Project Data Wrangling

Open Street Map: San Francisco

Inhaltsverzeichnis

1 Overview Dataset 1

1.1 OSM San Francisco 1

1.2 Facts 1

1.2.1 Nodes 1

1.2.2 Ways 1

1.2.3 Users 2

1.2.4 Restaurants or Attractions ..... **Fehler! Textmarke nicht definiert.**

2 Data Wrangling Python 3

2.1 Clean up Street (Code from Class) 3

2.2 Clean up City 4

2.3 Clean up Postcode 4

3 Data for SQLite 3 6

4 Dataanalysis SQL 9

5 Umsetzung ..... Fehler! Textmarke nicht definiert.

6 Evaluierung ..... Fehler! Textmarke nicht definiert.

7 Zusammenfassung und Ausblick ..... Fehler! Textmarke nicht definiert.

7.1 Erreichte Ergebnisse ..... Fehler! Textmarke nicht definiert.

7.2 Ausblick ..... Fehler! Textmarke nicht definiert.

7.2.1 Erweiterbarkeit der Ergebnisse ..... **Fehler! Textmarke nicht definiert.**

7.2.2 Übertragbarkeit der Ergebnisse ..... **Fehler! Textmarke nicht definiert.**

8 Literaturverzeichnis ..... Fehler! Textmarke nicht definiert.

Anlage 1 ..... Fehler! Textmarke nicht definiert.

Anlage 2 ..... Fehler! Textmarke nicht definiert.

1 Overview Dataset

1.1 OSM San Francisco

The analyzed dataset includes the Open Street Map Data of San Francisco and the Bay Area. OSM data is structured in nodes, ways and tags.

Nodes = points: A point is a geometrical point which is defined by his long- and latitude. Nodes can describe attractions, restaurants or other points of interest. Nodes can be start- or endpoints of a way.

Ways = polyline: These lists of nodes describes for examples roads or a river. A way has a start and an endpoint (node) and it can include 2 – 2000 nodes. Ways have a direction.

Tags = data element: A tag can have all types of data elements, which can be nodes, ways or relations. A tag includes all information's or attributes which describe for example a node. These attribute are for example a restaurant, there street, house number, which type of restaurant, for example Mexican and so on.

Nodes and ways are related to each other. This means a way has an "x-value" of nodes.
The cleaned .osm data will be stored into XLM for further analysis.

1.2 Facts

The OSM Data set of San Francisco has the following important data facts:

- Count Nodes
- Count Ways
- Count of different users
- Count of special data parts, like how many attractions and restaurants

1.2.1 Nodes

SQL-Query:

```
SELECT count(DISTINCT id)
FROM nodes;
```

Result: 3969715

1.2.2 Ways

SQL-Query:

```
SELECT count(DISTINCT id)
FROM ways;
```

Result: 492251

1.2.3 Users

```
SELECT count(distinct user)
FROM (
        SELECT distinct n.user
        FROM nodes n
        UNION
        SELECT distinct w.user
        FROM ways w
)
```

Result: 2190

1.2.4 Key Counts

```
SELECT key, count(key) as Count
FROM nodes_tags
GROUP BY key
ORDER BY Count DESC
```

	key	Count
1	highway	19691
2	street	19031
3	housenumber	18969
4	name	15134
5	city	14481
6	amenity	11022
7	crossing	9308
8	postcode	7764
9	state	6810
10	natural	5169
11	country	4332
12	shop	4149
13	created_by	3934
14	en	3387
15	taxon	3281
16	source	3047
17	operator	2691
18	railway	2299
19	website	2252
20	cuisine	2174

For San Francisco are 682 keys in the dataset available.

This result shows the frequency of each tag attribute that exists in the dataset. In this picture only the first 20 results are shown.

2 Data Wrangling Python

Data in general is not in a perfect way and can be analyzed. To solve this problem data wrangling is needed. In example for Data Warehousing this process has the name ETL (extract, transform, and load). This means the data, often from different sources, have to extract from the source, transform to the same shape / structure and loaded into the end system to analyze the date. “The same / structure” mean for example you have two different writings of a date, 31.12.2017 or 2017-12-31. We, as humans, now these are the same dates. An analyzing tool not, because if you run a query on the data 31.12.2017 the data for 2017-12-31 will be missing. The transformation step will clean this and bring it in the same shape.

