Externship Project

Presented By: Grace Mao, Elissa Ito, Juan Rached, Jawad Yousef, Mulan Jiang

US

Mulan Jiang
MechE

Juan Rached
EECS

Grace Mao
AeroAstro

Elissa Ito
MechE

Jawad Yousef
AeroAstro

The Past Week:

- Focused on:
  - Animal crossings
  - Identifying unsafe areas

(we wanted to try other things too but these two became the main focus)

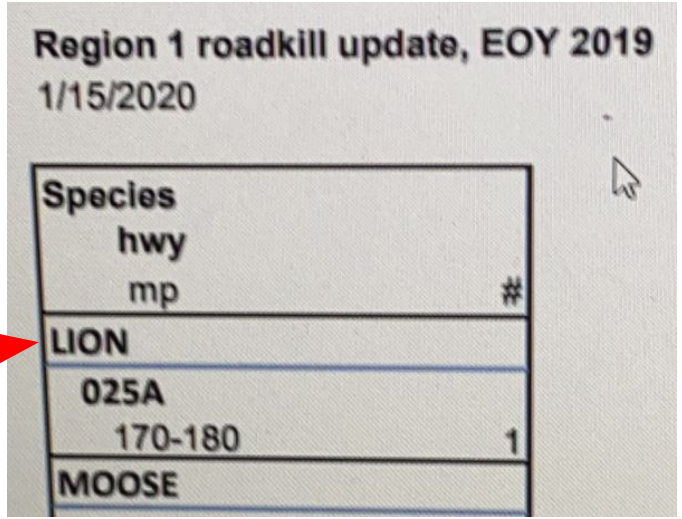

New Horizons
Welcome to
Animal Crossing

Initial research

- Most common roadkill: deer
- Animal crossing signs are placed due to citizens requests
- Collisions tend to happen during sunset to sunrise
- Deer travel together in herds/bevies
- Factors:
  - Nearly triple the amount of crashes in November compared to August (for insured drivers)
  - Mating season (Oct. - Jan.)
  - Time of day
  - Proximity to parks/green spaces

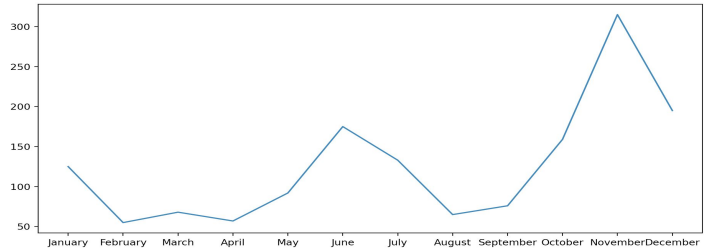
Proposal for Predictions

- An equation that has a max of 100 (where 100 is high risk of deer)
- Overlay with existing data on animal-strike accidents
- Collaborate with insurance companies to get their crash data
- Reached out to different state DOT's
- Clustering Algorithms

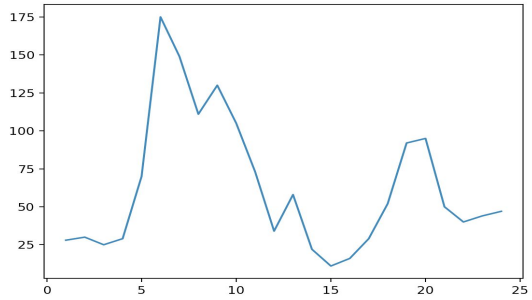


Data Evidence

Deer Crashes per Month in VT for the past 5 years

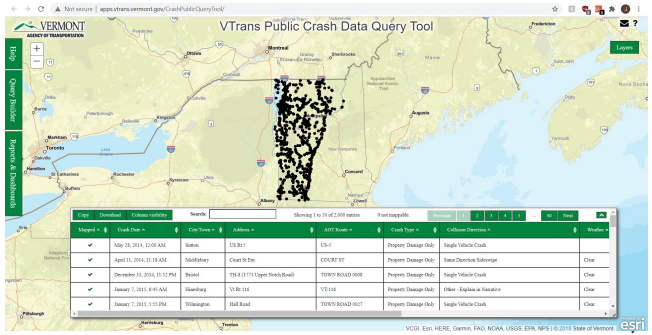


Deer Crashes per Hour in VT for the past 5 years



Streets w/ 40+ Deer Crashes in the past 5 years in VT:

