**Project ETL - Extract, Transform, Load**

**Team members**

Swati Swain

Emi Rivera

Nicholas Prentkowski

Terrence Cummings

**Minneapolis Police Incidents, Use of Force, and Demographics**

Project Objective:

To create a database that will facilitate an analysis of the relationship between police incidents, the use of force by police in those incidents, and the demographic characteristics of the neighborhood in which the incidents occurred such as race and income.

The team will extract data from various sources, clean and normalize the data, and load the data into a Postgres database. Such data can then be used by applications to allows analysis of relationships and trends between these factors.

We chose relational database for this project for the below reasons:

1. We are working with complex queries and reports. With SQL we can build one script that retrieves and presents the data easily. Running queries in NoSQL is doable, but much slower.
2. We have a high transaction application. SQL databases are a better fit for heavy duty or complex transactions because it’s more stable and ensure data integrity.
3. We need to ensure ACID compliance (Atomicity, Consistency, Isolation, Durability) or defining exactly how transactions interact with a database.
4. We don’t anticipate a lot of changes or growth.

The target data, for the city of Minneapolis, includes:

1. Police incident data by neighborhood
2. Police “use of force” data by neighborhood
3. Income levels by neighborhood
4. Race distribution by neighborhood

Please find following Table of Contents directing to ERD, SQL, and detailed ETL flow for each dataset.

Table of Contents

[Entity Relationship Diagram (ERD) 4](#_Toc42540020)

[SQL for PostgreSQL tables 5](#_Toc42540021)

[Minneapolis Communities & Neighborhoods 10](#_Toc42540022)

[Description 10](#_Toc42540023)

[Data sources 10](#_Toc42540024)

[Data extraction 10](#_Toc42540025)

[Data transformation 10](#_Toc42540026)

[Data loading 10](#_Toc42540027)

[Jupyter notebooks 11](#_Toc42540028)

[Police Incidents 12](#_Toc42540029)

[Description 12](#_Toc42540030)

[Data sources 12](#_Toc42540031)

[Data extraction 12](#_Toc42540032)

[Data transformation 12](#_Toc42540033)

[Data loading 12](#_Toc42540034)

[Police Use of Force 13](#_Toc42540035)

[Description 13](#_Toc42540036)

[Data sources 13](#_Toc42540037)

[Data extraction 13](#_Toc42540038)

[Data transformation 13](#_Toc42540039)

[Data loading 14](#_Toc42540040)

[Neighborhood Race Demographics 15](#_Toc42540041)

[Description 15](#_Toc42540042)

[Data sources 15](#_Toc42540043)

[Data extraction 15](#_Toc42540044)

[Data transformation 15](#_Toc42540045)

[Data loading 16](#_Toc42540046)

[Jupyter Notebook 16](#_Toc42540047)

[Neighborhood Income Demographics 17](#_Toc42540048)

[Description 17](#_Toc42540049)

[Data sources 17](#_Toc42540050)

[Data extraction 17](#_Toc42540051)

[Jupyter Notebook: 17](#_Toc42540052)

[Jupyter Notebook: 18](#_Toc42540053)

[Data transformation 18](#_Toc42540054)

[Jupyter notebooks: 18](#_Toc42540055)

[Data loading 18](#_Toc42540056)

[Jupyter notebooks: 19](#_Toc42540057)

[Web Application 20](#_Toc42540058)

