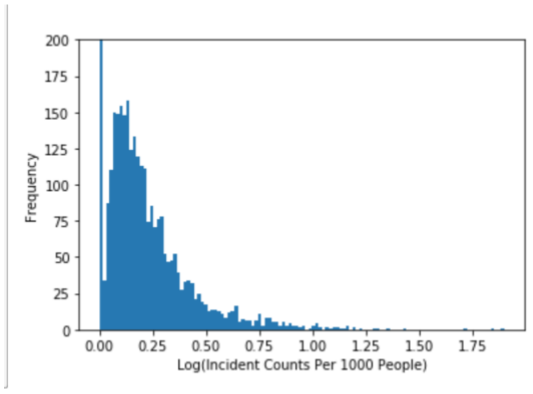
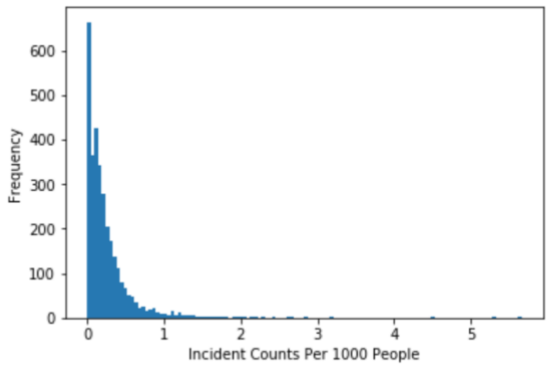
Intro:
Our project sought to build a model that was able to predict quantity of gun violence in a given area using a range of features both economic and demographic.

Initial Data Aggregation:
We started with a dataset of over 260k gun violence incidents in the United States from 2013-2017. Each incident came with information such as location, the number of people killed and injured, the age and gender of the shooter, as well as other additional information. We also had a list of census data from the American Community Survey 2015 5-year estimate, which contained a large number of demographic and economic features pertaining to a specific geographical region. We looked at the data broken down by county and census tract. A census tract is an area of about 4,000 inhabitants.
Next, we mapped each of these shootings to a county and census tract. To do so, we used the python library *censusgeocode*. This library provided a method that took in a pair of latitude, longitude coordinates and returned the corresponding census information, including the county and census tract id. We discarded all incidents that did not contain a latitude, longitude pair and mapped the remaining incidents to a county and tract. Once each incident was mapped, we went through each tract and county and counted the number of incidents occurring in each one of the regions. The code to perform all of this can be found at the end of the report.
After looking at these counts, we determined that we were only interested in shootings in which there was a fatality or injury. Some of the tracts with the highest number of incidents actually had very low injuries. We assumed this to be discrepancies in reporting and/or policing between regions, so we decided to eliminate incidents in which nobody was harmed to help reduce this potential source of noise.

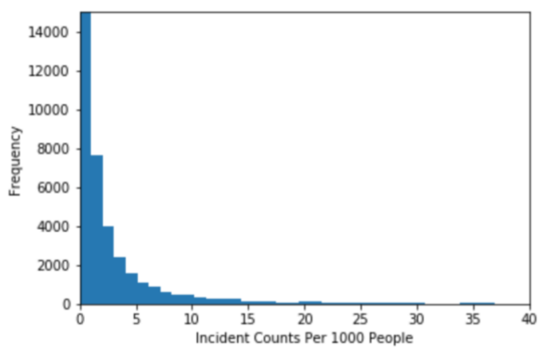
After doing this and removing census areas that were missing info, we obtained the following distributions.

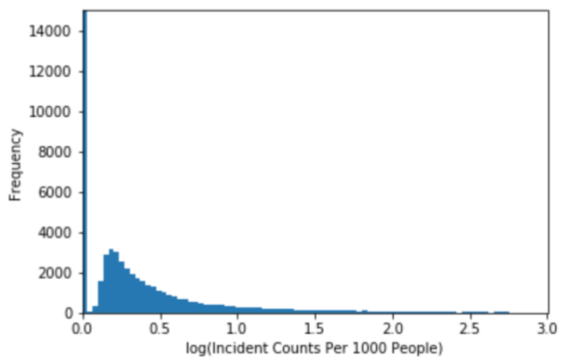
County-level:





Tract-level:





As is apparent by from the histograms, over half of the census tracts had 0 shootings. Because of this heavily skewed log distribution, we found it very difficult to find a model that came close to reflecting this, and pursued just a model that produced county level results.

