HW02 – Creating the Crime Dataset

NOTE: This data is not actually Canton, MI.
This is a derived dataset for educational purposes only.

You must have been quite convincing with your proposal. Police Chief Croft wants to look into using predictive models to make better decisions and be more efficient. But before the type of models can be determined, the Police Chief would like to see the data in one dataset.

You are to combine Calls, Dispositions, and Census Data into one dataset. The dataset will be used for different supervised learning models. You have decided that the data should be:

- Aggregated into one dataset.
- Grouped based on each week and subzone. Therefore, for each week there will be up to 25 subzones represented. (Figure 1 is a screenshot of the calls data grouped by year – week – subzone named SUB_YEAR_WEEK.)
- For each SUB_YEAR_WEEK, the types of calls and dispositions will be totaled. (See Figure 2)

	Subzone	Call_Number	Complaint	Date_Received	Day_Name	WEEK	MONTH	YEAR	YEAR_WEEK	SUB_YEAR_WEEK	F
1	ZONE1D	08C-000010	Burglary	2008-09-18 10:36:00	Thursday	38	9	2008	2008_38	2008_38_ZONE1D	
2	ZONE2B	08C-000011	Burglary	2008-09-18 10:44:00	Thursday	38	9	2008	2008_38	2008_38_ZONE2B	
3	ZONE1C	08C-000020	Assault	2008-09-18 11:52:00	Thursday	38	9	2008	2008_38	2008_38_ZONE1C	
5	ZONE3D	08C-000029	Burglar alarm	2008-09-18 12:23:00	Thursday	38	9	2008	2008_38	2008_38_ZONE3D	
6	ZONE4B	08C-000030	Welfare check	2008-09-18 12:26:00	Thursday	38	9	2008	2008_38	2008_38_ZONE4B	

Figure 1: Screenshot of Calls with added columns including SUB_YEAR_WEEK

	SUB_YEAR_WEEK	call_Armed subject	call_Assault	call_Burglar alarm	call_Burglary	call_Disturbance	call_Domestic	call_FW FIREWORKS	call_Fight	call_Loitering	call_Message delivery
0	2008_38_ZONE1D	0	1	2	1	6	2	0	4	0	0
1	2008_39_ZONE1D	1	1	16	2	2	7	0	1	0	0
2	2008_40_ZONE1D	0	1	6	5	2	3	0	0	0	1
3	2008_41_ZONE1D	0	0	11	5	1	8	0	0	0	0
4	2008_42_ZONE1D	0	1	5	5	4	5	0	1	1	0

Figure 2: Screenshot of all calls aggregated by SUB_YEAR_WEEK

By aggregating the data by SUB_YEAR_WEEK, the Police Department will be able to predict weekly calls or dispositions based on each subzone. This allows the department to have a better feel for activity in a subzone and then allocate resources accordingly. Also included as data to use for this project is census data. Therefore, the data for this project is presented in the following ERD. See the appendix for a detailed list of tables and columns.

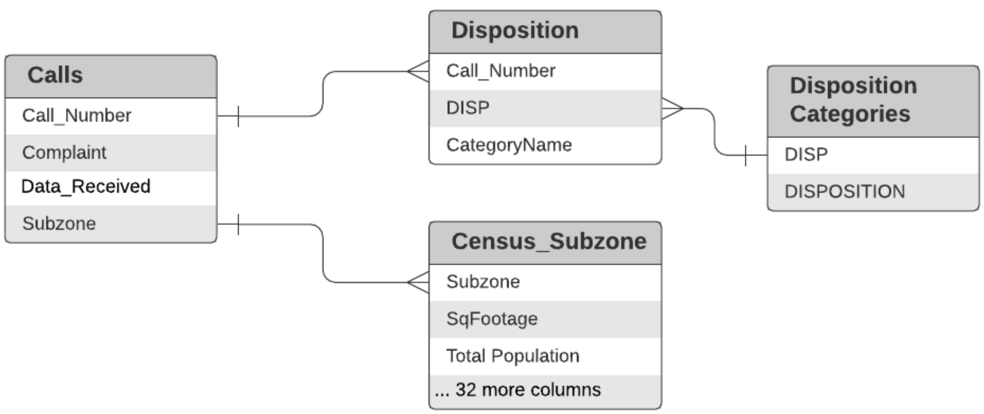

Figure 3: ERD for the data from the Canton Police Department

Table Description

- **Calls** - All calls for the past ten years are available. For example, if someone files a complaint for an assault, that would be one record.
- **Dispositions** - The result of the call/complaint. For example, if someone files a complaint for an assault, then the disposition would be what happened in the end, such as report filed or arrest.
- **Disposition Categories** – This is a translation table to format all dispositions to the same categorical names. For example, False Alarm in the DISP column may be FA, FA-Alarm, FalseAlarm, False Alarm, 71, 73, or 74 and the DISPOSITION column changes all of them to FA-FALSE ALARM.
- **Census Data** – 35 columns of census data totaled and summarized based on each of the 25 subzones.