# Entity Relationship Diagram (ERD)

A screenshot of a cell phone

Description automatically generated

# SQL for PostgreSQL tables

|  |
| --- |
| --COMMUNITY |
|  | DROP TABLE IF EXISTS COMMUNITY CASCADE; |
|  | CREATE TABLE COMMUNITY ( |
|  | community\_id INT NOT NULL, |
|  | name VARCHAR(50) NOT NULL, |
|  | CONSTRAINT pk\_COMMUNITY PRIMARY KEY ( |
|  | community\_id |
|  | ), |
|  | CONSTRAINT uc\_COMMUNITY\_name UNIQUE ( |
|  | name |
|  | ) |
|  | ); |
|  |  |
|  | --NEIGHBORHOOD |
|  | DROP TABLE IF EXISTS NEIGHBORHOOD CASCADE; |
|  | CREATE TABLE NEIGHBORHOOD ( |
|  | neighborhood\_id INT NOT NULL, |
|  | name VARCHAR(50) NOT NULL, |
|  | community\_id INT NOT NULL, |
|  | CONSTRAINT pk\_NEIGHBORHOOD PRIMARY KEY ( |
|  | neighborhood\_id |
|  | ), |
|  | CONSTRAINT uc\_NEIGHBORHOOD\_name UNIQUE ( |
|  | name |
|  | ) |
|  |  |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD CASCADE; |
|  | CREATE TABLE HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD ( |
|  | household\_income\_by\_neighborhood\_id INT NOT NULL, |
|  | neighborhood\_id INT NOT NULL, |
|  | IncomeLess35000\_count INT DEFAULT NULL, |
|  | IncomeLess35000\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess35to49\_count INT DEFAULT NULL, |
|  | IncomeLess35to49\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess50to74\_count INT DEFAULT NULL, |
|  | IncomeLess50to74\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess75to99\_count INT DEFAULT NULL, |
|  | IncomeLess75to99\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | Income100Plus\_count INT DEFAULT NULL, |
|  | Income100Plus\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | Median\_Income\_Total DECIMAL(10,2) DEFAULT NULL, |
|  | CONSTRAINT pk\_HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD PRIMARY KEY ( |
|  | household\_income\_by\_neighborhood\_id |
|  | ), |
|  | CONSTRAINT uc\_HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD\_neighborhood\_id UNIQUE ( |
|  | neighborhood\_id |
|  | ) |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS HOUSEHOLD\_INCOME\_BY\_COMMUNITY CASCADE; |
|  | CREATE TABLE HOUSEHOLD\_INCOME\_BY\_COMMUNITY ( |
|  | household\_income\_by\_community\_id INT NOT NULL, |
|  | community\_id INT NOT NULL, |
|  | IncomeLess35000\_count INT DEFAULT NULL, |
|  | IncomeLess35000\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess35to49\_count INT DEFAULT NULL, |
|  | IncomeLess35to49\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess50to74\_count INT DEFAULT NULL, |
|  | IncomeLess50to74\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | IncomeLess75to99\_count INT DEFAULT NULL, |
|  | IncomeLess75to99\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | Income100Plus\_count INT DEFAULT NULL, |
|  | Income100Plus\_percent DECIMAL(10,2) DEFAULT NULL, |
|  | Median\_Income\_Total DECIMAL(10,2) DEFAULT NULL, |
|  | CONSTRAINT pk\_HOUSEHOLD\_INCOME\_BY\_COMMUNITY PRIMARY KEY ( |
|  | household\_income\_by\_community\_id |
|  | ), |
|  | CONSTRAINT uc\_HOUSEHOLD\_INCOME\_BY\_COMMUNITY\_community\_id UNIQUE ( |
|  | community\_id |
|  | ) |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS RACE\_BY\_NEIGHBORHOOD CASCADE; |
|  | CREATE TABLE RACE\_BY\_NEIGHBORHOOD ( |
|  | race\_by\_neighborhood\_id SERIAL NOT NULL, |
|  | neighborhood\_id INT NOT NULL, |
|  | total\_cnt DECIMAL(10,2) DEFAULT NULL, |
|  | white\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | black\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | native\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | asian\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | other\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | two\_or\_more\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | hispanic\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | of\_color\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | CONSTRAINT pk\_RACE\_BY\_NEIGHBORHOOD PRIMARY KEY ( |
|  | race\_by\_neighborhood\_id |
|  | ), |
|  | CONSTRAINT uc\_RACE\_BY\_NEIGHBORHOOD\_neighborhood\_id UNIQUE ( |
|  | neighborhood\_id |
|  | ) |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS RACE\_BY\_COMMUNITY CASCADE; |
|  | CREATE TABLE RACE\_BY\_COMMUNITY ( |
|  | race\_by\_community\_id SERIAL NOT NULL, |
|  | community\_id INT NOT NULL, |
|  | total\_cnt DECIMAL(10,2) DEFAULT NULL, |
|  | white\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | black\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | native\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | asian\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | other\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | two\_or\_more\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | hispanic\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | of\_color\_pct DECIMAL(10,5) DEFAULT NULL, |
|  | CONSTRAINT pk\_RACE\_BY\_COMMUNITY PRIMARY KEY ( |
|  | race\_by\_community\_id |
|  | ), |
|  | CONSTRAINT uc\_RACE\_BY\_COMMUNITY\_community\_id UNIQUE ( |
|  | community\_id |
|  | ) |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS POLICE\_USE\_OF\_FORCE CASCADE; |
|  | CREATE TABLE POLICE\_USE\_OF\_FORCE ( |
|  | police\_use\_of\_force\_id SERIAL NOT NULL, |
|  | response\_date TIMESTAMP DEFAULT NULL, |
|  | case\_number VARCHAR(50) DEFAULT NULL, |
|  | problem VARCHAR(100) DEFAULT NULL, |
|  | subject\_race VARCHAR(50) DEFAULT NULL, |
|  | subject\_sex VARCHAR(50) DEFAULT NULL, |
|  | subject\_age INT DEFAULT NULL, |
|  | subject\_role VARCHAR(50) DEFAULT NULL, |
|  | primary\_offense VARCHAR(50) DEFAULT NULL, |
|  | type\_of\_resistance VARCHAR(50) DEFAULT NULL, |
|  | police\_use\_of\_force\_type VARCHAR(50) DEFAULT NULL, |
|  | force\_type\_action VARCHAR(50) DEFAULT NULL, |
|  | subject\_injury VARCHAR(50) DEFAULT NULL, |
|  | neighborhood\_id INT DEFAULT NULL, |
|  | neighborhood VARCHAR(50) DEFAULT NULL, |
|  | precinct VARCHAR(10) DEFAULT NULL, |
|  | CONSTRAINT pk\_POLICE\_USE\_OF\_FORCE PRIMARY KEY ( |
|  | police\_use\_of\_force\_id |
|  | ), |
|  | CONSTRAINT uc\_POLICE\_USE\_OF\_FORCE\_case\_number UNIQUE ( |
|  | case\_number |
|  | ) |
|  | ); |
|  |  |
|  | DROP TABLE IF EXISTS POLICE\_INCIDENT CASCADE; |
|  | CREATE TABLE POLICE\_INCIDENT ( |
|  | police\_incident\_id SERIAL NOT NULL, |
|  | casenumber VARCHAR(50) DEFAULT NULL, |
|  | reporteddate TIMESTAMP DEFAULT NULL, |
|  | offense VARCHAR(100) DEFAULT NULL, |
|  | neighborhood VARCHAR(100) DEFAULT NULL, |
|  | incident\_id INT DEFAULT NULL, |
|  | neighborhood\_id INT DEFAULT NULL, |
|  | community\_id INT DEFAULT NULL, |
|  | CONSTRAINT pk\_POLICE\_INCIDENT PRIMARY KEY ( |
|  | police\_incident\_id |
|  | ), |
|  | CONSTRAINT uc\_POLICE\_INCIDENT\_incident\_id UNIQUE ( |
|  | incident\_id |
|  | ) |
|  | ); |
|  |  |
|  |  |
|  | --Foreign Keys |
|  | ALTER TABLE NEIGHBORHOOD ADD CONSTRAINT fk\_NEIGHBORHOOD\_community\_id FOREIGN KEY(community\_id) |
|  | REFERENCES COMMUNITY (community\_id); |
|  |  |
|  |  |
|  | ALTER TABLE HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD ADD CONSTRAINT fk\_HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD\_neighborhood\_id FOREIGN KEY(neighborhood\_id) |
|  | REFERENCES NEIGHBORHOOD (neighborhood\_id); |
|  |  |
|  | ALTER TABLE HOUSEHOLD\_INCOME\_BY\_COMMUNITY ADD CONSTRAINT fk\_HOUSEHOLD\_INCOME\_BY\_COMMUNITY\_community\_id FOREIGN KEY(community\_id) |
|  | REFERENCES COMMUNITY (community\_id); |
|  |  |
|  | ALTER TABLE HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD ADD CONSTRAINT fk\_HOUSEHOLD\_INCOME\_BY\_NEIGHBORHOOD\_neighborhood\_id FOREIGN KEY(neighborhood\_id) |
|  | REFERENCES NEIGHBORHOOD (neighborhood\_id); |
|  |  |
|  | ALTER TABLE HOUSEHOLD\_INCOME\_BY\_COMMUNITY ADD CONSTRAINT fk\_HOUSEHOLD\_INCOME\_BY\_COMMUNITY\_community\_id FOREIGN KEY(community\_id) |
|  | REFERENCES COMMUNITY (community\_id); |
|  |  |
|  | ALTER TABLE RACE\_BY\_COMMUNITY ADD CONSTRAINT fk\_RACE\_BY\_COMMUNITY\_community\_id FOREIGN KEY(community\_id) |
|  | REFERENCES COMMUNITY (community\_id); |
|  |  |
|  | ALTER TABLE POLICE\_USE\_OF\_FORCE ADD CONSTRAINT fk\_POLICE\_USE\_OF\_FORCE\_neighborhood\_id FOREIGN KEY(neighborhood\_id) |
|  | REFERENCES NEIGHBORHOOD (neighborhood\_id); |
|  |  |
|  | ALTER TABLE POLICE\_INCIDENT ADD CONSTRAINT fk\_POLICE\_INCIDENT\_neighborhood\_id FOREIGN KEY(neighborhood\_id) |
|  | REFERENCES NEIGHBORHOOD (neighborhood\_id); |
|  |  |
|  | ALTER TABLE POLICE\_INCIDENT ADD CONSTRAINT fk\_POLICE\_INCIDENT\_community\_id FOREIGN KEY(community\_id) |
|  | REFERENCES COMMUNITY (community\_id); |