Results:

Initially, we performed basic linear regression to on the county level data. Because we were able to obtain an output distribution that looked mostly normal by taking ln(incident_counts/1000 people), we decided to use this as our target variable.

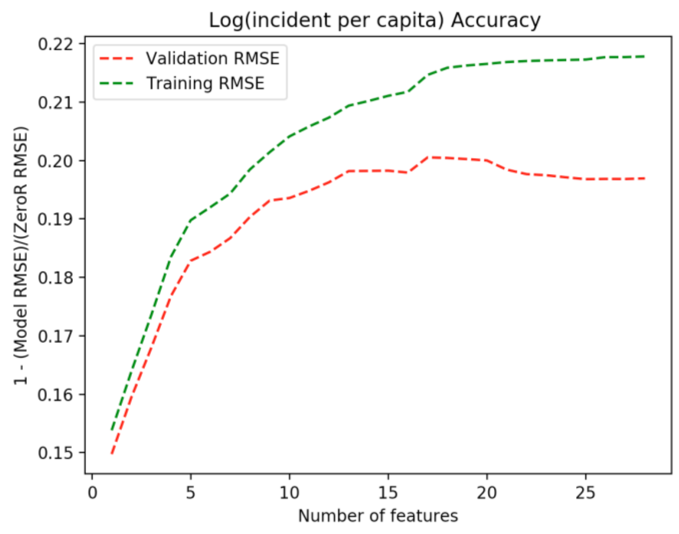
To evaluate our model's performance, we used two different accuracy measures, log accuracy and raw accuracy. For the first, we compared the root mean squared error (RMSE) of the predicted target vs the target values, and compared this against a baseline. For the second, we convert our model's predictions of y into raw incident counts, and evaluated this against a zeroR value for raw incident counts.

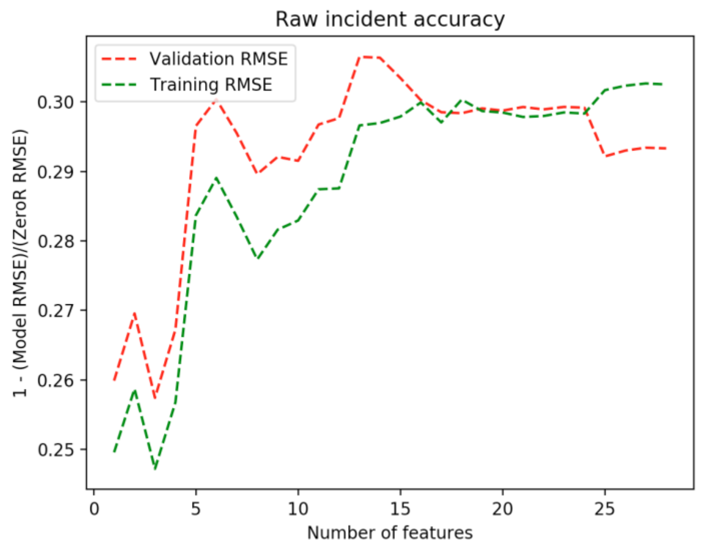
Log RMSE Accuracy	Raw RMSE Accuracy
$y = \ln\left(\frac{incident\_counts}{population} * 1000 + 1\right)$	$y = incident\_counts$
$y_{mean} = \text{mean}(\text{y})$	$y_{mean} = \text{mean}(\text{y})$
$y_{pred} \; => model\ prediction\ of\ y$	$y_{pred\_raw} = \frac{(e^{y_{pred}} - 1) * population}{1000}$
$RMSE = \frac{\sum_0^n (y_n - y_{pred\_n})^2}{n}$	$RMSE = \frac{\sum_0^n (y_n - y_{pred\_raw})^2}{n}$
$ZeroR = \frac{\sum_0^n (y_n - y_{mean})^2}{n}$	$ZeroR = \frac{\sum_0^n (y_n - y_{mean})^2}{n}$
$\boldsymbol{Accuracy = 1 - \frac{RMSE}{ZeroR}}$	$\boldsymbol{Accuracy = 1 - \frac{RMSE}{ZeroR}}$