Objective of this homework:

- Create a dataset that can be used for predictive modeling to predict weekly totals or if a call or disposition will occur.
- You will also incorporate the concept of spatial and time lag into the dataset. These are terms from crime literature are defined as:
  - **Spatial lag** is the effect of the surrounding areas (space) around a given Subzone and how it relates to crimes within the Subzone. Therefore, do surrounding areas with crime lead to crime in other areas.
  - **Time lag** is when activities happen in one week will have an effect on a future week. For example, if a crime happens in week 5 of 2020, will that lead to a crime in week 6?
  - *Will calls and/or dispositions from one week have an effect on a future week?*

Format of this Homework

It is very important that your Jupyter Notebook is formatted correctly with markdown, comments, and code that works. It is also very important to have the correct folder structure to get started.

The following provides guidance on how to format your HW.

- A **Section** is numbered as 1, 2, 3, etc.
- A **Subsection** is denoted as 5a., 5b, etc.

Each Section/Subsection of your homework should include:

- Section title as a Heading 2
- Subsection title as a Heading 3
- Each Section/Subsection may also include bulleted text for clarification.
- Code should include comments (#) to clarify steps.
- Markdown Summary at the end of the Section explaining the results and practical implications, these will be noted as subsections in the guidelines.

How to turn it in:

- Your Jupyter notebook file must be named HW02_LastnameFirstInitial.ipynb. For example, HW02_SmithJ.ipynb.
- **You are to turn in your Jupyter notebook file only. No data files and no folders.**
- It is assumed that you created your Jupyter notebook in a folder named HW02.

Create a folder on your computer

- Create a folder on your computer named HW02.
- Inside of HW02, make sure to include a data folder with all CSV files of data, including the CantonPoliceDept.db inside the data folder.

2. Create a Jupyter Notebook.

- Open up Anaconda and Jupyter Notebooks.
- In the HW02 folder, create a new notebook and name it HW02_LastNameFirstInitial.ipynb.

3. Import Libraries

- Create a code block to import the following libraries:
  - import numpy as np
  - import pandas as pd
  - import matplotlib.pyplot as plt
  - import seaborn as sns
  - import datetime as dt
  - from sqlalchemy import create_engine

4. Import Data provided in Homework02 Data folder in D2L

- Create a connection to the database, execute a SELECT statement to import the data. Make sure to include the column names. (Similar to Scenario 2, Code Block 2 through 9.)
- Import the data from the Calls table, similar to the following screenshot. Name it **df_calls**.

```
[3]: from sqlalchemy import create_engine
     from sqlalchemy import text

[4]: engine = create_engine('sqlite:///data/CantonPoliceDept.db')
     con = engine.connect()
     print('connection is ok')

     connection is ok

[5]: #print(engine.table_names())

     from sqlalchemy import inspect

     insp = inspect(engine)
     print(insp.get_table_names())

     ['Calls', 'Disposition']

[6]: rs = con.execute(text("SELECT * FROM Calls"))
     df_calls = pd.DataFrame(rs.fetchall()) ##fetches all data from the Calls table
     df_calls.columns = rs.keys()
     display(df_calls.head(2))
     df_calls.info()
```

Figure 4: Screenshot of importing the Calls table from CantonPoliceDept.db

- Import the Disposition table similar to how you imported the Calls table (only need Code block 6 from Figure 4). Name it **df_disp**.

- Import DispositionCategories and name it **df_disp_cat**.
- Import CensusData and name it **df_census**.
- Import Subzone_Distances and name it **df_spatial**.

  Note: The data is provided in the Homework02 Data folder in the D2L.

---

5. Create and format df_calls and df_disp

OBJECTIVE: Before you can group the data into subzone-year-week, you will need to get the calls and dispositions data ready.

For **df_calls**, we will:

- Create new columns based on date, then join columns together to create columns such as SUB_YEAR_WEEK.
- We will create dummy variables for weekday names, months, and all call complaints.

For **df_disp**, we will:

- We will merge **df_disp_cat** so that we have consistent names for all dispositions.
- We will create dummy variables for all dispositions categories that have a frequency above 100 in the dataset.

5.1 Formatting df_calls

- Create the following columns based on Date_Received (see Figure 5 for an example and for more help, watch [C1.S2.Py10-Converting Objects to Dates](), which is from Scenario02_p2.ipynb in Class 1):
  - Day_Name - this is `day_name()`
  - WEEK- use isocalender
  - MONTH
  - YEAR
- Create YEAR_WEEK and SUB_YEAR_WEEK, similar to Figure 5. Here is a sample code.

```python
# Create YEAR_WEEK variable
def create_year_week(row):
    if row['MONTH'] == 12 and row['WEEK'] == 1:
        return f"{row['YEAR'] + 1}_1"
    elif row['MONTH'] == 1 and row['WEEK'] == 53:
        return f"{row['YEAR'] - 1}_53"
    else:
        return f"{row['YEAR']}_{row['WEEK']}"

# Apply the function to create the 'YEAR_WEEK' column
df_calls['YEAR_WEEK'] = df_calls.apply(create_year_week, axis=1)
```