# Minneapolis Communities & Neighborhoods

## Description

The objective is to set up reference tables for Minneapolis Communities and Neighborhoods. The community\_id/neighborhood\_id(s) will be used as foreign keys in all tables to aggregate data at community and neighborhood level.

## Data sources

Data is obtained from Open Data Minneapolis Police Incident reports in form of csv files.

Source file location in project folder:

* 1. source\_files\MLPS\_Communities\_raw.csv
  2. source\_files\MLPS\_Neighborhoods\_raw.csv

## Data extraction

Use the raw csv files from MINNESOTA COMPASS (mncompass.org) for Community and Neighborhood data.

## Data transformation

* **Community Data transformation:**

1. Import MLPS\_Communities\_raw.csv from source\_files into a dataframe.
2. Extract the final list of fields from the dataframe to match the database table.
3. Rename fields in dataframe which don’t match the SQL table.
4. Start the index field from 1(increment by 1 as well) in the final df and rename index field to match the column in the table.

* **Neighborhood Data transformation:**

1. Import MLPS\_Neighborhoods\_raw.csv from source\_files into a dataframe.
2. Import the community table from Postgres to get the community names and IDs.
3. Join the neighborhood dataframe with the community dataframe on the community names to get the cmonnunity ID.
4. Validate count of records before and after the join to ensure no records were dropped.
5. Extract the final list of fields from the dataframe to match the database table.
6. Rename fields in dataframe which don’t match the SQL table.
7. Start the index field from 1(increment by 1 as well) in the final df and rename index field to match the column in the table.

## Data loading

Below steps were performed to load MLS neighborhoods and communities tables in Postgres:

1. Postgres tables: neighborhood & community were created using the SQL script generated from the ERD diagram.
2. Neighborhood and Community data was loaded to their respective tables using SQL Alchemy through Python/pandas.

## Jupyter notebooks

* **Community** **data**: 1\_MLS\_Community\_Data\_ETL.ipnyb
* **Neighborhood** **data**: 2\_MLS\_Neighborhood\_Data\_ETL.ipnyb

# Police Incidents

## Description

Any time the police are dispatched to an emergency call or respond on their own, an officer must file an incident report. The police in Minneapolis (MPD) keep track of these incidents utilizing an electronic records management systems. Prior to 2018 the MPD were using a system called CAPRS, in mid 2018 they began using PIMS which offered more flexibility and technological ability.

## Data sources

We used Open Minneapolis to obtain the Police Incident data. Open Minneapolis is a website that has a variety of Minneapolis data provided by the City of Minneapolis.

## Data extraction

The data was extracted from Open Minneapolis via CSV’s. We utilized data spanning 2015 to 2019. In total six data sets were utilized, one CSV per year. Additionally, the 2018 data was split into two CSV’s due to the recording system changing halfway through the year (there was one data set for the beginning of the year and another once the system changed).

## Data transformation

Data transformations were executed in Python.

* The first transformation to occur was a union of all of the six files into one police incident dataset spanning the four years. An additional challenge was posed due to the fact the system change also changed the formatting of the CSV’s. The variables remained relatively similar but the names and order of the variables changed.
* Most of the fields were irrelevant to our analysis so they were dropped.
* The neighborhood field was a pivotal piece to our project, any police incident with a missing neighborhood name was dropped.
* The last major transformation was merging the final police incident dataset with a neighborhood dataset in order to obtain a unique neighborhood and community ID. There were transformations made to the neighborhood names in both datasets as these did not match across all of the neighborhoods.
* Once all neighborhood names matched these datasets were joined together.
* The ultimate file contained the case number, reported date, offense, neighborhood, incident ID, neighborhood ID, and a community ID.

