Data Analytics

Paris for Kids

Xinly ROY

November, 2024

Table of content

Overview of the project

The main goal of this project is to develop a user-friendly Streamlit application that helps parents and caregivers find kid-friendly places across Paris. These places include restaurants, parks, events, and other family-oriented attractions. The project leverages a wide variety of data sources, integrates machine learning for predictive insights, and uses interactive visualizations to make the data accessible.

Motivation

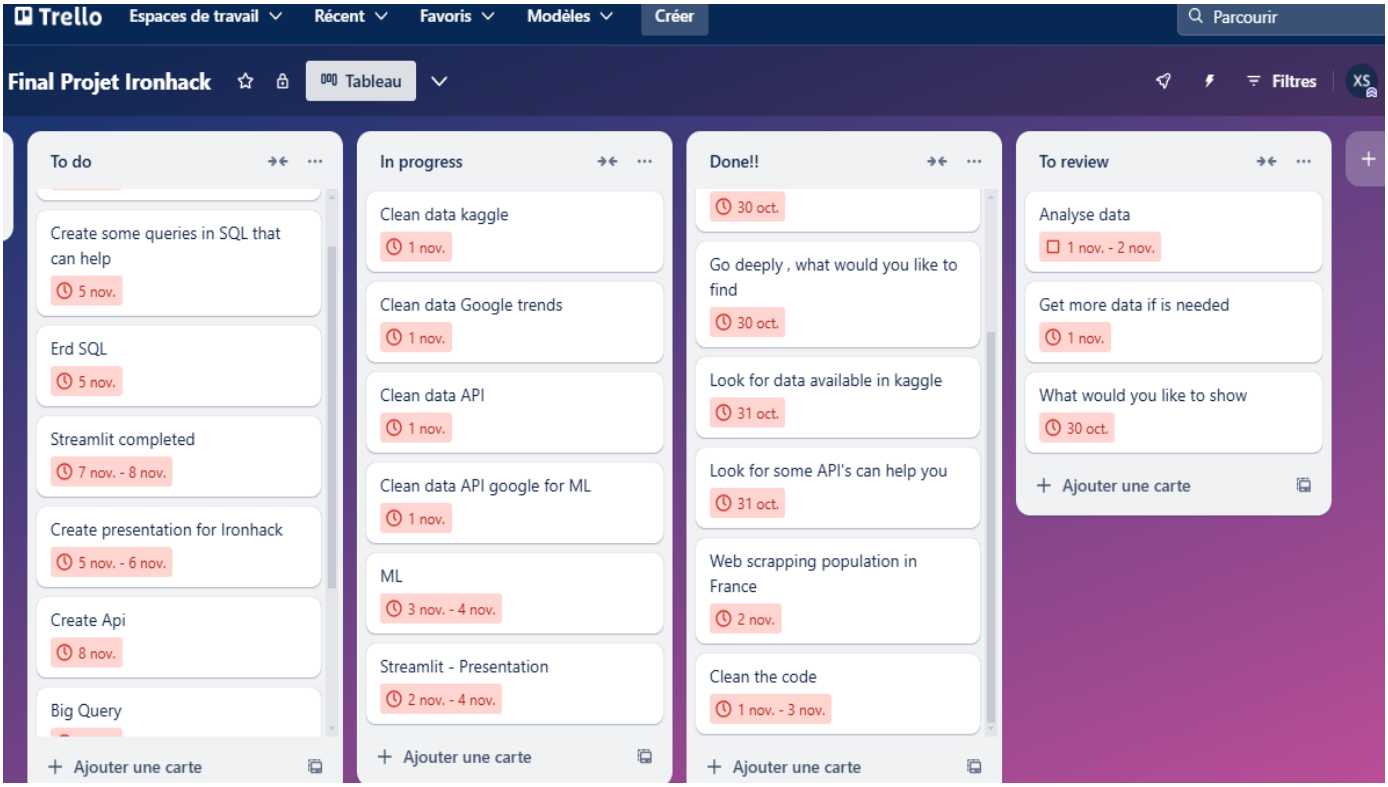
As a parent, I understand the challenges of finding locations that cater to families with children. This project seeks to alleviate these difficulties by offering a centralized platform where users can access reliable information about places that are welcoming to children. Whether it's discovering a family-friendly restaurant, attending an event, or visiting a park, this application aims to simplify planning for parents and caregivers.

Key Features

1. **Introduction**:
    - Explains the app's purpose and how it can benefit families.
2. **Data Exploration**:
    - Offers transparency by detailing the origins and structure of datasets used.
3. **Data Analysis**:
    - Explores trends in population, activities, and kid-friendly places by arrondissement or year.
4. **Recommendations**:
    - Provides personalized suggestions for officially categorized kid-friendly locations.
5. **Machine Learning**:
    - Predicts whether a non-categorized restaurant is kid-friendly based on user reviews.
6. **Data Download**:
    - Allows users to download datasets for further analysis through our API.

Project Management

I used Trello to organize and monitor the progress of the project:



Data and data sources

Data was sourced from multiple channels to ensure diversity and relevance:

1. Web Scraping:

- Extracted population data for Paris and France from websites like Wikipedia and L'Internaute.

2. API:

- **Google Places API**:
  - Fetched reviews and metadata for officially categorized kid-friendly places all over the world to train the ML model. And some data from Paris for the analysis and recommendations section.
  - Gathered data for non-categorized restaurants to test the machine learning model.

- **Google Trends API**:
  - Extracted seasonal trends in Google searches for kid-friendly activities.

3. Flat Files (CSV):

- Downloaded event data for Paris from Kaggle, including descriptions, dates, and locations.
- Downloaded event data for Paris from Opendata.Paris, including public green spaces (parks, squares, promenades, etc.) managed by the City of Paris.

4. Relational Database:

- Cleaned datasets were imported into MySQL for normalization and querying.

5. BigQuery:

- A denormalized version of the database was uploaded to BigQuery to facilitate more complex analytics.

