# Tasks & Mini-projects

Sign up to take on a task & mini-assignment - whether it's writing a code snippet or planning out a more abstract goal or writing documentation and creating the final presentation! Every contribution, small or large, helps make our project that much better.

| Task | Description | Requires running full dataset* | Assignee (can have multiple!) | Helper |
|---|---|---|---|---|
| Writing a schema | *List the variables we create, types, would be helpful to track whether we generated it or it existed in the original doc* | N | | |
| Standardization of 'Location Type' col | *Group similar categories (e.g. "RESIDENTIAL Building" & "Residential Building/Home"* | ~Y | Vivek Agarwal | |
| Standardization of 'Complaint Type' | *Clean up odd chars (e.g. \) and group similar* | ~Y | Chet | Victoria Levchenko, JP |
| Standardization/ Creation of date fields | Also creating fields with "year & week", week, season, month | N | Victoria Levchenko | |
| Create features around location data | | ~Y | JP | |
| Standardization of Address cols | Columns: `'Incident Zip', 'Incident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2', 'Intersection Street 1', 'Intersection Street 2', 'Address Type', 'City'` | ~Y | @kathelyn | Victoria Levchenko |
| Data quality checks | | ~Y | Akash | |
| Visualize | A visual of concentrations, if | Y | | |

| | | | | |
|---|---|---|---|---|
| location data by “Agency” (maybe interactive?) | any, of where diff. Agencies receive the most complaints | | | |
| Visualize location data by “Complaint Type” (maybe interactive?) | A visual of concentrations, if any, of different complaint types | Y | Vikram Setlur | |
| Holdout 311 dataset - using a combination of NYC and other cities’ 311 data | Find/create a hold out dataset to improve testing | N | Marvin Mananghaya | |

* “Requires running full dataset” is a flag to inform team members who may not be able to work off the full dataset if a task requires it. Otherwise the sample dataset will work well!

# Breaking down larger tasks into code snippets!

### Standardization of selected fields

- **Location type**

Created a function named ***standardizeLocations*** place in the file ***utils.py***. This function will accept a pandas dataframe. It would combine the 161 Location types into a smaller set of 45 different categories. Similar categories will be merged, for example RESIDENTIAL BUILDING, Residential Building/House, 3+ Family Apt. Building, 1-2 Family Dwelling etc have been merged under ‘Residential’. Most values less than 100 have been merged into ‘Other’. Standardized locations will be placed in a separate column named ‘***Standardized Locations***’ in the same dataframe. This file can be found here:
https://github.com/datakind/msvdd_311/tree/jp_datacleaning_eda

Spreadsheet used to do analysis can be found here. This sheet will help to easily update the mapping table and generate new code.
https://docs.google.com/spreadsheets/d/1Bkg3i_ebdcZzZPJ_X1nKrSfeseTgK9bUUNTX1UUB3ag/edit?usp=sharing

- Complaint type
- ?

Approach - Create a lookup table
1.) Generate distinct values of the selected field
2.) Generate look-up column - manually annotate
3.) Perform look-up

Things to discuss:
1.) Standardized values

# EDA

List any interesting observations you see with the data. Be sure to list your name/slack handle in case another team member has a question about it!

- missing values can be relevant to classify the complaint type. For example, when 'Vehicle Type' is not missing, most of the "Complaint Type" is about "For Hire Vehicle Complaint". - JP
    - Percentage of missing values for columns:
      *{'Closed_Date': 3.721856749729175, 'Descriptor': 0.7302612655286058, 'Location_Type': 22.01530929848839, 'Incident_Zip': 3.993754928154484, 'Incident_Address': 13.793823904429223, 'Street_Name': 13.798117033562251, 'Cross_Street_1': 39.68282361965166, 'Cross_Street_2': 39.91050256800675, 'Intersection_Street_1': 78.96953452462897, 'Intersection_Street_2': 79.02920901957809, 'Address_Type': 9.826972585508399, 'City': 4.7440507963039025, 'Landmark': 93.32575834548528, 'Facility_Type': 15.50163067354903, 'Due_Date': 64.1609579974556, 'Resolution_Description': 8.209321528182247, 'Resolution_Action_Updated_Date': 1.8967044509731807, 'BBL': 18.970335908733798, 'X_Coordinate__State_Plane_': 6.558327167922884, 'Y_Coordinate__State_Plane_': 6.558327167922884, 'Vehicle_Type': 99.99141374173394, 'Taxi_Company_Borough': 99.92000469382118, 'Taxi_Pick_Up_Location': 99.22365914844352, 'Bridge_Highway_Name': 99.85102841908382, 'Bridge_Highway_Direction': 99.83543004990048, 'Road_Ramp': 99.83557315420491, 'Bridge_Highway_Segment': 99.77546934634248, 'Latitude': 6.558327167922884, 'Longitude': 6.558327167922884, 'Location': 6.558327167922884}*