In case of the San Francisco Open Street Map data only a few attributes will be cleaned, but normally all attributes has to be checked and cleaned for getting the best result when analyzing the data.

2.1 Clean up Street (Code from Class)

The first kind of data that will be cleaned is the street. In the first step I looked at all street names to see if there are typing mistakes or a lower and upper writing of a street, and so on.

To find these different writings of one street I printed in the StreetCleaningTest.py all Streets and there counts. Here is a part of the result:

```
67 AVE: 1
68 Ave: 17
69 Ave.: 3
70 Avenue: 12470
71 B: 2
72 Bldg: 1
73 Blvd: 7
```

For example Avenue, Boulevard, Highway and a lot of other Streets can be cleaned. (I’m not sure for what the letter “M” or “H” stays, so maybe I missed here some cleaning options!) For cleaning I decided in the case of Avenue that the “correct” writing for analyzing is “Avenue”. This means every Street which is an Avenue but has a different writing need to change. AVE, Ave and Ave. which are 21 datasets that will not be find when analyzing data on Avenue. After changing the Analysis on Avenue gives a correct result.

To clean the street a mapping is needed. The cleaning process and the mapping can be found in the audit.py.

Here are the examples for streets that need to be cleaned and in which way they will be expected:

```
33 expected = ["Street", "Avenue", "Boulevard", "Drive", "Place", "Road",
34             "Highway", "East", "Broadway", "Place", "Way"]
35
36 mapping = { "St": "Street",
37             "st": "Street",
38             "St.": "Street",
39             "street": "Street",
```

After running the cleaning process I get the following result:

```
Willie Stargell Ave. => Willie Stargell Avenue
Lincoln Ave. => Lincoln Avenue
Edes Ave. => Edes Avenue
```

(small part of the result)

Here is visible that the cleaning process works for the example Avenue.

2.2 Clean up City

For cleaning the names of the different cities of the bay area, I used the same process like for streets. First I printed all the city names and there counts (CityCleaningTest.py) which returns me the following list:

```
45 Alamda: 1
46 alameda: 1
47 Alameda: 132
48 Albany: 238
49 Beach: 3
50 Berkeley: 5772
51 berkeley: 2
```

There I looked at the different Cities and found some difference too. In the next step I cleaned the data in the same way like for street, with the following mapping:

```
110
111 city_mapping = { "Alamda": "Alameda",
112                  "alameda": "Alameda",
113                  "berkeley": "Berkeley",
114                  "Emeyville": "Emeryville",
115                  "oakland": "Oakland",
116                  "OAKLAND": "Oakland",
117                  "Okaland": "Oakland",
118                  "ca": "CA"
119                  }
```

2.3 Clean up Postcode

The last data that I cleaned is the Postcode; a postcode for San Francisco has a length of 5. Here I printed in the first step all postcodes and there counts too. (PostcodeCleaningTest.py)

```
94113: 80
94114: 371
94115: 87
94115-4620: 1
```

Here is visible that the postcode has in some cases a length greater than 5. In some cases there is a “CA” before the postcode.

In the audit.py the postcode will be also cleaned. The postcode has a different regex than street and city, which has to be included:

```
search = re.match(r'^\D*(\d{5}).*', postcode)
```

These regex is important to clean the postcode.

