Disaster Data:

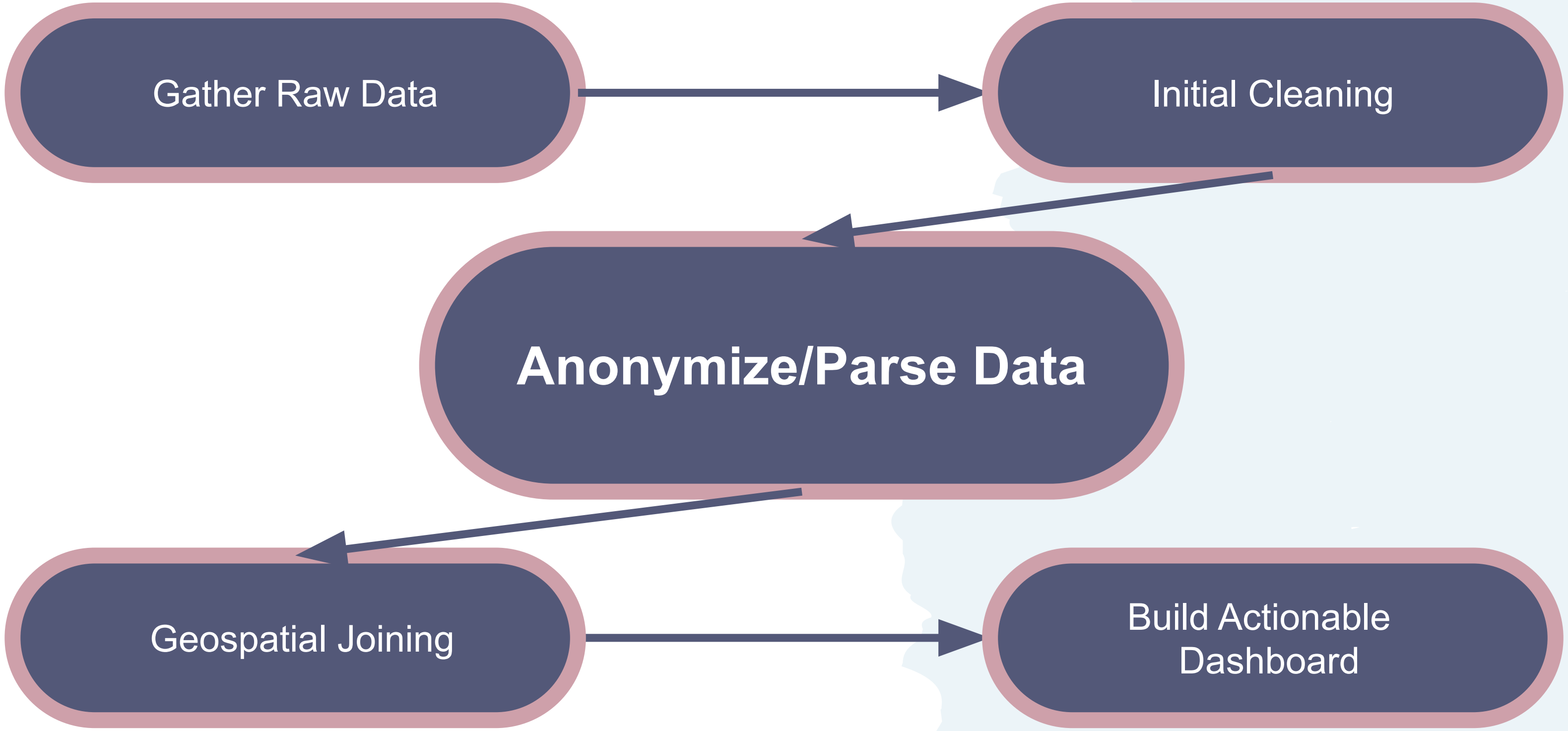
March 2021 Flood - South Nashville

Holliday Therrell
NSS Data Analytics Capstone

Inspiration

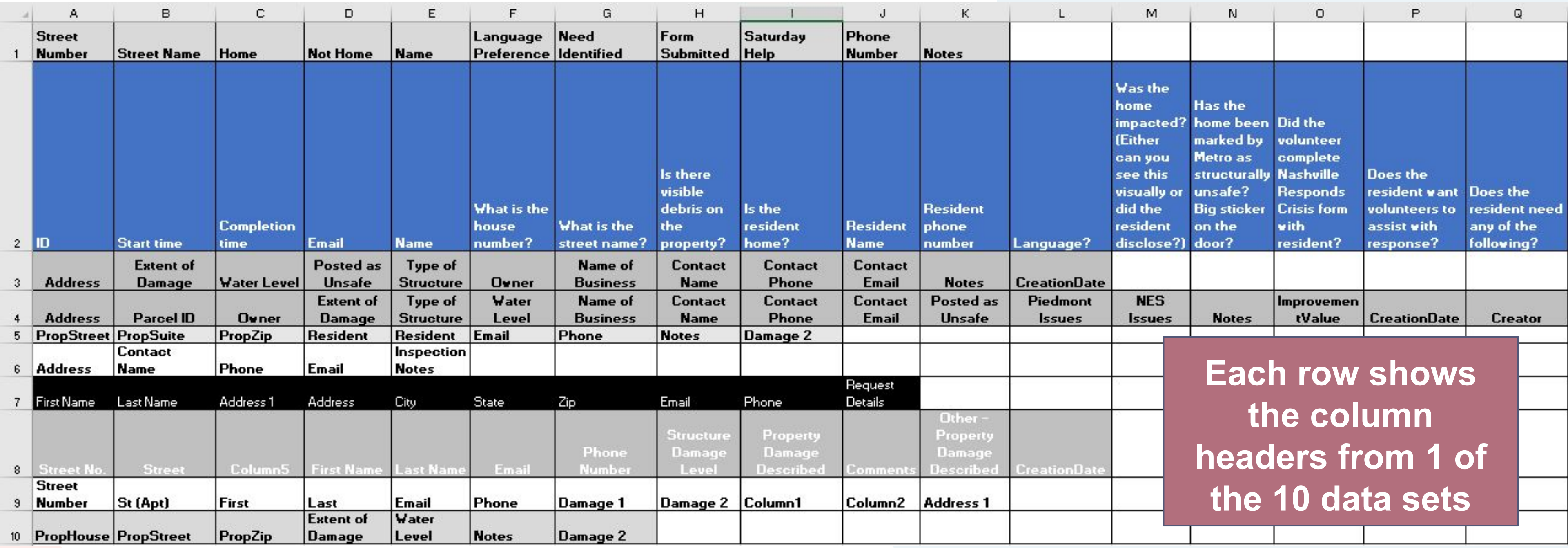
- Worked with Hands On Nashville during the March 2021 flooding
- Wanted to use my new skills to optimize the data process because...
- Fast, accurate reporting means more people getting more help more quickly
  - FEMA
  - Community Volunteer Projects
  - Relief Organizations
  - Metro Council Members

Process


Gather Raw Data
Initial Cleaning
Anonymize/Parse Data
Geospatial Joining
Build Actionable Dashboard

The Raw Data

- 10 Excel spreadsheets (.xls) from various nonprofit/government organizations



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Street Number	Street Name	Home	Not Home	Name	Language Preference	Need Identified	Form Submitted	Saturday Help	Phone Number	Notes						
2	ID	Start time	Completion time	Email	Name	What is the house number?	What is the street name?	Is there visible debris on the property?	Is the resident home?	Resident Name	Resident phone number	Language?	Was the home impacted? (Either can you see this visually or did the resident disclose?)	