Paris Rental Market Data Extraction

Web crawling with Scrapy ▪ Studapart + La Carte des Colocs

Data Extraction Project

Université Paris Dauphine-PSL

December 2025

github.com/Fatima-alharake/Data_Extraction_French_Rentals

Project Objective

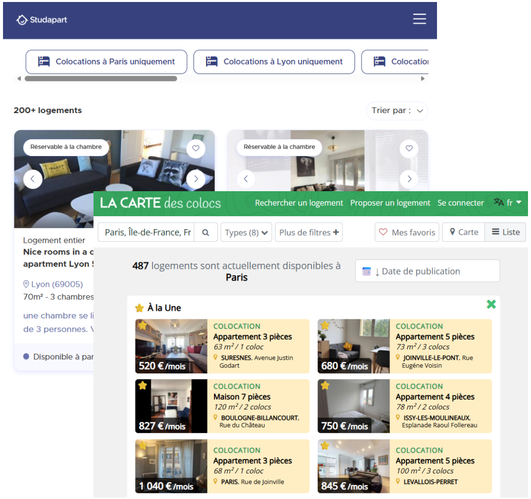
Goal

Extract rental listing data from major French platforms and build a dataset for price analysis.

Target websites

- **Studapart** – student housing
- **La Carte des Colocs** – shared housing (colocation)

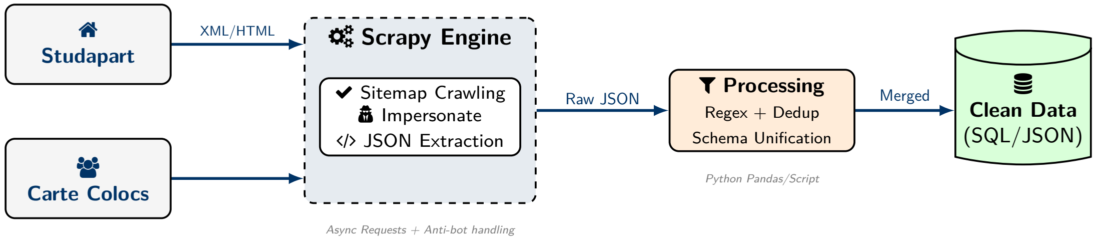
Other sites like Leboncoin were excluded. Their policy states: *`# It's forbidden to use search robots or other automatic methods to access Leboncoin.fr`*



520 € /mois — COLOCATION Appartement 3 pièces, 63 m² / 1 coloc, SURESNES, Avenue Justin Godart	680 € /mois — COLOCATION Appartement 5 pièces, 73 m² / 3 colocs, JOINVILLE-LE-PONT, Rue Eugène Voisin
827 € /mois — COLOCATION Maison 7 pièces, 120 m² / 2 colocs, BOULOGNE-BILLANCOURT, Rue du Château	750 € /mois — COLOCATION Appartement 4 pièces, 78 m² / 2 colocs, ISSY-LES-MOULINEAUX, Esplanade Raoul Follereau
1 040 € /mois — COLOCATION Appartement 3 pièces, 68 m² / 1 coloc, PARIS, Rue de Joinville	845 € /mois — COLOCATION Appartement 5 pièces, 100 m² / 3 colocs, LEVALLOIS-PERRET

End-to-end Data Pipeline

1. Target Sites



Ingestion
Multi-source crawling using sitemaps for efficiency.

Normalization
Unified JSON schema (Price, Lat/Lon, Surface).

Technical Architecture & Compliance

Scraping Engine

To ensure reliability and data integrity, we implemented a custom Scrapy stack:

- **Asynchronous I/O:** Using `AsyncioSelectorReactor` for high-performance non-blocking requests.
- **Politeness Policies:** Strict rate limiting (`DOWNLOAD_DELAY = 1`) to prevent server overload.