Missing values count (on 15% data) in descending order - the ranking should generalize to the larger dataset if it is a representative sample

| | colname |
|---|---|
| Vehicle.Type | 698731 |
| Taxi.Company.Borough | 698232 |
| Bridge.Highway.Name | 697750 |
| Road.Ramp | 697648 |
| Bridge.Highway.Direction | 697642 |
| Bridge.Highway.Segment | 697229 |
| Taxi.Pick.Up.Location | 693366 |
| Landmark | 652152 |
| Intersection.Street.2 | 552249 |
| Intersection.Street.1 | 551832 |
| Facility.Type | 507743 |
| Due.Date | 448351 |
| Cross.Street.2 | 278891 |
| Cross.Street.1 | 277300 |
| Location.Type | 155108 |
| BBL | 132563 |
| Street.Name | 96432 |
| Incident.Address | 96390 |
| Address.Type | 68670 |
| Resolution.Description | 57552 |
| Latitude | 45829 |
| Longitude | 45829 |
| Y.Coordinate..State.Plane. | 45829 |
| X.Coordinate..State.Plane. | 45829 |
| Location | 45829 |
| City | 33166 |
| Incident.Zip | 27924 |
| Closed.Date | 26008 |
| Resolution.Action.Updated.Date | 13254 |
| Descriptor | 9905 |
| Park.Facility.Name | 63 |
| Borough | 0 |
| Community.Board | 0 |
| Open.Data.Channel.Type | 0 |
| Park.Borough | 0 |
| Created.Date | 0 |
| Status | 0 |

- Consider using more generalizable geo fields/features that can apply to other cities
    - Currently looking into Location Type & Facility Type

  -JP & Marvin
- Column profiles (for sample)

## Dataset info

| | |
|---|---|
| Number of columns | 41 |
| Number of rows | 698791 |
| Total Missing (%) | 0 |
| Total size in memory | -1 Bytes |

## Column types

| | |
|---|---|
| Categorical | 0 |
| Numeric | 0 |
| Date | 0 |
| Array | 0 |
| Not available | 0 |

---

## Agency

categorical

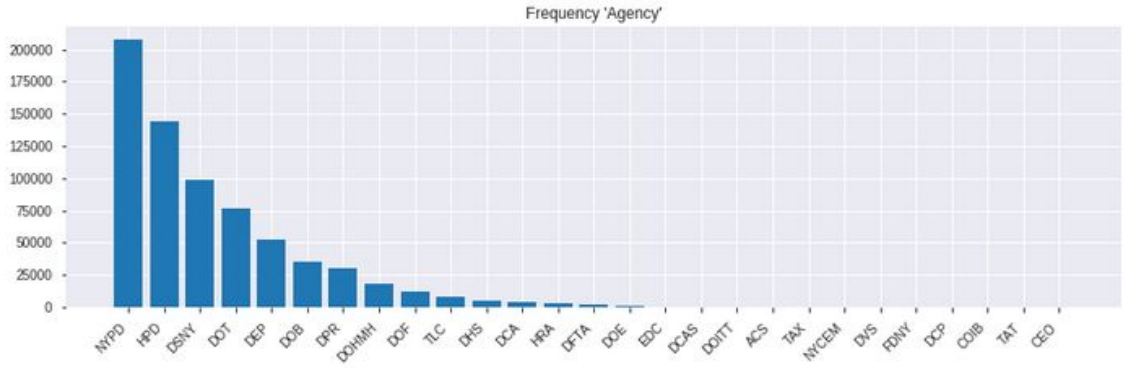
| | |
|---|---|
| Unique | 28 |
| Unique (%) | 0.0 |
| Missing | 0 |
| Missing (%) | 0.0 |

### Datatypes

| | |
|---|---|
| String | 698791 |
| Integer | |
| Decimal | |
| Bool | |
| Date | |
| Missing | 0 |
| Null | 0 |

### Frequency

| Value | Count | Frecuency (%) |
|---|---|---|
| NYPD | 207485 | 29.69% |
| HPD | 143975 | 20.6% |
| DSNY | 98732 | 14.13% |
| DOT | 76827 | 10.99% |
| DEP | 51994 | 7.44% |
| DOB | 35756 | 5.12% |
| DPR | 30404 | 4.35% |
| DOHMH | 17735 | 2.54% |
| DOF | 11788 | 1.69% |
| TLC | 8520 | 1.22% |
| "Missing" | 0 | 0.0% |