{'I-91': 221, 'VT-15': 55, 'I-89': 97, 'US-7': 159, 'US-2': 43, 'VT-9': 41, 'VT-100': 49}



Code

1. Retains only crashes that fit description i.e. ‘deer’

```
"""enter animal name in string, finds all crashes related to that animal, one can also enter 'animal' """
#use 'ANIMAL' for chicago, and e.g. 'Deer' for vt,
def crash_source_finder(loc,animal):
    field_to_col_num =  loc_to_info[loc][1]
    field_list = loc_to_info[loc][0]
    source = loc_to_info[loc][2]
    animals_colnums = loc_to_info[loc][3]
    anim_crashes = []
    with open(source) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        found = False
        for row in csv_reader:
            column_count = 0
            row_info = []
            for colnum in animals_colnums:
                if animal in row[colnum]:
                    found = True
            if found == True:
                for column in row:
                    if column_count in field_list:
                        row_info.append(column)
                    column_count += 1
                anim_crashes.append(row_info)
            found = False
        line_count += 1
    return anim_crashes

#chicago_animal_crashes = crash_source_finder('chicago','ANIMAL')
vt_deer_crashes = crash_source_finder('vt','Deer')
#nyc_animal_crashes = crash_source_finder('nyc','Animal')
```

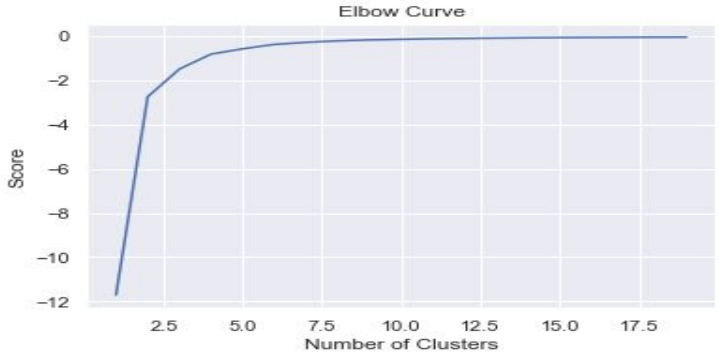
2. Talleys the number of crashes for given category

```
"""given crash table, talleys field name"""
def counter(loc,crash_table,field_name):
    field_to_col_num =  loc_to_info[loc][1]
    field_list = loc_to_info[loc][0]
    source = loc_to_info[loc][2]
    seen = set()
    counts = {}
    col_num = field_to_col_num[field_name]
    row_num = 0
    if field_name == 'date_time':
        monthseen = set()
        hourseen = set()
        monthcounts = {}
        hourcounts = {}
        for row in crash_table:
            if row_num == 0:
                pass
            else:
                month = row[col_num][5:7]
                hour = row[col_num][11:13]
                if month not in monthseen:
                    monthseen.add(month)
                    monthcounts[month] = 1
                else:
                    monthcounts[month] = monthcounts[month] + 1
                if hour not in hourseen:
                    hourseen.add(hour)
                    hourcounts[hour] = 1
                else:
                    hourcounts[hour] = hourcounts[hour] + 1
            row_num += 1
        return (monthcounts,hourcounts)
    for row in crash_table:
        if row_num == 0:
            pass
        else:
            field = row[col_num]
            if field not in seen:
                seen.add(field)
                counts[field] = 1
            else:
                counts[field] = counts[field] + 1
        row_num += 1
    return counts
```

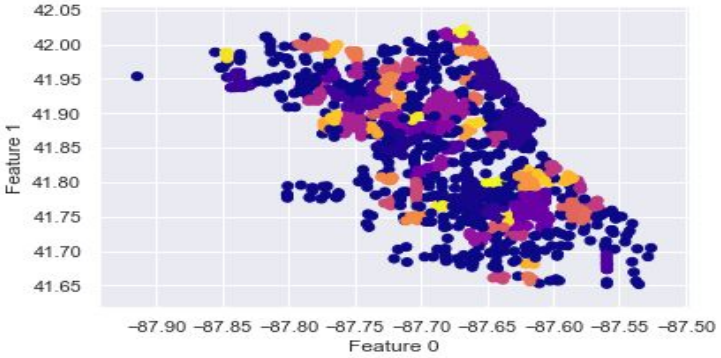
3. Lists only field types that occur at higher than input frequency

```
def high_freq(loc,crash_table,freqval,category):
    catcounts = counter(loc,crash_table,category)
    if len(catcounts) > 1:
        freqcounts0 = {}
        for catname in catcounts[0].keys():
            if catcounts[0][catname] > freqval:
                freqcounts0[catname] = catcounts[0][catname]
        freqcounts1 = {}
        for catname in catcounts[1].keys():
            if catcounts[1][catname] > freqval:
                freqcounts1[catname] = catcounts[1][catname]
    return freqcounts0,freqcounts1
    freqcounts = {}
    for catname in catcounts.keys():
        if catcounts[catname] > freqval:
            freqcounts[catname] = catcounts[catname]
    return freqcounts
```