	Subzone	Call_Number	Complaint	Date_Received	Day_Name	WEEK	MONTH	YEAR	YEAR_WEEK	SUB_YEAR_WEEK
1	ZONE1D	08C-000010	Burglary	2008-09-18 10:36:00	Thursday	38	9	2008	2008_38	2008_38_ZONE1D
2	ZONE2B	08C-000011	Burglary	2008-09-18 10:44:00	Thursday	38	9	2008	2008_38	2008_38_ZONE2B
3	ZONE1C	08C-000020	Assault	2008-09-18 11:52:00	Thursday	38	9	2008	2008_38	2008_38_ZONE1C
5	ZONE3D	08C-000029	Burglar alarm	2008-09-18 12:23:00	Thursday	38	9	2008	2008_38	2008_38_ZONE3D
6	ZONE4B	08C-000030	Welfare check	2008-09-18 12:26:00	Thursday	38	9	2008	2008_38	2008_38_ZONE4B

Figure 5: Screenshot of df_calls

- Create Dummy Variables from the Day_Name column and concatenate it back to df_calls (similar to Figure 6).
    - Do not drop any columns and do not include a prefix.
- Create Dummy Variables from the MONTH column and concatenate it back to df_calls (similar to Figure 6).
    - Do not drop any columns and add the prefix 'month'.

SUB_YEAR_WEEK	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday	month_1	month_2	month_3	month_4
2008_38_ZONE1D	0	0	0	0	1	0	0	0	0	0	0
2008_38_ZONE2B	0	0	0	0	1	0	0	0	0	0	0
2008_38_ZONE1C	0	0	0	0	1	0	0	0	0	0	0

Figure 6: Screenshot of df_calls with dummy variables for Day_Name and MONTH

- Create Dummy Variables from the Complaint column and concatenate it back to df_calls.
    - Do not drop any columns and add the prefix 'call'.

Note: In Section 5.1 on formatting df_calls, please check everything is correct so far!

5.2 Create df_calls_subweek

OBJECTIVE: This DataFrame will be our basis for combining aggregating the calls and dispositions together based on each subzone-year-week.

- Create **df_calls_subweek** by creating a copy of the following columns from df_calls
    - 'SUB_YEAR_WEEK','Subzone' ,'WEEK', 'MONTH', 'YEAR', 'YEAR_WEEK'
    - Note: at this point we have 264881 records in df_calls_subweek. But most of these re duplicates since there were many calls per SUB_YEAR_WEEK. Therefore, we need to drop duplicates.

- Drop the duplicates based on SUB_YEAR_WEEK.
    - `df_calls_subweek = df_calls_subweek.drop_duplicates('SUB_YEAR_WEEK')`
    - You should have 12187 records after removing duplicates.
- Create the call_ALL column
    - This column is a sum of all calls to the police department for a SUB_YEAR_WEEK. Therefore, we will need to group **df_calls** based on SUB_YEAR_WEEK and merge it with **df_calls_subweek (See Figure 7)**.

```
df_calls_subweek_gr = df_calls.groupby('SUB_YEAR_WEEK')['WEEK'].count().reset_index()
df_calls_subweek_gr = df_calls_subweek_gr.rename(columns={'WEEK':'call_ALL'})
df_calls_subweek_gr
```

	SUB_YEAR_WEEK	call_ALL
0	2008_38_ZONE1A	16
1	2008_38_ZONE1B	16
2	2008_38_ZONE1C	19
3	2008_38_ZONE1D	17
4	2008_38_ZONE2A	13

Figure 7: Screenshot of df_calls_subweek_gr which create a sum of all calls per SUB_YEAR_WEEK

- Merge df_calls_subweek_gr with df_calls_subweek (Figure 8)
    - Should be an 'inner' and on 'SUB_YEAR_WEEK'
    - Use the name **df_calls_subweek**

```
df_calls_subweek.head()
```

	SUB_YEAR_WEEK	Subzone	WEEK	MONTH	YEAR	YEAR_WEEK	call_ALL
0	2008_38_ZONE1D	ZONE1D	38	9	2008	2008_38	17
1	2008_38_ZONE2B	ZONE2B	38	9	2008	2008_38	1
2	2008_38_ZONE1C	ZONE1C	38	9	2008	2008_38	19
3	2008_38_ZONE3D	ZONE3D	38	9	2008	2008_38	11
4	2008_38_ZONE4B	ZONE4B	38	9	2008	2008_38	8

Figure 8: Screenshot of df_calls_subweek

5.3 Formatting df_disp

- Merge df_disp and df_disp_cat so that df_disp will have three columns.
    - Should be an 'inner' and on 'DISP'
    - Use the name **df_disp**

- Drop 'DISP'
- Create Dummy Variables from the DISPOSITION column and concatenate it back to df_disp. (See Figure 9)
  - Do not drop any columns and add the prefix 'disp'.

