

# How have I (datascientist) been searching for a rent in Edinburgh

PETER ALACS, BDP

PYDATA – 25<sup>TH</sup> MARCH 2019

My Landlord is selling her flat and I have to move out  
(the pic is an illustration)

Competitive market

I have to do multiple viewings / applications

I need an efficient way to find my new place

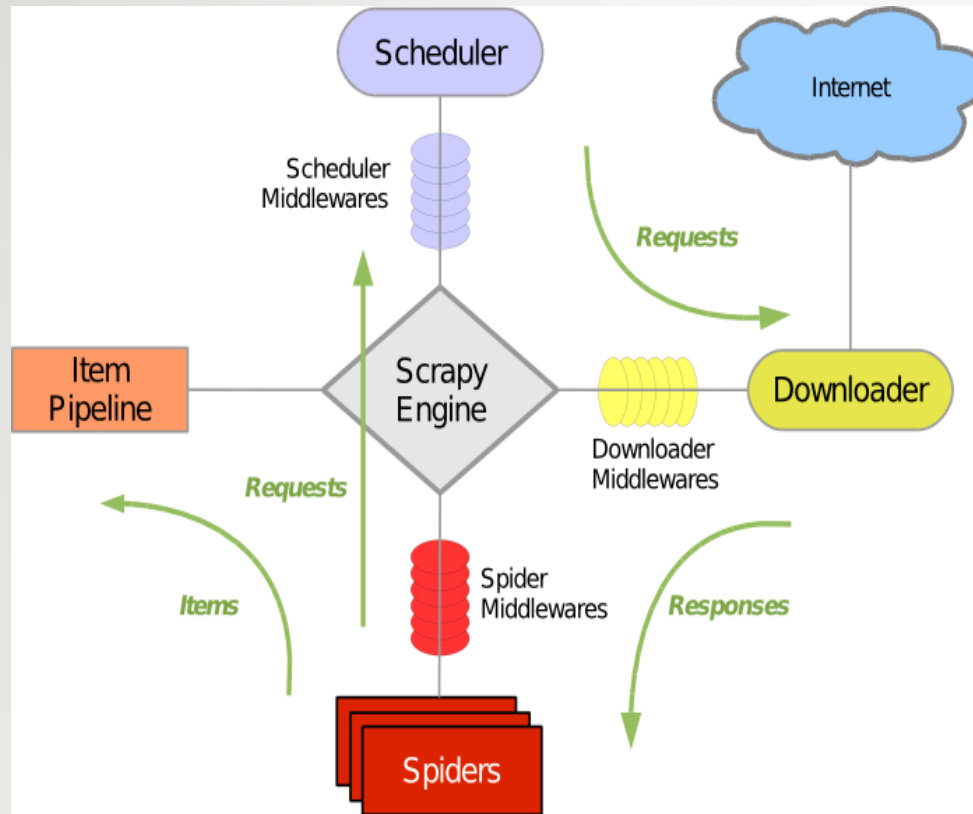


[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

# How to spice up flat-hunting? (for geeks only)

- The data:
  - Coming from multiple (4+) agencies
  - Possible augmentations?
- The model:
  - Is the price reasonable?
  - Could I construct a score that would rank my queries according to my preferences?
- The process:
  - I need a good overview of the opportunities and the upcoming