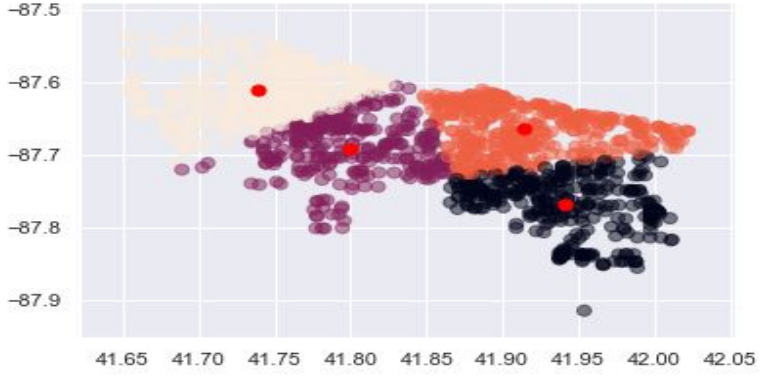
Clustering Elbow Curve



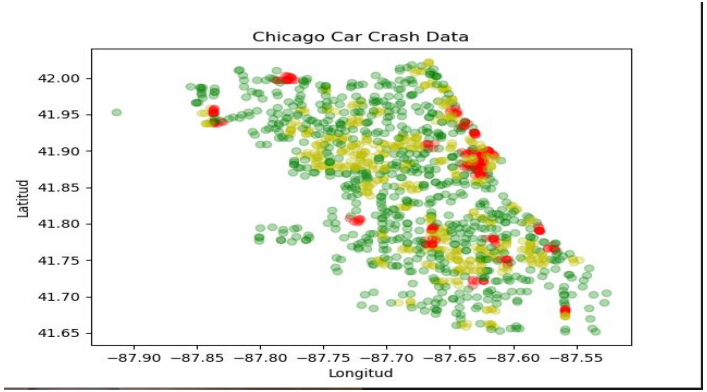
DBSCAN Clustering



K-means Clustering



Heat Map



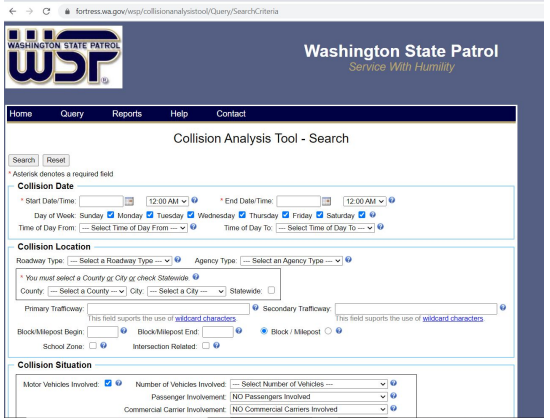
Washington State Stats 2016- 2021

Deer Accidents with relation to lighting - {'Dawn': 353, 'Dark-Street Lights On': 416, 'Daylight': 1952, 'Dark-No Street Lights': 2787, 'Dusk': 204, 'Dark-Street Lights Off': 48, 'Unknown': 13, 'Other': 2, '': 3, 'Dark - Unknown Lighting': 16}

The city with the most deer crashes - Spokane with 181 cases

The lowest bound of property damage- 5,707,000

In conversation concerning obtaining street sign geo-data



Clustering Code

```
column_names = pandas_list.pop(0)
data = pd.DataFrame(pandas_list, columns=column_names)

#Removing Un-needed colomns
not_needed = ['CRASH_RECORD_ID','RD_NO','CRASH_DATE_EST_I','TRAFFIC_CONTROL_DEVICE','DEVICE_CONDITION','LANE_CNT','ALIGNMENT','ROAD_DEFECT','REPORT_
              'INTERSECTION_RELATED_I','NOT_RIGHT_OF_WAY_I','HIT_AND_RUN_I','PHOTOS_TAKEN_I','STATEMENTS_TAKEN_I','DOORING_I','WORK_ZONE_I','WORK_ZO
              'WORKERS_PRESENT_I','NUM_UNITS','MOST_SEVERE_INJURY','INJURIES_REPORTED_NOT_EVIDENT','INJURIES_NO_INDICATION','LOCATION']

data.drop(not_needed, inplace=True, axis=1, errors='ignore')

#Debugging
#data.to_csv(r'Visual_data.csv', index = False)


new_data = [data['LATITUDE'][11:], data['LONGITUDE'][11:]]
#
headers = ['LATITUDE','LONGITUDE']
df3 = pd.concat(new_data, axis=1, keys=headers)
df3['LATITUDE'] = df3['LATITUDE'].astype(float)
df3['LONGITUDE'] = df3['LONGITUDE'].astype(float)

#Debugging
#df3.to_csv(r'longlat.csv', index = False)


#KMeans Clustering
#Elbow Curve
K_clusters = range(1,20)
kmeans = [KMeans(n_clusters=i) for i in K_clusters]
X_axis = df3[['LATITUDE']]
Y_axis = df3[['LONGITUDE']]
score = [kmeans[i].fit(X_axis).score(X_axis) for i in range(len(kmeans))]
plt.plot(K_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()

#The Actual Clustering
kmeans = KMeans(n_clusters=3).fit(df3)
centroids = kmeans.cluster_centers_
print(centroids)
up_top= kmeans.labels_.astype(float)
plt.scatter(df3['LATITUDE'], df3['LONGITUDE'], c= kmeans.labels_.astype(float), s=50, alpha=0.5)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
```

```
#Attempt at DBSCAM
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df3)
# cluster the data into five clusters
dbscan = DBSCAN(eps=0.123, min_samples = 5)
clusters = dbscan.fit_predict(X_scaled)

# plot the cluster assignments
plt.scatter(df3['LATITUDE'], df3['LONGITUDE'], c=clusters, cmap="plasma")
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Washington State Code

