



## Events in Paris

Name: Mathieu JOMAIN

June, 2022

## Table of Contents

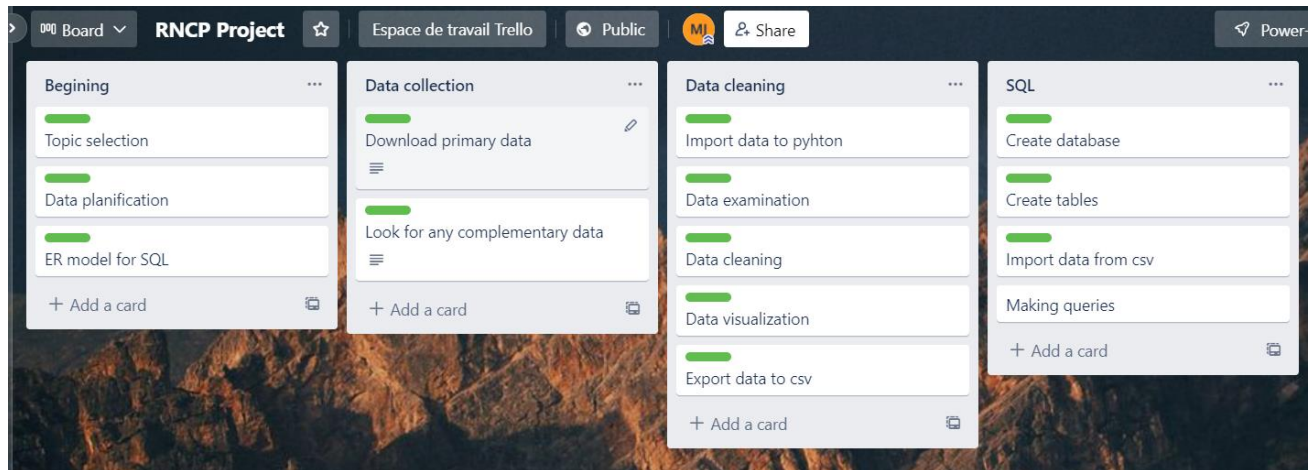
Introduction.....	3
Data .....	3
Intro, data sources and meta data .....	3
Cleaning .....	5
Data visualization .....	8
Data exportation .....	10
SQL.....	10
ER Model .....	10
Database and tables .....	11
Importing data to database.....	12
Queries .....	13

## Introduction

The project is the following: find and manipulate data on the subject of our choice.

The goal is to realize the different steps of the creation of a database, starting with the collection of data, the cleaning, the visualization and finally the creation of a database with several tables.

The first step is to plan the project using a project management tool. I chose to use Trello. You can find my board here : <https://trello.com/b/z8nmgMzG/rncp-project>



## Data

Intro, data sources and meta data

In order to realize this project, I went to the Paris Data website, which provides a lot of free data. I chose data about events taking place in Paris.

This data set is available online on the Opend Data Paris website, or you can reach it using this link:

[https://opendata.paris.fr/explore/dataset/que-faire-a-paris-/information/?disjunctive.tags&disjunctive.address\\_name&disjunctive.address\\_zipcode&disjunctive.address\\_city&disjunctive.pmr&disjunctive.blind&disjunctive.deaf&disjunctive.transport&disjunctive.price\\_type&disjunctive.access\\_type&disjunctive.programs](https://opendata.paris.fr/explore/dataset/que-faire-a-paris-/information/?disjunctive.tags&disjunctive.address_name&disjunctive.address_zipcode&disjunctive.address_city&disjunctive.pmr&disjunctive.blind&disjunctive.deaf&disjunctive.transport&disjunctive.price_type&disjunctive.access_type&disjunctive.programs).

In addition, I also used another data source regarding postal codes / city name / insee code, from the Data Gouv website; available here:

<https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/>

I chose this data because on the one hand it is a very tangible subject, so it is natural to understand the content of the columns.

On the other hand, it is a database with very little numerical data. This makes, in my opinion, the handling more complex, which makes it a real challenge for me.

