Visualizing Houston violent crime trends via data cleaning

Mason Biamonte
September 7, 2020

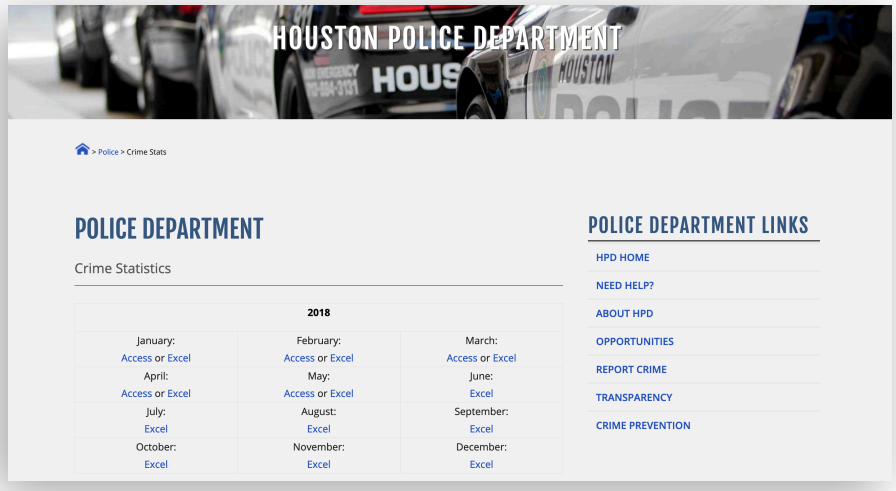
Violent crime is one of the symptoms indicating societal illness. It occurs in large part due to the failure of a society and its government to provide those components essential to human flourishing: food, shelter, education, job training and access to physical and mental healthcare. Heinous and petty crimes have also been and will continue to be committed by individuals well-endowed with these essentials. In this article I will illustrate fundamental data science methods using the example of violent crime statistics. The methods will be presented while providing insight into one of the major symptoms of societal illness. I will be looking at violent crime data in my hometown of Houston, Texas.

The goal of this article is to present

1. **data collection**: web scraping to download data files into a local directory
2. **data cleaning**: extract and store only the desired values
3. **data visualization**: plotting data so that the information is visually manifest

I. Data collection via web scraping and browser automation

To start, we go to the Houston Police Department crime statistics web page.



Monthly crime statistics are available in both Access and Excel formats. I will be working with Excel files. When you click on an Excel link, an Excel file containing the statistics for that month is downloaded to your local machine. Below is an example of the first few columns of the June 2009 data.

Date	Hour	Offense Type	Beat	Premise	Block Range	Street Name	Type	Suffix	# Of Offenses
6/24/2009	14	Murder	18F40	18A	2900-2999	CROSSVIEW	DR	-	1
6/24/2009	10	Murder	16E20	20R	4800-4899	MACRIDGE	BLVD	-	1
6/23/2009	21	Murder	17E10	18T	6700-6799	HILLCROFT	-	-	1
6/8/2009	00	Murder	3B40	11R	900-999	40TH 1/2	ST	E	1
6/3/2009	23	Murder	9C30	20R	3900-3999	MANITOU	DR	-	1
3/29/2009	02	Murder	18F50	18N	7600-7699	DASHWOOD	DR	-	1
6/28/2009	05	Murder	18F50	20A	5800-5899	FONDREN	RD	-	1
6/14/2009	18	Murder	6B10	13R	11000-11099	HARDY	RD	W	1
5/31/2009	10	Murder	19G50	19V	10500-10599	ROCKLEY	RD	-	1
6/20/2009	02	Murder	17E30	20A	8800-8899	BRAESWOOD	BLVD	S	1
6/11/2009	20	Murder	20G60	20A	600-699	NOTTINGHAM OAKS	TRL	-	1
6/3/2009	08	Murder	10H10	070	3700-3799	HARRISBURG	BLVD	-	2
6/6/2009	06	Murder	6B60	18A	1100-1199	LANGWICK	DR	-	1
6/6/2009	04	Murder	17E10	18N	6000-6099	HILLCROFT	-	-	1
6/30/2009	01	Murder	20G10	20A	9300-9399	TOWN PARK	DR	-	1
6/18/2009	19	Murder	8C60	100	5500-5599	VAN ZANDT	-	-	1
6/18/2009	20	Murder	9C20	20D	600-699	WOOLWORTH	-	-	1
6/22/2009	06	Murder	7C20	20R	5100-5199	LELIA	-	-	1
6/2/2009	22	Murder	19G50	20A	10600-10699	WILCREST	DR	-	1
6/13/2009	03	Murder	7C20	20D	3800-3899	WYLIE	-	-	1
9/8/2008	02	Murder	14D20	100	8400-8499	SCOTT	-	-	1
6/5/2009	02	Murder	2A30	18N	2700-2799	SHEPHERD	DR	N	1
6/29/2009	07	Murder	3B10	20R	4900-4999	HOOVER	-	-	1
4/7/2006	11	Murder	20G20	20A	1200-1299	WILCREST	DR	-	1
6/16/2009	17	Murder	18F30	18A	6100-6199	FAIRDALE	LN	-	2
6/6/2009	05	Rape	14D20	20R	9500-9599	CHESTERFIELD	DR	-	1
6/3/2009	22	Rape	20G30	20D	10800-10899	RICHMOND	AVE	-	1
6/26/2009	18	Rape	7C10	20A	5300-5399	COKE	ST	-	1
6/26/2009	21	Rape	17E30	20A	8700-8799	BRAESWOOD	BLVD	S	1
6/5/2009	17	Rape	6B20	140	7100-7199	SHEPHERD	DR	N	1
6/13/2009	00	Rape	6B10	20R	800-899	DUNKLEY	DR	-	1

