dans une même variable (par data.extend())
* début historique : 2023-01-04T10:50 (toutes les données commencent le 4 janvier à 10h50)
* **Base Parc Velo :** 
  + nombre d'enregistrements attendus : 365 jours x 24 heures x 60 mn / 1 mn (temps de réponse entre 2 enregistrements) = 525.600 enregistrements
  + les limites d'extraction sont de 10.000 lignes par extraction, il faut donc au moins 53 extractions pour récupérer une base complète
  + le plus simple c'est de générer une boucle avec une extraction par jour, avec un cumul dans une même variable (par data.extend())
  + début historique : 2023-01-04T10:50 (toutes les données commencent le 4 janvier à 10h50)
* **extraction des données de l’API :**

url = "https://portail-api-data.montpellier3m.fr/parking\_timeseries/"+parking\_id+"/attrs/availableSpotNumber"

params = {

"fromDate": from\_date,

"toDate": to\_date

}

response = requests.get(url, params=params)

data = response.json()

* **cumul dans une base totale pour chaque parking:**

# Ajoutez les nouvelles valeurs des listes.

data\_cumul["index"].extend(data["index"])

data\_cumul["values"].extend(data["values"])

## Transformation des Données :

## 2.1 Transformation des JSON en Dataframe :

Le format JSON n’est pas vraiment adapté pour effectuer des calculs statistiques ou même contrôler la qualité des données.

On a privilégié le traitement sous forme de DataFrame de la bibliothèque Pandas.

Ce qui permet également une continuité des traitements informatiques sous Python (comme pour l’extraction des données).

Le code Python est par ailleurs plus facilement réutilisable en vue de futures mises à jour.

Le passage du JSON au DataFrame s’effectue simplement en combinant les 2 listes : timestamp et value dans un dictionnaire puis en passant le dictionnaire au DataFrame :

pkg\_data = json.load(parking\_file)

timestamps = np.array(pkg\_data["index"])

values = np.array(pkg\_data["values"])

data = {"timestamp": timestamps, "values": values}

dataframe = pd.DataFrame(data)

## Vérification de la qualité des données :

* Valeurs manquantes ou aberrantes : les disponibilités ne peuvent être > capacité, création d’un filtre :   
  dataframe = dataframe[(dataframe['values'] <= capacite\_max)]

Les fichiers par parking contiennent un nombre variable de jours (en moyenne 255 jours).

Si les bases de données ne sont pas complètes, on dispose de suffisamment de données pour pouvoir

* Types et formats des variables : pour traiter plus facilement les données, celles-ci après avoir été chargées sont transférées du format JSON au format « dataframe » de la librairie Pandas sous Python.
* Cohérence des dates/horaires : le timstamp permet une codification cohérente des dates et des heures. Les délais entre 2 enregistrements et d’environ 5 mn pour les parkings et d’une minute pour les stations de vélos.

## 2.3 Structure des données :

* **availableSpotNumber :** Capacité maximum des parkings (extrait de la liste des parkings et de la liste des stations de vélo),
* **Timestamp**: contient toutes les informations de date et heure

Ex : « 2023-02-23T16:11:29.000+00:00 »

# Reformate le timestamp au format pandas (supprime notamment le 'T') :

dataframe["timestamp"] = pd.to\_datetime(dataframe["timestamp"])

* **availableSpotNumber**: nombre de places disponibles

ex.         260.0

## 2.4 Transformation des données :

* Extraction des variables temporelles : nous avons choisi de traiter les données selon :
  + **Mois :** analyse de la variation mensuelle sur les 12 mois de l’année 2023
  + **Jour :** analyse de la variation selon le jour de semaine, du lundi au dimanche
  + **Heure :** analyse de la variation sur les 24 heures de la journée
* Calcul de nouvelles variables dérivées :
  + **Nb\_data :** nombre de données / temps (heure, jour, mois)
  + **min :** nombre minimum de places disponibles / temps (heure, jour, mois)
  + **max :** nombre maximum de places disponibles / temps (heure, jour, mois)
  + **mean :** nombre moyen de places disponibles / temps (heure, jour, mois)
  + **std :** écart-type du nombre de places disponibles / temps (heure, jour, mois)
  + **sum :** nombre total de places disponibles / temps (heure, jour, mois)
  + **Q1 :** 1er quartile (25 % ) de la distrib. du nombre de places disponibles / temps (heure, jour, mois)
  + **median :** 2ème quartile ou médian (50 % ) de la distrib. du nombre de places disponibles / temps (heure, jour, mois)
  + **Q3 :** 3ème quartile (75 % ) de la distrib. du nombre de places disponibles / temps (heure, jour, mois)
  + **Taux de disponibilité :** nb de places disponibles / capacité maximum \* 100
  + **Taux d’occupation :** 100 - Taux de disponibilité
  + **Penurie :** nbr de périodes de temps ou le parking est saturation (places disponibles = 0),
  + Taux\_penurie : penurie / nb data \*100 (soit % de temps ou le parking est à saturation).
* Traitements Python pour la création des variables :

