Management of Scientific Data

Exam Project: 311 Service Requests in New York City

Exam Project

Structure

1. Data Management Plan
2. The Dataset: 311 Service Requests in New York City
3. Socrata Open Data API (SODA)
4. Data Quality Control
5. Data Analysis
6. Preserving and publishing

Data Management Plan

Creation of the Data Management plan

- The wizard by CLARIN-D (https://www.clarin-d.net/de/aufbereiten/datenmanagementplan-entwickeln) was used as a guide for which fields should be included
    - Not all fields were applicable since this is a small project
    - Some fields were added or re-ordered to better accomodate this project

Data Management Plan

Structure

- Project Information
- Research Data Information
- Documentation
- Storage and Backup
- Data sharing
- Licensing

Project Information

Project Supervisor
John Wigg

Institution
Friedrich Schiller University Jena

Context
Exam project for the course Management of Scientific Data

Research Question
"What is the influence of national holidays on 311 service requests?"

Research Data Information

Produced data

Description

- Python Jupyter Notebook (.ipynb) used for this DMP as well as to access, analyze and visualise data
- PDF version of the notebook (.pdf)

Data formats

- `.ipynb` - Python Jupyter Notebooks
- `.pdf` - Portable Document Format

Pre-existing data

Sources

- *311 Service Requests from 2010 to Present*
- provided by the City of New York Department of Information Technology and Telcommunications (DoITT)
- accessible at the NYC OpenData portal: https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9
- the data was accessed via the Socrata Open Data API (SODA)

License

- data provided under the *Open Data Law*
- terms of use: https://opendata.cityofnewyork.us/overview/#termsofuse

Reusability for other researchers

- data will stay freely available at https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

Creation of derived works

- FAQ states that there are no restrictions on how the data can be used: https://opendata.cityofnewyork.us/faq/
- terms of use are not clear about this

Relationship between produced and pre-existing data

- generated data are filtering, analysis and visualization scripts applied to the pre-existing data.

Documentation

- use Jupyter (https://jupyter.org) or any other compatible software to access notebooks
- Python 3.7.4
- in the notebooks, the following Python libraries were used:
    - matplotlib 3.2.1, pandas 1.0.1, numpy 1.18.1, sodapy 2.1.0

Storage and Backup

- generated data is kept in a public GitHub repository: https://github.com/john-wigg/mosd-exam
- data can be read by everyone, write access is restricted by GitHub's internal access management

Storage and Backup

- generated data is kept in a public GitHub repository: https://github.com/john-wigg/mosd-exam
- data can be read by everyone, write access is restricted by GitHub's internal access management

Data Sharing

- the complete generated project data is freely available at https://github.com/john-wigg/mosd-exam

Licensing

- produced data is licensed under the MIT license, a very permissive license

The Dataset

311 Service Requests in New York City from 2010 to present

- 311 provides access to *non-emergency* municipal services
- data provides, for each call, information about
    - time/duration, reason, and resolution of complaints
    - who handled the complaint
    - and more...
- provided by the DoITT through the NYC OpenData Portal