```
#To deal with Washington State Data
#Very Detailed Such as Deer, Elk, Bear, Raccoon, etc.

def washington_break_down(filed):
    pandas_list = []
    header = None
    damage_counter = 0
    loc_dict = {}
    light_dict = {}
    city_dict = {}

    counter = 0
    with open(filed, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            if counter == 0:
                header = row
                counter += 1
            else:
                pandas_list.append(row)

    data = pd.DataFrame(pandas_list, columns=header)
    for x in data['Primary Trafficway']:
        if x.strip() in loc_dict:
            loc_dict[x.strip()] += 1
        else:
            loc_dict[x.strip()] = 1

    for x in data['Secondary Trafficway']:
        if x.strip() in loc_dict:
            loc_dict[x.strip()] += 1
        else:
            loc_dict[x.strip()] = 1

    for x in data['Lighting Condition']:
        if x in light_dict:
            light_dict[x] += 1
        else:
            light_dict[x] = 1

    for x in data['Damage Threshold Met']:
        if x == 'Y':
            damage_counter += 1000
    for x in data['City']:
        if x in city_dict:
            city_dict[x] += 1
        else:
            city_dict[x] = 1

    return damage_counter, city_dict, damage_counter, light_dict, loc_dict
```

$H = 55M_v + T + 1.25D$

H = Hazard Value (max of 100)

M_v = Month Value based on data

T = Time of day (20 for dusk and dawn, 10 for night, 0 for day)

D = Distance in miles from city ($D \leq 20$, if greater than 20 just put 20)

Month	Value
Jan	6.4/13.8
Feb	5.3/13.8
Mar	5.1/13.8
April	4.8/13.8
May	5.4/13.8
June	5.9/13.8
July	4.6/13.8
Aug	4/13.8
Sept	5.2/13.8
Oct	9.9/13.8
Nov	1
Dec	8.1/13.8

Future Considerations

- Algorithms need refinement
  - Perhaps be more specific to smaller areas
- State Crash Analysis Queries
  - Each State has a different way of organizing data, some including geolocational info while others not
  - Some queries rely heavily on officer input, leds to randomness in entries. Needs an Autocorrect Algorithm
- Heat maps
  - Identify “hotspots” of deer populations
- Insurance Data Collaboration
- Deer Hazard Equation Refinement

Unsafe Conditions

Initial Research

- Crime reports
- Maps of crime distribution
- Where security systems are typically installed
    - Inversely related to crime rates
- If insurance reports thefts and vandalism damages
- Police department data

Factors to Consider

- Crime rates in each area
- Type of crimes (want to primarily focus on violent crime, vandalism in parking lots, car thefts and attempted car hijackings, shootings in areas)
- Urban planning (ex. New York City and KCMO), redlining from decades ago that have allowed certain areas to thrive whereas others struggle to improve economic status
    - Generally highlights higher risk neighborhoods
    - Bias in a lot of crime maps due to the fact that some just go based on number

Proposal

- Gather data from law enforcement/cities
- Create maps of vehicle crime
- Overlay this with existing maps
- Isolate areas of notable interest
  - Compare with what is known about cities
  - If accurate, then can rely on data clusters to locate these areas

Code

