6.2 Data Extraction and Clean-up

All Data sources do not follow the same access and clean-up pattern. Each of the data sources need specific processing steps to make them consistent and useful. No matter how good the data source is, there will be a need to check for gaps in the data. Assess impact of these gaps on the overall analytical goals. Define a strategy to address these gaps. If the data does not meet the requirements as defined, do we have the flexibility to redefine the analytic strategy or the requirement itself. Nice try! Don't touch the requirements. Could we go back to sourcing ? Explore additional data sources to enrich existing data sources or extrapolate the given data to fill for gaps.

Ref: Jupyter Notebook

Population Data: Section 6.2.1

Home-price Data: Section 6.2.2

COVID Data: Section 6.2.3

Venue Data: Section 6.2.4

Each of the csv data sources (as listed in the previous section) can be read using standard pandas function(read_csv) to read CSV files. Clean-up the dataframes for blank, Null, or invalid values. Some of these steps specific to individual data sources can be seen in Section 6.2 of Jupyter Notebook. Some of the clean-up tasks include,

- Dropping NAs
- Clean-up County names to be consistent
- Eliminate Unallocated COVID data records
- Drop blank Home-price records
- … and more, based on specific datasets

Some of the clean-up tasks could lead to loss of data, which has been ignored in this analysis, with an expectation that these incomplete lost records will not have a noticeable impact on the analysis. This would require a more concerted view when a production grade application is built.

Extracting venue data from Foursquare has been quite a challenge. It takes over 6-8Hrs for larger states like California, or New York. This can be a serious constraint while developing the program. Hence, make a set of backup venue counts database (CSV files in this case) to store the Foursquare data relevant to this analysis. These backup files can then be used to recreate venue data, when required.

Here is a code snippet to define credentials for accessing Foursquare API,

```
#Set Credentials
CLIENT_ID = 'XXXXXXXXXXXXXXXXXX' # your Foursquare ID
CLIENT_SECRET = 'XXXXXXXXXXXXXXXXXX' # your Foursquare Secret
VERSION = '20180605' # Foursquare API version
print('Your Credentials:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

Define a reusable function to extract data from Foursquare,

```
def get_venues_count_ll(lat,lon, radius, categoryId):
    explore_url = '
https://api.Foursquare.com/v2/venues/explore?client_id={}&client_secret={}&v={}&ll={},{}&radius={}&c
ategoryId={}'.format(
                CLIENT_ID,
                CLIENT_SECRET,
                VERSION,
                lat,
                lon,
                radius,
                categoryId)

    # make the GET request
    return requests.get(explore_url).json()['response']['totalResults']
```

Foursquare calls can sometimes time-out or give network error. Write the extract program, in a way that can give some flexibility to start mid-way, in case of failures, as illustrate in Jupyter Notebook, Section 6.3.4.

```
import time
RADIUS = 2000 # 1000
processing_start_index = 0 # 0
file_start_index = 0 # 0
block = 200 # 200
message_block = 25 # 50
df_size= len(EXTRACT_venues_df)

processing_blocks = [(x, min(x+block,df_size)) for x in range(processing_start_index,
df_size, block)]

main_df = pd.DataFrame()

for n, p in enumerate(processing_blocks):
    start = p[0]
    end = p[1]

    print(" Start Index: {}, Start Time: {}".format(start, time.ctime(time.time())))
    process_df = EXTRACT_venues_df.iloc[start:end,:].copy(deep = True)
# anaconda warning, dont write to a slice, chain indexing risk

     for i, row in process_df.iterrows():

        if i%message_block == 0:
            print("     Start Processing Index: {}, ZIP: {}, at: {}".format(i,
row['ZIP'], time.ctime(time.time())))

        for c in venue_cats.items():
            try:
                state = process_df.State.loc[i]
#                 zip = process_df.ZIP.loc[i]
                lat = process_df.Latitude.loc[i]
                lon = process_df.Longitude.loc[i]
                process_df.loc[i, c[0]] = get_venues_count_ll(lat, lon, radius=RADIUS,
categoryId=c[1])
```

Continued…code snippet

```
            except:
                process_df.loc[i, c[0]] = 0

    pd.concat([main_df, process_df])