Data cleaning

The data cleaning process was one of the most critical steps in this project. It involved transforming raw datasets from various sources into a structured and consistent format suitable for analysis, machine learning, and integration into a relational database. Below is a general overview of the steps applied across all datasets, highlighting the common challenges faced and the techniques used to address them:

General Cleaning Workflow

1. **Handling Missing Values**:

- Missing values in key columns were either filled with meaningful defaults ("Unknown", 0) or derived from other columns when possible. For instance, if an address was missing, its zip code was used as a fallback.
- Rows with excessive missing data in critical fields, such as names or unique identifiers, were dropped.

```
df_faire["Nom du lieu"] = df_faire["Nom du lieu"].fillna(df_faire["Code postal"])
df_faire["Description"] = df_faire["Description"].fillna(df_faire["Chapeau"])
```

2. **Standardizing Formats**:

- All zip codes were standardized to ensure consistency across datasets. Some datasets included incorrect zip codes (7014 instead of 75014), which were corrected during the cleaning process.
- Dates were converted to a consistent datetime format, allowing for easier analysis. For invalid dates, "Unknown" was used as a placeholder.

```
# Mapping dictionary to convert month numbers to month names
month_mapping = {1: 'January', 2: 'February', 3: 'March', 4: 'April', 5: 'May', 6: 'June',
    7: 'July', 8: 'August', 9: 'September', 10: 'October', 11: 'November', 12: 'December'}
# Apply the mapping to 'Month_fin' and 'Month_debut' columns
df_faire['Month_fin'] = df_faire['Month_fin'].map(month_mapping)
df_faire['Month_debut'] = df_faire['Month_debut'].map(month_mapping)
# Mapping to convert arrondissements to zip codes
arrondissement_to_zip = {
    "1er": "75001", "2e": "75002", "3e": "75003", "4e": "75004", "5e": "75005",
    "6e": "75006", "7e": "75007", "8e": "75008", "9e": "75009", "10e": "75010",
    "11e": "75011", "12e": "75012", "13e": "75013", "14e": "75014", "15e": "75015",
    "16e": "75016", "17e": "75017", "18e": "75018", "19e": "75019", "20e": "75020"
}

# Apply the mapping to the 'arrondissement' column to create a new 'zip_code' column
df_kf_places_paris['zipcode'] = df_kf_places_paris['arrondissement'].map(arrondissement_to_zip)
```

3. **Removing Duplicates**:

- Some duplicated rows were eliminated to maintain data integrity.

4. **Filtering Irrelevant Data**

- Certain rows or columns irrelevant to the project's goals were removed.

```
df_parks_with_playground=df_parks_with_playground.drop(columns=['ID_ATELIER_HORTICOLE',"Nombre d'entités",'IDA3D_ENB','ID_EQPT',
                                                               'Compétence','Année de changement de nom','ID_DIVISION','Année de rénovation',
                                                               'Surface horticole','Présence cloture','last_edited_user','last_edited_date'])
```

5. **Enriching Data**

- New columns were added to enhance the datasets.

```
df_kf_places_paris['unique_id'] = df_kf_places_paris.apply(lambda row: f"{row['name'][:4].replace(' ', '')}_{row['zipcode']}_{row.name}", axis=1)
df_kf_places_paris['type_id'] = df_kf_places_paris.apply(lambda row: f"{row['type'][:2].replace(' ', '')}-{row['zipcode']}", axis=1)
df_kf_places_paris['df_id']='kfex3075'
df_kf_places_paris['df_unique_id']=df_kf_places_paris['df_id']+'_'+(df_kf_places_paris.index+1).astype(str)
```

6. **Metadata / Data Dictionary**

- For each dataset, I documented the column names, data types, and their meanings. Here's an example for one of the datasets:

Dataset Name: paris_data

- **Source:** Web Scraping (Wikipedia, Paris.fr)
- **Description:** Contains demographic and geographic data for Paris.
- **Columns:**
    1. **arrondissement (integer):** District number of Paris.
    2. **name_arrondissement (string):** Official name of the arrondissement.
    3. **surface_ha (float):** Surface area of the arrondissement in hectares.
    4. **population_2020 (integer):** Population of the arrondissement in 2020.
    5. **density_2021_hab_km2 (integer):** Population density in 2021.
    6. **zipcode (string):** Postal code of the arrondissement.

This type of documentation was created for all datasets.

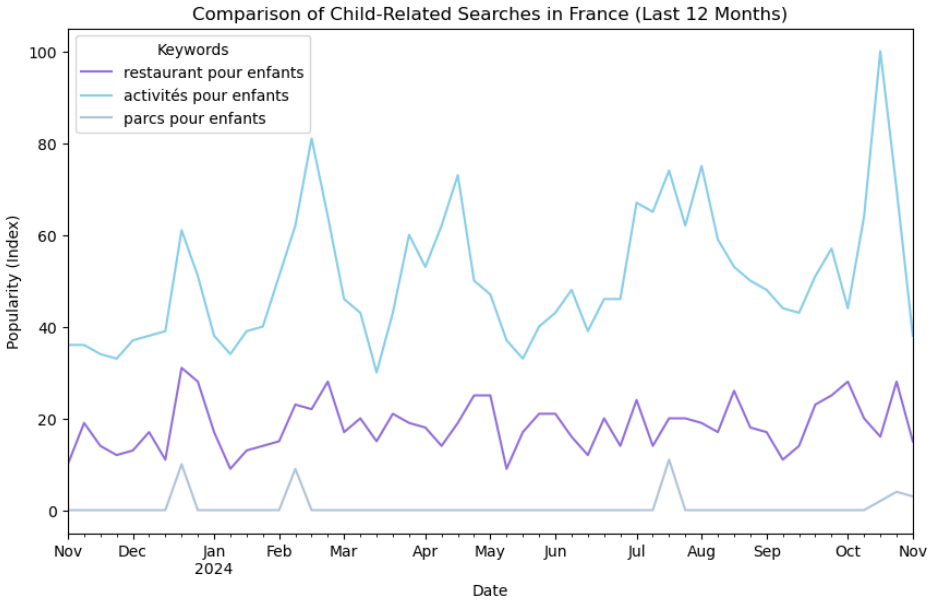
Exploratory data analysis and Visualization