TLS Fingerprinting

Although *La Carte des Colocs* permits bots in `robots.txt`, standard Python requests were rejected due to TLS/SSL fingerprinting (anti-bot protection).
Solution: We integrated **Scrapy-Impersonate** to emulate a standard browser signature (`safari15_5`), ensuring legitimate access to public pages.

Ethical Compliance

- **Robots.txt:** We respect `User-agent: *`.
- **Scope:** Only public listing URLs accessed.
- **Excluded:** No access to `/admin`, `/sidekiq`, or private user data.

Goal: Replicate human browsing behavior to gather open data without disrupting service.

Adaptive Strategy: Handling Different Structures

1. Studapart: Standard Approach

- **Navigation:** Uses the native `SitemapSpider`. It automatically follows all URLs matching the `/fr/` rule.
- **Extraction:** Visual parsing. We scrape what the user sees in the DOM.
- **Logic:** Requires **Regex** to clean formatting (spaces, € symbols).

2. Carte des Colocs: Hybrid Approach

- **Navigation:** Manual XML parsing. We filter the sitemap for specific keywords (e.g., `/paris/`) *before* generating requests.
- **Extraction:** Intercepts a hidden **JSON** object embedded in the HTML.
- **Logic:** Direct access to typed data (int/float).

```
# 1. Scrape visual text
raw = response.css(".price::text").get()

# 2. Clean with Regex (Risk of errors)
price = re.search(r"\d+", raw).group()
```

```
# 1. Extract hidden JSON string
raw = response.css('#d::attr(data-json)').get
    ()
data = json.loads(raw)

# 2. Key access (100% reliable)
price = data.get('cost_total_rent')
```

Scrapy Settings

```python
# Polite crawling
CONCURRENT_REQUESTS_PER_DOMAIN = 1
DOWNLOAD_DELAY = 1

# Browser impersonation
DOWNLOAD_HANDLERS = {
 "http":  "scrapy_impersonate.
      ImpersonateDownloadHandler",
 "https": "scrapy_impersonate.
      ImpersonateDownloadHandler",
}

# Reactor (impersonation-compatible)
TWISTED_REACTOR = (
 "twisted.internet.asyncioreactor.
      AsyncioSelectorReactor"
)
```

Why these settings?

- **Delay + low concurrency**: reduce risk of blocks
- **Impersonation handler**: realistic browser fingerprint
- **Asyncio reactor**: required for the chosen handler

Unified Output Schema

One record = one listing

Field	Meaning
`AdUrl`	Canonical URL (primary key)
`AdTitle`	Listing title
`RentalPrice_EUR`	Monthly rent (EUR)
`RentalAddress`	Address / area string (raw)
`Latitude`	**Geo Latitude (decimal)**
`Longitude`	**Geo Longitude (decimal)**
`RentalSize_m2`	Size (m^2)
`RentalRooms`	Number of rooms
`RentalType`	Type (Appt, Duplex, Studio...)
`Furnished`	Meublé or null

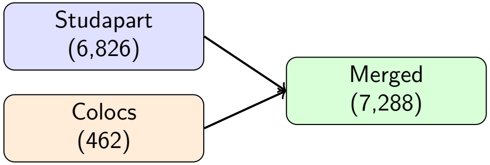
Merging the Data Sources

Inputs (from the crawl)

- `output_studapart.json` — **6,826** records (all France)
- `data_paris.json` — **462** records (Paris only)

Merge strategy (what we keep)