This accuracy evaluated our ability to predict the raw incident counts of a census region, while the first accuracy graded our ability to predict the log transformed output. Obviously, positive values indicate the model outperforming zeroR.

To perform initial feature selection, we wrote an algorithm that started with zero features and greedily added the feature that most improved testing accuracy. On each subset of features, we performed basic linear regression with k-Fold validation using 3 folds. To get the training and testing accuracies for each subset, we averaged over the 3 folds.

Below are the plots displaying the accuracy values as a function of the number of features that the algorithm had added. Both error plots are generated with same linear model for each set of features, but using the different accuracy metrics described above. The code we used to perform feature selection can be found at this link:
https://github.com/olcole/GunViolencePrediction_ML/blob/master/featureSelection.py





Below displays the input features we used with their ranking according to this greedy algorithm:

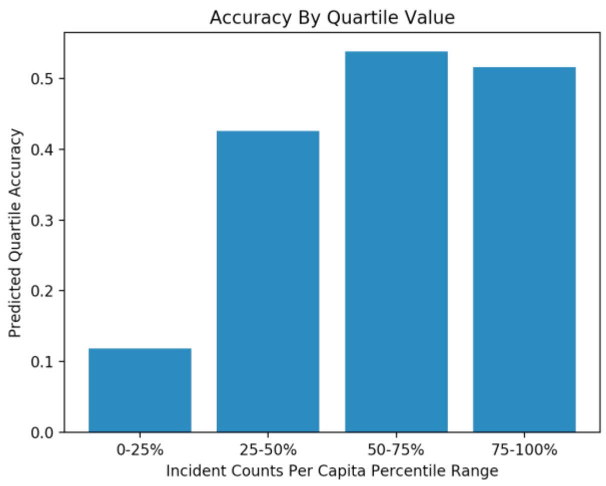
Feature Name	Description	Rank
African-American	% of population that is African American	1
SelfEmployed	% of population that is self employed	2
MeanCommute	The average commute time for workers	3
PercentMen	% of population that are men	4
Drive	% of workers that drives to work	5
Office		6
Hispanic	% of population that is hispanic	7
ChildPoverty	% of children that are below the poverty line	8
Walk	% of workers that walk to work	9
Production	% of workers that work in production	10

Poverty	% of people below the poverty line	11
White	% of population that is white	12
PublicWork	% of workers that work do public work	13
PrivateWork	% of workers that work do private work	14
WorkAtHome	% of workers that work form hom	15
Asian	% of population that is Asian	16
Native	% of population that are Native Americans	17
Pacific	% of population that are Pacific	18
OtherTransp	% of workers that does not commute via walking, transit or car	19
Service	% of workers that work in the service industy	20
IncomePerCap	Income per capita	21
Unemployment	Unemployment Rate	22
PercentEmployed	Percentage of People that are employed	23
FamilyWork	% of workers who work in a family business	24
Carpool	% of workers who carpool to work	25
Transit	% of workers who take public transit to work	26
Construction	% of workers who carpool to work	27
Professional		

The regression looks to fit the data best between 10-15 features. As expected the training error continued to increase with additional features. However, the validation error actually surpassed the training error when using the accuracy metric on raw incident counts. This was surprising, but since our linear model was trained using the log distribution and not the raw distribution, it is definitely possible.

We also ran the same feature selection algorithm using combinations of features. This technique did not produce any change in our accuracies metrics, and the top 5 features did not change using this approach, so we decided not use any of these combinations in future tests.