	Call_Number	DISPOSITION	disp_15-ARREST	disp_66-CANCEL	disp_9-VERBAL	disp_ADV-ADVICE GIVEN	disp_AST-ASSIST	disp_CI-CITY CITATION	disp_CLR-CLEARED BY DISPATCH	disp_CM-CIVIL MATTER	disp
281571	16C-007256	RF-REPORT FILED	0	0	0	0	0	0	0	0	
858660	1115172	RES-RESOLVED	0	0	0	0	0	0	0	0	
14616	11C-047199	FA-FALSE ALARM	0	0	0	0	0	0	0	0	
595032	16C-099896	CLR-CLEARED BY DISPATCH	0	0	0	0	0	0	1	0	
531242	16C-102957	NR-NO REPORT	0	0	0	0	0	0	0	0	

Figure 9: Screenshot of df_disp

5.4 Creating df_calls_disp

- Merge **df_calls** and **df_disp**.
  - Should be an 'inner' and on 'Call_Number'.
  - Should have a total of 267,266 records and 85 columns.
  - Use the name **df_calls_disp**

5.5 Summary for Section 5

- In markdown explain how **df_calls_disp** was created and how it can be useful.

---

6. Census Data

OBJECTIVE: Format the Census Data so that it is ready to be merge with the call and disposition data.

6.1 Formatting df_census

- Remove any column that has less than 20 non-null values.
- For HouseholdIncome_Median_25to44, fill in any null values with the mean.

6.2 Convert Columns from totals to ratios

- For many of the columns in **df_census**, the totals are not helpful unless they are compared to the population or housing units. Therefore, we will convert the following columns from a total to a ratio. (See Table 1 for details)
  - Example: We will convert Population_Male from totals to a ratio with the following line of code:
    - ```
      df_census['Population_Male'] =
      df_census['Population_Male']/df_census['Population']
      ```
      ```

Table1: Columns and the Types of Conversion

Column	Conversion
Population_Male	Ratio based on 'Total Population'
Population_Female	Ratio based on 'Total Population'
Workers who travel to work	Ratio based on 'Worked'
Drove alone to Work	Ratio based on 'Worked'
Carpooled to Work	Ratio based on 'Worked'
Enrolled in school	Ratio based on 'Population_3andover'
Enrolled in nursery school, preschool	Ratio based on 'Population_3andover'
Enrolled in kindergarten	Ratio based on 'Population_3andover'
Enrolled in college, undergraduate years	Ratio based on 'Population_3andover'
Graduate or professional school	Ratio based on 'Population_3andover'
Not enrolled in school	Ratio based on 'Population_3andover'
Households_wageorsalaryincome	Ratio based on 'Households_earnings
Households_selfemploymentincome	Ratio based on 'Households_earnings
Households_interest_dividends	Ratio based on 'Households_earnings
Households_SSI	Ratio based on 'Households_earnings
Households_publicassistanceincome	Ratio based on 'Households_earnings

6.3 Summary for Section 6

- In markdown explain how you would use **df_census**. Do you think it is useful to use census data with **df_calls_disp** for a predictive model?

7. Create the combine dataset based on SUB_YEAR_WEEK

OBJECTIVE: It is now time to combine df_calls_subweek, df_calls_disp, and df_census based on SUB_YEAR_WEEK. This is the DataFrame that will be used to for future modeling.

7.1 Create df_calls_disp_week

- Create **df_calls_disp_week** by creating a copy of **df_calls_disp**.
- Drop the following columns from **df_calls_disp_week:**
    - 'Subzone','Call_Number', 'Complaint', 'Date_Received', 'Day_Name', 'WEEK', 'MONTH', 'YEAR', 'YEAR_WEEK', 'DISPOSITION'
- Group df_calls_week by SUB_YEAR_WEEK by summing up all of the columns as integers.
    - ```
      df_calls_disp_week =
      df_calls_disp_week.groupby('SUB_YEAR_WEEK').sum().astype(int).reset_index()
      ```
- **df_calls_disp_week** should include 12,169 records and 75 columns.
      ```

- Merge **df_calls_subweek** and **df_calls_disp_week**
    - Should be an 'inner' based on 'SUB_YEAR_WEEK'
    - Use the name **df_calls_disp_week**
- Merge **df_calls_disp_week and df_census**
    - Should be an 'inner' based on 'Subzone'
    - Use the name **df_calls_disp_week**
    - Should have 111 columns and 12,169 records.

	SUB_YEAR_WEEK	Subzone	WEEK	MONTH	YEAR	YEAR_WEEK	call_ALL	Friday	Monday	Saturday	Sunday	Thursday
0	2008_38_ZONE1D	ZONE1D	38	9	2008	2008_38	17	4	0	8	2	3
1	2008_39_ZONE1D	ZONE1D	39	9	2008	2008_39	47	8	6	3	6	6
2	2008_40_ZONE1D	ZONE1D	40	9	2008	2008_40	35	8	5	1	6	5
3	2008_41_ZONE1D	ZONE1D	41	10	2008	2008_41	34	6	4	0	3	6
4	2008_42_ZONE1D	ZONE1D	42	10	2008	2008_42	36	4	4	9	5	4

Figure 10: Screenshot of df_calls_disp_week

7.2 Summary for Section 7

- In markdown explain how you can use **df_calls_disp_week** for a predictive model.