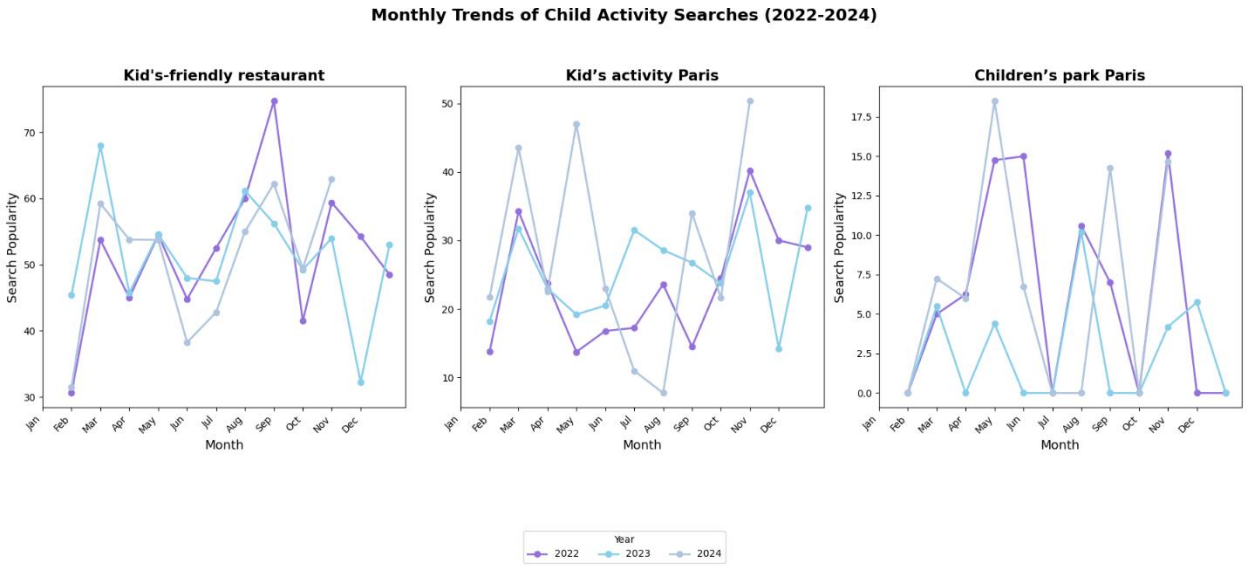
EDA was conducted to uncover meaningful insights from the cleaned datasets. Key patterns and trends were identified, which informed recommendations for families in Paris.

Highlights from the EDA include:

Analyzing Seasonal Trends:

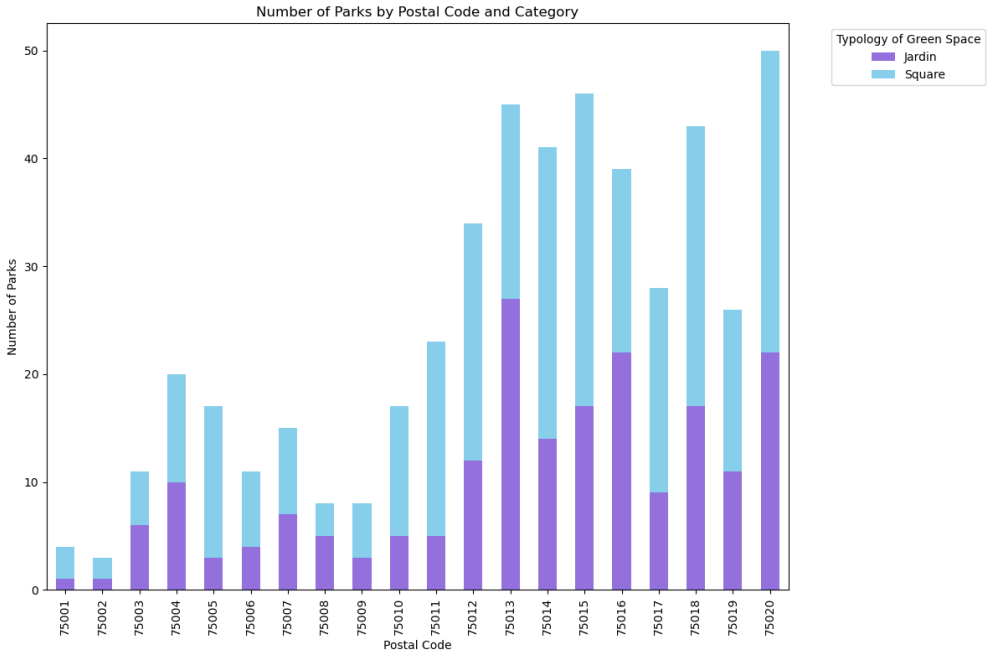
- Using Google Trends data, seasonal interest in kid-friendly restaurants and activities was analyzed. Line charts illustrated peaks in searches during warmer months, aligning with school holidays and family outings.

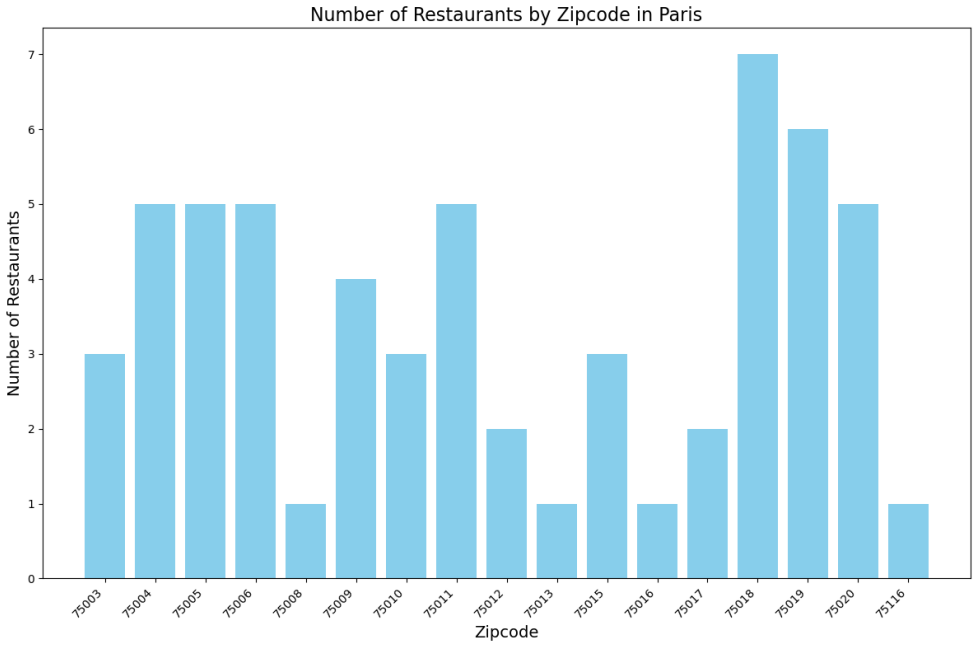




Distribution Across Paris:

- Bar charts revealed the distribution of population, restaurants, and parks across Paris' arrondissements. Highly populated districts, such as the 15th and 19th, were shown to have better accessibility to kid-friendly places.