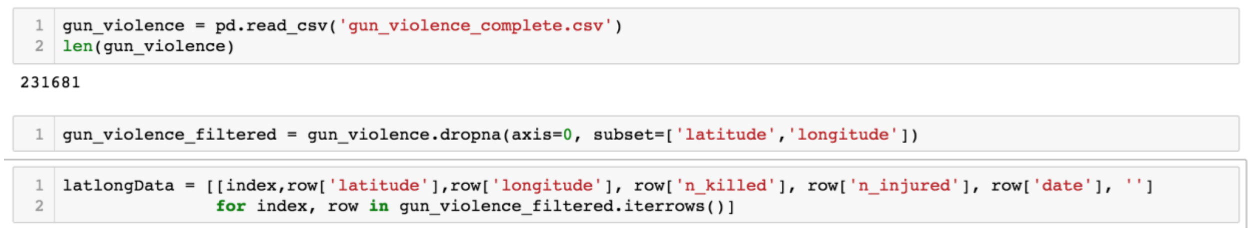
The plot below displays where our linear regressor made mistakes. To generate this plot, we used our linear model trained on the full subset of features. We broke the incident counts per capita output into quartile bins and generated a new accuracy by measuring whether the regressor's predicted value was in the same quartile as the actual value. Unlike the accuracies used in the previous plot, this accuracy is more of a true accuracy, representing the percentage of correct classifications. We broke down the accuracies by the quartile of the actual value.



The regressor makes most of it’s errors on the lowest quartile. This is because our data is skewed to the right, pulling the mean above the median. While the majority of the counts are closer to 0, in order to minimize the least-squares error the regressor must overestimate these values to stay closer to the mean. This effect was actually somewhat by taking the natural log of the output, but even taking the natural log was not enough to fully correct for it, as indicated by the plots. It’s possible that using a base 10 log instead of the natural log may have been a better approach.

The overall accuracy for this technique was 40%. This outperforms the zeroR approach, which gives a 25% accuracy (each data point has a 25% of being in each quartile bin).

Data aggregation code:



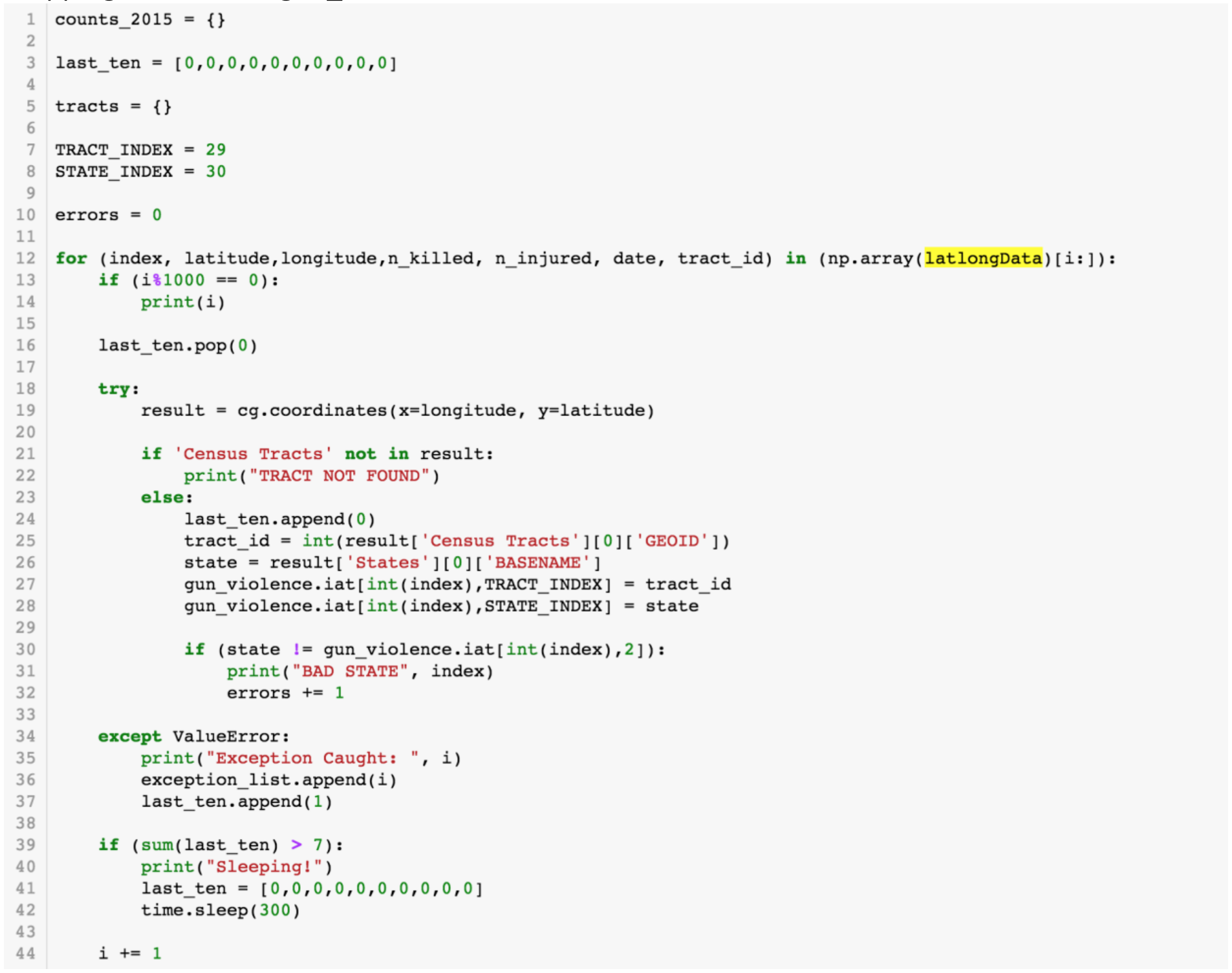
```
gun_violence = pd.read_csv('gun_violence_complete.csv')
len(gun_violence)
```

```
231681
```

```
gun_violence_filtered = gun_violence.dropna(axis=0, subset=['latitude','longitude'])
```