Has the home been marked by Metro as structurally unsafe? Big sticker on the door?	Did the volunteer complete Nashville Responds Crisis form with resident?	Does the resident want volunteers to assist with response?	Does the resident need any of the following?
3	Address	Extent of Damage	Water Level	Posted as Unsafe	Type of Structure	Owner	Name of Business	Contact Name	Contact Phone	Contact Email	Notes	CreationDate					
4	Address	Parcel ID	Owner	Extent of Damage	Type of Structure	Water Level	Name of Business	Contact Name	Contact Phone	Contact Email	Posted as Unsafe	Piedmont Issues	NES Issues	Notes	ImprovementValue	CreationDate	Creator
5	PropStreet	PropSuite	PropZip	Resident	Resident	Email	Phone	Notes	Damage 2								
6	Address	Contact Name	Phone	Email	Inspection Notes												
7	First Name	Last Name	Address 1	Address	City	State	Zip	Email	Phone	Request Details							
8	Street No.	Street	Column5	First Name	Last Name	Email	Phone Number	Structure Damage Level	Property Damage Described	Comments	Other - Property Damage Described	CreationDate					
9	Street Number	St (Apt)	First	Last	Email	Phone	Damage 1	Damage 2	Column1	Column2	Address 1						
10	PropHouse	PropStreet	PropZip	Extent of Damage	Water Level	Notes	Damage 2										