    #save to file
    filename = P_select_state+'_bkup_venues_counts_{}.csv'.format(str(n+file_start_index))
    file_location = os.path.join(P_path,filename)
    process_df.to_csv(file_location)

    print(" End Index: {}, End Time: {}\n          File: {}".format(end-1,
time.ctime(time.time()), filename))
```

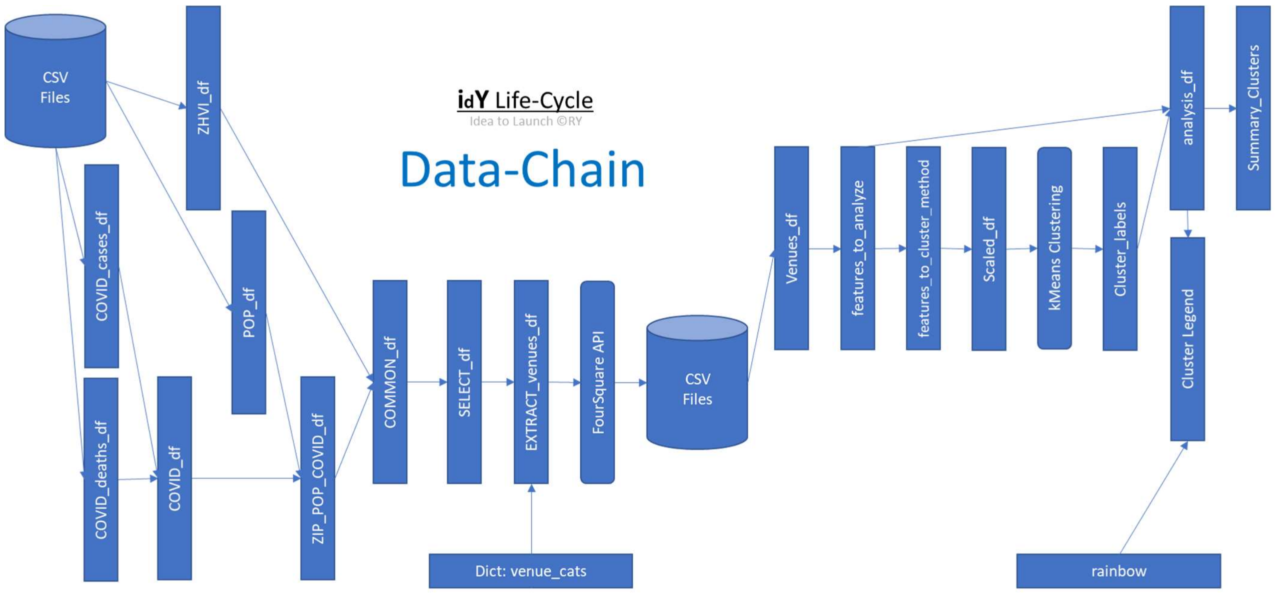
6.3 Data Sculpting

Sculpting implies taking cleaned data and preparing it for the required analytics goals. Some of the key aspects to keep in mind while sculpting your data,

1. Performance, how long does it take to run ?
2. Flexibility of data structure to expand the scope of analysis
3. Is there a need to change data extract and clean-up strategy ?

"idY" runs with a strong belief in doing things right, in-line with an evolving (not static/rigid) expectation, hence, all the feedback loops that take you back – "a step forward and another back is better than rushing 20 steps forward to a crash landing." Define data-chain and evaluate integration touchpoints to the ongoing app design. It maybe necessary to redo your data sculpting strategy based on the needs of Simulation and Modelling exercise.

Data Chain to meet the analytical demands of this project is illustrated below.



Ref: Jupyter Notebook, Section 6.3

Some of the key sculpting tasks include,

- Derive COVID Cases and Deaths by ZIP – 6.3.1

   - COVID data is available by county. Calculate population data by County and use it as the denominator to distribute the COVID data by ZIP.
      - COVID-ZIP = (COVID-CASES(or DEATHS) * POPULATION-ZIP)/POPULATION-COUNTY
- Build a common data frame by joining all data components
   - COMMON = Join by ZIP(Population, Home-prices, COVID)
- Final Data Filter
   - Create a more focussed/reduced dataset "SELECT" from the COMMON all-inclusive dataset
   - If there is scope for reducing the data size based on certain criteria in the interest of performance; this analysis will include all the ZIP records from Washington DC dataset
   - Some of the examples of this filtering could be,
      - Include only those records that belong in the Population inter-quartile range
      - Include only those records that have home prices within a required range
- Venue Data
   - Pull data from Foursquare using the functions and credentials defined in the previous section (Jupyter Notebook: 6.2.4)
   - Use SELECT dataframe as the ZIP reference to call the Foursquare API
   - The given call for CA state ZIPs created 9 files, for the selected call parameters,
      - Venue categories limited to venue_cats dictionary,