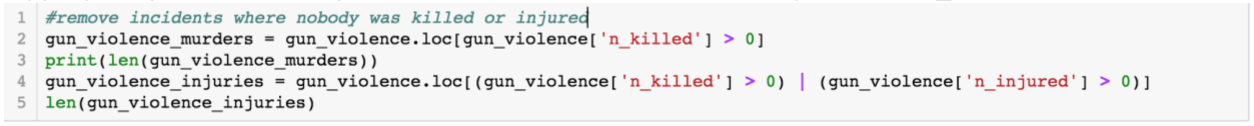
```
latlongData = [[index,row['latitude'],row['longitude'], row['n_killed'], row['n_injured'], row['date'], '']
               for index, row in gun_violence_filtered.iterrows()]
```

Mapping each row in gun_violence dataset to a census tract

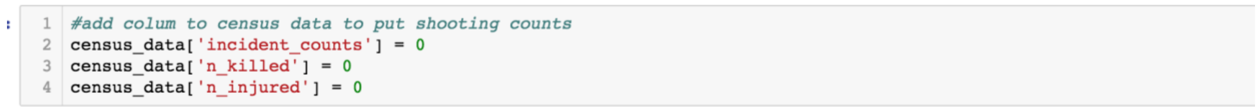


```
counts_2015 = {}

last_ten = [0,0,0,0,0,0,0,0,0,0]

tracts = {}

TRACT_INDEX = 29
STATE_INDEX = 30

errors = 0

for (index, latitude,longitude,n_killed, n_injured, date, tract_id) in (np.array(latlongData)[i:]):
    if (i%1000 == 0):
        print(i)

    last_ten.pop(0)

    try:
        result = cg.coordinates(x=longitude, y=latitude)

        if 'Census Tracts' not in result:
            print("TRACT NOT FOUND")
        else:
            last_ten.append(0)
            tract_id = int(result['Census Tracts'][0]['GEOID'])
            state = result['States'][0]['BASENAME']
            gun_violence.iat[int(index),TRACT_INDEX] = tract_id
            gun_violence.iat[int(index),STATE_INDEX] = state

            if (state != gun_violence.iat[int(index),2]):
                print("BAD STATE", index)
                errors += 1

    except ValueError:
        print("Exception Caught: ", i)
        exception_list.append(i)
        last_ten.append(1)

    if (sum(last_ten) > 7):
        print("Sleeping!")
        last_ten = [0,0,0,0,0,0,0,0,0,0]
        time.sleep(300)

    i += 1
```