8. Create Spatial Lag

OBJECTIVE: One of the keys attributes of predicting future crimes or activities is to see what is happening in the surrounding subzones. For example, if there are a lot of burglaries in one neighborhood (Subzone ZONE2A) there is an assumption by the Canton Police that the Subzones next to ZONE2A will have burglaries next week. Therefore, we want to create columns that show activity around each Subzone. This is called spatial lag.

We will create spatial lag features for call_ALL (all calls) and call_burglaries (calls received for burglaries). ***NOTE: this is tricky, please follow the steps very carefully.***

8.1 Create df_spatial

- We have already imported Subzones_distances.csv and named it **df_spatial** from Section 4.
    - This Dataframe is a crosstab that shows the distance from each Subzone. For example, index 2 is ZONE1D. If you go over to the 5th column (ZONE5B), it has an amount of 11908.006760. That is the distance in feet from each other.
    - If an amount is less than 12,000 then we will say it is considered a complementary Subzone (or a Subzone that is close).

- This section will identify these relationships and then create a way to aggregate call_ALL and call_Burglary.

	ZONE	ZONE5D	ZONE3X	ZONE1D	ZONE5B	ZONE6A	ZONE2C	ZONE3C	ZONE2B
0	ZONE5D	0.000000	61752.50885	27952.862110	19900.400120	53707.055550	35654.001300	40667.285820	46895.96176
1	ZONE3X	61752.508850	0.00000	38991.875170	41993.263080	13123.373080	45672.221490	21197.338720	51900.60212
2	ZONE1D	27952.862110	38991.87517	0.000000	11908.006760	28114.045530	13406.932120	19171.327440	25792.10749
3	ZONE5B	19900.400120	41993.26308	11908.006760	0.000000	33912.976880	24570.068460	20825.143480	37160.63945
4	ZONE6A	53707.055550	13123.37308	28114.045530	33912.976880	0.000000	32971.356490	14521.043490	38778.09899
5	ZONE2C	35654.001300	45672.22149	13406.932120	24570.068460	32971.356490	0.000000	29324.061470	12593.02489
6	ZONE3C	40667.285820	21197.33872	19171.327440	20825.143480	14521.043490	29324.061470	0.000000	39161.84481
7	ZONE2B	46895.961760	51900.60212	25792.107490	37160.639450	38778.098990	12593.024890	39161.844810	0.00000
8	ZONE7B	22525.868880	53691.31900	14787.102120	19157.881510	42309.235270	14362.005430	33867.268280	24467.10028
9	ZONE7D	7746.724534	57821.14229	21532.948730	16151.513120	48648.659030	28036.341130	36637.234190	39151.68497
10	ZONE4D	39803.548350	32676.64218	11871.281350	22763.474600	20025.292230	12997.801970	17965.751450	21227.73101
11	ZONE1A	32080.033990	32737.53815	6326.566288	13320.022370	22306.576810	17422.446330	12933.237220	28818.35915

Figure 11: Screenshot of df_spatial

- Create a list of the column names from **df_spatial** and name it **df_spatial_col**.
  - `df_spatial_col = df_spatial.columns`
  - Drop 'ZONE' from **df_spatial_col**.
- Create a melt of df_spatial (This converts the crosstab into three total columns).
  - Data source is df_spatial
  - id_vars is 'ZONE'
  - value_vars is df_spatial_col (This is a list of all of the subzones. It is easier to create list and then reference it here, which is why we created **df_spatial_col**).
- Rename the columns in **df_spatial_melt**
  - 'ZONE' to 'Subzone'
  - 'value' to 'Subzone_Dist',
  - 'variable' to 'Subzone_Comp'

	Subzone	Subzone_Comp	Subzone_Dist
0	ZONE5D	ZONE5D	0.00000
1	ZONE3X	ZONE5D	61752.50885
2	ZONE1D	ZONE5D	27952.86211
3	ZONE5B	ZONE5D	19900.40012
4	ZONE6A	ZONE5D	53707.05555

Figure 12: Screenshot of df_spatial_melt

- Drop the rows where Subzone_dist = 0 in **df_spatial_melt**
  - Since all Subzones that are compared to itself have a Subzone_dist = 0, they can be dropped.

  - The easiest way to do this is to instead of dropping anything =0, you can keep anything that is greater than 0.
    - `df_spatial_melt = df_spatial_melt[df_spatial_melt['Subzone_Dist'] > 0]`
- Drop all rows where Subzone_dist >=12,000.
  - A complement to a Subzone is any Subzone that is within less than 12,000 feet.
  - Therefore we want to drop any Subzone_dist that is greater than or equal to 12,000 or keep any Subzone_dist that is less than 12,000.
  - You should have 122 records remaining.

8.2 Create df_burglary_week

OBJECTIVE: We will now want to sum up all calls and all burglaries for the surrounding areas around a given Subzone.