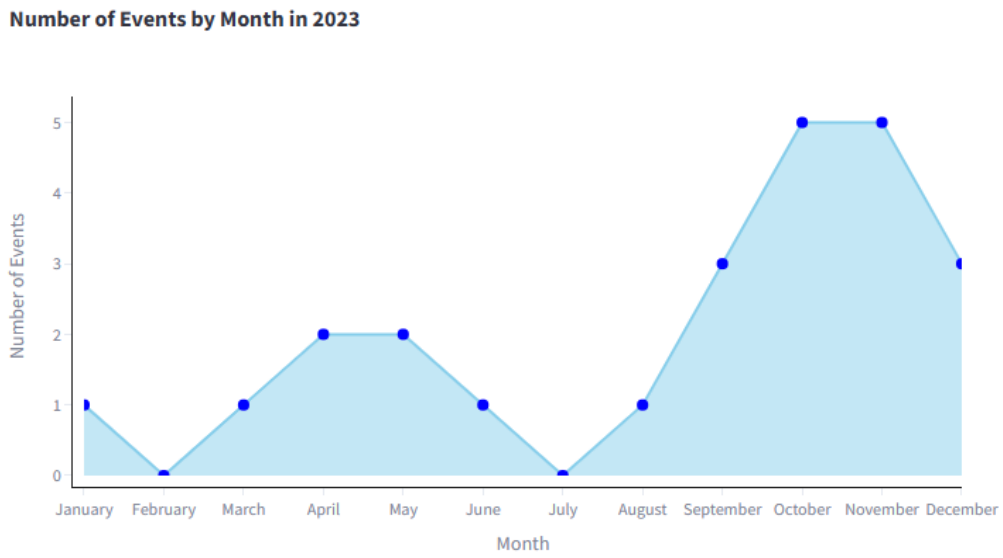


Activity Types:

- The frequency and variety of activities were analyzed by arrondissement, showing which districts had more family-oriented facilities (playgrounds, parks, and restaurants).

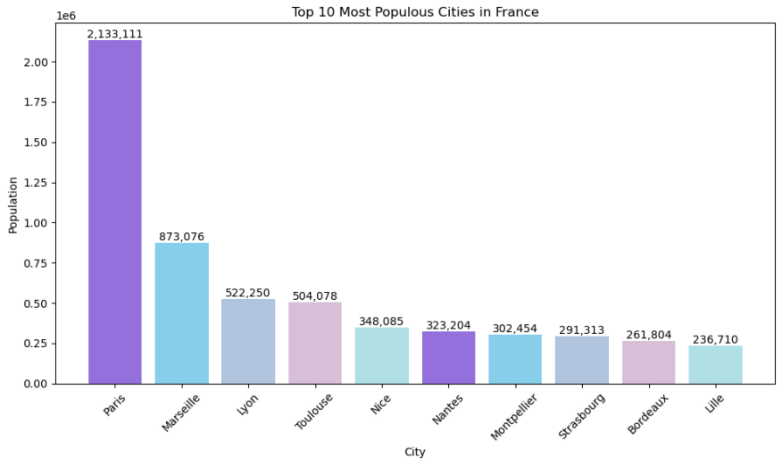
Event Trends:

- Monthly trends in kid-friendly events were visualized, highlighting periods with more family-oriented events, such as summer and back-to-school months.



Centralization of Paris:

- Comparative analyses with other major French cities underscored Paris' unique position as the most populous and activity-dense city, emphasizing its potential to cater to family-friendly needs.



Database type selection

The data was stored in a MySQL database, chosen for its reliability and efficiency in handling relational data. To prepare the datasets for MySQL, additional cleaning and transformations were required. This included addressing challenges such as emojis and special characters in user-generated reviews, which caused encoding issues, and ensuring column lengths matched the database constraints. Data types were standardized, such as converting zip codes to strings to accommodate varying formats.

```
CREATE DATABASE kids_friendly_db_new;
USE kids_friendly_db_new;
CREATE TABLE detailed_places (

    name VARCHAR(255),

    type VARCHAR(255),

    vicinity VARCHAR(255),

    formatted_address VARCHAR(255),

    website VARCHAR(255),

    phone_number VARCHAR(50),

    opening_hours TEXT,

    reviews TEXT,

    zipcode INT,

    unique_id VARCHAR(255),

    type_id VARCHAR(255),
```

```
CREATE TABLE `type` (
    type_id VARCHAR(255) PRIMARY KEY,
    `type` VARCHAR(255) NOT NULL,
    zipcode VARCHAR(10),
    CONSTRAINT fk_type_zipcode FOREIGN KEY (zipcode) REFERENCES paris_data(zipcode)
);
```

```
CREATE TABLE parks (
    unique_id VARCHAR(255) PRIMARY KEY,
    name_park VARCHAR(255),
    typologie_espace_vert VARCHAR(255),
    categorie VARCHAR(255),
    zipcode VARCHAR(10),
    ouverture_24h_24h VARCHAR(50),
    url_plan VARCHAR(255),
    geo_point VARCHAR(255),
    adresse VARCHAR(255),
    `type` VARCHAR(255),
    type_id VARCHAR(255),
    df_id VARCHAR(255),
    df_unique_id VARCHAR(255),
    CONSTRAINT fk_parks_zipcode_new FOREIGN KEY (zipcode) REFERENCES paris_data(zipcode),
    CONSTRAINT fk_parks_type_id FOREIGN KEY (type_id) REFERENCES `type`(type_id));
```

SQLAlchemy was used to facilitate the seamless transfer of cleaned data into MySQL. Foreign key relationships were established between tables to maintain referential integrity.

```python
#Connection to SQL
load_dotenv()

DB_HOST = os.getenv("DB_HOST")
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_NAME = os.getenv("DB_NAME")


conn = mysql.connector.connect(
    host=DB_HOST,
    user=DB_USER,
    password=DB_PASSWORD,
    database=DB_NAME
)
cursor = conn.cursor()
```