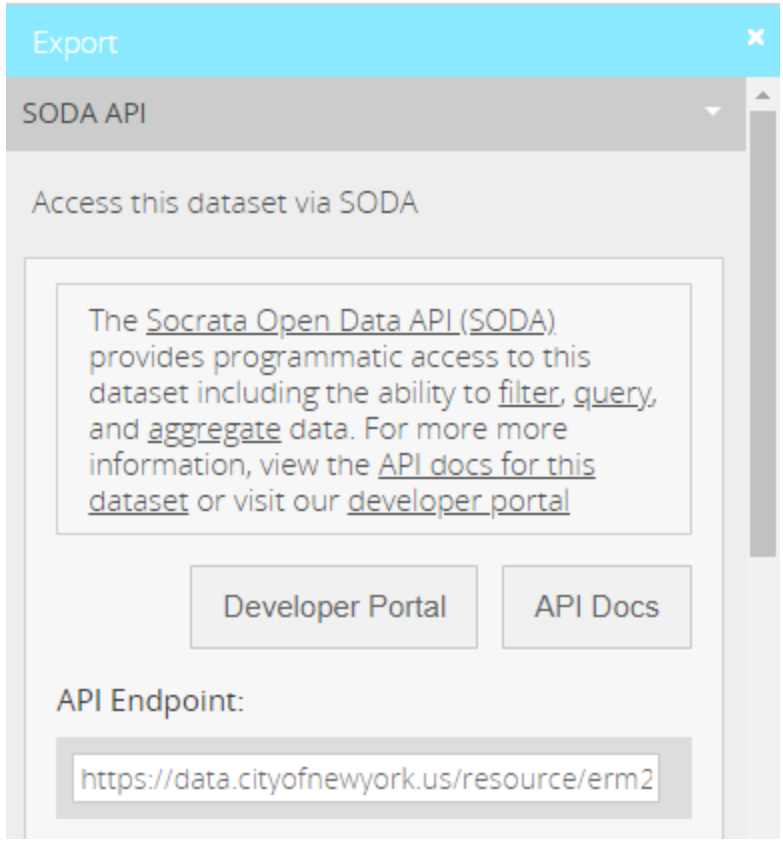
Problem: Size of the Dataset

- the dataset is *very* large
    - ~ 12 GB as .csv download
    - data is updated daily
- loading whole dataset at once not a good idea

- Is there a way to filter relevant data without downloading the whole set?
    - **Socrata Open Data API (SODA)**

Socrata Open Data API (SODA)

The Socrata Open Data API (SODA) provides programmatic access to this dataset including the ability to filter, query, and aggregate data.



sodapy

- `sodapy` provides Python bindings of the API.

sodapy

- `sodapy` provides Python bindings of the API.

In [1]:
```
from sodapy import Socrata
```

sodapy

- `sodapy` provides Python bindings of the API.

In [1]:
```
from sodapy import Socrata
```

Each Socrata dataset is hosted on a domain and has an identifier:

sodapy

- `sodapy` provides Python bindings of the API.

In [1]:
```
from sodapy import Socrata
```

Each Socrata dataset is hosted on a domain and has an identifier:

In [2]:
```
socrata_domain = 'data.cityofnewyork.us'
socrata_dataset_identifier = 'erm2-nwe9'
```

Optionally, an application token can be generated by registering at https://opendata.socrata.com/.

This removes data limits:

Optionally, an application token can be generated by registering at https://opendata.socrata.com/.

This removes data limits:

In [3]:
```
with open("application.token", "r") as f:
    socrata_token = f.read()
```

Create a `client` that can be used to access the data:

Create a `client` that can be used to access the data:

In [4]:

```
client = Socrata(socrata_domain, socrata_token)
print("Domain: {domain:}\nSession: {session:}\nURI Prefix: {uri_prefix:}".format(**
```

```
Domain: data.cityofnewyork.us
Session: <requests.sessions.Session object at 0x000001BE6AEED048>
URI Prefix: https://
```

We can now use SoQL clauses (https://dev.socrata.com/docs/queries/) to query and filter the data "over the air".

Example: Calls that were created on January 10th 2015 between 12 AM and 2 PM:

We can now use SoQL clauses (https://dev.socrata.com/docs/queries/) to query and filter the data "over the air".

Example: Calls that were created on January 10th 2015 between 12 AM and 2 PM:

In [5]:
```
import pandas as pd
```

We can now use SoQL clauses (https://dev.socrata.com/docs/queries/) to query and filter the data "over the air".

Example: Calls that were created on January 10th 2015 between 12 AM and 2 PM:

In [5]:
```
import pandas as pd
```

In [6]:
```
query = "created_date between '2015-01-10T12:00:00' and '2015-01-10T14:00:00'"
results = client.get(socrata_dataset_identifier, where = query)
df = pd.DataFrame.from_dict(results)
```

In [7]:
```
df.head()
```

Out[7]:

	unique_key	created_date	closed_date	agency	agency_name	complaint_type	descriptor	locatic
0	29690137	2015-01-10T12:00:00.000	2015-01-10T12:00:00.000	DSNY	BCC - Brooklyn South	Derelict Vehicles	14 Derelict Vehicles	Street
1	29689451	2015-01-10T12:00:00.000	2015-01-12T10:13:00.000	DOT	Department of Transportation	Street Light Condition	Street Light Out	NaN
2	29691167	2015-01-10T12:00:03.000	2015-01-16T05:27:49.000	DOT	Department of Transportation	Broken Muni Meter	Coin or Card Did Not Register	Street
3	29686604	2015-01-10T12:00:06.000	2015-01-10T15:50:40.000	NYPD	New York City Police Department	Noise - Residential	Banging/Pounding	Resident Building
4	29685180	2015-01-10T12:00:30.000	2015-02-20T19:33:19.000	DOT	Department of Transportation	Highway Condition	Graffiti - Highway	Highway

5 rows × 38 columns

Retreive metadata and data properties with SODA

Retreive metadata and data properties with SODA

Socrata also allows access to metadata:

Retreive metadata and data properties with SODA

Socrata also allows access to metadata:

In [8]:
```
md = client.get_metadata(socrata_dataset_identifier)
md.keys()
```

Out[8]:
```
dict_keys(['id', 'name', 'attribution', 'averageRating', 'category', 'createdA
t', 'description', 'displayType', 'downloadCount', 'hideFromCatalog', 'hideFrom
DataJson', 'indexUpdatedAt', 'newBackend', 'numberOfComments', 'oid', 'provenan
ce', 'publicationAppendEnabled', 'publicationDate', 'publicationGroup', 'public
ationStage', 'rowClass', 'rowIdentifierColumnId', 'rowsUpdatedAt', 'rowsUpdated
By', 'tableId', 'totalTimesRated', 'viewCount', 'viewLastModified', 'viewType',
'approvals', 'columns', 'grants', 'metadata', 'owner', 'query', 'rights', 'tabl
eAuthor', 'tags', 'flags'])
```

The metadata contains e.g. information about the columns:

The metadata contains e.g. information about the columns:

In [9]:
```
print("Number of columns: ", len(md["columns"]))
print("-----------------------------------------------------")
for d in md["columns"][:5]:
    print(d["fieldName"], end="")
    if ("description" in d):
        print(": " + d["description"], end="")
    else:
        print(": NO DESCRIPTION")
```

```
Number of columns:  46
-----------------------------------------------------
unique_key: Unique identifier of a Service Request (SR) in the open data set
created_date: Date SR  was created
closed_date: Date SR was closed by responding agency
agency: Acronym of responding City Government Agency
agency_name: Full Agency name of responding City Government Agency
```

We can also use SODA to count the rows of the dataset without downloading it:

We can also use SODA to count the rows of the dataset without downloading it:

In [10]:
```
client.get(socrata_dataset_identifier, select="count(*)")
```

Out[10]:
```
[{'count': '23501686'}]
```

Data Quality Control

Data Quality Control

NOTE: This data does not present a full picture of 311 calls or service requests, in part because of operational and system complexities associated with remote call taking necessitated by the unprecedented volume 311 is handling during the Covid-19 crisis. The City is working to address this issue. (Source: https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9)

→ we should avoid data from 2020

Quality Control using SODA

Quality Control using SODA

- use the metadata to start with quality control
- e.g. find columns with missing descriptions

In [11]:
```
print("Columns with missing descriptions:")
for d in md["columns"]:
    if ("description" not in d):
        print("{} ({}) with data type {}.".format(d["fieldName"], d["name"], d["dat
```

```
Columns with missing descriptions:
:@computed_region_efsh_h5xi (Zip Codes) with data type number.
:@computed_region_f5dn_yrer (Community Districts) with data type number.
:@computed_region_yeji_bk3q (Borough Boundaries) with data type number.
:@computed_region_92fq_4b7q (City Council Districts) with data type number.
:@computed_region_sbqj_enih (Police Precincts) with data type number.
```

- we can still retreive the fields `name` and `dataTypeName` to get a better idea
    - field name seems to indicate special or computed fields
- Looking at specific entries may give more clues.

→ **Fields do not seem relevant to the research question.**

Since the data set should contain only records since 2010, we can also check for invalid or missing creation dates:

Since the data set should contain only records since 2010, we can also check for invalid or missing creation dates:

In [12]:
```
query = "created_date < '2010-01-01T00:00:00' OR created_date IS NULL"
results = client.get(socrata_dataset_identifier, where = query)
results
```

Out[12]:
```
[]
```

Since the data set should contain only records since 2010, we can also check for invalid or missing creation dates:

In [12]:
```
query = "created_date < '2010-01-01T00:00:00' OR created_date IS NULL"
results = client.get(socrata_dataset_identifier, where = query)
results
```

Out[12]:
```
[]
```

In this case, all entries seem to have a valid date associated.

Retreiving the relevant Data using SODA

Retreiving the relevant Data using SODA

- download only the interesting part of the data

Retreiving the relevant Data using SODA

- download only the interesting part of the data

What is the influence of national holidays on 311 service requests?
Example: Independence Day 2019

Retreiving the relevant Data using SODA

- download only the interesting part of the data

What is the influence of national holidays on 311 service requests?
Example: Independence Day 2019

- retreive data around July 4th 2019.