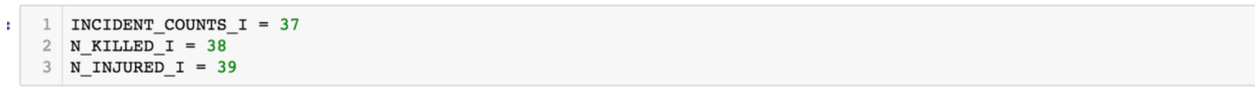
Aggregating counts from gun violence incident list and adding to census_data table



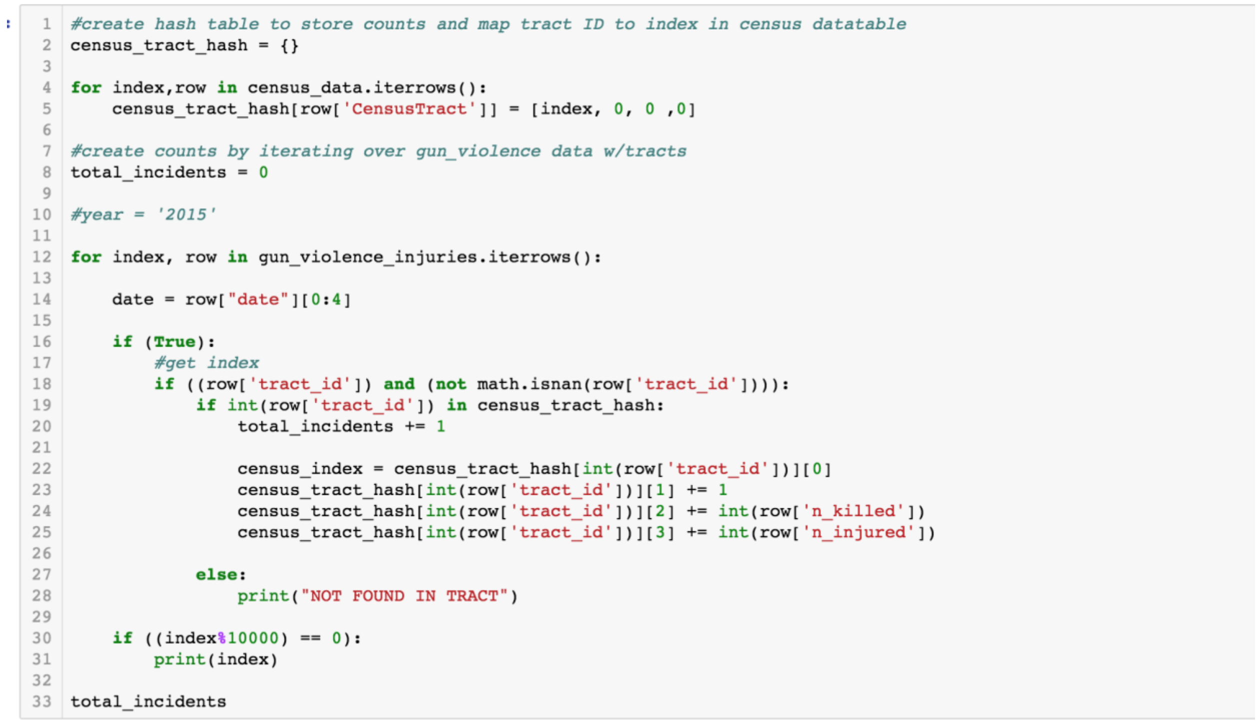
```
#remove incidents where nobody was killed or injured
gun_violence_murders = gun_violence.loc[gun_violence['n_killed'] > 0]
print(len(gun_violence_murders))
gun_violence_injuries = gun_violence.loc[(gun_violence['n_killed'] > 0) | (gun_violence['n_injured'] > 0)]
len(gun_violence_injuries)
```