Before moving on to the data science, let's take a moment to reflect on all the suffering caused by each one of the line items above and another moment to reflect on the countless unreported such incidents.

Note that some of the rows have dates lying outside of June 2009. This could be due to previous incidents getting reported at later dates. We will have to handle these cases in our data cleaning and processing step. To begin collecting data, we observe that all Excel links have URLs that when visited initiate a download of the associated file. So first, we need to collect all the Excel link URLs on the webpage above. We will do this using the `requests` and `bs4` (a.k.a. `BeautifulSoup`) libraries. The Excel links can be got using the following function.

```
def get_excel_links(h_url, b_url):

    res = requests.get(h_url)
    soup = bs4.BeautifulSoup(res.text, 'lxml')

    links = list()

    for link in soup.find_all('a', href=True):
        if link == '#':
            pass
        if str(link['href']).startswith('xls'):
            links.append(b_url + link['href'])

    return links
```

The function takes two URLs as input: a "home URL" and a "base URL". Since these two URLs are fixed they can be defined in a configuration file called `config.py`. Here are their definitions:

```
base_url = "https://www.houstontx.gov/police/cs/"
home_url =  base_url + "crime-stats-archives.htm"
```

When called with these two URLs, the array returned by `get_excel_links` should have 114 links. With the links in hand, we need to write browser automation code to visit and download them nicely into a directory of our choice. One obstacle with this particular data set, is that the file names and extensions are in two different formats. In addition, the files with the longer names including the string `NIBRS_Public_Data_Group_A&B` have a more complicated structure and result in gobbledygook upon importing with the Pandas library. Since these files only comprise the last 6 months out of the 114 months in the data set, we will ignore these files for now. To properly ignore them, we save them in a sub-directory called `messy`. Here is a function for downloading the files into local directories from the URL links

```
def download_excel_files(links, b_url, m_dict, data_dir):

    """
    takes array of links, a base URL, a dictionary of month names and a
    download directory path and downloads the files from those links

    two types of Excel files: parsable and messy; the two types need to be
    downloaded into separate directories
    """

    for link in links:

        r = requests.get(link, stream=True)

        # extract the filename from the link
        file_name = link.replace(b_url + 'xls/', '')

        print(f"Downloading {file_name}...")

        # standardize filenames into format mm-yyyy.xls
        if file_name.endswith('xlsx'):

            file_name = file_name.replace('.NIBRS_Public_Data_Group_A&B','')

            with open(data_dir + '/messy/' + file_name,'wb') as f:
                f.write(r.content)
        else:
            file_name = m_dict[file_name[0:3]]+ '-' + '20' + file_name[3:]
            file_name = file_name.replace('xls', 'xlsx')

            with open(data_dir + file_name,'wb') as f:
                f.write(r.content)
```

where `data_dir` is the path to the download directory and the dictionary `m_dict` is used to convert between 3-character alphabetic and 2-character numeric string representations of the months given by

```
month_dict = {'jan': '01', 'feb': '02', 'mar': '03', 'apr': '04',
              'may': '05', 'jun': '06', 'jul': '07', 'aug': '08',
              'sep': '09', 'oct': '10', 'nov': '11', 'dec': '12'}
```