- Create **df_burglary_week** by creating a copy of the following columns from **df_calls_disp_week**.
  - 'SUB_YEAR_WEEK', 'Subzone','call_ALL','call_Burglary'
- Merge **df_spatial_melt** and **df_burglary_week** together to create **df_spatial_burg**.
  - df_spatial_melt, df_burglary_week (in this order)
  - left_on=['Subzone_Comp']
  - right_on=['Subzone']
  - how='inner'
- Drop 'Subzone_y' and 'Subzone_dist'
- Rename 'Subzone_x' to 'Subzone' (See Figure 13)

	Subzone	Subzone_Comp	SUB_YEAR_WEEK	call_ALL	call_Burglary
0	ZONE7D	ZONE5D	2011_33_ZONE5D	1	0
1	ZONE7D	ZONE5D	2012_44_ZONE5D	1	0
2	ZONE7D	ZONE5D	2013_17_ZONE5D	2	0
3	ZONE7D	ZONE5D	2013_19_ZONE5D	1	0
4	ZONE7D	ZONE5D	2013_21_ZONE5D	1	0

Figure 13: Screenshot of df_spatial_burg

- Create a dictionary that contains all subzones as a keys and the list of their neighbors as the values using the following code

  dict = {}

```
for i in range(len(df_spatial_melt)):
   zone = df_spatial_melt.iloc[i,0]
   if zone not in dict:
      dict[zone] = []
      dict[zone].append(df_spatial_melt.iloc[i,1])
   else:
      dict[zone].append(df_spatial_melt.iloc[i,1])
```

- Create a dictionary where you compute and save the sum of call_ALL of each neighboring subzones. You may use the following sample code.

```
#Create a dictionary where you save the sum of call_ALL of each neighboring subzone

dict2 = {}
for i in range(len(df_spatial_burg)):
    key = df_spatial_burg.iloc[i,2]#keys are the values in column SUB_YEAR_WEEK
    zone = key[-6:] #gives the output ZONE7D from the SUB_YEAR_WEEK combination 2018_9_ZONE7D
    #print(zone)
    year_week = key[:-6] #gives the output 2018_9_ from the SUB_YEAR_WEEK combination 2018_9_ZONE7D
    neighbours = []
    if zone in dict:
        neighbours = dict[zone]
    #print("neighbours=",neighbours)
    accumulate = 0
    if len(neighbours)>0:
        if key not in dict2:
            dict2[key] = 0
            for neigh in neighbours:
                value = year_week+neigh
                #print(value)
                call_all = df_spatial_burg[df_spatial_burg.SUB_YEAR_WEEK==value]['call_ALL']
                #print(call_all)
                if(call_all.empty==False):
                    call_all = call_all.values[0]
                    dict2[key]+=int(call_all)
```

- The dict2 contains the SUB_YEAR_WEEK and the call_ALL from the complementary subzones. Convert the dictionary to the data frame and rename the columns using the following code.

results1 = pd.DataFrame.from_dict(dict2, orient ='index').reset_index()
results1 = results.rename(columns = {'index':'SUB_YEAR_WEEK', 0:'call_ALL_comp'})

- Create a dictionary dict3 that where you compute and save the sum of call_Burglary of each neighboring subzones modifying the code given above. Then the dict3 contains the SUB_YEAR_WEEK and the call_Burglary from the complementary subzones. Convert the dictionary to the data frame, name it results2, and rename the columns 'index' to 'SUB_YEAR_WEEK', and 0 to 'call_ Burglary _comp' as before.

- Merge the data frames results1 and results2 in order, on 'Sub_Year_Week' and name it **df_spatial_burg_comp.** The **df_spatial_burg_comp should have 11,626 rows and 3 columns.**

- Merge **df_calls_disp_week** and **df_spatial_burg_comp**
  - df_calls_disp_week, df_spatial_burg_comp (in that order)
  - on = ['SUB_YEAR_WEEK']
  - how='left'
  - Name it **df_calls_disp_week**
- Since there are some call_ALL_comp and call_Burglary_comp values that are null because of no calls or burglaries from complementary Subzones, we will fill in those values with a 0.
  - Fill in null values for call_ALL_comp and call_Burglary_comp with 0.

df_calls_disp_week will have 12,169 rows and 113 columns.

8.3 Summary for Section 8

- How can spatial lag be helpful in a predictive model using **df_calls_disp_week**.

9. Create Time Lag

OBJECTIVE: You now have a dataset named **df_calls_disp_week** that includes:

- All calls aggregated based on SUB_YEAR_WEEK
- All dispositions aggregated based on SUB_YEAR_WEEK
- All census data based on Subzone
- All complementary Subzones (within 12,000 feet) with ALL calls and burglary calls. (spatial lag)

Now comes the challenging part. When you look to make a prediction, you are looking to use data that is available to you to make an estimate (prediction) that will happen. For this project, the objective is to use data that is available to us (previous calls, dispositions, and census data) to predict a future amount or if a crime will occur in the future.
Therefore, we will use current data to predict next week's target variable.