The postcode data runs through pretty much the same cleaning process like the cities and streets, the only difference is that here is no mapping needed.

```
183
184 bad_postcode = ['94103-3124', 'CA 94066', '94118-4504']
185
186 for i in bad_postcode:
187     clean = update_postcode(i)
188     print clean
```

For testing different cases of possible Postcodes, I created a short set with only three different postcodes, and try to clean them. The result was like I expect it:

```
94103
94066
94118
```

3 Data for SQLite 3

After all the cleaning steps the data is stored into 5 different csv documents:

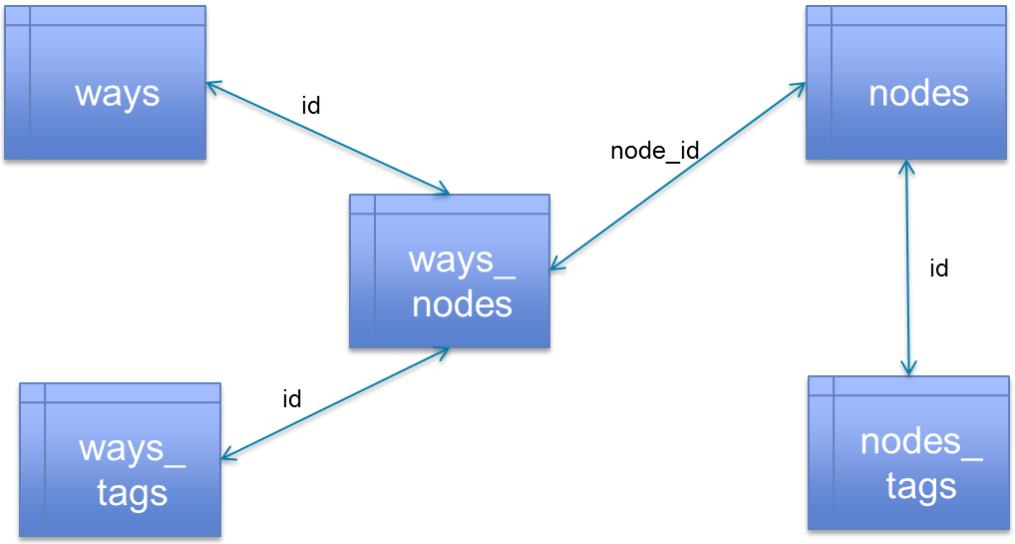
- nodes.csv
- nodes_tagss.csv
- ways.csv
- ways_nodes.csv
- ways_tags.csv

To run SQL queries on the extracted and cleaned data it will be loaded into a SQLite3 database. The name of the database is “SFO”. I imported the five csv’s into the SQLite 3 Studio. By importing the tables it will create the tables, I gave them the same names like the csv’s.

Tables:

- nodes
- nodes_tags
- ways
- ways_nodes
- ways_tags

After creating the table I run some queries to explore the structure of the data, where I can find the data and how the five tables can be joined. The “data model” and there dependencies look like this:



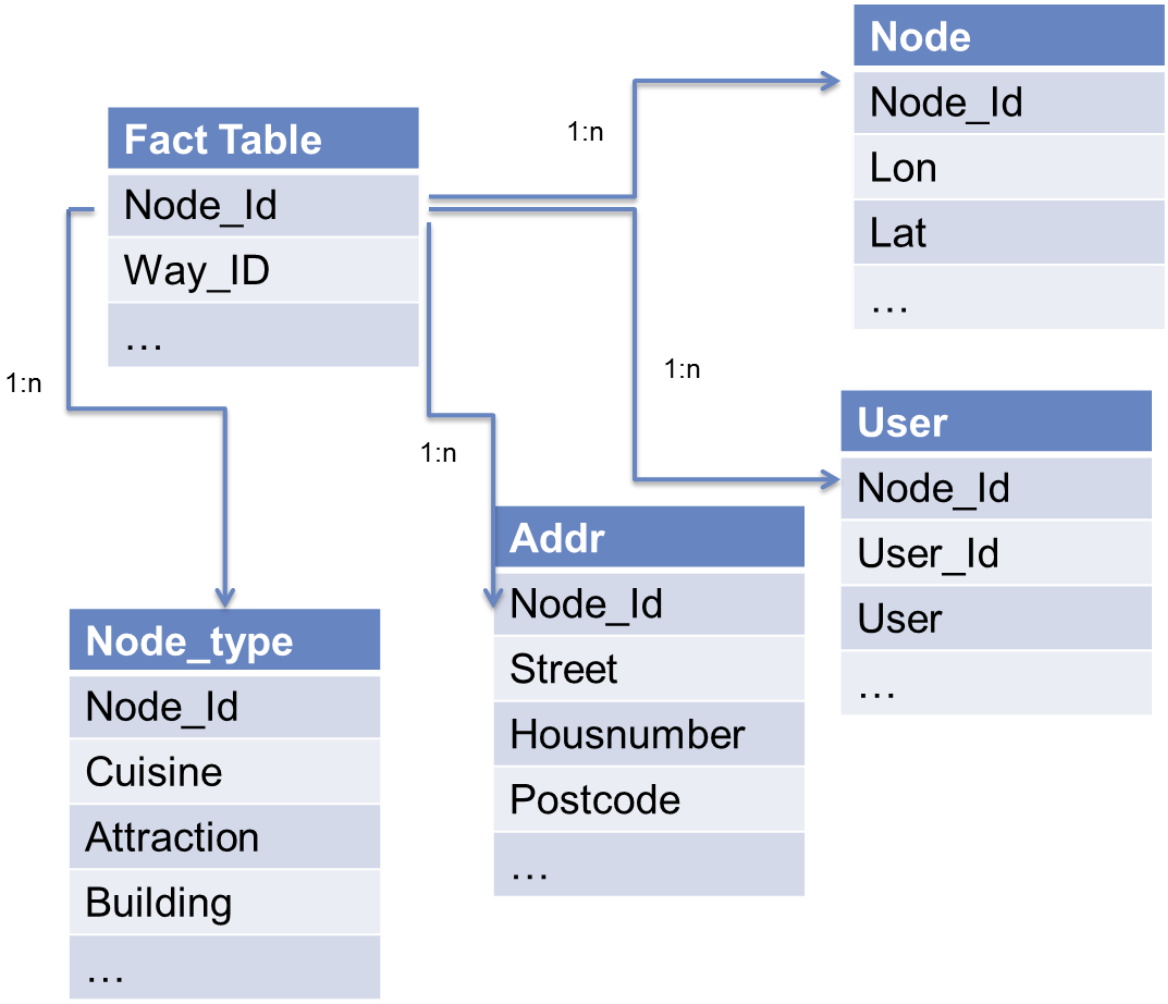
4 Improving Ideas and Problems

4.1 Data model

For understanding and analyzing the data of OSM it will be easier when a typical Data Warehouse will be created.

The structure of a typical Warehouse is that in the middle stays a fact table with IDs and KPIs and around them Dimensions which describes the Keys and KPIs. Normally has a Data Warehouse a Star- or a Snowflake-schema.

In example of nodes the following schema shows how a good DWH for OSM could be look:



Not all data for nodes are included it should only show how a data model can look like. Also all information for ways and so one have to be modeled. The benefits for an analyst is that is much

easier to read and understand how the data is structured, SQL queries will be much more performant because the data can be accessed on an easier way and not so many sub-selects are needed.

4.2 Include additional data

For giving the data more interesting information, additional data can be added. In example for restaurant analysis, an important information can be how a restaurant is rated / ranked, is it a good or a worst restaurant. This information can be found on sides like YELP and can be included to the data.

Also interesting can be more detailed information for attractions.
Something that can be also really important is the information if a building, train station or anything else is accessible whit a handicap. For example some train stations has no elevators only stairs.

4.3 Problems

A big issue of OSM is that it is an open source community, where are no checks on consistent data inputs. This is the reason why for example one and the same street can't be find as the equal street, because different users stored different names (Market Street, Market St, and so on). Here is a process needed which checks the new inputs, if they make sense or are they correct. Or a system is needed where is defined how an input has to look like, only "Street" is acceptable and not "St.".

Tools like this have in most cases a lot of costs and the users has to be trained for using it. Users have to learn standardizing techniques and how they have to work with this tool.

Issues by adding for example the information of restaurant ranking can be that it has to be updated very often. Rankings can change very fast. These updates has to be included relatively often to the dataset.