## Data loading

The ultimate file was exported as a CSV and was loaded into our PostgresSQL database.

# Police Use of Force

## Description

The objective is to obtain data regarding the incidents during which Minneapolis police officers deemed the use of force necessary.

## Data sources

The primary source utilized to meet the specified objective was a data frame found on the webpage titled, *Police Use of Force*, which can be found on the website titled, Open Minneapolis. The page can be found via the following hyperlink: <http://opendata.minneapolismn.gov/datasets/police-use-of-force/data?geometry=-103.617%2C-5.468%2C10.289%2C48.789&orderBy=ResponseDate&orderByAsc=false>

## Data extraction

The source data was extracted via a comma-separated values file (.CSV) that was first downloaded locally and finally pushed onto our project team’s repository. It is specifically stored in the folder titled, “source\_file.”

## Data transformation

Transformation (cleaning) involved the following steps:

1. Declaring and assigning a variable to the CSV…
2. Reading in the CSV by using the read\_csv function, which will produce and store a Pandas data frame…
3. Dropping unessential data fields…
4. Renaming remaining fields to match entity-relationship diagram (ERD)…
5. Using .dtypes code to determine type of value held in the response\_date field…
6. Using astype function and Numpy to convert response\_date field to datetime64…
7. Declaring and assigning a variable to the MLS\_Neighborhoods CSV…
8. Changing the field name titled: name, in new data frame to, neighborhood…
9. Using the replace function to match the spelling and punctuation of the ten neighborhoods that conflict with one another when trying to merge the two data frames…
10. Merging the two data frames on the field, neighborhood and via a left join…
11. Use double brackets to rearrange the order of the fields of data frame to match the ERD…
12. Use .dtypes to check, or refer to the last time it was used to see what type of values can be found in the subject\_age field…
13. Change all NaN(s) within subject\_age field to 0 via fillna, which will allow for conversion to int64…
14. Convert subject\_age field to int64 via astype function…
15. Change all NaN(s) within neighborhood\_id field to 0 via fillna…
16. Convert values in neighborhood\_id from float64 to int64 via .astype…
17. Rename final data frame to something more concise and clearer…
18. Export as CSV to the folder, target\_files…

## Data loading

The steps are as follows:

1. Create tables in PostgreSQL using the SQL script based on the ERD presented at the beginning of this document.
2. Use Sqlalchemy (from sqlalchemy import create\_engine) to connect to PostgreSQL database.
3. Use Pandas df.to\_sql to populate PostgreSQL tables with Pandas dataframe values.

# Neighborhood Race Demographics

## Description

The objective is to obtain data regarding the racial mix of Minneapolis neighborhoods and communities.

## Data sources

Data is obtained from MINNESOTA COMPASS (mncompass.org). We need to scrape data from the following endpoints:

1. Scrape links to Minneapolis neighborhood-specific webpage on mncompass.org found on:
   1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis-saint-paul>
2. Scrape links to Minneapolis community-specific webpage on mncompass.org found on:
   1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis-saint-paul>
3. Scrape race data for each Minneapolis neighborhood links obtained in step 1. For example, Armatage neighborhood at:
   1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis/armatage>
4. Scrape race data for each Minneapolis community links obtained in step 2. For example, Camden at:
   1. <http://www.mncompass.org/profiles/communities/minneapolis/camden>

## Data extraction

Selenium webdriver (from selenium import webdriver) was used to scrape data at the URL. This is because the data is populated by Javascript and therefore not accessible by Splinter.

Extraction followed the following process:

1. Scrape the individual neighborhood and community links and store in lists of URLs.
2. Send the webdriver to each link in the lists and scrape the race data from each page.

The neighborhood and community race data is then stored in a Pandas dataframe and written to csv files.

## Data transformation

Transformation (cleaning) involved the following steps:

1. Read in the csv’s from extraction as Pandas dataframes.
2. The scraped data contained the word ‘suppressed’ in some table cells. Replace this with NaN so all missing data is represented by NaN.
3. Convert text-styled numbers into numeric type.
4. Add a ‘total’ column as the sum of the individual race columns.
5. Use pd.merge to bring in neighborhood and community ID’s that will be used in PostgreSQL keys.
6. Delete extraneous columns.
7. Reorder columns for presentability.

## Data loading

Steps:

1. Create tables in PostgreSQL using the SQL script based on the ERD presented at the beginning of this document.
2. Use Sqlalchemy (from sqlalchemy import create\_engine) to connect to PostgreSQL database.
3. Use Pandas df.to\_sql to populate PostgreSQL tables with Pandas dataframe values.

## Jupyter Notebook

1. 6\_MLS\_Race\_Data\_Extract.ipynb
2. 6a\_MLS\_Race\_Data\_Transform.ipynb

# Neighborhood Income Demographics

## Description

The objective is to obtain data regarding the household income of Minneapolis neighborhoods and communities.

## Data sources

Data is obtained from MINNESOTA COMPASS (mncompass.org). We need to scrape data from the following endpoints:

1. Scrape links to Minneapolis neighborhood and community specific webpage on mncompass.org from:

* 1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis-saint-paul>

1. Scrape household income data for each Minneapolis neighborhood links obtained in step 1. For example, Downtown West neighborhood at:
   1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis/downtown-west>
2. Scrape household income data for each Minneapolis community links obtained in step 2. For example, Loring Park at:
   1. <http://www.mncompass.org/profiles/neighborhoods/minneapolis/loring-park>

## Data extraction

The webpages in MN Compass are populated via java script. Selenium package is used to automate web browser interaction from Python. Selenium will start a browser session. The python\_button.click() is used to tell Selenium to click the JavaScript link on the page. After arriving at the individual MLS neighborhood page, Selenium hands off the page source to Beautiful Soup. Beautiful Soup will grab all the rendered data on the page that matches the required class and IDs.