In [13]:
```
query = "created_date between '2019-06-20T0:00:00' and '2019-07-18T23:59:59'"
results = client.get(socrata_dataset_identifier, limit=500000, where = query)
df = pd.DataFrame.from_dict(results)
```

In [14]:
```
df.head()
```

Out[14]:

	unique_key	created_date	closed_date	agency	agency_name	complaint_type	descriptor	cross_street_1
0	43022702	2019-06-20T00:00:00.000	2019-06-24T11:20:00.000	DOT	Department of Transportation	Traffic Signal Condition	LED Lense	LINDEN BLVD
1	43032740	2019-06-20T00:00:00.000	2019-07-15T00:00:00.000	DOHMH	Department of Health and Mental Hygiene	Standing Water	Sewer or Drain	PRINCE STREET
2	43021298	2019-06-20T00:00:00.000	2019-06-20T00:48:00.000	DOT	Department of Transportation	Traffic Signal Condition	Controller	NaN
3	43024836	2019-06-20T00:00:00.000	2019-07-09T00:00:00.000	DOHMH	Department of Health and Mental Hygiene	Standing Water	Other - Explain Below	FOSTER AVENUE
4	43024837	2019-06-20T00:00:00.000	2019-07-11T00:00:00.000	DOHMH	Department of Health and Mental Hygiene	Standing Water	Other - Explain Below	37 AVENUE

5 rows × 41 columns

In [15]:
```
df.shape
```

Out[15]:
```
(192190, 41)
```

```
In [15]: df.shape

Out[15]: (192190, 41)
```

The dataset is still very large, but manageable!

Quality Control using Pandas

Quality Control using Pandas

- do "local" quality control on the smaller dataset

Plot a matrix highlighting all missing values (yellow are missing):

Quality Control using Pandas

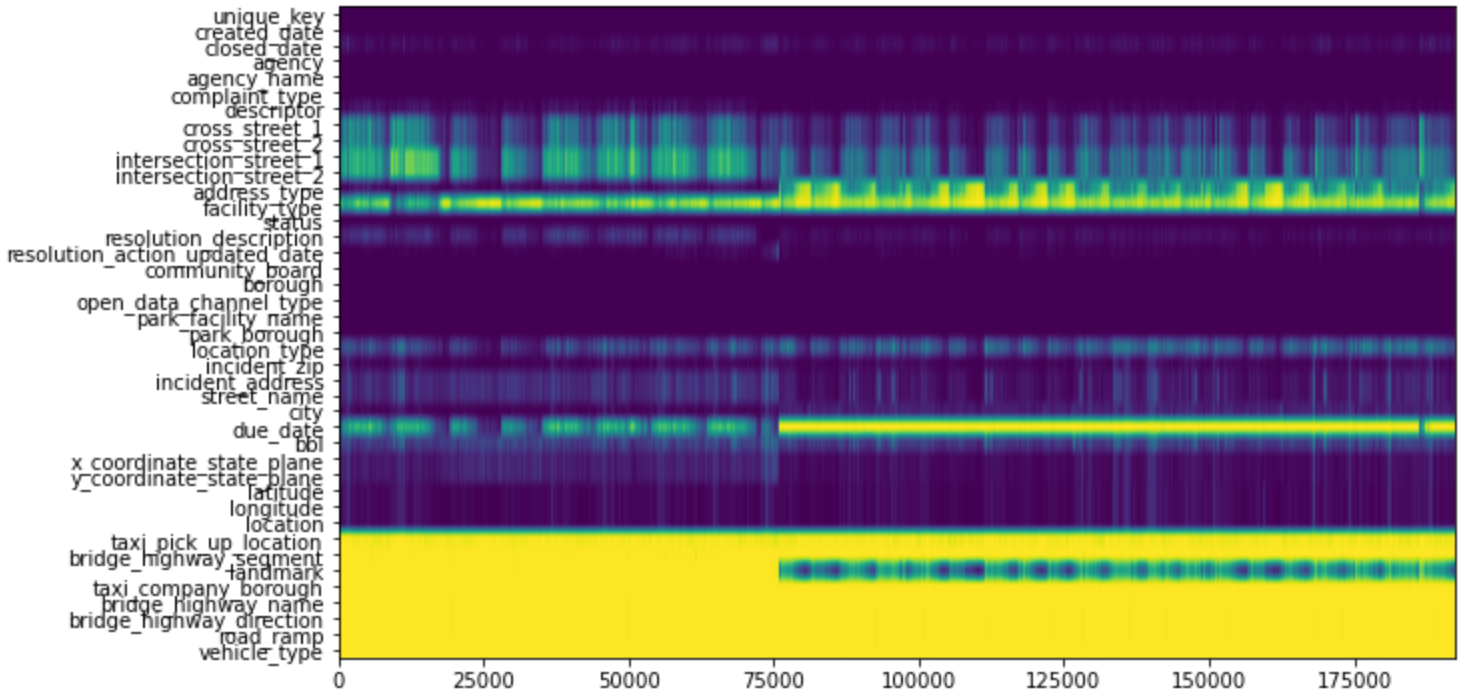
- do "local" quality control on the smaller dataset