5 Data Analysis with SQL

5.1 Questions to answer with SQL queries

1. How many restaurants are in San Francisco and which kind of food are they deserve?
2. How many restaurants are on one Street and which is the name of this restaurant?
3. Which coordinates has the Restaurant “Baby Blues BBQ”?

5.2 SQL queries

5.2.1 How many restaurants are in San Francisco and which kind of food are they deserve?

```
SELECT value as "Restaurant Type", count(value) as Count
FROM nodes_tags
WHERE key = 'cuisine'
GROUP BY value
ORDER BY Count DESC;
```

The result shows there are 269 different cuisine types and how often each kind of restaurant can be found in San Francisco. (Result should be smaller, because there are some options to clean the “key” for restaurants.

	Restaurant Type	Count
1	coffee_shop	229
2	mexican	188
3	pizza	166
4	chinese	145
5	italian	118
6	japanese	115
7	thai	99
8	sandwich	98
9	burger	97
10	american	96
11	vietnamese	61
12	indian	60
13	sushi	60
14	asian	45
15	french	36
16	seafood	32
17	ice_cream	31
18	regional	22
19	mediterranean	20
20	korean	18

5.2.2 How many restaurants are on one Street and which is the name of this restaurant?

```
SELECT max(x.street) as street,
        max(x.restaurant_type) as restaurant_type,
        max(x.name) as name
FROM
        (SELECT nt.id,
        CASE WHEN nt.key ='street'
        THEN nt.value
        END as street,
        CASE WHEN nt.key ='cuisine'
        THEN nt.value
        END as restaurant_type,
        CASE WHEN nt.key ='name'
        THEN nt.value
        END as name

FROM nodes_tags nt
WHERE nt.key in ('cuisine', 'street', 'name')
AND nt.id in (SELECT id FROM nodes_tags nt2 WHERE key ='cuisine'))x
GROUP BY x.id
ORDER BY 1 DESC;
```

	street	restaurant_type	name
1	Yosemite Avenue	coffee_shop	Trouble Coffee Company
2	Yerba Buena Lane	indian	Amber India
3	Yerba Buena Lane	mexican	Tropisueño
4	Yerba Buena Lane	american	Bluestem Brasserie
5	Yerba Buena Lane	pizza	Delarosa
6	Westlake Center	mexican	Chipotle
7	Westlake Center	american;asian;	Broaster Foods
8	West Portal Avenue	sushi	Fuji Sushi
9	West Portal Avenue	chinese	Tsing's Chinese Restaurant
10	West Portal Avenue	italian	Paradise
11	West Portal Avenue	coffee_shop	Starbucks Coffee
12	West Portal Avenue	pizza	Goat Hill Pizza
13	West Portal Avenue	italian	Trattoria da Vittorio
14	West Portal Avenue	coffee_shop	Peet's Coffee & Tea
15	West Portal Avenue	mediterranean	Bursa
16	West Portal Avenue	burger	Calibur
17	West Portal Avenue	greek	Orexi
18	West Portal Avenue	mexican	El Toreador
19	West Portal Avenue	indian	Clay Oven
20	West Portal Avenue	peruvian	Fresca

5.2.3 Which coordinates has the Restaurant “Baby Blues BBQ”?

```
SELECT x.id, max(x.lat) as lat , max(x.lon) as lon,
        max(x.street) as street,
        max(x.restaurant_type) as restaurant_type,
        max(x.name) as name
FROM
(SELECT nt.id,
lat, lon,
CASE WHEN nt.key ='street'
THEN nt.value
END as street,
CASE WHEN nt.key ='cuisine'
THEN nt.value
END as restaurant_type,
CASE WHEN nt.key ='name'
THEN nt.value
END as name
FROM nodes_tags nt join nodes n on n.id=nt.id
WHERE nt.key in ('cuisine', 'street', 'name')
AND nt.id in (SELECT id FROM nodes_tags nt2 WHERE key ='cuisine')
)x
WHERE x.id = 4945060565
GROUP BY x.id

order by 1 desc;
```

	id	lat	lon	street	restaurant_type	name
1	4945060565	37.7468661	-122.4188886	Mission Street	barbecue	Baby Blues BBQ