Extraction of neighborhood household income data:

1. Scrape all individual neighborhood part-links from <http://www.mncompass.org/profiles/neighborhoods/minneapolis-saint-paul> page with **class\_='minneapolis-neighborhoods-listing'**
2. Append **'http://www.mncompass.org/profiles/neighborhoods/'** to the part links from step 1 to create complete links for neighborhood pages.
3. Loop through the neighborhood links in step 2 to fetch household income data for all available MLS links using a combination of Selenium and Beautiful Soup.
4. Save the final data set as **‘MLPS\_Hsld\_Income\_by\_Neighborhood.csv’** in resources folder.

### Jupyter Notebook:

* 3a\_MLS\_Income\_by\_Neighborhood\_Extract.ipnyb

Extraction of community household income data:

1. Scrape all individual neighborhood part-links from <http://www.mncompass.org/profiles/neighborhoods/minneapolis-saint-paul> page with **class\_=** ‘**minneapolis-communities-listing’**
2. Append **'http://www.mncompass.org/profiles/neighborhoods/'** to the part links from step 1 to create complete links for neighborhood pages.
3. Loop through the neighborhood links in step 2 to fetch household income data for all available MLS links using a combination of Selenium and Beautiful Soup.
4. Save the final data set as **‘MLPS\_Hsld\_Income\_by\_Community.csv’** in resources folder.

### Jupyter Notebook:

* 4a\_MLS\_Income\_by\_Community\_Extract.ipnyb

## Data transformation

Below steps were performed to clean/transform household income data for MLS neighborhoods and communities:

1. Import the csvs saved in the ‘Extract’ steps into a dataframe.
2. Replace **\n**, commas(**,**), **%**, **$** in source data with **''**
3. Trim whitespace from all values across all fields in the dataframe.
4. Median income field: Extract the median income by removing '$ ' and converting to float.
5. Replace 'suppressed' and blank fields with ‘**NaN**’ across the dataframe.
6. Convert all household income count fields to numeric.
7. Convert all household percent fields to float.
8. Import the neighborhood table from Postgres to get the neighborhood names and IDs.
9. Join the household income dataframe with the neighborhood dataframe on the neighborhood names to get the neighborhood ID.
10. Validate count of records before and after the join to ensure no records were dropped.
11. Extract the final list of fields from the dataframe to match the database table.
12. Rename fields in dataframe which don’t match the SQL table.
13. Start the index field from 1(increment by 1 as well) in the final df and rename index field to match the column in the table.

### Jupyter notebooks:

* **Neighborhood** **data**: 3b\_MLS\_Income\_by\_Neighborhood\_Transform\_Load.ipnyb
* **Community** **data**: 4b\_MLS\_Income\_by\_Community\_Transform\_Load.ipnyb

## Data loading

Below steps were performed to load household income data for MLS neighborhoods and communities:

1. Postgres tables: household\_income\_by\_neighborhood & household\_income\_by\_community were created using the SQL script generated from the ERD diagram.
2. Household income data for neighborhoods and communities were loaded to their respective tables using SQL Alchemy through Python/pandas.

### Jupyter notebooks:

* **Neighborhood** **data**: 3b\_MLS\_Income\_by\_Neighborhood\_Transform\_Load.ipnyb
* **Community** **data**: 4b\_MLS\_Income\_by\_Community\_Transform\_Load.ipnyb

# Web Application

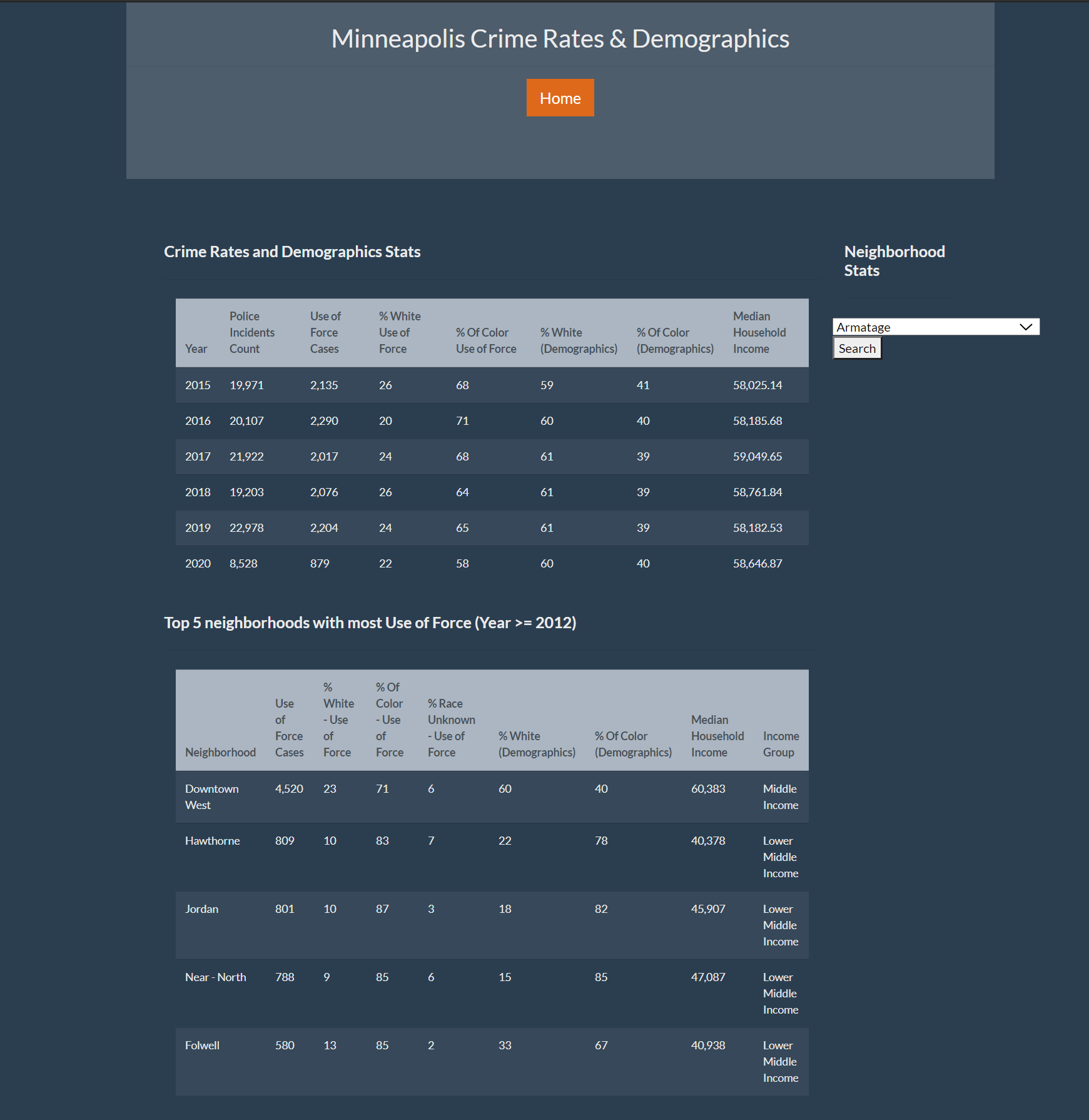
The app summarizes police incidents/ use of force by police/ demographics (race and household income) for Minneapolis and as well as individual neighborhoods in Minneapolis by Year(>=2015). App file name: app.py