Plot a matrix highlighting all missing values (yellow are missing):

In [16]:
```
%%capture
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(10, 6));
ax = plt.gca();
ax.imshow(np.array(df.isna()).transpose(), aspect='auto');
ax.set_yticks((range(0, len(df.columns))));
ax.set_yticklabels(df.columns);
```

In [17]:
```
fig
```

Out[17]:



- some `closed_date` seem to be missing
- it is hard to see but `descriptor` also has some missing values
- other fields may be specific to certain complaint types

- use pandas to gather which types of complaints where made how often

- use pandas to gather which types of complaints where made how often

In [18]:
```
values = df["complaint_type"].value_counts()
len(values.keys())
```

Out[18]: 196

- use pandas to gather which types of complaints where made how often

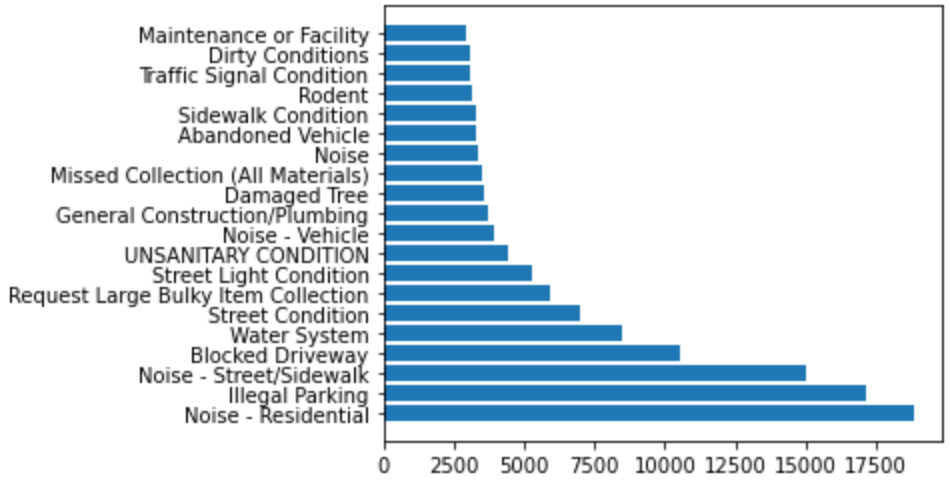
```
In [18]: values = df["complaint_type"].value_counts()
         len(values.keys())

Out[18]: 196
```

We can see that there are 196 types of complaints hat occured in the time period.

- list the 20 most common complaint types:

- list the 20 most common complaint types:

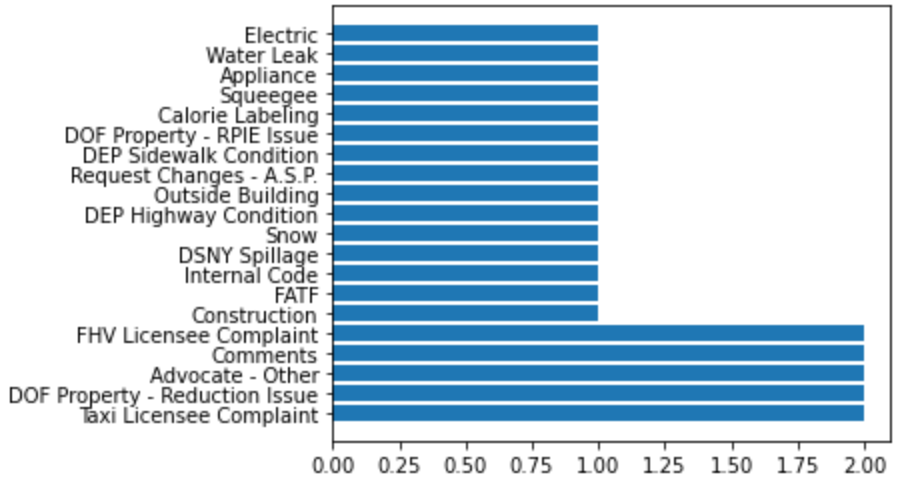
In [19]:
```
plt.figure(figsize=(5, 4))
plt.barh(values.keys()[:20], values[:20]);
```