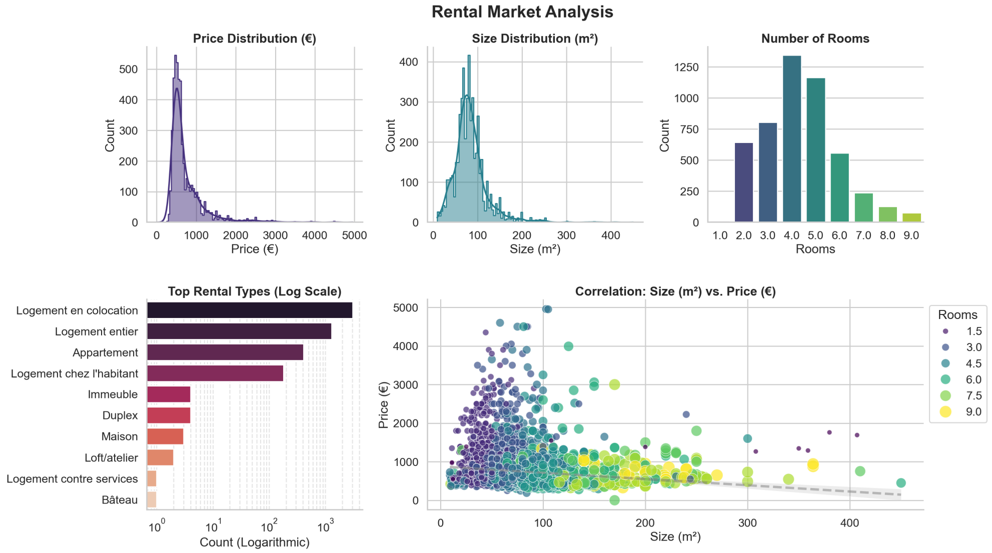
- Normalize into the unified schema
- Create unique `id` (URL hash) + add `source`



Output

`merged_rentals.json`

Exploratory Data Analysis



SQL: Different Databases Per Sites

One table per site

`AdId` (primary key)	Internal surrogate key
`AdUrl`	Canonical URL (unique)
`AdUrlHash`	Automatic hash
`AdTitle`	Listing title
`RentalPrice_EUR`	Monthly rent (EUR)
`RentalAddress`	Address / area string (raw)
`Latitude`	**Geo latitude (float)**
`Longitude`	**Geo longitude (float)**
`RentalSize_m2`	Size (m^2)
`RentalRooms`	Number of rooms
`RentalType`	Type (Appt, Duplex, Studio)
`Furnished`	Furnished or null

Notes on design choices

- Site-specific schemas avoid excessive nulls in case of scraping more site specific data
- Primary keys must be immutable; URLs can change
- `AdUrlHash` prevents duplicates
- Null inputs replaced by – for strings

SQL: Source Merged Database Schema (SQLite)

Table

```sql
CREATE TABLE rentals (
  id TEXT PRIMARY KEY,
  source TEXT,
  url TEXT,
  title TEXT,
  price_eur REAL,
  arrondissement TEXT,
  rental_type TEXT,
  latitude REAL,
  longitude REAL
);
```

Extracting Arrondissement

Problem

Addresses come in many formats:

- `75011 Paris`
- `Paris 11e Arrondissement`
- `Rue de Rivoli, Paris` (no number!)

Solution 1: Regex on Address

```python
# Pattern: postal code 750XX
match = re.search(r"750(\d{2})", address)
if match:
    num = int(match.group(1))
    if 1 <= num <= 20:
        return str(num).zfill(2)
```

Solution 2: GPS Coordinates

When address has no postal code, use lat/lon to find nearest arrondissement center.

```python
# Arrondissement centers (lat, lon)
CENTERS = {
  "01": (48.860, 2.342),
  "02": (48.868, 2.341),
  ...
  "20": (48.864, 2.398),
}

# Find nearest center
for arr, (clat, clon) in CENTERS.items():
    dist = sqrt((lat-clat)**2 + (lon-clon)**2)
    if dist < min_dist:
        nearest = arr
```