```python
with open('chicago_crimes.csv') as file:
    csv_reader = csv.reader(file, delimiter=',')
    count = 0
    latitude = []
    longitude = []
    for row in csv_reader:
        if count == 0:
            count += 1
        else:
            #if "VEHICLE" in row[4] or "VEHICLE" in row[5]:
            if "TO VEHICLE" in row[5] or 'VEHICULAR' in row[5]:
                if row[14] == '' or row[15] == '':
                    count += 1
                else:
                    latitude.append(float(row[14]))
                    longitude.append(float(row[15]))
```

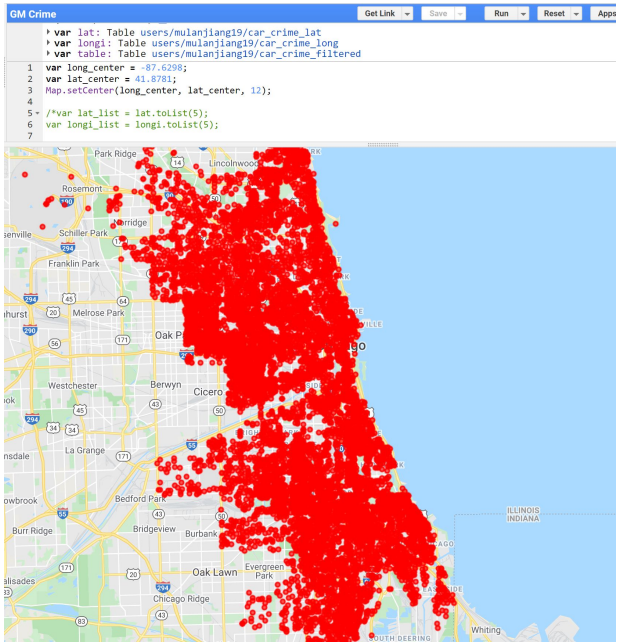
- Filter through data for vehicle crimes
  - First tried to search just for vehicle crimes
  - Excluded motor vehicle thefts
  - Export to a csv with location of all vehicular crime
- Visualization:
  - Import CSVs into Google Earth Engine
  - Transform CSV table to list of coordinates
  - Attempted to isolate latitude and longitude - did not work
  - Created function to return individual coordinates and compile them
  - Mapped compiled coordinates to visualize data

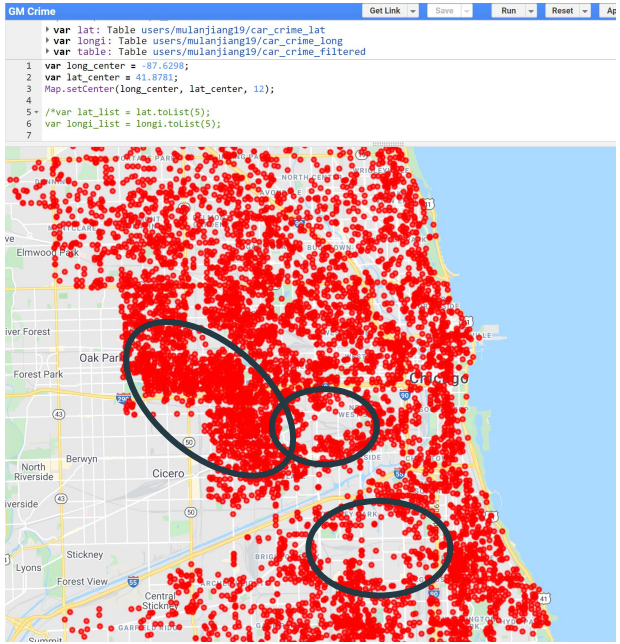
GM Crime All — Get Lir

Imports (3 entries)
- var lat: Table users/mulanjiang19/car_crime_lat
- var longi: Table users/mulanjiang19/car_crime_long
- var table: Table users/mulanjiang19/car_crime_filtered

```
var long_center = -87.6298;
var lat_center = 41.8781;
Map.setCenter(long_center, lat_center, 12);

/*var lat_list = lat.toList(5);
var longi_list = longi.toList(5);

var lat_point = ee.Number(lat_list.get(1));
var longi_point = ee.Number(longi_list.get(1));
var coords = lat_point.merge(longi_point)
print(coords);*/

var numb_list = ee.List.sequence(1, 12915);

var coords_list = table.toList(12916);

function get_coords(numb) {
  var coord = ee.Number(coords_list.get(numb));
  return coord;
}

var compiled_coords = numb_list.map(get_coords);

print(compiled_coords);

var coords_collection = ee.FeatureCollection(compiled_coords);