- rare complaint types may reveal erroneous entries

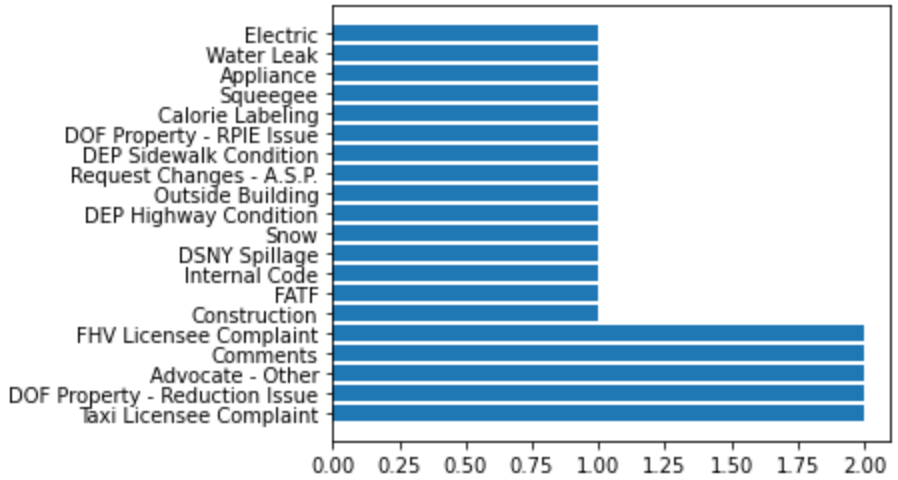
- rare complaint types may reveal erroneous entries

In [20]:
```
plt.figure(figsize=(5, 4))
plt.barh(values.keys()[-20:], values[-20:]);
```



- rare complaint types may reveal erroneous entries

In [20]:
```
plt.figure(figsize=(5, 4))
plt.barh(values.keys()[-20:], values[-20:]);
```



- in this case, all entries seem to be reasonable
- clustering is the more proper way to do this

Data Analysis

Data Analysis

What is the influence of national holidays on 311 service requests?
Example: Independence Day 2019

- Is there a spike in overall complaints?

- Is there a spike in overall complaints?

- convert the `created_date` column to pandas dates first

In [21]:
```
df["created_date_format"] = pd.to_datetime(df["created_date"])
```

- Is there a spike in overall complaints?
- convert the `created_date` column to pandas dates first

In [21]:
```
df["created_date_format"] = pd.to_datetime(df["created_date"])
```

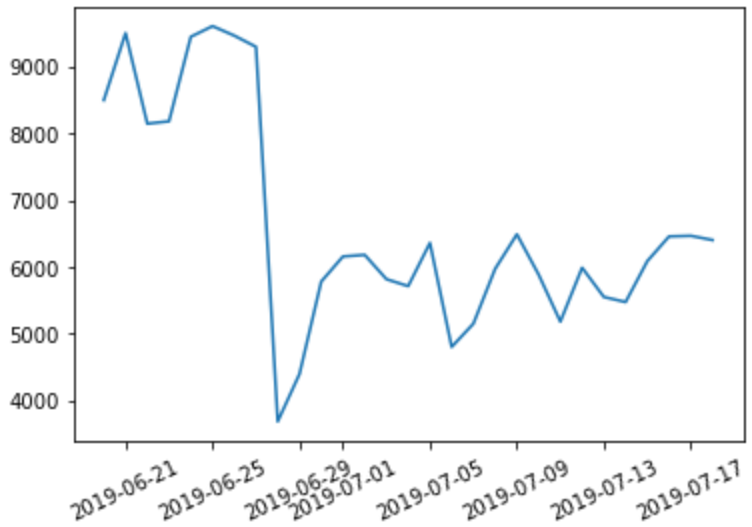
- resample and count the complaints per day

In [22]:
```
df_grouped = df.resample('D', on='created_date_format').count()
```

- plot the total number of complaints per day

- plot the total number of complaints per day

In [23]:
```
plt.plot(df_grouped.index, df_grouped["unique_key"]);
plt.xticks(rotation=25);
```



- sharp decrease may be evidence of a change in methodology
    - also visible in matrix plot
- **but no evidence of a change of overall calls on July 4th**

- look at more specific complaint types
- noise complaints might be interesting

- look at more specific complaint types
- noise complaints might be interesting

In [24]:
```
import re
df["complaint_type"].value_counts()[df["complaint_type"].value_counts().keys().str.
```

Out[24]:
```
Noise - Residential          18874
Noise - Street/Sidewalk      14985
Noise - Vehicle               3947
Noise                         3375
Noise - Commercial            2736
Noise - Park                   659
Noise - Helicopter             146
Noise - House of Worship        64
Name: complaint_type, dtype: int64
```