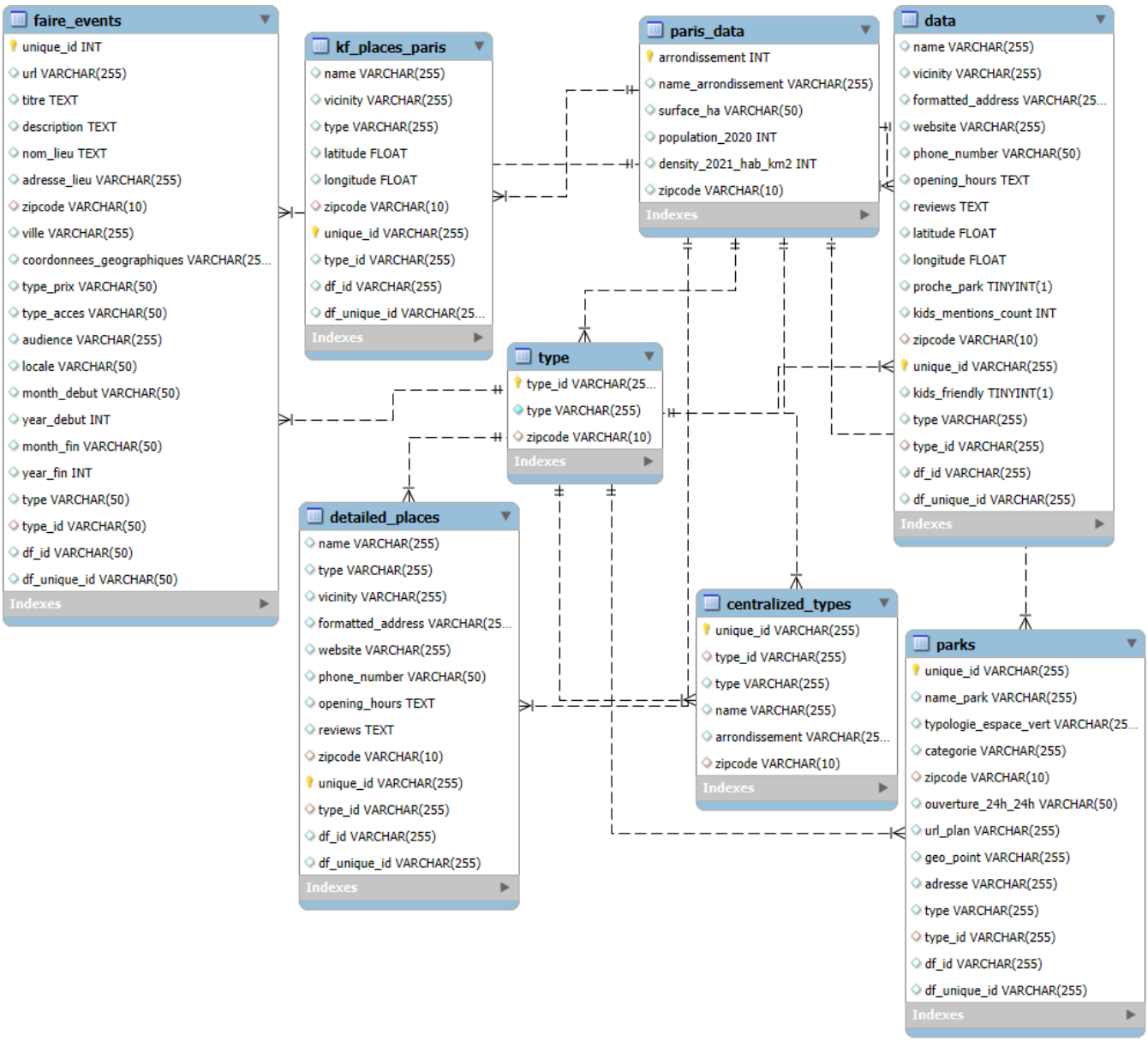
```python
# Founction to clen every csv
def clean_data(df):
    def remove_emojis(text):
        emoji_pattern = re.compile(
            "["
            "\U0001F600-\U0001F64F"  # Emoticons
            "\U0001F300-\U0001F5FF"  # Symboles and pictogrammes divers
            "\U0001F680-\U0001F6FF"  # Transports and symboles
            "\U0001F1E0-\U0001F1FF"  # Flgs
            "\U00002500-\U00002BEF"  # Other stuff
            "\U00002702-\U000027B0"  # Symboles divers
            "\U0001F900-\U0001F9FF"  # Emoji divers
            "\U0001FA70-\U0001FAFF"  # More emojis
            "\U00002600-\U000026FF"  # Symboles divers
            "]+",
            flags=re.UNICODE
        )
        return emoji_pattern.sub(r'', text)

    for column in df.columns:
        if pd.api.types.is_numeric_dtype(df[column]):
            df[column] = pd.to_numeric(df[column], errors="coerce").fillna(0)
        elif pd.api.types.is_string_dtype(df[column]):
            df[column] = df[column].astype(str).str.strip().replace("nan", "").fillna("")
            df[column] = df[column].apply(remove_emojis)

            if column == 'zipcode':
                df[column] = df[column].str[:10]
    return df

# Function to insert the data to sql
def insert_data(df, table_name):
    placeholders = ", ".join(["%s"] * len(df.columns))
    columns = ", ".join([f"`{col}`" for col in df.columns])

    # DUPLICATE KEY UPDATE
    update_clause = ", ".join([f"`{col}`=VALUES(`{col}`)" for col in df.columns if col != 'unique_id'])

    query = f"""
    INSERT INTO `{table_name}` ({columns}) VALUES ({placeholders})
    ON DUPLICATE KEY UPDATE {update_clause};
    """

    for _, row in df.iterrows():
        cursor.execute(query, tuple(row))
    conn.commit()
```

To simplify analysis, a centralized table, centralized_types, was created, consolidating key data such as unique IDs, types, names, and zip codes from multiple tables. This table bridged the gap between normalized database storage and advanced analytical requirements, setting the stage for exploratory analysis and visualization. This structured approach ensured the data was clean, consistent, and fully prepared for downstream tasks.

```sql
#Create new table to centralized the all tables on type
CREATE TABLE centralized_types (
    unique_id VARCHAR(255) PRIMARY KEY,
    type_id VARCHAR(255),
    `type` VARCHAR(255),
    `name` VARCHAR(255),
    arrondissement VARCHAR(255),
    zipcode VARCHAR(10)
);

INSERT INTO centralized_types (unique_id, type_id, `type`, `name`, arrondissement, zipcode)
SELECT
    d.unique_id,
    d.type_id,
    d.`type`,
    d.name,
    pd.name_arrondissement,
    d.zipcode
FROM `data` d
LEFT JOIN paris_data pd ON d.zipcode = pd.zipcode;
```

Entities. ERD

The Entity-Relationship Diagram (ERD) show the relational structure of the database, featuring key entities and their relationships. The primary tables include detailed_places, faire_events, parks, kf_places_paris, data, and type, all connected through shared attributes such as type_id and zipcode. The paris_data table links arrondissements and population data with other entities through the zipcode field, while the centralized_types table consolidates data from all sources, enabling streamlined querying and analysis.