Map.addLayer(coords_collection, {color: 'FF0000'});
```

Visualization



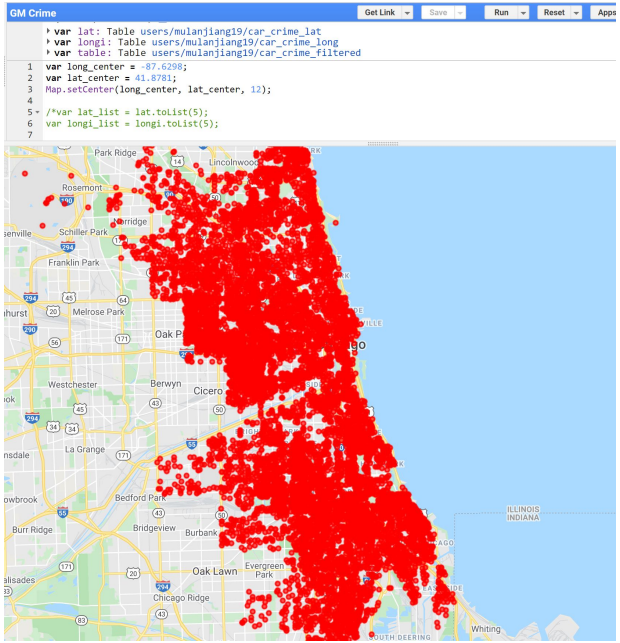


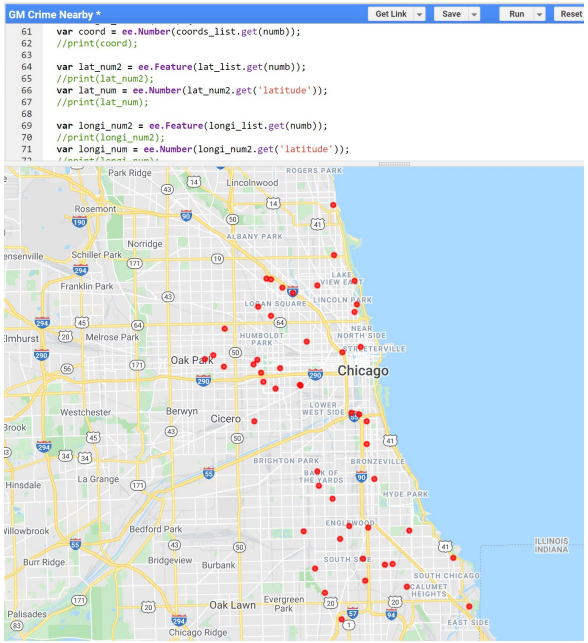
https://code.earthengine.google.com/b467e4b356c826ccfae4a728afa80369

Map visualizes vehicle-related crimes from the past X years and reveals hotspots for crime

Can also see areas with less crime (college areas, residential areas, etc.)

Radius Visualization





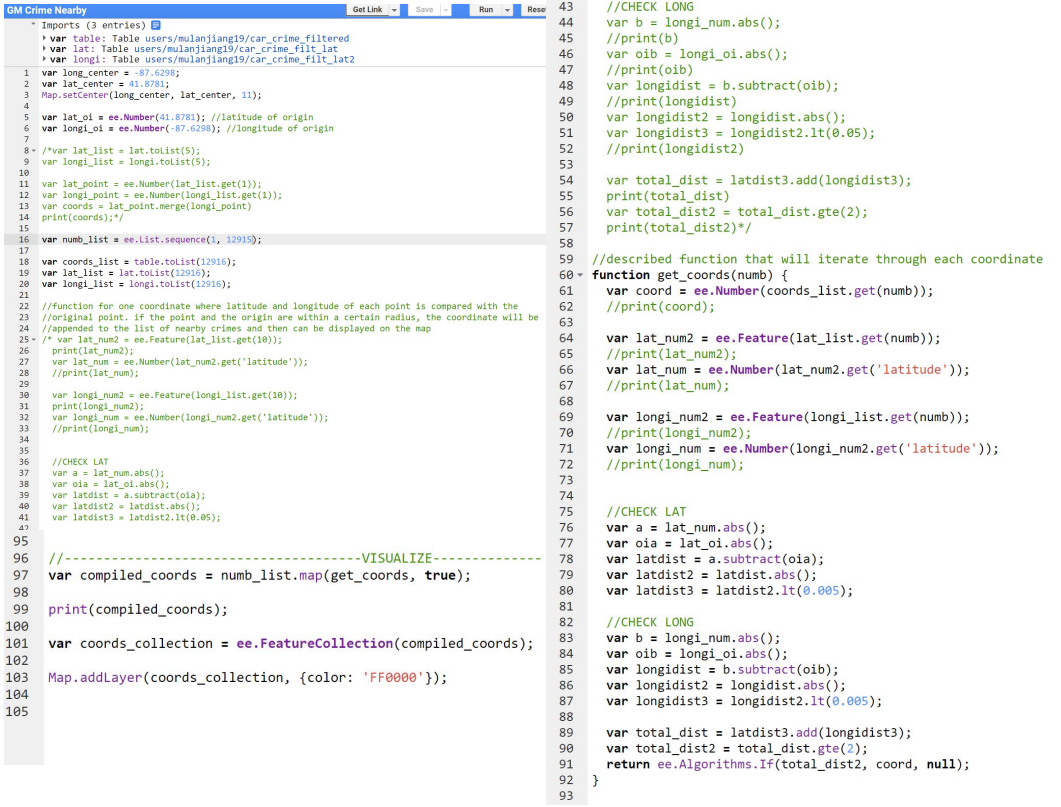
https://code.earthengine.google.com/d188fe464f8739a253b4635e6e2abdc9

Takes in input coordinates and compares them to the coordinates of all of the crime locations. If the crime was within a certain radius of the input coordinates, then it is visualized on the map.

Narrows down data

Radius Visualization Code