Meta data 1: events

Column name	Explanation
ID	Basically an unique integer number for each event
URL	Website link where we can find information about the event
Titre / chapeau	Name and some information about the event and what it is about
Date début / fin	Starting and ending date of the event
Occurences	At first list of dates of the event, then turned into the number of occurences
Mots cles	Keyword to indicate the nature / related field of the event
Nom / adresse du lieu	Name of the place and adress
Postal code / ville	Postal code and city of the adress
Access PMR / mal voyant / mal entendant	If the location suits people with different disabilities needs
URL / telephone / email de contact / facebook	Website link, phone number, email address and facebook page of the organisator
Type de prix	Field to know if the event as either a paid or a free entrance

Meta data 2: insee

Column name	Explanation
Code insee	INSEE unique code of the location (something like a postal code)
Nom de la commune	Name of the city
Code postal	Postal code of the city
Coordonnées GPS	Exact GPS coordinates

## Cleaning

The first step we have to do once we have the data store on our computer is to import the data to python using the command read from pandas.

Then we start by exploring the data to try to better understand how the information is structured, what are their types....

```
#Checking every column data types to get an overall idea of the content  
df.dtypes
```

ID	int64
URL	object
Titre	object
Chapeau	object

According to me, the next step is to identify potential columns to be deleted for lack of relevance.

```
# We will continue to drop column that we identified as not relevant  
drop_col_list = ['Description', 'Description de la date', "URL de l'image",  
                 "Texte alternatif de l'image", "Crédit de l'image", 'Coordonnées géographiques',  
                 'Détail du prix', 'Image de couverture', 'title_event']  
  
for i in drop_col_list:  
    del df[i]
```

Another factor that can lead to the removal of a column is its lack of information. We should therefore consider looking at the missing values of the whole dataframe.

```
#Checking missing values to see if some columns have more than 50% of missing values  
df.isna().sum()
```

ID	0
URL	0
Titre	0
Chapeau	9
Description	27
Date de début	94
Date de fin	94
Occurrences	238

I then did some formatting on the columns containing text, to reduce the case of the characters and to remove the potential accents.

```
#Lower content of some columns
lower_list = ['Titre', 'Chapeau', 'Mots clés', 'Nom du lieu', 'Adresse du lieu', 'Ville']

def lower_content(data, list):
    for i in list:
        data[i] = data[i].str.lower()

lower_content(df, lower_list)
```

```
#In this step we are going to change every letter with an accent by the normal letter. That way we won't have problem with sql afterwards

accent_col_list = ['Titre', 'Chapeau', 'Mots clés', 'Nom du lieu',
                  'Adresse du lieu', 'Ville', "Type d'accès"]

df[accent_col_list] = df[accent_col_list].apply(lambda x: x.str.normalize('NFKD').str.encode('ascii', errors='ignore').str.decode('utf-8'))
```

I then transformed the information contained in text in different ways. I extracted the information from the date columns in order to have only one date and delete the additional useless content.

```
#Working on the content of column Date de début and Date de fin
df['Date de début'] = df['Date de début'].str.findall(r'[0-9]{4}-[0-9]{2}-[0-9]{2}')

#Dropping column with date as list + create new column date
df1 = df.apply(lambda x: pd.Series(df['Date de début'][0]),axis=1).stack().reset_index(level=1, drop=True)
df1.name = 'Date debut'
df=df.drop('Date de début', axis=1).join(df1)
```

I have cleaned the list of keywords linked to each event by keeping only one value, in order to be able to process this information with much more ease.

```
#Making some changes in Mots clés to only keep one tag per row

activities_list = ['musique', 'conférence', 'expo', 'sport', 'théâtre', 'enfants',
                  'balade', 'loisirs', 'art', 'atelier', 'cinéma', 'spectacle musical',
                  'littérature', 'clubbing', 'cirque', 'nature', 'concert']

#function to replace activities by a single given activity
def keywords(alist):
    for activity in alist:
        s='.*(?='+str(activity)+').*'
        # print(s) this was to print my patterns to check if it's working
        df['Mots clés'] = np.where(df['Mots clés'].str.contains(activity), df['Mots clés'].str.replace(s, activity, regex=True), df['Mots clés'])
```

I have made changes to the phone number and postal code columns by removing potential spaces, letters and any other unwanted symbols.

```
#We need to replace spaces and dots by nothing
df['Téléphone de contact'] = df['Téléphone de contact'].str.replace(' ', '', regex=True)
df['Téléphone de contact'] = df['Téléphone de contact'].str.replace('.', '', regex=True)
```