SQL Queries

Here are some of the sql queries that helped with the analysis of this project:

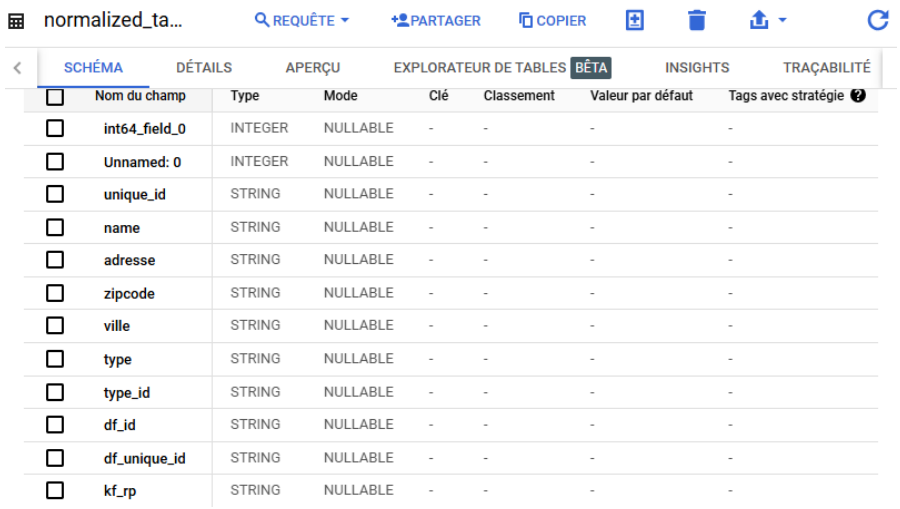
```sql
#Name and number actvities by name_arrondissement
SELECT
    pd.name_arrondissement AS Arrondissement,
    t.`type` AS Type,
    COUNT(DISTINCT dp.unique_id) AS Total
FROM detailed_places dp
JOIN `type` t ON dp.type_id = t.type_id
JOIN paris_data pd ON dp.zipcode = pd.zipcode
GROUP BY pd.name_arrondissement, t.`type`
ORDER BY pd.name_arrondissement, Total DESC;

#Top 5 restaurants by number of reviews
SELECT
    dp.name AS Restaurant,
    dp.formatted_address AS Adresse,
    dp.reviews AS Avis,
    dp.zipcode AS Zipcode,
    COUNT(reviews) AS Nombre_Avis
FROM detailed_places dp
JOIN `type` t ON dp.type_id = t.type_id
WHERE t.`type` = 'Restaurant'
GROUP BY dp.name, dp.formatted_address, dp.reviews, dp.zipcode
ORDER BY Nombre_Avis DESC
LIMIT 5;
```

```sql
#Number of differents activities by number of habitants per arrondissement
SELECT
    pd.zipcode AS Arrondissement,
    pd.population_2020 AS Number_Habitants,
    COUNT(CASE WHEN ct.`type` = 'Restaurant' THEN ct.unique_id END) AS Number_Restaurants,
    COUNT(CASE WHEN ct.`type` = 'Event' THEN ct.unique_id END) AS Number_Events,
    COUNT(CASE WHEN ct.`type` = 'Park' THEN ct.unique_id END) AS Number_Parks,
    COUNT(CASE WHEN ct.`type` = 'Museum' THEN ct.unique_id END) AS Number_Museums,
    COUNT(CASE WHEN ct.`type` = 'Zoo' THEN ct.unique_id END) AS Number_Zoos,
    COUNT(ct.unique_id) AS Nombre_Total_Activites
FROM paris_data pd
LEFT JOIN centralized_types ct ON pd.zipcode = ct.zipcode
GROUP BY pd.zipcode, pd.population_2020
ORDER BY Number_Habitants DESC;
```

```sql
#Events by acces type and price
SELECT
    fe.type_acces AS Type_Acces,
    fe.type_prix AS Type_Prix,
    COUNT(fe.unique_id) AS Total_Events
FROM faire_events fe
GROUP BY fe.type_acces, fe.type_prix
ORDER BY Total_Events DESC;

#Numbers of Events and Parks by most tourist arrondissement
SELECT
    pd.name_arrondissement AS Arrondissement,
    COUNT(DISTINCT p.unique_id) AS Total_Parcs,
    COUNT(DISTINCT fe.unique_id) AS Total_Events
FROM paris_data pd
LEFT JOIN parks p ON pd.zipcode = p.zipcode
LEFT JOIN faire_events fe ON pd.zipcode = fe.zipcode
WHERE pd.zipcode IN ('75001', '75004', '75008', '75016', '75018')
GROUP BY pd.name_arrondissement
ORDER BY Total_Events DESC, Total_Parcs DESC;
```

Big Data System

A denormalized version of the database was uploaded to BigQuery for advanced analytics.



normalized_ta... RÉQUÊTE PARTAGER COPIER

SCHÉMA | DÉTAILS | APERÇU | EXPLORATEUR DE TABLES BÊTA | INSIGHTS | TRAÇABILITÉ

Nom du champ	Type	Mode	Clé	Classement	Valeur par défaut	Tags avec stratégie
int64_field_0	INTEGER	NULLABLE	-	-	-	-
Unnamed: 0	INTEGER	NULLABLE	-	-	-	-
unique_id	STRING	NULLABLE	-	-	-	-
name	STRING	NULLABLE	-	-	-	-
adresse	STRING	NULLABLE	-	-	-	-
zipcode	STRING	NULLABLE	-	-	-	-
ville	STRING	NULLABLE	-	-	-	-
type	STRING	NULLABLE	-	-	-	-
type_id	STRING	NULLABLE	-	-	-	-
df_id	STRING	NULLABLE	-	-	-	-
df_unique_id	STRING	NULLABLE	-	-	-	-
kf_rp	STRING	NULLABLE	-	-	-	-