Example – Predicting Burglaries:
Let's say that we are looking to predict the number of burglaries in a subzone so that the Canton Police department can allocate officers and other resources appropriately. Therefore, we want a good estimate for burglaries in a specific Subzone for next week so we can plan this week.

To keep it simple – If you are trying to predict Burglaries for week 39, you will need Features from week 38. Therefore, use all columns from df_calls_disp_week data week 38 to predict next week's burglaries. Then compare to the target variable from week 39, call_Burglary.

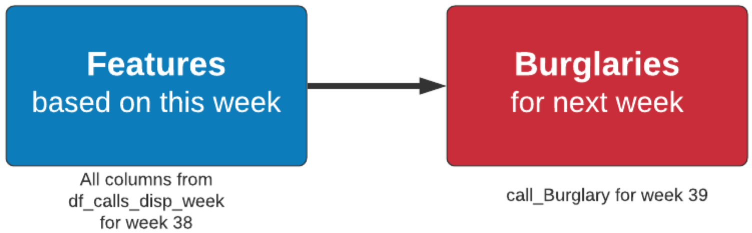

Figure 14: Example showing Time Lag between week 38 (current week) and week 39 (prediction week)

9.1 Create df_calls_subweek_target

- Create a new DataFrame named **df_calls_subweek_target** which is a copy of **df_calls_subweek**.
- Merge **df_calls_subweek_target** with **df_calls_disp_week**[['SUB_YEAR_WEEK', 'call_Burglary']].
    - o Merge it on 'SUB_YEAR_WEEK' by using 'inner'
    - o Name it **df_calls_subweek_target**
- Format, add, and drop **df_calls_subweek_target** to look similar to Figures 15 and 16.

	SUB_YEAR_WEEK	Subzone	WEEK	MONTH	YEAR	YEAR_WEEK	call_ALL	call_Burglary	SUB_YEAR_WEEK_target	call_ALL_target	call_Burglary_target
0	2008_38_ZONE1D	ZONE1D	38	9	2008	2008_38	17	1	2008_39_ZONE1D	47	1
1	2008_38_ZONE2B	ZONE2B	38	9	2008	2008_38	1	1	2008_39_ZONE2B	2	0
2	2008_38_ZONE1C	ZONE1C	38	9	2008	2008_38	19	0	2008_39_ZONE1C	29	1
3	2008_38_ZONE3D	ZONE3D	38	9	2008	2008_38	11	0	2008_39_ZONE3D	20	2
4	2008_38_ZONE4B	ZONE4B	38	9	2008	2008_38	8	0	2008_39_ZONE4B	10	0

Figure 15: Screenshot of df_calls_subweek_target

	SUB_YEAR_WEEK	Subzone	WEEK	MONTH	YEAR	YEAR_WEEK	call_ALL	call_Burglary	SUB_YEAR_WEEK_target	call_ALL_target	call_Burglary_target
0	2008_38_ZONE1D	ZONE1D	38	9	2008	2008_38	17	1	2008_39_ZONE1D	47	1
22	2008_39_ZONE1D	ZONE1D	39	9	2008	2008_39	47	1	2008_40_ZONE1D	35	5
47	2008_40_ZONE1D	ZONE1D	40	9	2008	2008_40	35	5	2008_41_ZONE1D	34	3
64	2008_41_ZONE1D	ZONE1D	41	10	2008	2008_41	34	3	2008_42_ZONE1D	36	4
84	2008_42_ZONE1D	ZONE1D	42	10	2008	2008_42	36	4	2008_43_ZONE1D	38	2

Figure 16: Screenshot of df_calls_subweek_target filtered by Subzone = ZONE1D

Things to consider when creating **df_calls_subweek_target:**

- call_All_target and call_Burglary_target are the total calls and burglary calls for the next week.
    - For example, in Figure 16, the first row, which is SUB_YEAR_WEEK = 2008_38_ZONE1D has a SUB_YEAR_WEEK_target = 2008_39_ZONE1D.
    - That means for in Week 38, you are trying to predict the target calls for Week 39.
    - Notice that the call_ALL_target in row 1 = 47, which is the target for Week 39. This was derived from the call_ALL from the second row, where call_ALL = 47 for the SUB_YEAR_WEEK = 2008_39_ZONE1D.
- Create a YEAR_WEEK_target first before creating SUB_YEAR_WEEK_target. Therefore, for a YEAR_WEEK of 38, the YEAR_WEEK_target would be 39.
    - `df_calls_subweek_target['YEAR'].astype(str) + "_" + (df_calls_subweek_target['WEEK']+1).astype(str)`
    - Watch out for weeks 52 and 53, the last week of the year, you may need to increase the YEAR by +1, and the WEEK will need to be 1.

9.2 Create df_pred_calls

- Create **df_pred_calls** by copying the following columns from **df_calls_subweek_target**.
    - 'SUB_YEAR_WEEK','SUB_YEAR_WEEK_target', 'call_ALL_target','call_Burglary_target'
- Merge **df_pred_calls** with **df_calls_disp_week**
    - Mege on 'SUB_YEAR_WEEK' by using an 'inner' join.
    - Name it **df_pred_calls**.