I changed the nature of the content of the "occurrence" column to have a number instead of a list of dates.

```
#Do some changes in the Occurrences column to actually have a count instead of dates
df['Occurrences'] = df['Occurrences'].str.findall(r'[0-9]{4}-[0-9]{2}-[0-9]{2}')
```

```
df['Occurrences']=df['Occurrences'].replace(np.nan, 0)
```

```
df['Nb occurrences'] = None
for i, k in zip(df['Occurrences'], range(0, len(df['Occurrences']))):
    if i != 0:
        x=len(i)
        df['Nb occurrences'][k]=x

df.drop('Occurrences', axis=1)
```

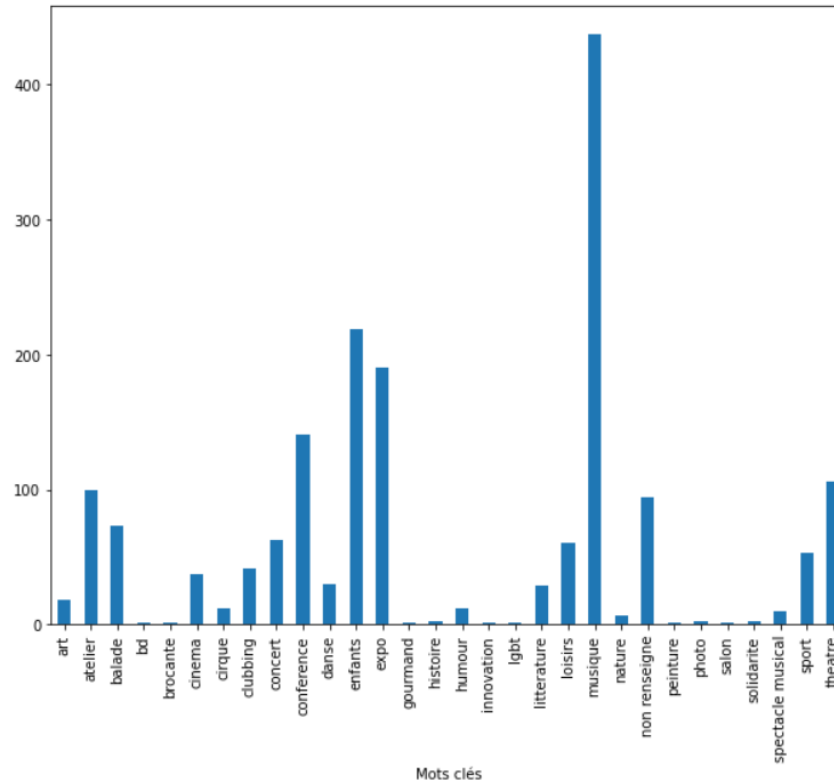
In general, in data cleaning, there is a large part of the work that corresponds to the treatment of null values. Because I have mainly categorical values in my data, I had to change the NaN values by something else. Here I chose to replace it by “non renseigné”.

```
#Replace missing values in columns by non renseigné
col_missing_values = ['Chapeau', 'Mots clés', 'Nom du lieu', 'Adresse du lieu',
                      'Ville', 'Url de contact', 'Téléphone de contact',
                      'Email de contact', 'URL Facebook associée', 'Type de prix',
                      "Type d'accès", 'URL de réservation']

def replace_values(col_list):
    for i in col_list:
        df.loc[(df[i].isnull()), i] = 'non renseigné'
```