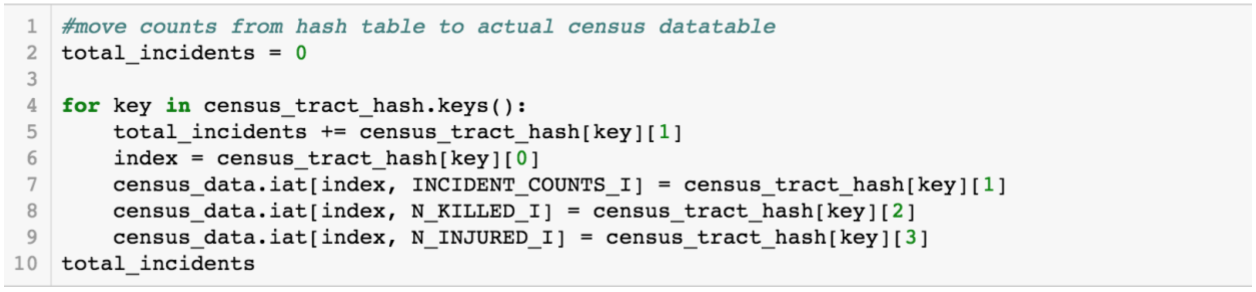
```
#add colum to census data to put shooting counts
census_data['incident_counts'] = 0
census_data['n_killed'] = 0
census_data['n_injured'] = 0
```



```
INCIDENT_COUNTS_I = 37
N_KILLED_I = 38
N_INJURED_I = 39
```



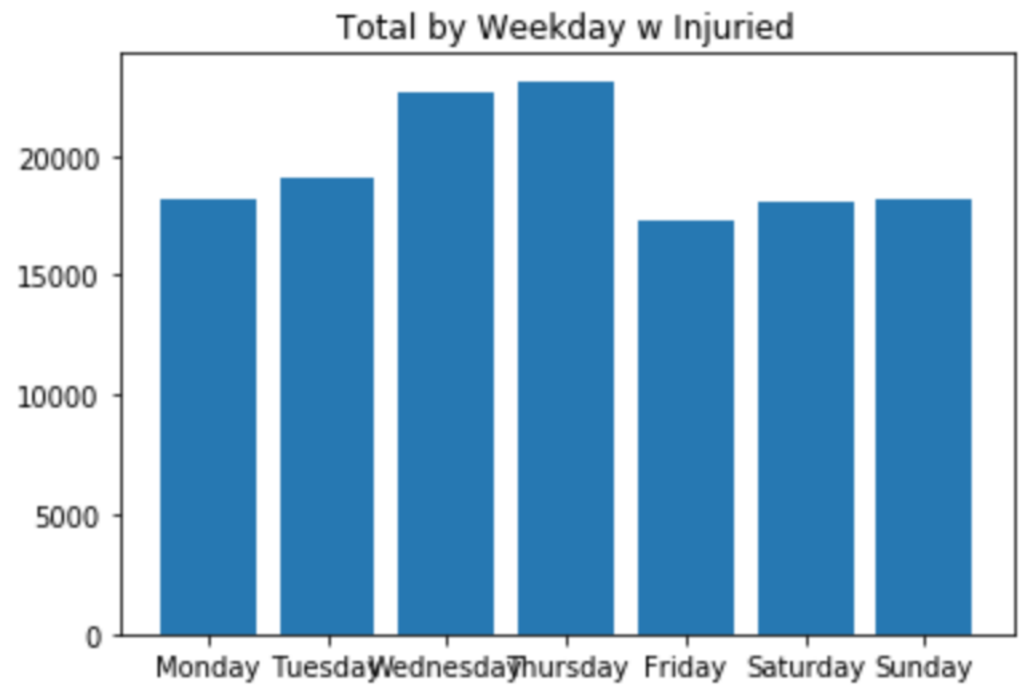
```
#create hash table to store counts and map tract ID to index in census datatable
census_tract_hash = {}

for index,row in census_data.iterrows():
    census_tract_hash[row['CensusTract']] = [index, 0, 0 ,0]

#create counts by iterating over gun_violence data w/tracts
total_incidents = 0

#year = '2015'

for index, row in gun_violence_injuries.iterrows():

    date = row["date"][0:4]

    if (True):
        #get index
        if ((row['tract_id']) and (not math.isnan(row['tract_id']))):
            if int(row['tract_id']) in census_tract_hash:
                total_incidents += 1

                census_index = census_tract_hash[int(row['tract_id'])][0]
                census_tract_hash[int(row['tract_id'])][1] += 1
                census_tract_hash[int(row['tract_id'])][2] += int(row['n_killed'])
                census_tract_hash[int(row['tract_id'])][3] += int(row['n_injured'])

            else:
                print("NOT FOUND IN TRACT")

    if ((index%10000) == 0):
        print(index)

total_incidents
```