Each row shows the column headers from 1 of the 10 data sets

Initial Cleaning (Excel)

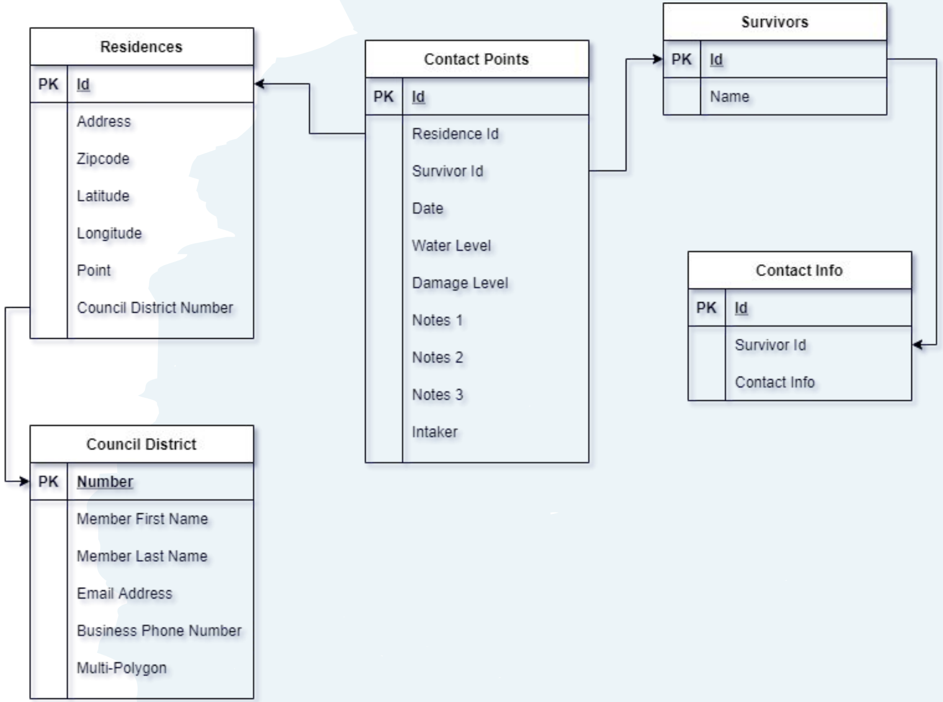
For each spreadsheet, I needed to…

- Select relevant columns
- Scan through comments to remove Personally Identifying Information (PII)
- Fix consistency errors that I knew would cause problems later
  - Apartment numbers with dashes (e.g. H-5) don't play nice in Google Maps
  - Address format varied if entered by human
  - Capitalization errors resulting in new categories (e.g. "major" vs "Major")
- Save cleaned data to a CSV with UTF-8 encoding (Jupyter friendly)

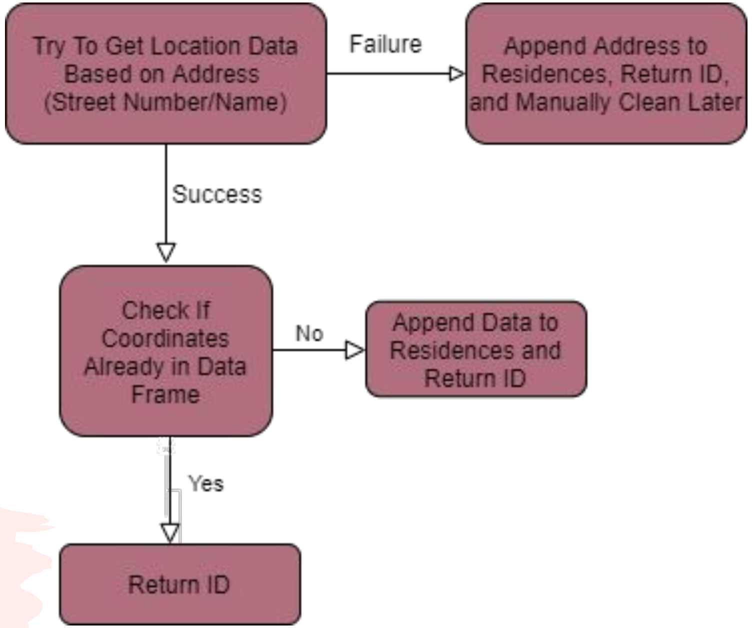
Anonymizing/Parsing the Data (Python)