Note: You can create a 'WEEK_target' and then make a 'SUB_WEEK_target' as well. Please verify if this function correctly serves the purpose of creating the 'WEEK_target'. If so, then use it; else correct it and use it!

```
def determine_week_target(row):
    last_week_of_year = df_calls_subweek_target[df_calls_subweek_target['YEAR'] == row['YEAR']]['WEEK'].max()
    if last_week_of_year in [52, 53]:
        if row['WEEK'] == last_week_of_year and row['MONTH'] == 12:
            return str(row['YEAR'] + 1) + "_1"
        elif row['WEEK'] == last_week_of_year:
            return str(row['YEAR'] + 1) + "_1"
    elif row['WEEK'] in [52, 53] and row['MONTH'] == 1:
        return str(row['YEAR']) + "_1"

    return str(row['YEAR']) + "_" + str(row['WEEK'] + 1)

# Apply the function to each row to determine WEEK_target
df_calls_subweek_target['WEEK_target'] = df_calls_subweek_target.apply(determine_week_target, axis=1)
```

Finally you can create a 'SUB_WEEK_target' as follows:
df_calls_subweek_target['SUB_WEEK_target'] = df_calls_subweek_target['WEEK_target'] + "_" + df_calls_subweek_target['Subzone']

```
df_pred_calls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11887 entries, 0 to 11886
Columns: 116 entries, SUB_YEAR_WEEK to call_Burglary_comp
dtypes: Int64(2), UInt32(1), float64(20), int32(2), int64(87), object(4)
memory usage: 10.4+ MB
```

```
df_pred_calls.head()
```

	SUB_YEAR_WEEK	SUB_YEAR_WEEK_target	call_ALL_target	call_Burglary_target	Subzone	WEEK	MONTH	YEAR	YEAR_WEEK	call_ALL	Friday
0	2008_38_ZONE1D	2008_39_ZONE1D	47	1	ZONE1D	38	9	2008	2008_38	17	4
1	2008_38_ZONE2B	2008_39_ZONE2B	2	0	ZONE2B	38	9	2008	2008_38	1	0
2	2008_38_ZONE1C	2008_39_ZONE1C	29	1	ZONE1C	38	9	2008	2008_38	19	4
3	2008_38_ZONE3D	2008_39_ZONE3D	20	2	ZONE3D	38	9	2008	2008_38	11	3
4	2008_38_ZONE4B	2008_39_ZONE4B	10	0	ZONE4B	38	9	2008	2008_38	8	4

Figure 17: Screenshot of df_pred_calls

9.3 Create Burg_status and ALL_status

- Our final step is to create two columns for df_pred_calls that indicate if at least one call or one burglary happened with a given SUB_YEAR_WEEK. This will be used for Classification Modeling.
- For **df_pred_calls** create a column named ALL_status.
  - If a SUB_YEAR_WEEK has at least one call_ALL_target, then ALL_status should equal 1, otherwise 0.
  - This is easiest to create with a defined function (See Scenario 3 presentation for a Defined Function).
- For **df_pred_calls** create a column named Burg_status.
  - If a SUB_YEAR_WEEK has at least one call_Burglar_taget, then Burg_status should equal 1, otherwise 0

```
df_pred_calls[['SUB_YEAR_WEEK', 'SUB_YEAR_WEEK_target' ,'call_ALL_target',
               'call_Burglary_target', 'call_ALL_comp', 'call_Burglary_comp',
               'Burg_Status' ,'ALL_Status']].head()
```

	SUB_YEAR_WEEK	SUB_YEAR_WEEK_target	call_ALL_target	call_Burglary_target	call_ALL_comp	call_Burglary_comp	Burg_Status	ALL_Status
0	2008_38_ZONE1D	2008_39_ZONE1D	47	1	119.0	9.0	1	1
1	2008_38_ZONE2B	2008_39_ZONE2B	2	0	0.0	0.0	0	1
2	2008_38_ZONE1C	2008_39_ZONE1C	29	1	156.0	15.0	1	1
3	2008_38_ZONE3D	2008_39_ZONE3D	20	2	128.0	10.0	1	1
4	2008_38_ZONE4B	2008_39_ZONE4B	10	0	9.0	1.0	0	1

Figure 18: Screenshot of a select number of columns for the final df_pred_calls

9.4 Summary for Section 9

- Congratulations, you made it! How can this final dataset **df_pred_calls** be used to make predictions?

- Explain in a few paragraphs (with bullets/dashes) what you think the final df_pred_calls can be used for in helping the police chief make better decisions.

10. Save Jupyter Notebook

- Click the Save button
- Click File – Close and Halt

11. Submit notebook through Dropbox on D2L.

- Go back into D2L and click on Dropbox in the menu.
- Click Homework02.
- Submit your .ipynb file only. For example, if your name is Jane Smith, you should only submit HW02_SmithJ.ipynb. Do not submit the data or any folders, just the Jupyter notebook.
- **It is extremely important that you setup the data (URLs) correctly.**