Ignoring the `messy` files, all the others are in the format `mm-yyyy.xlsx`. I want to change these to the format `yyyy-mm.xlsx` so that the file system will list them in chronological order when sorting alphanumerically. The following function will do the trick

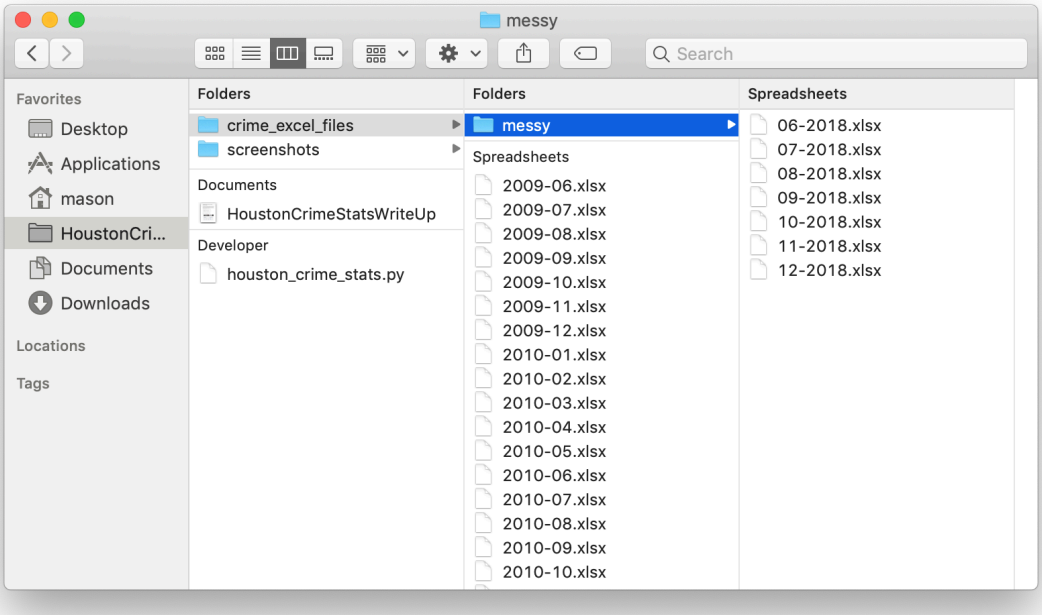
```
def rename_files(data_dir):

    files_in_data_dir = os.listdir(data_dir)
    excel_file_names = [file for file in files_in_data_dir if file.endswith(".xlsx")]

    for name in excel_file_names:

        new_name = name.split('-')
        new_name[1] = new_name[1].replace('.xlsx','')
        new_name[0], new_name[1] = new_name[1], new_name[0]
        new_name = "-".join(new_name) + ".xlsx"
        os.rename(os.path.join(data_dir, name),os.path.join(data_dir, new_name))
```

Now the non-messy files are listed chronologically and the messy files are stored in a sub-directory.



Our data collection step is now complete. To organize the code, I placed all the functions shown above into a file called `helper_funcs.py` and imported the `requests`, `os` and `bs4` libraries. Then, I created another file named `data_collection.py` to handle this data collection step. The entire contents of this file are provided in the screenshot below

```
import bs4
import os
import requests
from config import (base_url, home_url,
                    crime_data_dir, month_dict)
from helper_funcs import (get_excel_links,
                          download_excel_files,
                          rename_files)

####################
# DATA COLLECTION #
####################

excel_links = get_excel_links(home_url, base_url)
download_excel_files(excel_links, base_url,
                     month_dict, crime_data_dir)
rename_files(crime_data_dir)
```

where `crime_data_dir` is the path to the directory in which the Excel files were downloaded.