The app gets its data from views built in Postgres (script loc: SQL\View\_Scripts\_for\_App) on top of the tables.

The app has two routes as below:

1. **Root path (\) –** This uses **index.html** in templates folder to display data.

**Snapshot of Root (Home page) at a glance:**



**Root page broken down into individual components:**

**1-a**. **Jumbotron component with Home button**: Summary header with a link to Home page (root)

**1-b.** **Table summarizing** Total police incidents in Minneapolis by Year along with police use of force data split by percentages for use of force race and demographics data (race and median household income).

Underlying view: **vw\_minneapolis\_stats**

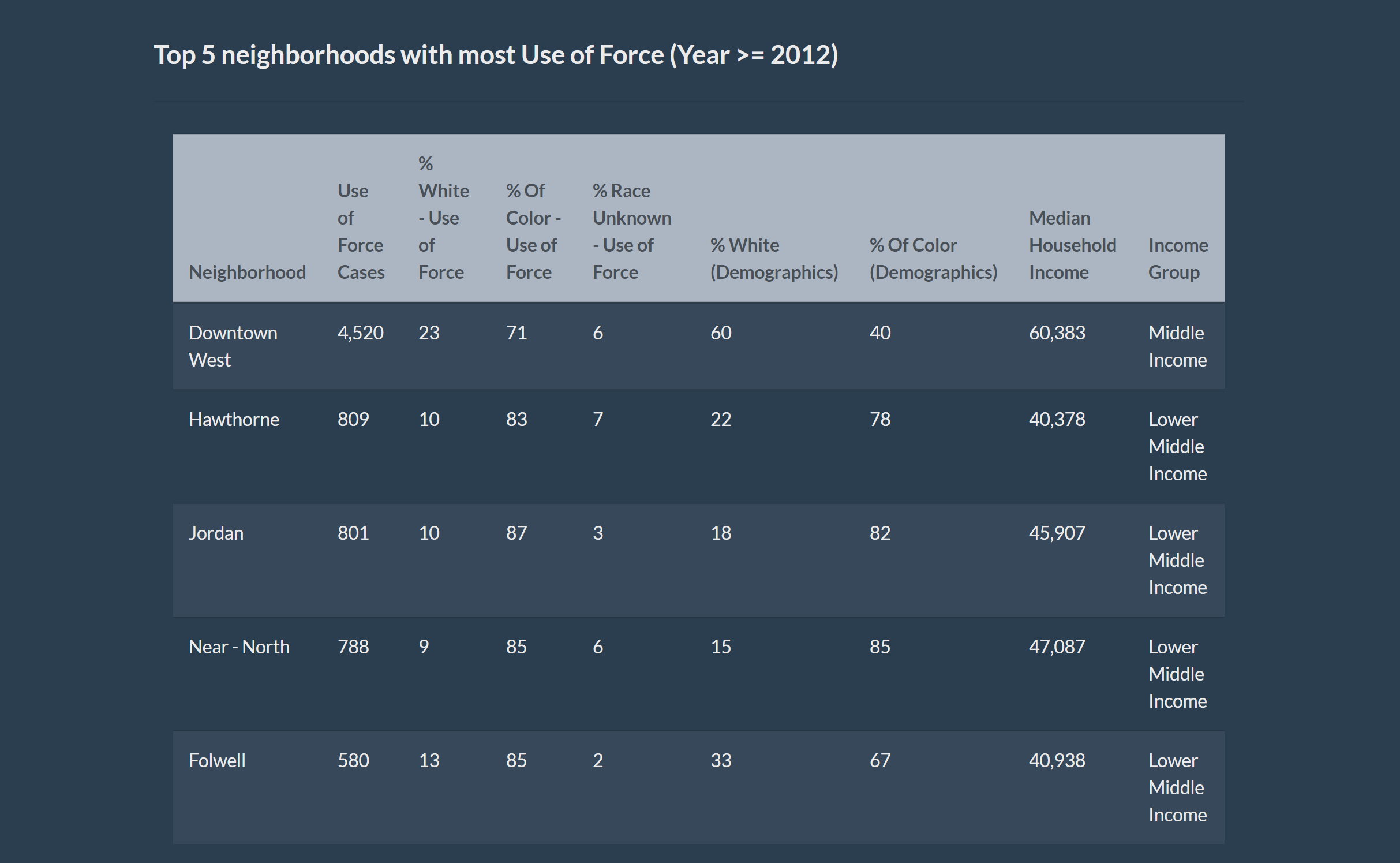
1-a



1-b

**1-c: Table summarizing** data for maximum police use of force in top 5 neighborhoods in Minneapolis along with Demographics data.