# The agencies' DATA: go Scrapy!



```
import scrapy
from scrapy.loader import ItemLoader
from flapp.items import SouthSideFlatItem

class SouthSideSpider(scrapy.Spider):
    name = "southsideflats"

    def start_requests(self):
        urls = [
            'https://southsideflats.com/latest-properties/'
        ]
        for url in urls:
            yield scrapy.Request(url, callback=self.parse)

    def parse(self, response):
        flatlinks = response.xpath('//h3[@class="entry-title"]/a/@href').extract()[:3]
        for link in flatlinks:
            yield scrapy.Request(link, callback=self.parse1)

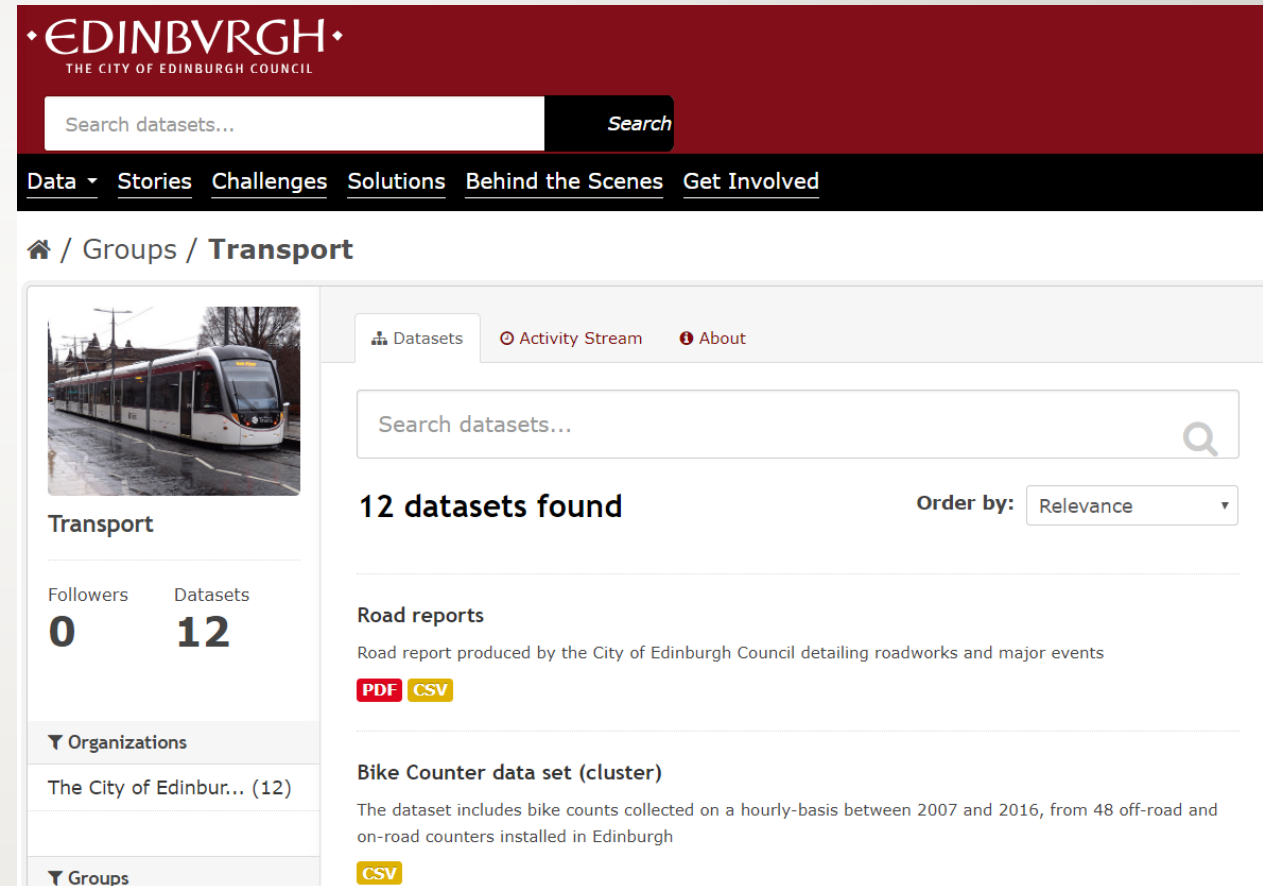
    def parse1(self, response):
        l = ItemLoader(item=SouthSideFlatItem(source=self.name, link=response.request.url), response=response)
        l.add_xpath('price', '//span[@class="page-price"]/text()')
        l.add_xpath('address_street', '//span[@class="item-street"]/text()')
        l.add_xpath('address_pcode', '//span[@class="item-pcode"]/text()')
        l.add_xpath('available', '//div[contains(@class,"date-available")]/text()')
        l.add_xpath('descr_short', '//div[contains(@class,"tab-content")]/h2[@class="entry-title"]/text()')
        l.add_xpath('description', '//div[contains(@class,"tab-content")]/p')

        return l.load_item()
```



# Open DATA

- UK government strategy
- <https://edinburghopendata.info/>
  - (not updated regularly)
  - Transportation: accidents
  - Planning (construction projects)
  - Public toilets
  - Etc...