KidsFriendlyParis | Tapez / pour rechercher des ressources, des documents, des produits, etc | Recherche

Explorateur

Requête ... tre | normaliz... ble | *AnalysKF

Analys KF | EXÉCUTER | ENREGISTRER LA REQUÊTE | TÉLÉCHARGER | PARTAGER | PLANIFIER | Requête termi

```
16    OR adresse IS NULL
17    OR zipcode IS NULL
18    OR ville IS NULL
19    OR type IS NULL;
20
21  #Distribution of types across zip codes
22  SELECT
23    zipcode,
24    type,
25    COUNT(*) AS num_places
26  FROM `kidsfriendlyparis.Kids_friendly_data.normalized_table`
27  GROUP BY zipcode, type
28  ORDER BY zipcode, num_places DESC;
29
30  #Unique places with adresse
31  SELECT DISTINCT
32      unique_id,
33      name AS Place_Name,
34      adresse AS Address,
35      zipcode AS Zip_Code,
36      ville AS City
37  FROM `kidsfriendlyparis.Kids_friendly_data.normalized_table`
38  ORDER BY City, Place_Name;
39
40
41  # Places by city
42  SELECT
43      ville AS City
```

Appuyez sur Alt+F1 pour afficher les options d'accessibilité

Résultats de la requête | ENREGISTRER LES RÉSULTATS | EXPLORER LES DONNÉES

Machine Learning

The Machine Learning component of this project focuses on predicting whether restaurants not officially categorized as kid-friendly can still provide a welcoming environment for families with children. Two different models were implemented to analyze customer reviews and provide predictions: Naive Bayes and Logistic Regression with TF-IDF.

Training the model

```
X_train = train.drop(columns = "kids_friendly")
y_train = train["kids_friendly"]
```

```
#create the model

model = MultinomialNB()
model.fit(X_train, y_train)
```

```
model.score(X_train, y_train)
```

```
data_eval = train
```

```
pred=model.predict(data_eval.drop(columns = "kids_friendly"))
```

```
# Applying TF-IDF instead of CountVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=20000)
X_tfidf = tfidf_vectorizer.fit_transform(df_concat_cleaned["clean_blob"]).toarray()

# Training the model with TF-IDF
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, df_concat_cleaned["kids_friendly"], test_size=0.2, random_state=42
model = LogisticRegression(class_weight='balanced', max_iter=1000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

The decision to use both models highlights the strengths and weaknesses of different machine learning approaches. Naive Bayes excels in simplicity and speed, while Logistic Regression is better suited for complex patterns and richer datasets. I wanted also to use both to can in real-time how their predictions might align or differ. Their occasional disagreements reflect their distinct methodologies. Naive Bayes focuses on individual keywords, while Logistic Regression captures broader linguistic patterns.

API

The API was designed to host multiple endpoints, each linked to a specific dataset. These endpoints allow users to download datasets in CSV format dynamically and efficiently. Flask was chosen for its lightweight and flexible nature, making it an ideal framework for building and deploying this API.

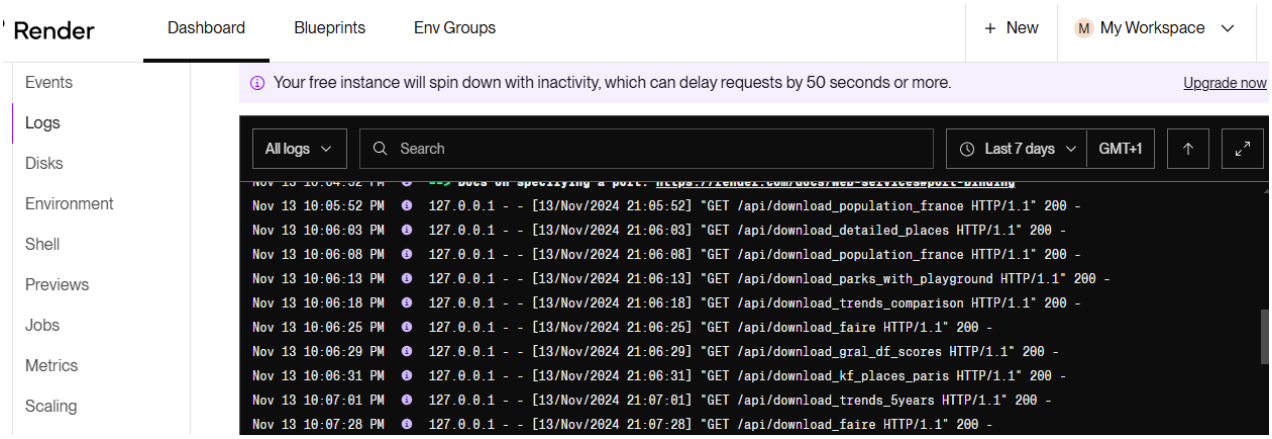
Key Features:

- The API allows users to download datasets relevant to the project, such as information on kid-friendly places, parks, events, and population trends. Each dataset has its dedicated endpoint, ensuring easy access.

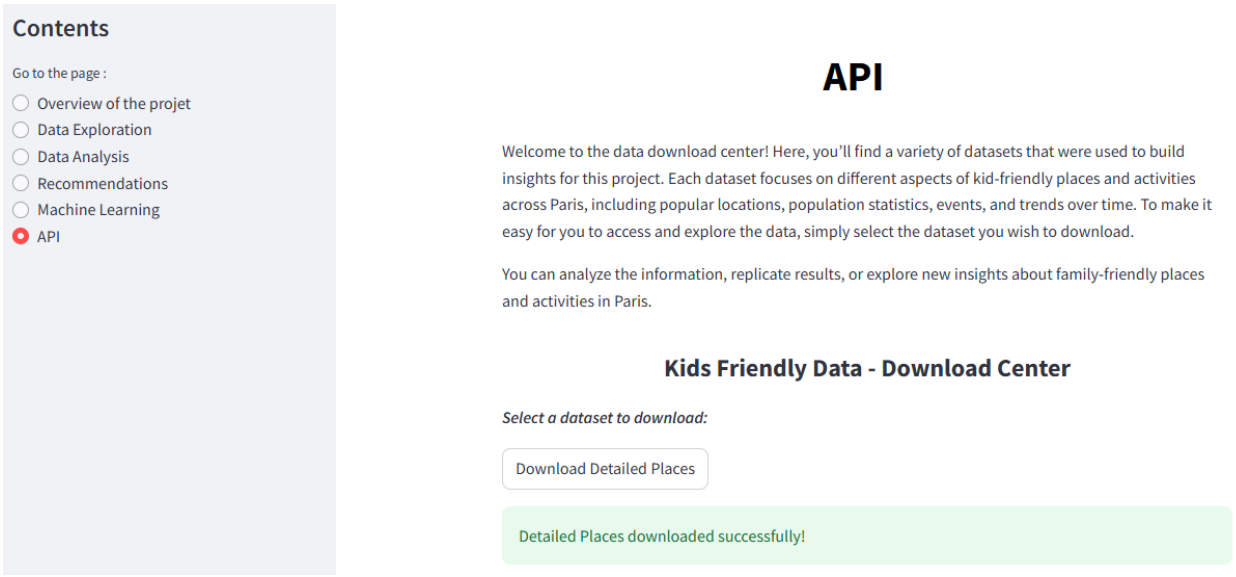
- Instead of saving files on the disk, the API uses in-memory buffers (io.BytesIO) to generate and serve CSV files directly. This approach reduces latency and avoids unnecessary file operations on the server.