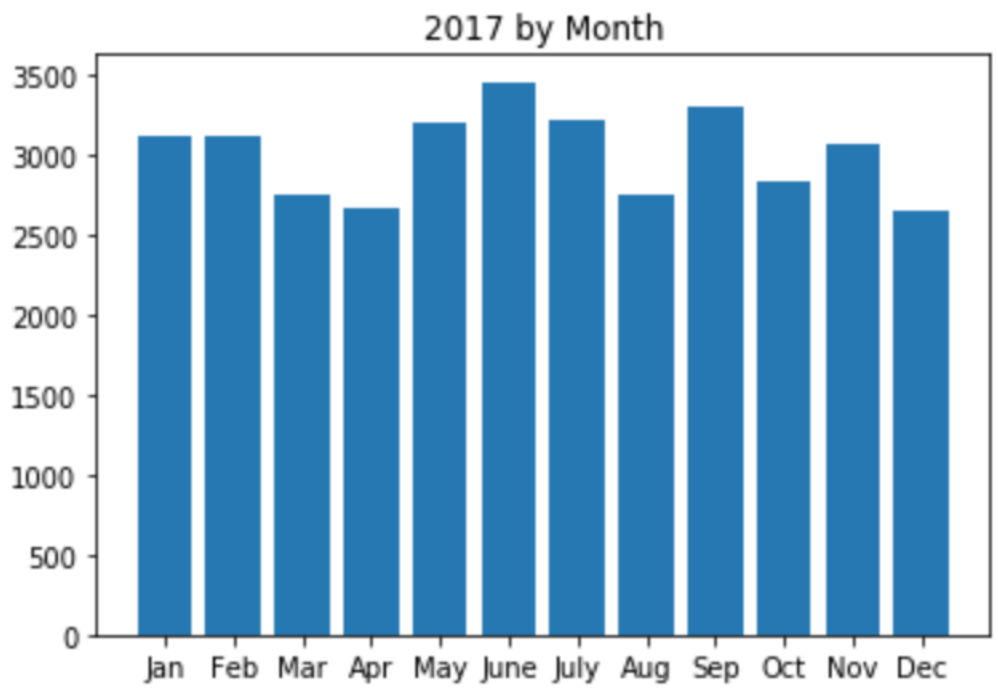


```
#move counts from hash table to actual census datatable
total_incidents = 0

for key in census_tract_hash.keys():
    total_incidents += census_tract_hash[key][1]
    index = census_tract_hash[key][0]
    census_data.iat[index, INCIDENT_COUNTS_I] = census_tract_hash[key][1]
    census_data.iat[index, N_KILLED_I] = census_tract_hash[key][2]
    census_data.iat[index, N_INJURED_I] = census_tract_hash[key][3]
total_incidents
```

Each tract was in county so converting from a census tracts to counties was straight forward

Other Visualizations of Gun Violence In America:


Total by Weekday w Injuried
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday


2017 by Month
Jan
Feb
Mar
Apr
May
June
July
Aug
Sep
Oct
Nov
Dec