**Landmark**

null

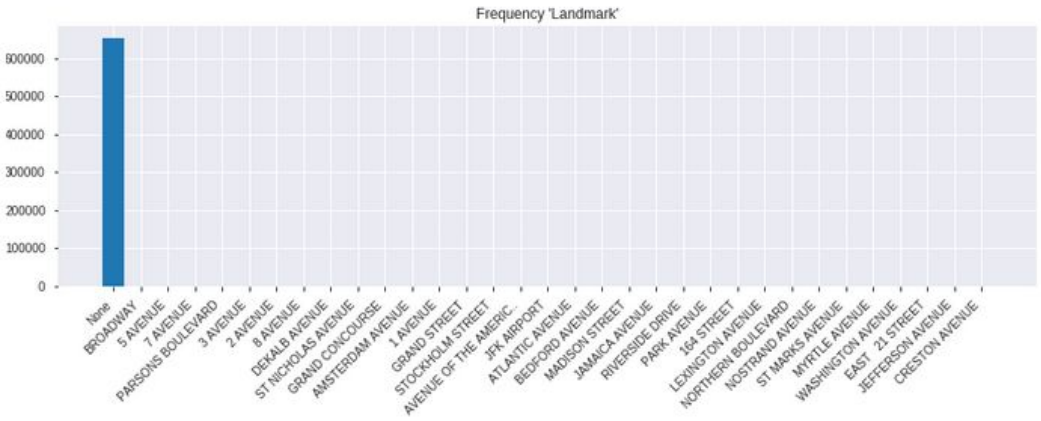
| | |
|---|---|
| Unique | 4656 |
| Unique (%) | 0.67 |
| Missing | 652152 |
| Missing (%) | 93.33 |

**Datatypes**

| | |
|---|---|
| String | 46639 |
| Integer | |
| Decimal | |
| Bool | |
| Date | |
| Missing | 0 |
| Null | 652152 |

**Frequency**

| Value | Count | Frecuency (%) |
|---|---|---|
| None | 652152 | 93.33% |
| BROADWAY | 537 | 0.08% |
| 5 AVENUE | 315 | 0.05% |
| 7 AVENUE | 224 | 0.03% |
| PARSONS BOULEVARD | 218 | 0.03% |
| 3 AVENUE | 210 | 0.03% |
| 2 AVENUE | 199 | 0.03% |
| 8 AVENUE | 179 | 0.03% |
| DEKALB AVENUE | 178 | 0.03% |
| ST NICHOLAS AVENUE | 175 | 0.03% |
| "Missing" | 652152 | 93.33% |




## Code snippets 🎉

*Standardization/Creation of date fields <Python syntax>*

There are 4 date fields to clean:

```
    # ALTER object types to datatime for date columns
    df['Created Date'] = pd.to_datetime(df['Created Date'])
    df['Created Date'] = pd.to_datetime(df['Closed Date'])
    df['Due Date'] = pd.to_datetime(df['Due Date'])
    df['Resolution Action Updated Date'] = pd.to_datetime(df['Resolution
Action Updated Date'])
```

Helper functions to clean Complaint.Type

```
def clean_col(col):
    col = col.tolist()
    ls_col = []
    for c in col:
        c = re.sub('[^a-zA-Z- ]','',c)
        c = c.split()
        c = ' '.join(c)
        c = c.lower()
        ls_col.append(c)
    return ls_col

import nltk
from nltk.stem import PorterStemmer
from nltk.tag import pos_tag
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

def cleaner_col(col, lemmatize=True, stem=False):
    ps = PorterStemmer()
    wnl = WordNetLemmatizer()
    col = col.tolist()
    ls_col = []
    for c in col:
        c = re.sub('[^a-zA-Z- ]','',c)
        c = c.split()
        c = ' '.join(c)
        c = c.lower()
        ls_word = []
        eng_stopwords = set(stopwords.words('english'))

        for word in c.split():
            word = re.sub(r'[-/]','',word)
            if word not in eng_stopwords:
                if lemmatize is True:
                    word=wnl.lemmatize(word)
                elif stem is True:
                    if word == 'oed':
                        continue
```

```
                    word=ps.stem(word)

            ls_word.append(word)
        words = ' '.join(ls_word)
        ls_col.append(words)

    return ls_col

Use case:
df['CitySvcReq'] = cleaner_col(df['Complaint.Type'])
```