```python
app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome to the Kids-Friendly Data API!"

# Download of df_detailed_places
@app.route("/api/download_detailed_places", methods=["GET"])
def download_detailed_places():
    df_detailed_places = pd.read_csv("ddetailed_kids_friendly_places_paris_with_reviews.csv")
    output = io.BytesIO()
    df_detailed_places.to_csv(output, index=False)
    output.seek(0)
    return send_file(output, mimetype='text/csv', as_attachment=True, download_name="detailed_places.csv")

# Download of trends_5years
@app.route("/api/download_trends_5years", methods=["GET"])
def download_trends_5years():
    trends_5years = pd.read_csv("ttrends_kids_friendly_paris_5years.csv")
    output = io.BytesIO()
    trends_5years.to_csv(output, index=False)
    output.seek(0)
    return send_file(output, mimetype='text/csv', as_attachment=True, download_name="trends_5years.csv")

# Download of population France
@app.route("/api/download_population_france", methods=["GET"])
def download_population_france():
    df = pd.read_csv("population_france.csv")
    output = io.BytesIO()
    df.to_csv(output, index=False)
```

- The API was deployed on Render, making it accessible online. The deployment process ensures that the API can handle user requests from various locations while integrating seamlessly into the Streamlit application.



API section into the streamlit application:

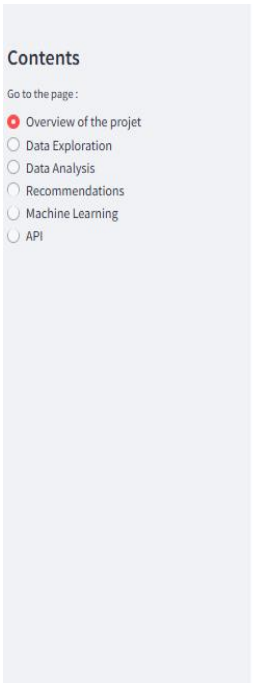


Streamlit

The core of my project is the “Kids-friendly application” in Streamlit, that wants to provide parents and caregivers a comprehensive tool to discover kid-friendly locations in Paris.

Here you will find the link to explore it:

https://kidsfriendlyapp.streamlit.app/



Welcome to the Kid-Friendly Places in Paris App!

This application is designed to help parents and caregivers discover and explore kid-friendly spots across Paris, whether you're searching for restaurants, parks, events, or other family-friendly places. The app is structured to offer a complete analysis of various locations and activities that are welcoming to children. Here's a brief overview of what each section provides:

1. Overview of the projet:

This page gives an introduction of the application and its purpose.

2. Data Exploration:

In this section, you'll find details about the datasets used for the project, including their sources, allowing transparency in data provenance.

3. Data Analysis:

Here, you can explore detailed analyses on Paris' population by arrondissement, monthly trends in Google searches for kid-friendly places, and evaluations of events and kid-friendly locations by year or arrondissement.

Conclusions

In this project, I addressed two key aspects of creating a family-friendly experience in Paris: identifying accessible locations for families and predicting new kid-friendly options using data-driven insights.

The first aspect focused on analyzing existing data to uncover patterns in how family-friendly places (such as restaurants, parks, and events) are distributed across Paris. The visualizations highlighted disparities between arrondissements, with areas like the 15th and 19th arrondissements standing out for their higher concentration of family-friendly locations. Seasonal trends in events and Google search patterns revealed that interest in kid-friendly activities peaks during warmer months, aligning with school holidays and family outings. This analysis provided a clear understanding of where families can access resources and when demand is likely to rise.

The second aspect leveraged machine learning to predict whether non-categorized restaurants might still be suitable for children. By analyzing customer reviews, models such as Naive Bayes and Logistic Regression captured sentiment patterns that helped classify these locations. This extended the value of the application by offering additional, data-backed recommendations for parents.

Together, these findings and tools culminated in an interactive Streamlit application that combines analysis, recommendations, and machine learning insights. The app simplifies the decision-making process for parents and caregivers, enabling them to plan outings more effectively.

This project demonstrated how integrating multiple datasets, cleaning and normalizing them, and applying advanced analytics can result in practical solutions for real-world problems. It also underscored the importance of balancing technical rigor with user-focused design, ensuring the insights are both meaningful and actionable for end-users.

GDPR

All datasets were anonymized and collected from public sources, ensuring compliance with GDPR guidelines. No personal or sensitive data was used.

References

Scraping:

- https://www.linternaute.com/ville/classement/villes/population?page=
- https://fr.wikipedia.org/wiki/Arrondissements_de_Paris

API

- https://developers.google.com/maps/documentation/places/web-service/overview?hl=fr
  https://pypi.org/project/pytrends/

Flat files

- https://www.kaggle.com/code/mpwolke/que-faire-paris?select=que-faire-a-paris-.csv
- https://opendata.paris.fr/explore/dataset/espaces_verts/export/?disjunctive.type_ev&disjunctive.categorie&disjunctive.adresse_codepostal&disjunctive.presence_cloture

GitHub Link

All code and documentation for the project are available on GitHub :

https://github.com/XinlyR/Kids_friendly_app