# Calcul des statistiques :

dataF\_comp["nb\_data"] = dataframe.groupby("hour")["values"].count()

dataF\_comp["min"] = dataframe.groupby("hour")["values"].min()

dataF\_comp["max"] = dataframe.groupby("hour")["values"].max()

dataF\_comp["mean"] = round(dataframe.groupby("hour")["values"].mean(),1)

dataF\_comp["std"] = round(dataframe.groupby("hour")["values"].std(),1)

dataF\_comp["sum"] = dataframe.groupby("hour")["values"].sum()

dataF\_comp["Q1"] = dataframe.groupby("hour")["values"].quantile(0.25)

dataF\_comp["median"] = dataframe.groupby("hour")["values"].median()

dataF\_comp["Q3"] = dataframe.groupby("hour")["values"].quantile(0.75)

# Calcul du taux d'occupation dans le dataframe source :

dataframe["taux\_dispo"] = (dataframe["values"] / capacite\_max)\* 100

# Aggregation dans le dataframe traité

dataF\_comp["taux\_dispo"] = round(dataframe.groupby("hour")[["taux\_dispo"]].agg({

"taux\_dispo": "mean"}),2)

dataF\_comp["taux\_occ"] = 100- dataF\_comp["taux\_dispo"]

# Compter le nb de valeurs = 0 (Penurie) :

dataF\_comp["penurie"] = dataframe.groupby("hour")["penurie"].sum()

# Calcul du taux de pénurie dans le dataframe source

dataF\_comp["taux\_penurie"] = (dataF\_comp["penurie"] / dataF\_comp["nb\_data"])\* 10

## 2.5 Organisation des données :

On génère 1 fichier par niveau de temps pour chaque parking, soit 3 fichiers data par parking.

Ex. data par jour :

Une image contenant capture d’écran, Caractère coloré, carré

Description générée automatiquement

On a également généré une série de graphiques pour parking ou station de vélo et selon les 3 niveaux de temps :

* Graphique min, moy, max
* Graphique médiane + intervalle interquartile
* Graphique Taux de disponibilité
* Graphique Taux d’occupation
* Graphique Taux de pénurie

# Comparaison entre parkings :

Pour pouvoir comparer les parkings entre eux selon leur niveau d’occupation et la saturation et rechercher des comportements homogènes, on a réalisé une compilation des différents parkings et stations de vélos.

## Compilation des données :

Chaque ligne du nouveau DataFrame correspond à un parking ou une station de vélos.

A partir des données par jour de semaine, on a distingué 2 périodes : Jour de semaine / Week-end.

A partir des données par heure, on a distingué 3 périodes : nuit / jour / soir.

A partir des données par mois, on a distingué 2 périodes : Été / autres saisons.

On a également considéré la localisation géographique du parking : Centre de Montpellier/Périphérie

Exemple du code permettant de compiler les taux d’occupation moyen en semaine et en week-end pour les parkings :

sem\_occ = round(bdd['taux\_occ'].iloc[0:5].mean(),2)

we\_occ = round(bdd['taux\_occ'].iloc[6:7].mean(),2)

Tx\_penurie = round(bdd['taux\_penurie'].mean(),2)

#création des nouvelles colonnes du DF :

if ind == 0:

d\_G['sem\_occ'] = np.nan

d\_G['we\_occ'] = np.nan

d\_G['tx\_penurie'] = np.nan

# remplissage

d\_G.loc[ind,'sem\_occ'] = sem\_occ

d\_G.loc[ind,'we\_occ'] = we\_occ

d\_G.loc[ind,'tx\_penurie'] = Tx\_penurie

La base résultant de ce traitement se présente sous la forme suivante pour les parkings :

**Une image contenant texte, capture d’écran

Description générée automatiquement**

## Requêtes sur la base compilée :

Afin de faciliter l’analyse dans le cadre de cette étude et de son exploitation éventuelle par le commanditaire de l’étude, on a généré un système de requête assez simple à partir du DataFrame compilé.

Il suffit d’écrire la condition ou filtre souhaité dans la variable ‘Query’ comme ci-dessous :

Query = 'we\_occ > 50 and tx\_penurie>0'

Query = 'we\_occ > sem\_occ \* 0.9'

Query = 'zone == "Centre"'

Puis d’exécuter et afficher les résultats :

print(d\_V.query(Query))

Il est également possible d’effectuer un nouveau calcul sur la base des éléments correspondant au Query :

## nouveaux calculs sur la base :

param='we\_occ'

print(d\_V.query(Query)[param].mean())

**Analyse selon la localisation :**

* Cartographie du niveau moyen d’occupation par parking (min, moy, max)
* Cartographie des seuils dépassés par parking (nb heures > seuil)
* Corrélation avec facteurs géographiques (population, transports...)

## **Analyses explicatives**

## **Synthèse et recommandations**