I decided to...

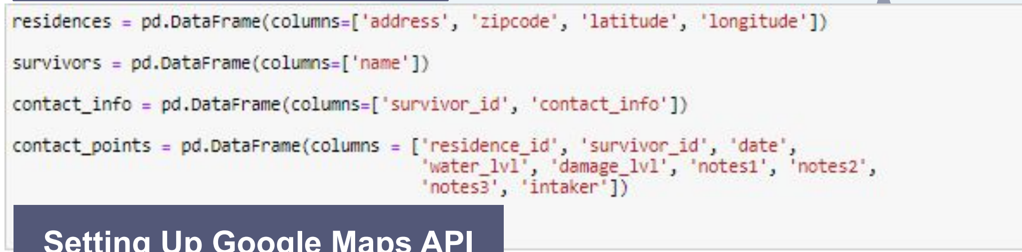
- Separate the data into related tables
  - Provide anonymity for public analysis
  - Preserve PII for authorized users (government, relief orgs, etc)
- Build functions that would work for every data set
  - Maximize consistency
  - Minimize duplication



find_or_add_residence() Function

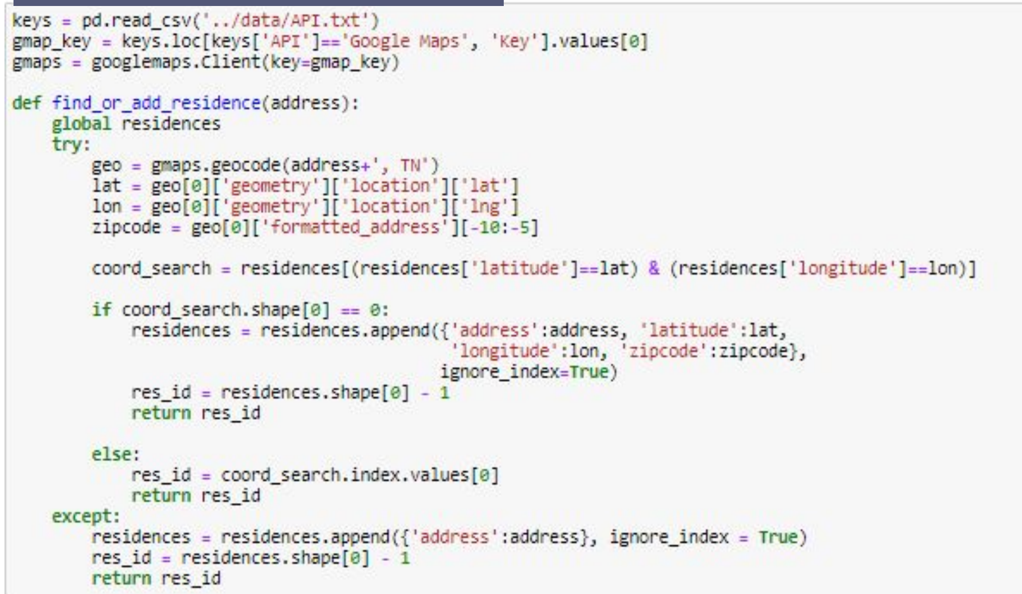

Try To Get Location Data Based on Address (Street Number/Name)
Failure
Append Address to Residences, Return ID, and Manually Clean Later
Success
Check If Coordinates Already in Data Frame
No
Append Data to Residences and Return ID
Yes
Return ID