Underlying view: **vw\_police\_use\_of\_force\_summary**

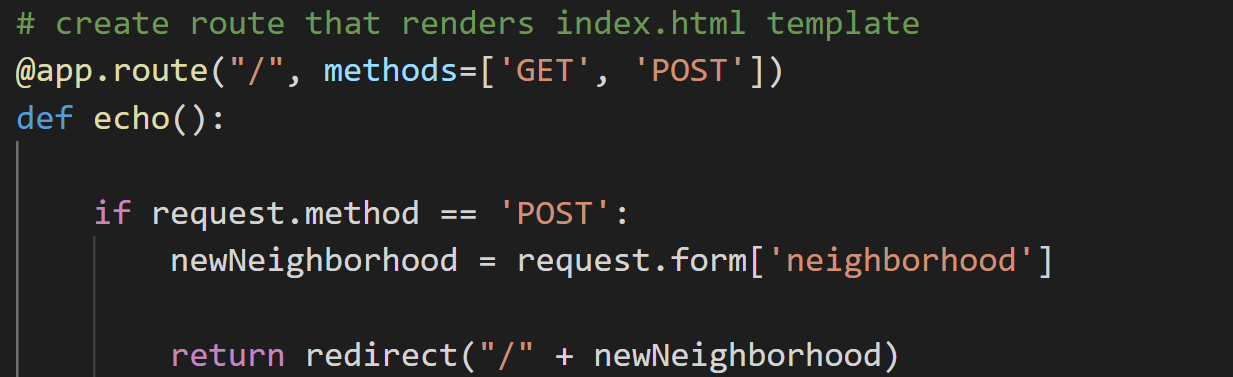


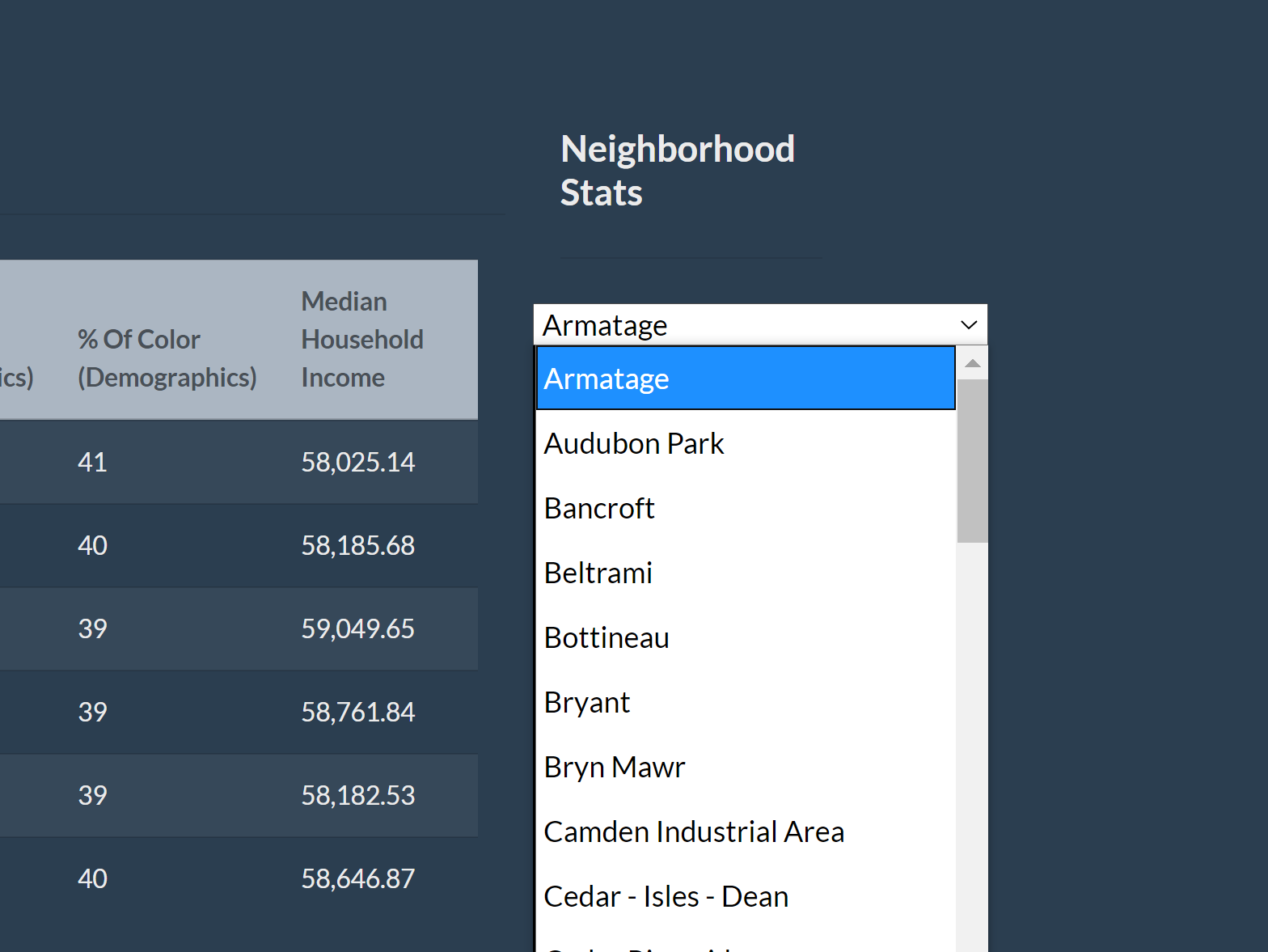
1-c

**1-d: Dropdown menu** listing out all neighborhoods in Minneapolis. Based on the neighborhood selected by the user, the app will redirect to another page summarizingpolice incidents and the demographics data for the selected neighborhoods in Minneapolis by Year (>=2015).