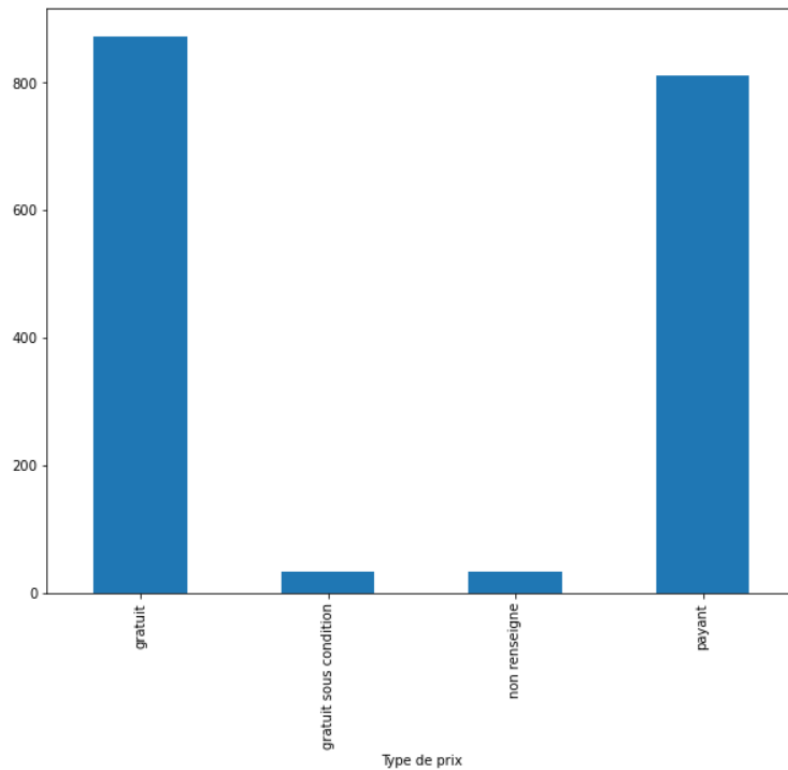
## Data visualization

After finishing the data cleaning, I made some graphs in order to have a global idea of the data distribution. The first graph is the repartition of keyword. We can see that events related to music are dominant.





The next thing I wanted to know is the distribution of paid and free entrance. I was surprised to see that the repartition is almost even.



Then I used crosstabs to gather some more information. Furthermore, this way is more suitable to analyze categorical data.

The first one is about the repartition of type of entrance for different location, here I used postal code. We can see that in the district 1 of Paris, there is almost 2 paying events for 1 free event.

Code postal	75000	75001	75002	75003	75004	75005	75006	75007	75008	75009	...
<b>Type de prix</b>											
<b>gratuit</b>	60	55	3	38	64	47	9	8	23	18	...
<b>gratuit sous condition</b>	1	15	0	2	3	1	0	2	0	0	...
<b>non renseigné</b>	13	1	1	3	5	1	0	0	1	2	...
<b>payant</b>	38	98	10	50	36	40	14	13	26	19	...

## Data exportation

And finally, I decided to create what will be my future database tables. I used the same process for my four tables: create a list with the columns that I want and then create a new dataframe with only those elements.

```
#Table organisator
col_organisator = ['URL', 'Url de contact', 'Téléphone de contact', 'Email de contact', 'URL Facebook associée']
df_organisator = df[col_organisator]
df_organisator
```

Once the new dataframe is set, I exported it as csv file.

```
#Exporting that table to csv
df_organisator.to_csv(r'C:\Users\matui\Downloads\organisator_table.csv', index=False, sep=';')
```