```
venue_cats = {
'Arts & Entertainment': '4d4b7104d754a06370d81259',
'Event': '4d4b7105d754a06373d81259',
'Nightlife Spot': '4d4b7105d754a06376d81259',
'Outdoors & Recreation': '4d4b7105d754a06377d81259',
'College & University': '4d4b7105d754a06372d81259',
'Travel & Transport': '4d4b7105d754a06379d81259',
'Professional & Other Places': '4d4b7105d754a06375d81259',
'Food': '4d4b7105d754a06374d81259',
'Shop & Service': '4d4b7105d754a06378d81259',
'Residence': '4e67e38e036454776db1fb3a',
'School': '4bf58dd8d48988d13b941735',
'Medical Center': '4bf58dd8d48988d104941735',
'Hospital': '4bf58dd8d48988d196941735',
'Spiritual Center': '4bf58dd8d48988d131941735',
}
```

     The above list of categories includes the top most hierarchy published by Foursquare, plus some more at the lower levels that may be of interest to enhance or impact liveability, such as, "Schools", "Medical Center", "Hospital", "Spiritual Center", which were part of "Professional & Other Places".
      - Radius of 2 KMs
      - this radius may not be appropriate for suburbs or rural areas, but we will proceed for now
   - Build venue count dataframe and store the data to local CSV file database
   - Extract venue data from the stored CSV file database
   - Remove null or NA data
   - Remove empty venue data
      - Remove records with zero venues data across all venues

7. Simulation and Modelling

It's time to execute our analytics strategy. For this analysis/project, lets focus on KMeans algorithm for clustering. We will execute upon the following key steps to fulfil this analysis,

1. Build a Feature Matrix
2. Evaluate for best k-value
3. Build Cluster: Perform clustering with the selected value of k
4. Evaluate individual clusters

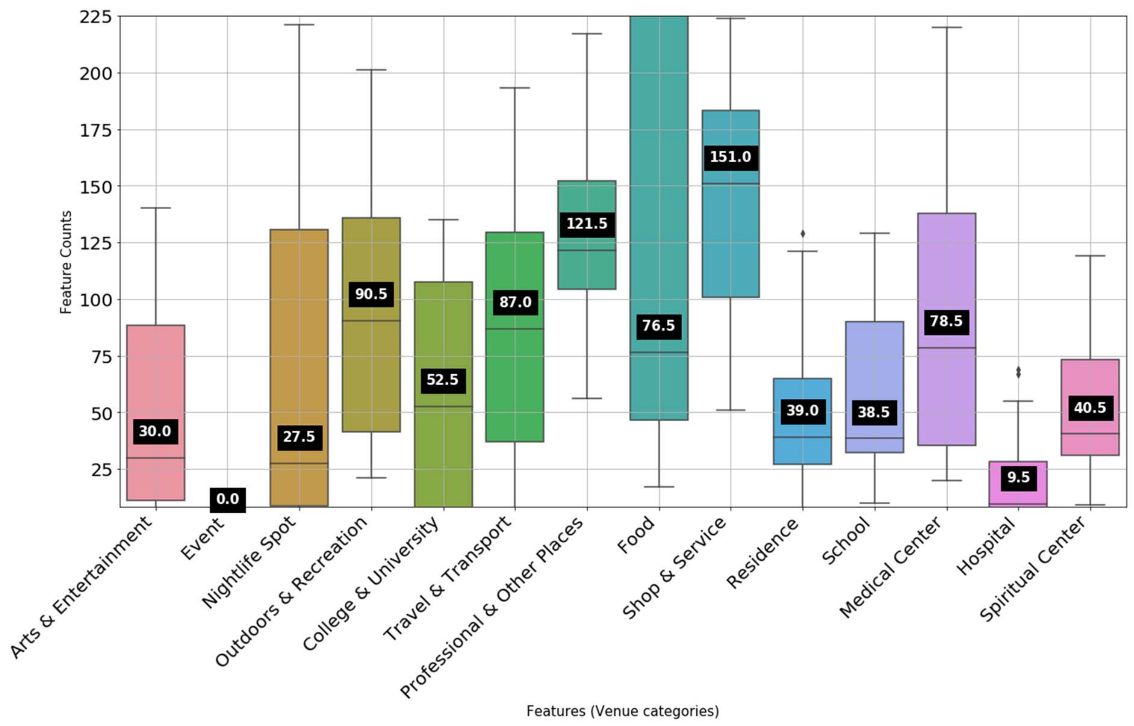
7.1 Build a Feature Matrix

Features are key characteristics or attributes to be used as a basis for identifying clusters. Our dataset includes 2 types of features:

1. Foursquare venue data which tells us about various venues available at a given neighbourhood, to enrich liveability
2. Non-venue data like population, population density, COVID cases/deaths, and home affordability, which will be further influence choice of a neighbourhood. We will not use non-venue data for clustering but will analyze the same for clusters derived using venue data, to make the final recommendations/decisions on liveability.

Ref: Jupyter Notebook, Section 7.1.2

Box-plot of Venue Feature Counts



Analysis:

- *Most venue categories seem to have a wide distribution across records, in Washington DC area.*
- *"Event" venue category seems to have a negligible presence, in DC area. We will retain it in the model to let the model be sufficiently generalized across states. A more intelligent or dynamic filter can be applied while designing the main app*