This is to standardize address columns

---

```
# converting dtype float to str
df['Incident Address'] = df['Incident Address'].astype(str)
df['Street Name'] = df['Street Name'].astype(str)
df['Cross Street 1'] = df['Cross Street 1'].astype(str)
df['Cross Street 2'] = df['Cross Street 2'].astype(str)

def normalize_case(df,text_field):
 df[‘text_field’] = df[‘text_field’].str.lower()
 return df

def remove_punc(df,text_field):
 df[‘text_field’] = df[‘text_field’].str.replace('[^\w\s]','')

 return df
```

Possible checklist to clean text data:

1. Removing all irrelevant chars such as any non alphanumeric chars

2. Tokenize text by separating it into individual words
3. Remove words that are not relevant, such as “@”
4. Convert all chars to lowercase
5. Consider combining misspelled or alternately spelled words to a sing representation
6. Consider lemmatization
7. Removing stop words

Some latitude & longitude features:

```
borough_to_location = {'MANHATTAN':[40.7831,-73.9712],
                       'BROOKLYN':[40.6782,-73.9442],
                       'BRONX': [40.8448,-73.8648],
                       'QUEENS':[40.7282,-73.7949],
                       'STATEN ISLAND':[40.5795,-74.1502],
                       'Unspecified':[0,0]}


def sphere_dist(first_lat, first_lon, second_lat, second_lon):
    #Define earth radius (km)
    R_earth = 6371
    #Convert degrees to radians
    first_lat, first_lon, second_lat, second_lon = map(np.radians,[first_lat, first_lon,
second_lat, second_lon])
    #Compute distances along lat, lon dimensions
    dlat = second_lat - first_lat
    dlon = second_lon - first_lon

    #Compute haversine distance
    a = np.sin(dlat/2.0)**2 + np.cos(first_lat) * np.cos(second_lat) *
np.sin(dlon/2.0)**2

    return 2 * R_earth * np.arcsin(np.sqrt(a))

def dummy_manhattan_distance(first_lat, first_lon, second_lat, second_lon):
    a = sphere_dist(first_lat, first_lon, second_lat, second_lon)
    b = sphere_dist(first_lat, first_lon, second_lat, second_lon)
    return a + b

def bearing_array(first_lat, first_lon, second_lat, second_lon):
    R_earth = 6371  # in km
    lng_delta_rad = np.radians(second_lon - first_lon)
    first_lat, first_lon, second_lat, second_lon = map(np.radians, (first_lat, first_lon,
second_lat, second_lon))
    y = np.sin(lng_delta_rad) * np.cos(second_lat)
    x = np.cos(first_lat) * np.sin(second_lat) - np.sin(first_lat) * np.cos(second_lat)
* np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))
```

```
data.loc[:,'borough_centers_latitude'] = data.park_borough.apply(lambda x:borough_to_location[x][0])
data.loc[:,'borough_centers_longitude'] = data.park_borough.apply(lambda x:borough_to_location[x][1])


data['spherical_distance'] = sphere_dist(data['latitude'], data['longitude'],
                                         data['borough_centers_latitude'] ,
data['borough_centers_longitude'])
data['dummy_manhattan_distance'] =
dummy_manhattan_distance(data['latitude'], data['longitude'],
                                         data['borough_centers_latitude'] ,
data['borough_centers_longitude'])
data['bearing_array'] = bearing_array(data['latitude'], data['longitude'],
                                         data['borough_centers_latitude'] ,
data['borough_centers_longitude'])
```

## Model deployment

- One could use a special-purpose framework such as MLflow (https://www.mlflow.org/docs/latest/index.html) which supports models from multiple libraries, build a simple Flask app (https://github.com/amirziai/sklearnflask/) or use cloud providers such as AWS/Azure/GCP.
    - Akash

Other cities 311 download links

Python code:

```
try:
        import urllib2 as urllib
except ImportError:
        import urllib.request as urllib
import requests
url = ‘’
urllib.urlretrieve(url,'./<filename>.csv')
```

San Francisco
https://data.sfgov.org/api/views/vw6y-z8j6/rows.csv?accessType=DOWNLOAD&bom=true&format=true

Austin
https://data.austintexas.gov/api/views/i26j-ai4z/rows.csv?accessType=DOWNLOAD&bom=true&format=true

Baton Rogue
https://data.brla.gov/api/views/7ixm-mnvx/rows.csv?accessType=DOWNLOAD&bom=true&format=true

Chicago
https://data.cityofchicago.org/api/views/v6vf-nfxy/rows.csv?accessType=DOWNLOAD&bom=true&format=true

Baltimore
https://data.baltimorecity.gov/api/views/9agw-sxsr/rows.csv?accessType=DOWNLOAD&bom=true&format=true

Buffalo
https://data.buffalony.gov/api/views/whkc-e5vr/rows.csv?accessType=DOWNLOAD&bom=true&format=true