- There is also a complaint type especially for illegal fireworks

- There is also a complaint type especially for illegal fireworks

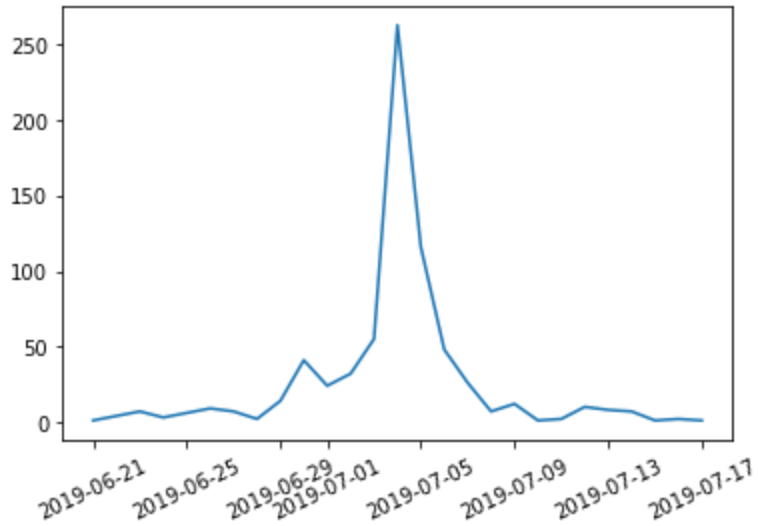
In [25]:
```
df["complaint_type"].value_counts()[df["complaint_type"].value_counts().keys().str.
```

Out[25]:
```
Illegal Fireworks    709
Name: complaint_type, dtype: int64
```

- plot the number of illegal firework complaints per day

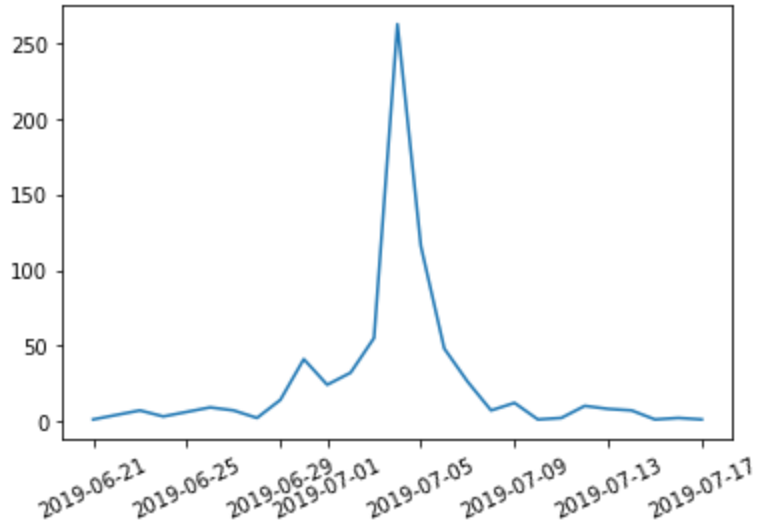
- plot the number of illegal firework complaints per day

In [26]:
```
df_fireworks_grouped = df[df["complaint_type"] == "Illegal Fireworks"].resample('D'
plt.plot(df_fireworks_grouped.index, df_fireworks_grouped["unique_key"]);
plt.xticks(rotation=25);
```



- plot the number of illegal firework complaints per day

In [26]:
```
df_fireworks_grouped = df[df["complaint_type"] == "Illegal Fireworks"].resample('D'
plt.plot(df_fireworks_grouped.index, df_fireworks_grouped["unique_key"]);
plt.xticks(rotation=25);
```