SQL Queries for Analysis

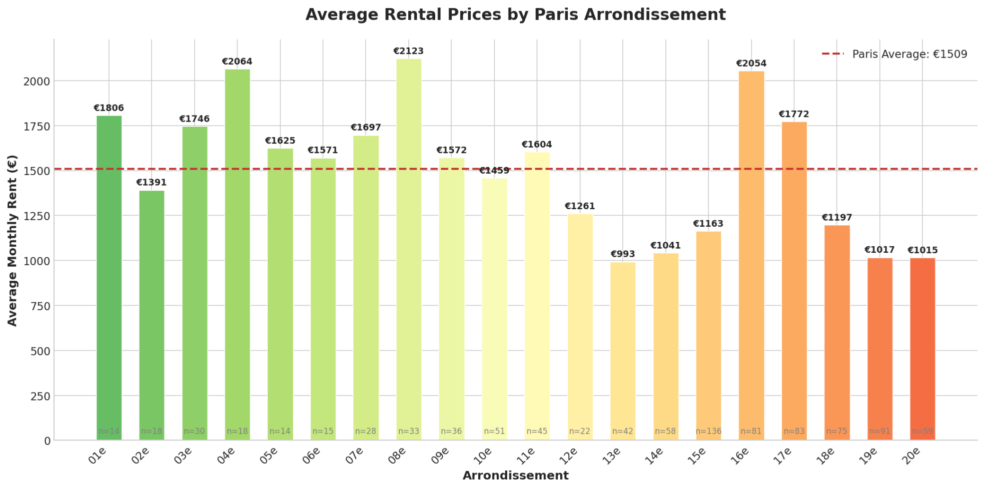
Query 1: Average Price by Arrondissement

```sql
SELECT arrondissement,
       COUNT(*) as listing_count,
       ROUND(AVG(price_eur), 2) as avg_price
FROM rentals
WHERE arrondissement IS NOT NULL
  AND price_eur > 0
GROUP BY arrondissement
ORDER BY CAST(arrondissement AS INTEGER);
```

Query 2: Shared Only (La Carte des Colocs)

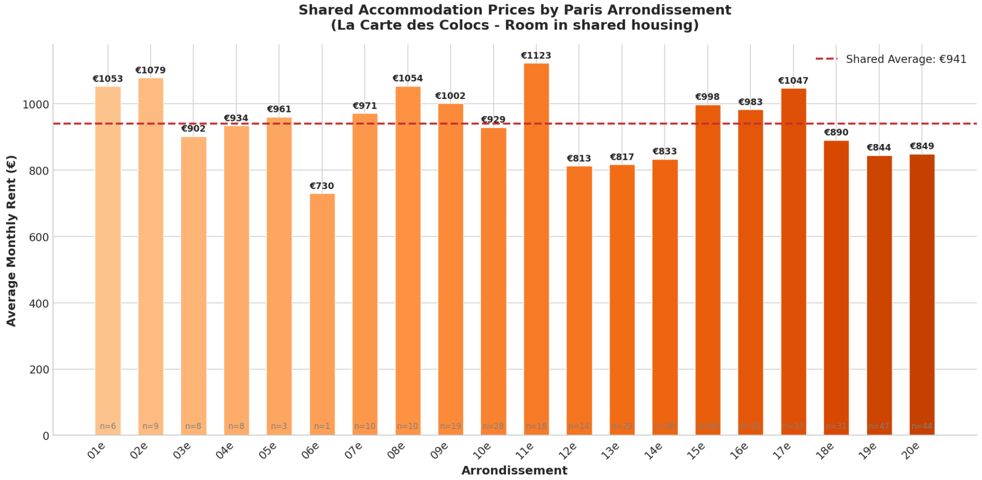
```sql
SELECT arrondissement, ROUND(AVG(price_eur), 2) as avg_price
FROM rentals
WHERE arrondissement IS NOT NULL
  AND source = 'lacartedescolocs'  -- Shared housing only
GROUP BY arrondissement;
ORDER BY CAST(arrondissement AS INTEGER);
```

Visualization 1: Price by Arrondissement (All Data)



Insight: Central arrondissements (1–8) and west (16e, 17e) show higher average rents.

Visualization 2: Price by Arrondissement (Shared Only)



Thank You!

Questions?