The screenshot shows the Edinburgh Open Data website. The header is dark red with the Edinburgh City Council logo. Below the header is a search bar and a navigation menu. The main content area shows the 'Transport' group page, which includes a search bar, a list of datasets (12 found), and details for two datasets: 'Road reports' and 'Bike Counter data set (cluster)'.

**EDINBURGH**  
THE CITY OF EDINBURGH COUNCIL

Search datasets... **Search**

[Data](#) [Stories](#) [Challenges](#) [Solutions](#) [Behind the Scenes](#) [Get Involved](#)

[Home](#) / [Groups](#) / **Transport**

**Transport**

Followers **0** Datasets **12**

**Organizations**

The City of Edinburgh (12)

**Groups**

**Datasets** **Activity Stream** **About**

Search datasets... **Q**

**12 datasets found** **Order by:** Relevance

**Road reports**

Road report produced by the City of Edinburgh Council detailing roadworks and major events

**PDF** **CSV**

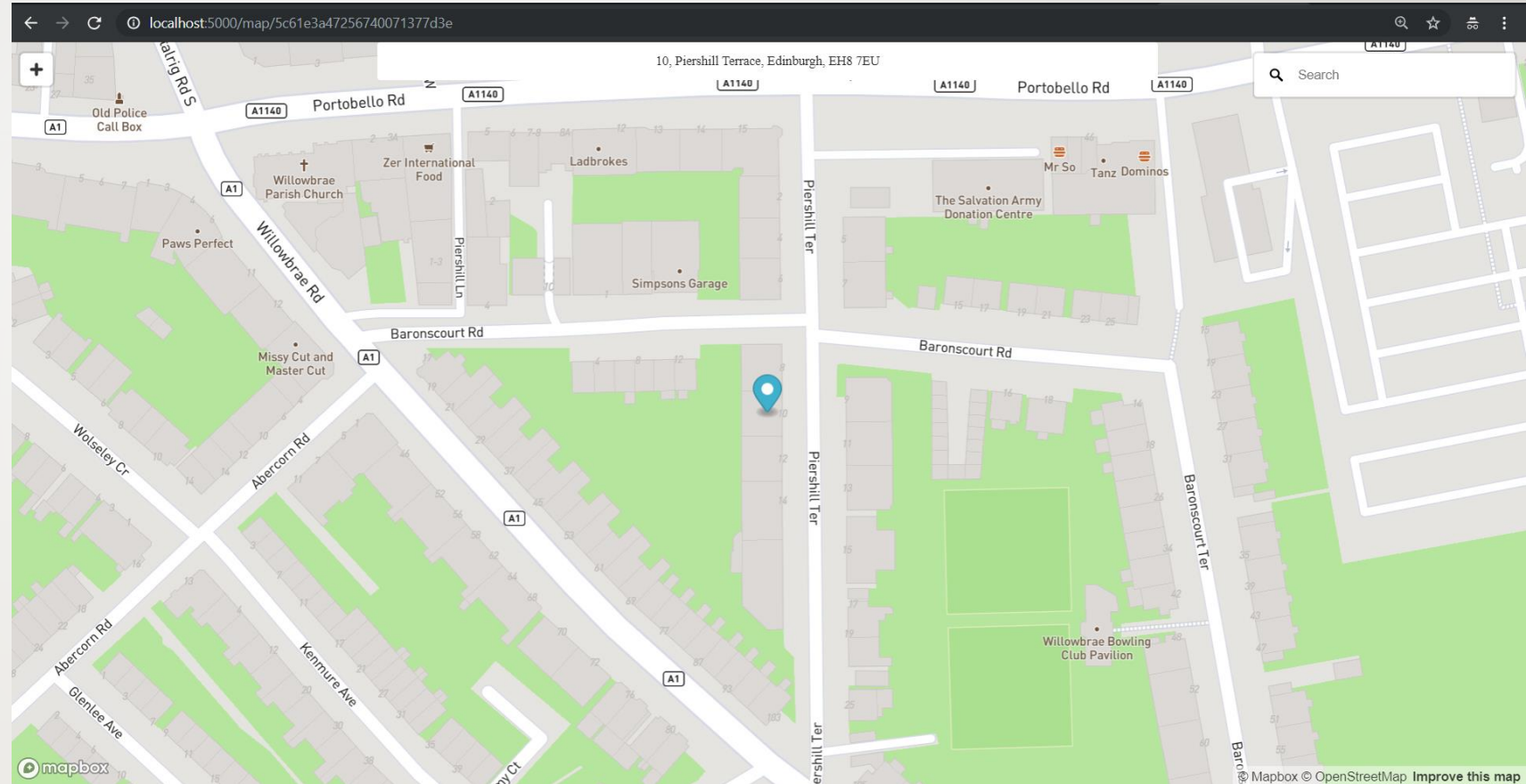
**Bike Counter data set (cluster)**

The dataset includes bike counts collected on a hourly-basis between 2007 and 2016, from 48 off-road and on-road counters installed in Edinburgh

**CSV**

# Maps

- Mapbox for visualizing
- Google Maps API for geocoding and data augmentation (e.g. distance from the centrum)



# The DATAbase: MongoDB

- NoSQL, Document-based
- Fits perfectly to ever changing / forming modelling needs, unknown data types (new agencies)
- MongoEngine ODR

```
class TaxCat(db.Document):
    cat = db.StringField()
    tax = db.DecimalField()
    def __unicode__(self):
        return f"{self.cat} (£{self.tax/12:.2f})"

class Viewing(db.EmbeddedDocument):
    created = db.DateTimeField(default = datetime.utcnow)
    viewingat = db.DateTimeField()
    contact = db.StringField()
    result = db.StringField()