## SQL

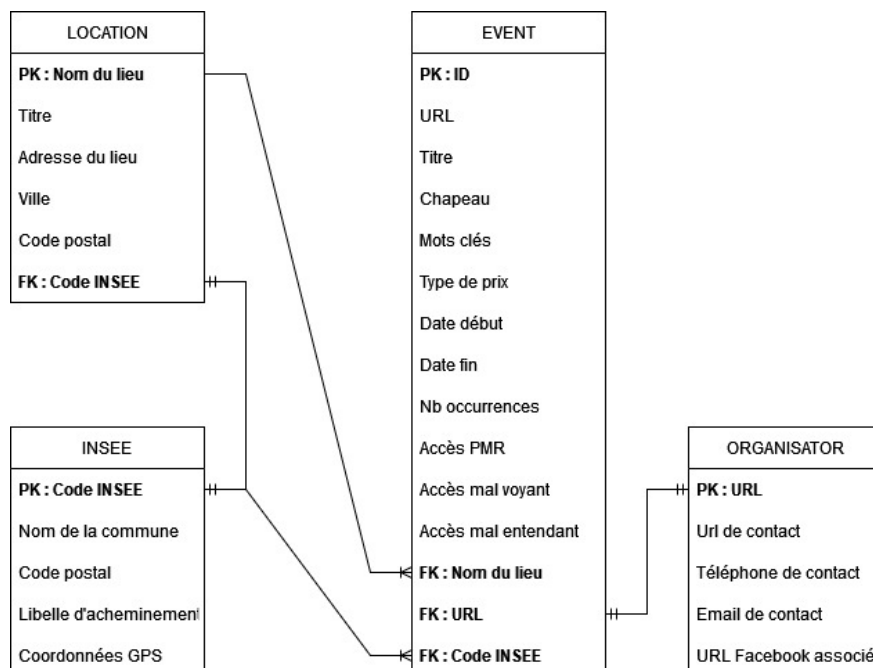
### ER Model

Before diving into data cleaning and visualization in python, it is important to create an initial entity relationship diagram.

This allows us to have some guidelines. It is indeed better to handle the data when we have a first idea of the final result.

This ER model will then change according to the elements we obtain and the relationships we can set up between them.

Here is my final diagram:



## Database and tables

The first step in working with SQL is to create the database. After that we need to let SQL know that we want to do some work on that particular database by writing the code “use”.

Now we can create the tables that we want by defining the columns names and their data type. We can select a column to be the primary key and also a foreign key from another table.

```
create database if not exists paris_event;  
use paris_event;
```

```
create table if not exists insee (  
  code_insee varchar(50),  
  nom_commune varchar(100),  
  code_postal int,  
  libelle varchar(100),  
  coord_gps varchar(100),  
  PRIMARY KEY (code_insee)  
);
```

Example of a more complex table:

```
create table if not exists event (  
  ID int,  
  titre varchar(100),  
  chapeau varchar(100),  
  mots_cles varchar(100),  
  type_prix varchar(100),  
  date_debut date,  
  date_fin date,  
  nb_occurrences int,  
  acces_pmr int,  
  acces_mal_voyant int,  
  acces_mal_entendant varchar(100),  
  lieu varchar(100),  
  code_insee int,  
  url varchar(100),  
  PRIMARY KEY (ID),  
  FOREIGN KEY (lieu) references location(lieu),  
  FOREIGN KEY (code_insee) references insee(code_insee),  
  FOREIGN KEY (url) references organisator(url)  
);
```

## Importing data to database

There are many ways to import data into SQL:

- create the table and import the data



- or with a code in SQL

```
LOAD DATA INFILE 'C:\Users\matui\Downloads\insee.csv'
INTO TABLE insee
FIELDS TERMINATED BY ';'
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

- or even with a code in python, but it requires some extra steps (I used that way)

First, we need to initialize the connexion between python and sql.

```
#Enabling a connection between python and sql
conn = sql.connect(host = "localhost",
                   user = "root",
                   password = "007m008Mat!",
                   database = "paris_event")
```

```
#Second way to create the connection
engine = db.create_engine('mysql://root:007m008Mat!@localhost:3306/paris_event')
```

Then we need to create a dataframe from the cleaned csv file and export it to SQL.

```
#Import table
df1 = pd.read_csv(r'C:\Users\matui\Downloads\organisator_table.csv', sep=';')
```

```
#Export to sql
df1.to_sql('organisator', engine, if_exists='replace', index=False)
```

## Queries

### Query 1:

Elements used: email, ID, titre (title of the event)

This first query shows how to reach a specific element from another table and add it to the input we want. Here we wanted to access the email addresses of organisers of events related to sports that occur more than 2 times.

### Query 2:

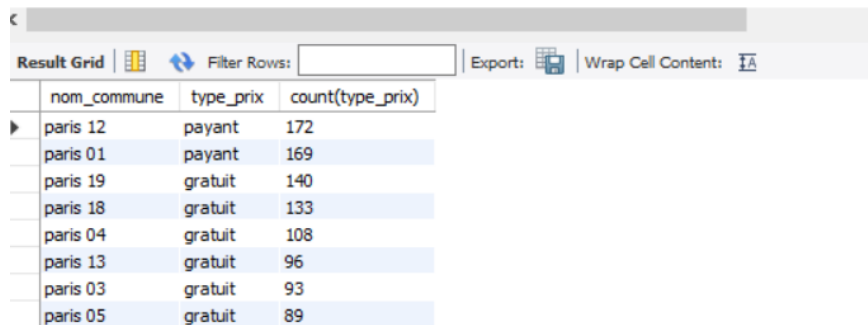
Elements used: keyword + count

For the second query, I wanted to classify the events according to the keywords in order to know which ones works the best.

We can see that the events related to music are strongly dominant, more than 400, almost twice the number of events related to the second biggest category (which are the activities for children)

### Query 3:

```
19 #Query 3: Where most of events take place based on the type of price
20 • select nom_commune, type_prix, count(type_prix)
21 from event
22 join insee
23 on insee.code_insee = event.code_insee
24 group by nom_commune
25 order by count(type_prix) desc;
```



The screenshot shows a 'Result Grid' interface with a table of event data. The table has three columns: 'nom\_commune', 'type\_prix', and 'count(type\_prix)'. The data is sorted by the count in descending order. The first row is Paris 12 with 172 paid events, followed by Paris 01 with 169 paid events, Paris 19 with 140 free events, Paris 18 with 133 free events, Paris 04 with 108 free events, Paris 13 with 96 free events, Paris 03 with 93 free events, and Paris 05 with 89 free events.

nom_commune	type_prix	count(type_prix)
paris 12	payant	172
paris 01	payant	169
paris 19	gratuit	140
paris 18	gratuit	133
paris 04	gratuit	108
paris 13	gratuit	96
paris 03	gratuit	93
paris 05	gratuit	89

Here I wanted to know which district of Paris is the best for each type of budget. In one hand I want to know the district where the most of events (paid entrance or free entrance) take place.

We can see that in Paris 12e there are 172 paid events, followed closely by Paris 1e with 169 paid events.

As for the free events, in the 19th district of Paris there are 140 free events and 133 for Paris 18th.

With this analysis you can identify the places to go to participate in events according to your budget.

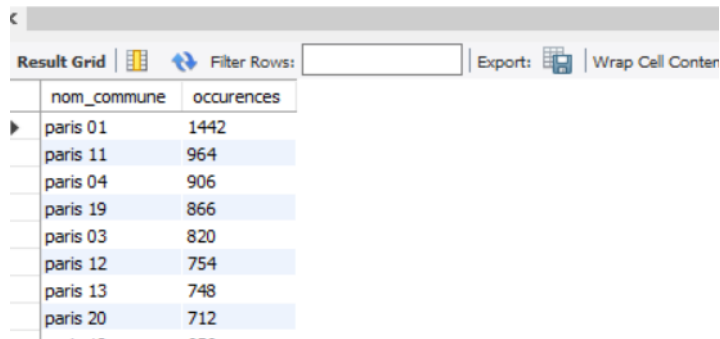
#### Query 4:

That query is very similar to query 3, but instead of displaying the type of price I wanted to know the repartition of events within the districts by the keyword.

With the output we can see that if you want to go to an event related to art for example, the best thing is to go in Paris 13e.

#### Query 5:

```
37 #Query 5: Again same thing but with occurrences
38 • select nom_commune, sum(nb_occurrences) as occurrences
39 from event
40 join insee
41 on insee.code_insee = event.code_insee
42 group by nom_commune
43 order by sum(nb_occurrences) desc;
```



The screenshot shows a software interface with a SQL query editor at the top and a 'Result Grid' below it. The query is a SQL statement that joins 'event' and 'insee' tables, groups by 'nom\_commune', and orders by the sum of 'nb\_occurrences' in descending order. The 'Result Grid' displays a table with two columns: 'nom\_commune' and 'occurrences'. The data is sorted by the number of occurrences, with 'paris 01' having the highest count at 1442, followed by 'paris 11' at 964, 'paris 04' at 906, 'paris 19' at 866, 'paris 03' at 820, 'paris 12' at 754, 'paris 13' at 748, and 'paris 20' at 712. The interface also includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox.

nom_commune	occurrences
paris 01	1442
paris 11	964
paris 04	906
paris 19	866
paris 03	820
paris 12	754
paris 13	748
paris 20	712

The last query shows the sum of occurrences in each district in order to know in which district takes place the most events. Thanks to the last 3 queries you can select the district that suits the best your desires in term of content, price and frequency.