- It doesn't visually show a radius
- Manual checking of the first 20 points is consistent
- Running with all 12916 points leads to visual errors
  - Still need to debug
- Shows right trend
  - Smaller radiuses lead to fewer points but also leave out points that should clearly be in the radius
- Errors
  - Coding longitude and latitude separately, so if either longitude or latitude were in the radius, it would count it as a nearby point



GM Crime Nearby

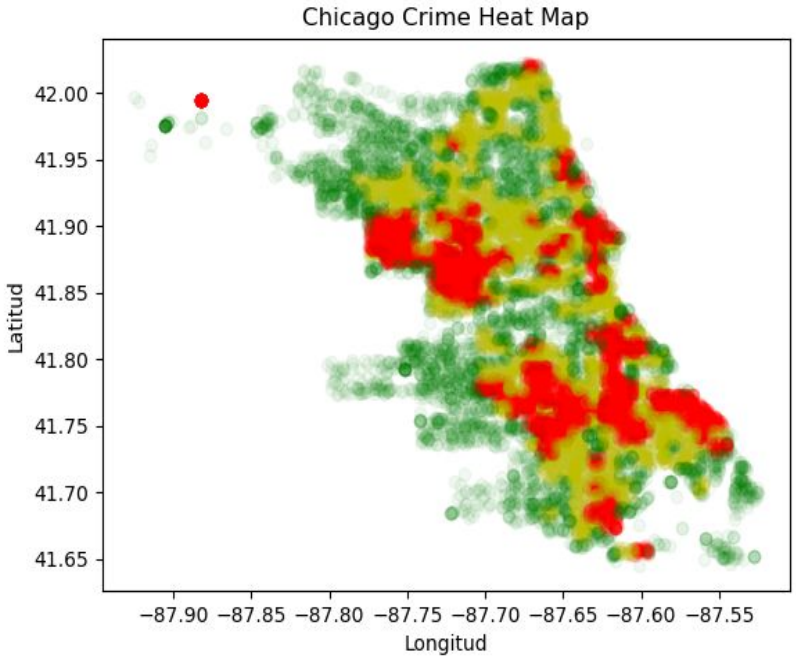
Get Link | Save | Run | Reset

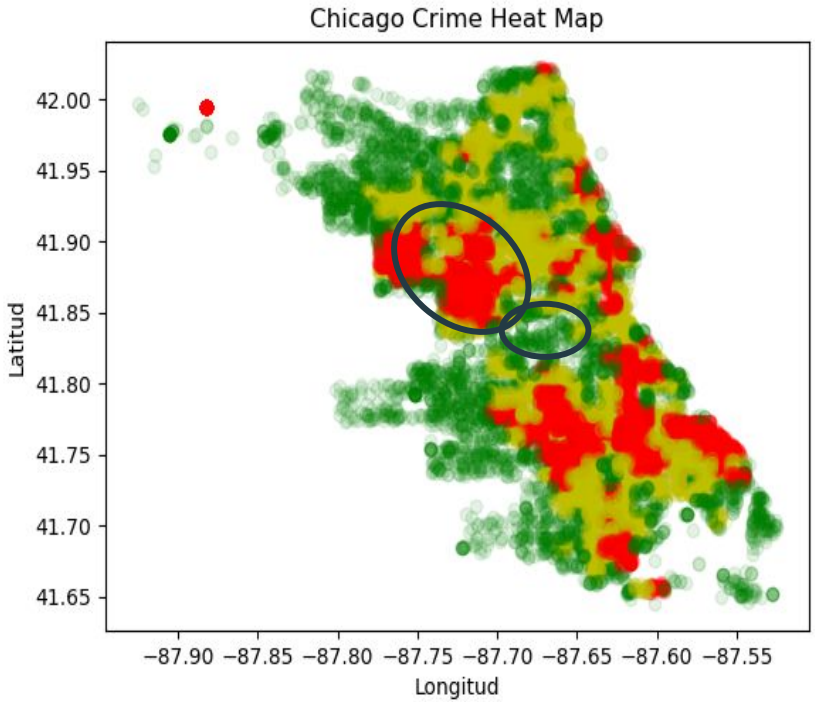
Imports (3 entries)
- var table: Table users/mulanjiang19/car_crime_filtered
- var lat: Table users/mulanjiang19/car_crime_filt_lat
- var longi: Table users/mulanjiang19/car_crime_filt_lat2