- large uptick on July 4th
- almost no complaints on other dates

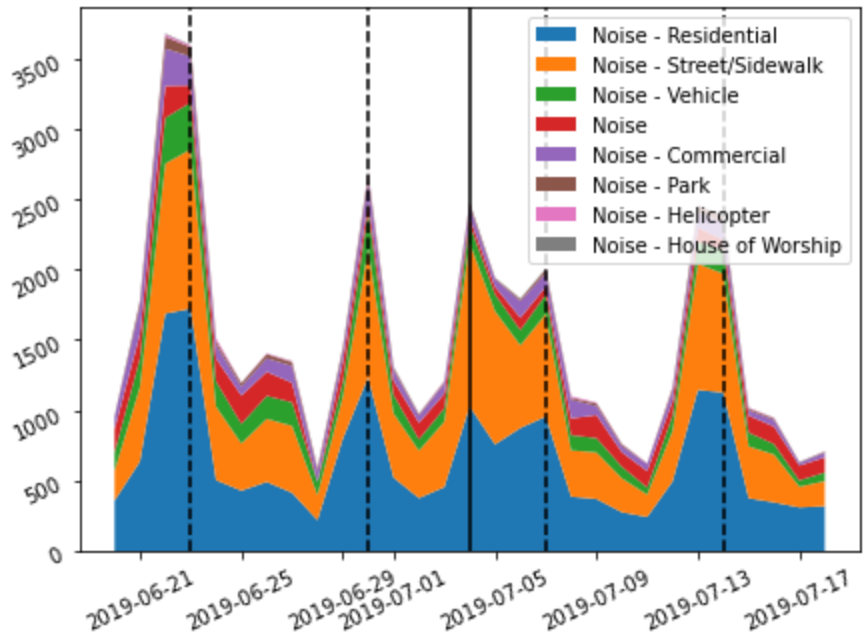
- stackplot of all noise complaints by type

- stackplot of all noise complaints by type

In [27]:
```
%%capture
keys = df["complaint_type"].value_counts()[df["complaint_type"].value_counts().keys
s = pd.Series(index=df_grouped.index, dtype='int')
noise_counts = []
for key in keys:
    noise_counts.append(s.add(df[df["complaint_type"] == key].resample('D', on='cre
fig = plt.figure(figsize=(7, 5))
ax = plt.gca()
ax.stackplot(df_grouped.index, noise_counts);
ax.tick_params(rotation=25);
ax.legend(keys);
ax.axvline(pd.to_datetime('2019-06-23'), color='black', linestyle='dashed');
ax.axvline(pd.to_datetime('2019-06-30'), color='black', linestyle='dashed');
ax.axvline(pd.to_datetime('2019-07-07'), color='black', linestyle='dashed');
ax.axvline(pd.to_datetime('2019-07-14'), color='black', linestyle='dashed');
ax.axvline(pd.to_datetime('2019-07-04'), color='black');
```

In [28]:
```
fig
```

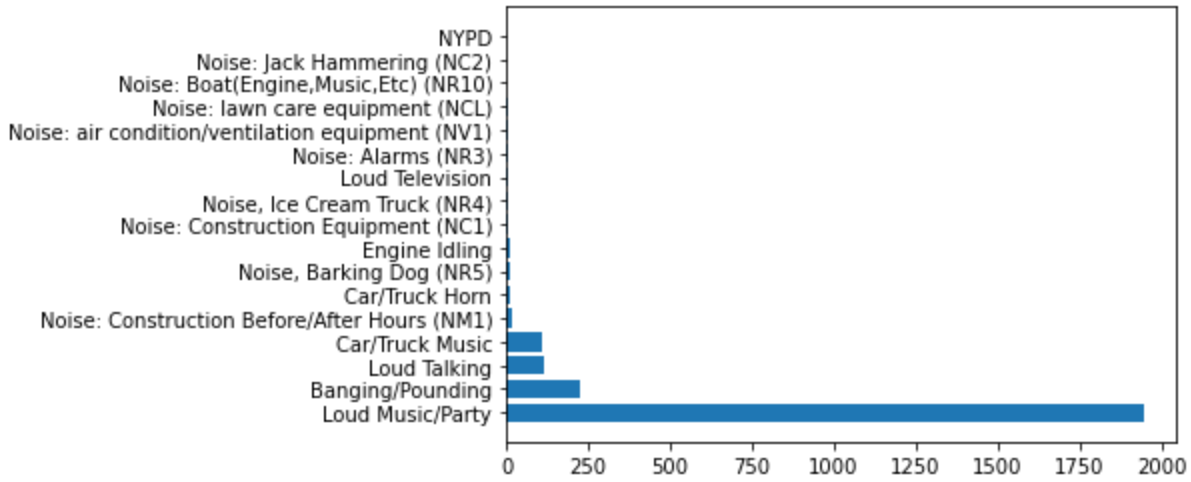
Out[28]:



- decode noise complaints by type

- decode noise complaints by type

In [29]:
```
date_selector = (df["created_date_format"] > pd.to_datetime('2019-07-04T00:00:00'))
counts = df[df["complaint_type"].str.contains("Noise") & date_selector]["descriptor
plt.barh(counts.keys(), counts);
```



Preserving and publishing

GitHub repo: https://github.com/john-wigg/mosd-exam

- the GitHub repository preserves past versions
- the data is decentralized
- the data can be accessed by everyone
- MIT License allows data to be freely used by other researchers
- for larger projects, publishing the results in a journal may lead to better visibility