Initializing data frames



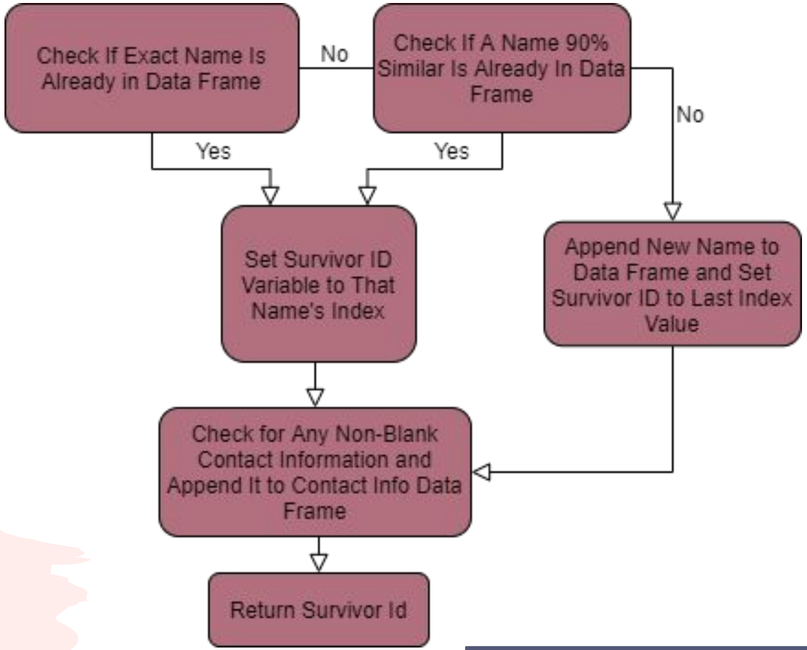
```
residences = pd.DataFrame(columns=['address', 'zipcode', 'latitude', 'longitude'])

survivors = pd.DataFrame(columns=['name'])

contact_info = pd.DataFrame(columns=['survivor_id', 'contact_info'])

contact_points = pd.DataFrame(columns = ['residence_id', 'survivor_id', 'date',
                                          'water_lvl', 'damage_lvl', 'notes1', 'notes2',
                                          'notes3', 'intaker'])
```