II. Data cleaning with Pandas

Now we're ready to clean the data and extract a subset of the information. We would like to import a single spreadsheet into a Pandas data frame using the `read_excel` function. When I tried this directly, I ran into the following error for a significant subset of files:

```
WARNING *** file size (3085901) not 512 + multiple of sector size (512)
WARNING *** OLE2 inconsistency: SSCS size is 0 but SSAT size is non-zero
```

To bypass these errors, I found the following hack here

```
wb = xlrd.open_workbook(filename, logfile=open(os.devnull, 'w'))
df = pd.read_excel(wb)
```

Once we have the spreadsheet converted into a Pandas dataframe, we need to drop all the columns of extraneous information. In this case, we are only interested in the date of occurence, the offense type and the number of offenses. We are also only going to keep track of violent offenses. Keeping track of only violent offenses will involve dropping those rows associated with non-violent incidents. The crimes considered violent and non-violent are listed below

```
nonviolent_offenses = ['Auto Theft', 'Theft', 'Burglary']
violent_offenses = ['Rape', 'Murder', 'Assault',  'Robbery']
```

where we will be later renaming `'Aggravated Assault'` to `'Assault'`. Here is the list of all column names common to many of the files at the beginning of the dataset (around 2009)

```
['Date', 'Unnamed: 1', 'Hour', 'Offense Type', 'Beat', 'Premise',
'Block Range', 'Street Name', 'Type', 'Suffix', '# Of Offenses']
```

The subset of columns we are interested in is `['Date', 'Offense Type', '# Of Offenses']`. For simplicity, we will be renaming the column `'Offense Type'` to `'Offense'` and `'# Of Offenses'` to `'#'`. However, in going through the data I noticed that not all last columns are labelled `'# Of Offenses'`. In fact, here is the exhaustive list of all alternative namings for this column:

```
['# Of Offenses', '# Of', '# Offenses', '# offenses', 'Offenses']
```

To handle these irregularities, we will look for any occurrence of the above items and rename them to `'#'`. This issue being handled, we can now drop extraneous columns. Taking the June 2009 data as an example, here is a print out of the Pandas dataframe with dropped columns

```
            Date Offense  #
0     2009-06-24  Murder  1
1     2009-06-24  Murder  1
2     2009-06-23  Murder  1
3     2009-06-08  Murder  1
4     2009-06-03  Murder  1
...          ...     ... ..
12564 2009-06-30   Theft  1
12565 2009-06-29   Theft  1
12566 2009-06-30   Theft  1
12567 2009-06-30   Theft  1
12568 2009-06-30   Theft  1

[12569 rows x 3 columns]
```

The following function will take as input the Pandas dataframe created from the June 2009 Excel file and produce the three-column dataframe above as output

```
def drop_extraneous_cols(df):

    """
    takes Pandas dataframe created from Excel file,
    drops non-violent offenses and location information
    and renames the aggravated assault incidents
    """

    cols = list(df.columns)

    col_names = ['# Of Offenses', '# Of', '# Offenses',
                 '# offenses', 'Offenses']

    col_name = [name for name in col_names if name in cols][0]

    cols_to_keep = ['Date', 'Offense Type', col_name]
    cols_to_drop = [col for col in cols if col not in cols_to_keep]

    df.drop(columns=cols_to_drop, inplace=True)
    df.rename(columns={"Offense Type": "Offense", col_name: "#"},
              inplace=True)

    df.loc[(df["Offense"] == 'Aggravated Assault'), "Offense"] = 'Assault'
```

Note the usage of the *`inplace`*`=True` argument. This is necessary when passing dataframes to functions because it tells Pandas to update the original dataframe instead of creating a copy and leaving the original unchanged. The last line is row entry renaming rather than column dropping.

Our two remaining steps in the data cleaning operation are: dropping extraneous rows and summing up all offenses occurring in a given month. For dropping extraneous rows, we use the following code to drop rows associated with non-violent offenses and rows with null entries

```
def drop_extraneous_rows(df):

    """
    function taking Pandas dataframe created from Excel file
    and dropping null/non-violent rows
    """

    nonviolent_offenses = ['Auto Theft', 'Theft', 'Burglary']

    for offense in nonviolent_offenses:

        index_names = df[df['Offense'] == offense].index
        df.drop(index_names, inplace=True)

    date_bools = pd.isna(df["Date"])
    df.drop(date_bools[date_bools==True].index, axis=0, inplace=True)

    offenses = set(df["Offense"])
    non_str_offenses = [offense for offense in offenses if type(offense) != str]

    for non_str_offense in non_str_offenses:

        index_names = df[df['Offense'] == non_str_offense].index
        df.drop(index_names, inplace=True)
```

Now we need to total up the violent offenses for each month. Before doing that, there are a few things that need to be done. For some of the data files, strings associated with the offense type contained extra whitespace. These whitespaces need to be cleaned up. The following code will do the trick.

```
def cleanup_whitespaces(df):

    offenses = set(df["Offense"])

    for offense in offenses:

        new_offense_name = offense.strip()
        df.loc[(df["Offense"] == offense), "Offense"] = new_offense_name

    df.loc[(df["Offense"] == 'Aggravated Assault'), "Offense"] = 'Assault'
```