    def __unicode__(self):
        return self.viewingat

class AddressLocation(db.EmbeddedDocument):
    address = db.StringField()
    lng = db.FloatField()
    lat = db.FloatField()
```

# The process: Flask

- Lightweight:
  - less code in the beginning,
  - Expandable: all components available, but need to manage it manually
- Supported by MongoEngine (e.g. admin-views: all creation, update, list views are free)



# The model: Contjoint Analysis

- Using the available Flask views compare 2 flats at once
  - Add tags for reasoning
  - Pick one as 'BETTER'
- Tags are converted to latent traits
- Price = Price Acknowledged + Extra

The screenshot shows a web application interface for comparing two flats. At the top, there is a navigation bar with tabs: Admin, Flats, Flat, Flat Tag, Tax Cat, and Con Joint. The 'Con Joint' tab is selected. Below the navigation bar, there are three sub-tabs: List, Create, and Edit. The 'Edit' tab is selected. The main form contains several fields:

- Created:** A text input field containing the date and time '2019-03-07 14:43:30'.
- Deleted:** An empty text input field.
- Flat1:** A dropdown menu showing 'Fala Place, Edinburgh, EH...'.
- Flat1Tags:** A text input field containing two tags: 'x quiet' and 'x nice'.
- Flat2:** A dropdown menu showing 'LOTHIAN ROAD, TOLLCR...'.
- Flat2Tags:** A text input field containing one tag: 'x furnished'.
- Is1Btr:** A dropdown menu showing 'Flat 1 is BETTER than Flat 2'.

At the bottom of the form, there are four buttons: 'Save' (blue), 'Save and Add Another' (grey), 'Save and Continue Editing' (grey), and 'Cancel' (red).

# Results

- Fun: a lot in the weekends with developing
- Views: some
- Rejected: 1
- Accepted: 0



# Morals

- The three great virtues of a programmer: laziness, impatience, and hubris. (Lary Wall)

Thank You!