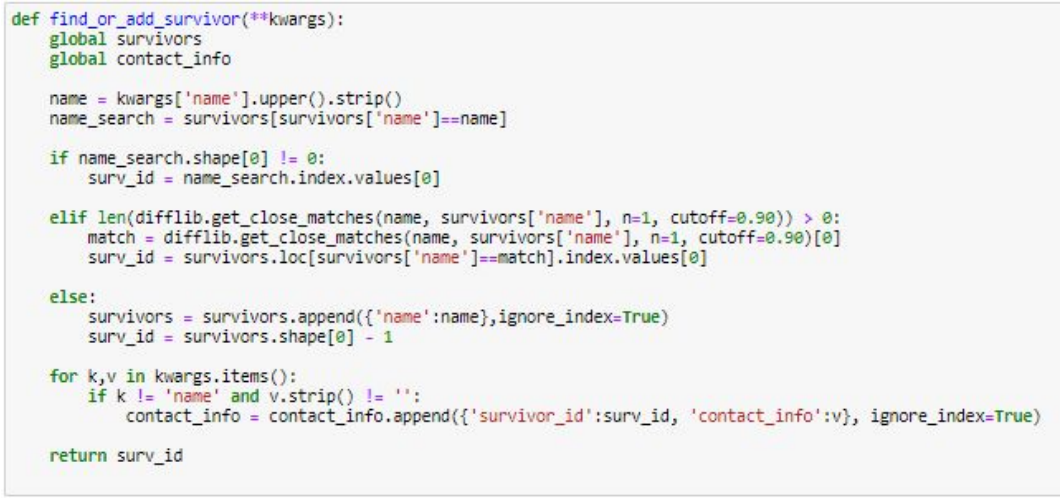
Setting Up Google Maps API



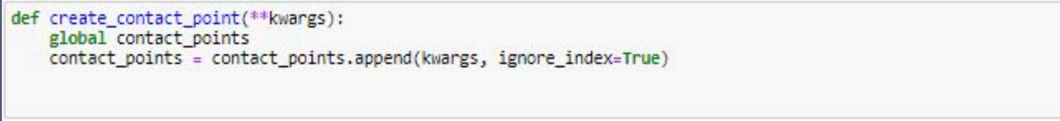
```
keys = pd.read_csv('../data/API.txt')
gmap_key = keys.loc[keys['API']=='Google Maps', 'Key'].values[0]
gmaps = googlemaps.Client(key=gmap_key)

def find_or_add_residence(address):
    global residences
    try:
        geo = gmaps.geocode(address+', TN')
        lat = geo[0]['geometry']['location']['lat']
        lon = geo[0]['geometry']['location']['lng']
        zipcode = geo[0]['formatted_address'][-10:-5]

        coord_search = residences[(residences['latitude']==lat) & (residences['longitude']==lon)]

        if coord_search.shape[0] == 0:
            residences = residences.append({'address':address, 'latitude':lat,
                                            'longitude':lon, 'zipcode':zipcode},
                                           ignore_index=True)
            res_id = residences.shape[0] - 1
            return res_id

        else:
            res_id = coord_search.index.values[0]
            return res_id
    except:
        residences = residences.append({'address':address}, ignore_index = True)
        res_id = residences.shape[0] - 1
        return res_id
```

Survivors, Contact Info, & Contact Points Functions


Check If Exact Name Is Already in Data Frame
No
Check If A Name 90% Similar Is Already In Data Frame
No
Yes
Yes
Set Survivor ID Variable to That Name's Index
Append New Name to Data Frame and Set Survivor ID to Last Index Value
Check for Any Non-Blank Contact Information and Append It to Contact Info Data Frame
Return Survivor Id



```
def find_or_add_survivor(**kwargs):
    global survivors
    global contact_info

    name = kwargs['name'].upper().strip()
    name_search = survivors[survivors['name']==name]

    if name_search.shape[0] != 0:
        surv_id = name_search.index.values[0]

    elif len(difflib.get_close_matches(name, survivors['name'], n=1, cutoff=0.90)) > 0:
        match = difflib.get_close_matches(name, survivors['name'], n=1, cutoff=0.90)[0]
        surv_id = survivors.loc[survivors['name']==match].index.values[0]

    else:
        survivors = survivors.append({'name':name},ignore_index=True)
        surv_id = survivors.shape[0] - 1

    for k,v in kwargs.items():
        if k != 'name' and v.strip() != '':
            contact_info = contact_info.append({'survivor_id':surv_id, 'contact_info':v}, ignore_index=True)

    return surv_id
```



```
def create_contact_point(**kwargs):
    global contact_points
    contact_points = contact_points.append(kwargs, ignore_index=True)
```

Use Residence ID, Survivor ID, and intake notes to create contact point

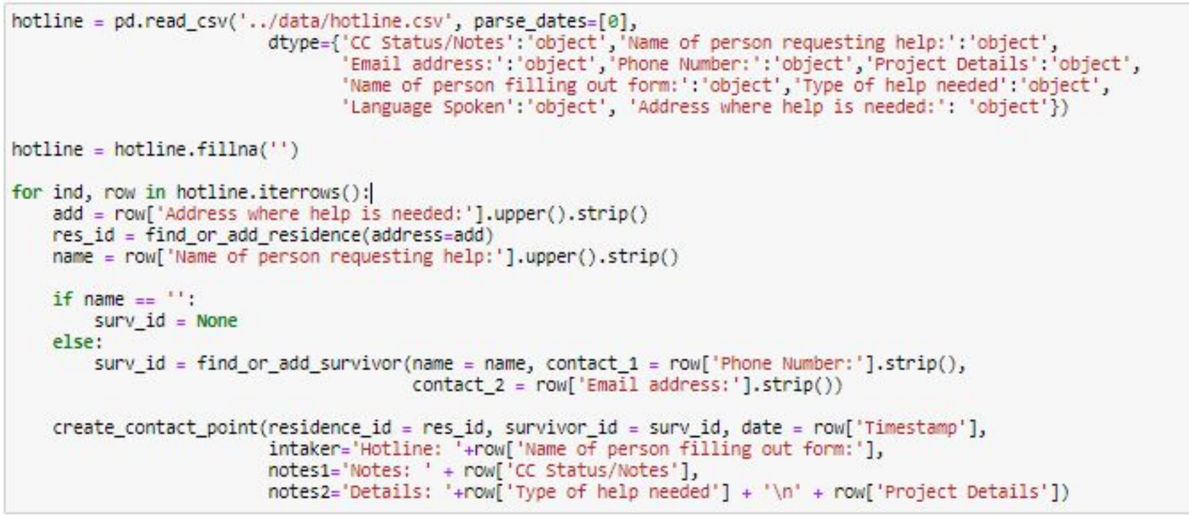
Parsing Template Used for All 10 Data Sets