Underlying table: **neighborhood**

**Section in app code redirecting the user input to the selected neighborhood dynamically:**

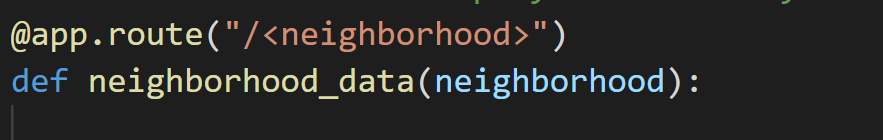


**Webpage with drop down menu:**

1-d

1. **Dynamic Path (to each neighborhood based on user selection in the dropdown in Home page) -** This uses **neighborhood.html** in templates folder to display data.

**Snapshot of app route -**



Below snapshot showing sample webpage where **neighborhood = ‘Downtown West’**

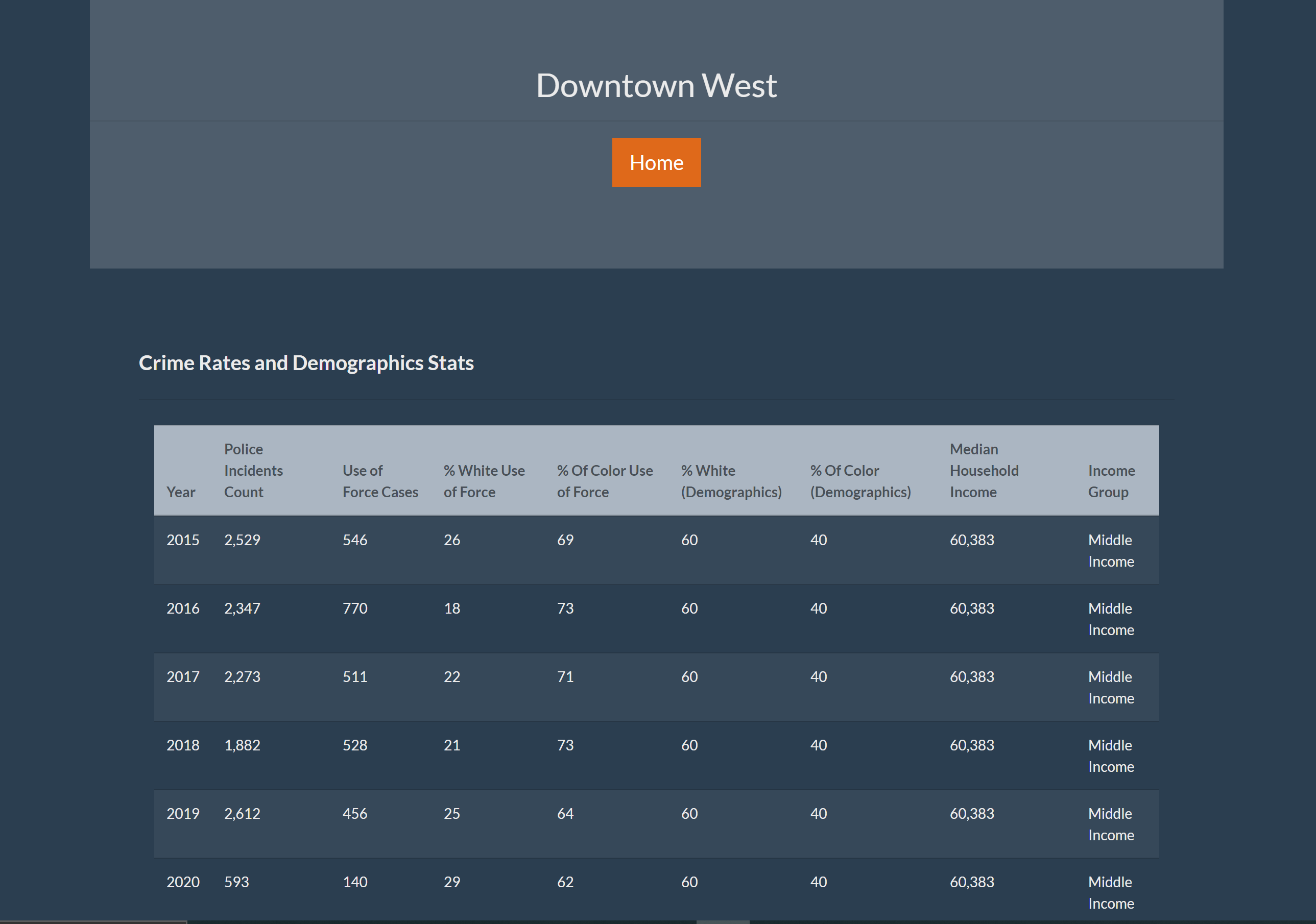
**2-a: Name** of the **selected neighborhood** (dynamically populated)

**2-b: Home button**: Button redirecting to the Home page

**2-c: Table summarizing** total police incidents in the **selected neighborhood** in Minneapolis by ‘Year’ along with police use of force data split by percentages for use of force race and demographics data (race and median household income).

**Underlying view:** vw\_mls\_neighborhood\_stats

2-a



2-b

2-c