Second, I want to reformat how the incident dates are stored because I do not want to work with Pandas' TimeStamp objects and while demonstrating how to split data into multiple columns. The code below has two functions: the first one converts the dates into numeric format while the second one splits this numeric information into Year and Month columns. The second function also drops rows that have missing Date values.

```
def assemble_numeric_dates(df):

    dates = list()

    for date in list(df["Date"]):

        temp = list()

        if str(date)[4] == '-':
            temp = [int(item) for item in str(date).split(' ')[0][0:7].split('-')]
            dates.append(temp)

        elif str(date)[2] == '/':
            temp = [int(str(date).split('/')[2]), int(str(date).split('/')[0])]
            dates.append(temp)

    return dates
```

```
def reformat_date_columns(dates, df):

    # drop rows with missing date values 'NaT'
    date_bools = pd.isna(df["Date"])
    df.drop(date_bools[date_bools==True].index, axis=0, inplace=True)

    # replace original date entries in df with numeric lists
    df["Year"] = [date[0] for date in dates]
    df["Month"] = [date[1] for date in dates]
    df.drop(columns=["Date"], inplace=True)

    # throw out entries with dates preceding June 2009
    drop_indices = df[(df["Year"] < 2009) |
                      ((df["Year"] == 2009) & (df["Month"] < 6))].index

    df.drop(drop_indices, axis=0, inplace=True)
```

The very last step of our data cleaning operation involves summing up all the offenses of a given type and storing them in an output dataframe. The function `append_monthly_sum` below achieves this:

```
def append_monthly_sum(year, month, df_in, df_out):

    # specify list of violent crime offenses
    violent_offenses = ['Rape', 'Murder', 'Assault',  'Robbery']

    for offense in violent_offenses:

        offense_counts = df_in[(df_in["Year"] == year) &
                               (df_in["Month"] == month) &
                               (df_in["Offense"] == offense)
                              ]["#"]

        df_out = df_out.append({"Year": year,
                                "Month": month,
                                "Offense": offense,
                                "#": int(sum(offense_counts))},
                                ignore_index=True
                                )

    return df_out
```

Putting it all together, we create a file called `data_collection.py` to carry out the above steps. First we import libraries and functions, then we create an empty Pandas dataframe and loop over all data files in the directory

```
import config
import glob
import os
import pandas as pd
import xlrd
from config import (base_url, home_url,
                    crime_data_dir, month_dict)
from helper_funcs import (drop_extraneous_cols, drop_extraneous_rows,
                          cleanup_whitespaces, assemble_numeric_dates,
                          reformat_date_columns, append_monthly_sum)

#################
# DATA CLEANING #
#################

# create empty dataframe storing cleaned data
col_names = ["Year", "Month", "Offense", "#"]
df_full = pd.DataFrame(columns=col_names)

# loop over all parsable Excel files
path = crime_data_dir + "*.xlsx"
for month_file in sorted(glob.glob(path)):
```

Inside the loop, for each file in the directory (corresponding to crime statistics occurring in that month), we call the functions defined above

```
    file_name = month_file.split("/")[-1].split(".")[0].split("-")

    year = int(file_name[0])
    month = int(file_name[1])

    print([month, year])

    # import spreadsheet as Pandas dataframe
    wb = xlrd.open_workbook(month_file, logfile=open(os.devnull, 'w'))
    df = pd.read_excel(wb)

    drop_extraneous_cols(df)

    drop_extraneous_rows(df)

    cleanup_whitespaces(df)

    # convert TimeStep dates to numeric lists [yyyy, mm]
    dates = assemble_numeric_dates(df)
    reformat_date_columns(dates, df)

    # sum all incidents of the same type for each month
    df_full = append_monthly_sum(year, month, df, df_full)
```

Once all the months have been appended to the output dataframe, we can export to a pickle file using the `to_pickle` command to save the dataframe. We have now successfully cleaned the data.

III. Data visualization with `matplotlib`

With the data now cleaned, let's plot the monthly reported violent crime incidents over the nine year period in question to visually capture the time series dynamics. We will be plotting the data using the `matplotlib` library.

Monthly Assault Cases (June 2009 — May 2018): The dominant trend visually manifest from the data is a decline in reported cases of assault for the first five years of the data followed by an increase back to initial levels.