Read in CSV and set column types

Deal with blanks/duplicates

Send address, name, and contact info to respective functions to get anonymized IDs

Send IDs and remaining notes to create contact point



```
hotline = pd.read_csv('../data/hotline.csv', parse_dates=[0],
                      dtype={'CC Status/Notes':'object','Name of person requesting help:':'object',
                             'Email address:':'object','Phone Number:':'object','Project Details':'object',
                             'Name of person filling out form:':'object','Type of help needed':'object',
                             'Language Spoken':'object', 'Address where help is needed:': 'object'})

hotline = hotline.fillna('')

for ind, row in hotline.iterrows():
    add = row['Address where help is needed:'].upper().strip()
    res_id = find_or_add_residence(address=add)
    name = row['Name of person requesting help:'].upper().strip()

    if name == '':
        surv_id = None
    else:
        surv_id = find_or_add_survivor(name = name, contact_1 = row['Phone Number:'].strip(),
                                       contact_2 = row['Email address:'].strip())

    create_contact_point(residence_id = res_id, survivor_id = surv_id, date = row['Timestamp'],
                         intaker='Hotline: '+row['Name of person filling out form:'],
                         notes1='Notes: ' + row['CC Status/Notes'],
                         notes2='Details: '+row['Type of help needed'] + '\n' + row['Project Details'])
```

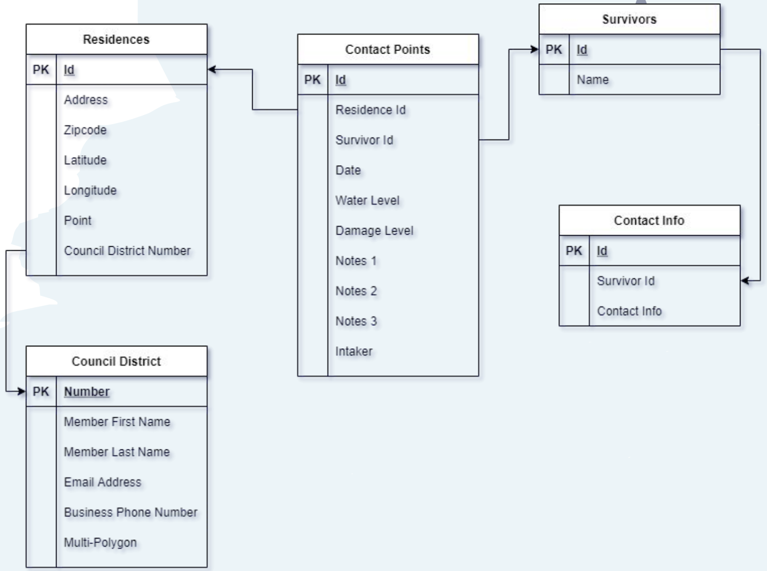
Geospatial Joining (Python)

- Turned the Residences DataFrame into a GeoDateFrame by creating a Point geometry column from the latitude and longitude columns
- Downloaded a GEOJSON file of Nashville Council District (Multi-Polygons)
- Used geopandas.sjoin() to match Council Districts to Residences using the “within” operation

Results & Dashboard (Tableau)

The final data included...

- 1,896 Points of Contact
- 880 Residences (1700+ in original process)
- 650 Survivors
- 1,218 Survivor Contact Information



I used this to create a dashboard that…

- hides PII from main view while keeping it accessible to authorized users.
- provides mapping and filtering capabilities.
- leads to action items.

Questions & Comments

A Very Special Thank You To

HandsOn
NASHVILLE

For Letting Me Use This Data and Inspiring My Data Analytics Journey