```
var long_center = -87.6298;
var lat_center = 41.8781;
Map.setCenter(long_center, lat_center, 11);

var lat_oi = ee.Number(41.8781); //latitude of origin
var longi_oi = ee.Number(-87.6298); //longitude of origin

/*var lat_list = lat.toList(5);
var longi_list = longi.toList(5);

var lat_point = ee.Number(lat_list.get(1));
var longi_point = ee.Number(longi_list.get(1));
var coords = lat_point.merge(longi_point)
print(coords);*/

var numb_list = ee.List.sequence(1, 12915);

var coords_list = table.toList(12916);
var lat_list = lat.toList(12916);
var longi_list = longi.toList(12916);

//function for one coordinate where latitude and longitude of each point is compared with the
//original point. if the point and the origin are within a certain radius, the coordinate will be
//appended to the list of nearby crimes and then can be displayed on the map
/* var lat_num2 = ee.Feature(lat_list.get(10));
 print(lat_num2);
 var lat_num = ee.Number(lat_num2.get('latitude'));
 //print(lat_num);

 var longi_num2 = ee.Feature(longi_list.get(10));
 print(longi_num2);
 var longi_num = ee.Number(longi_num2.get('latitude'));
 //print(longi_num);


 //CHECK LAT
 var a = lat_num.abs();
 var oia = lat_oi.abs();
 var latdist = a.subtract(oia);
 var latdist2 = latdist.abs();
 var latdist3 = latdist2.lt(0.05);


//-------------------------------------VISUALIZE-------------
var compiled_coords = numb_list.map(get_coords, true);

print(compiled_coords);

var coords_collection = ee.FeatureCollection(compiled_coords);

Map.addLayer(coords_collection, {color: 'FF0000'});
```

```
  //CHECK LONG
  var b = longi_num.abs();
  //print(b)
  var oib = longi_oi.abs();
  //print(oib)
  var longidist = b.subtract(oib);
  //print(longidist)
  var longidist2 = longidist.abs();
  var longidist3 = longidist2.lt(0.05);
  //print(longidist2)

  var total_dist = latdist3.add(longidist3);
  print(total_dist)
  var total_dist2 = total_dist.gte(2);
  print(total_dist2)*/

//described function that will iterate through each coordinate
function get_coords(numb) {
  var coord = ee.Number(coords_list.get(numb));
  //print(coord);

  var lat_num2 = ee.Feature(lat_list.get(numb));
  //print(lat_num2);
  var lat_num = ee.Number(lat_num2.get('latitude'));
  //print(lat_num);

  var longi_num2 = ee.Feature(longi_list.get(numb));
  //print(longi_num2);
  var longi_num = ee.Number(longi_num2.get('latitude'));
  //print(longi_num);


  //CHECK LAT
  var a = lat_num.abs();
  var oia = lat_oi.abs();
  var latdist = a.subtract(oia);
  var latdist2 = latdist.abs();
  var latdist3 = latdist2.lt(0.005);

  //CHECK LONG
  var b = longi_num.abs();
  var oib = longi_oi.abs();
  var longidist = b.subtract(oib);
  var longidist2 = longidist.abs();
  var longidist3 = longidist2.lt(0.005);

  var total_dist = latdist3.add(longidist3);
  var total_dist2 = total_dist.gte(2);
  return ee.Algorithms.If(total_dist2, coord, null);
}
```

Heat Map Visualization





HM Visualization Code

Procedure:

- Coordinate grid of 2,500 grids.
- Adds each coordinate data to its respective grid.
- Data points within a grid are plotted in dark green, light green, and red depending on the density of data points per grid.

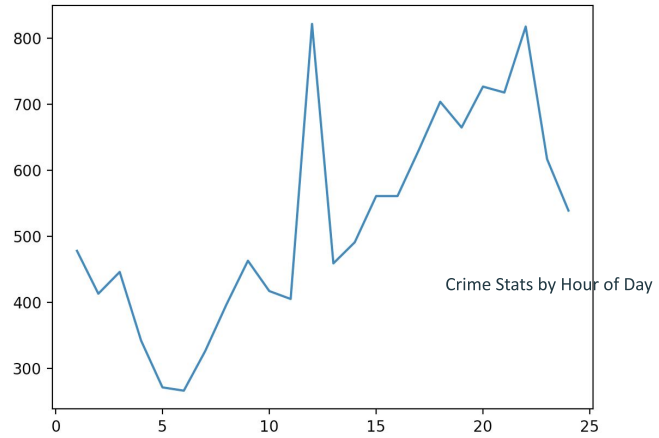
Observations:

- Despite using a different dataset, the Heat Map visual data exhibits similar trends as the Radius Visualization data. Low and High density groups can be observed on both graphs in neighboring coordinates.

```
import matplotlib.pyplot as plt
import math

def loadData(filename):
    crime_data = open(filename)
    next(crime_data)

    coords = [line.split(',') for line in crime_data]
    crimeLocs = [(float(coord[0]), float(coord[1][:-1])) for coord in coords]
    lat = [float(coord[0]) for coord in coords]
    long = [float(coord[1][:-1]) for coord in coords]

    return (lat, long, crimeLocs)

class Map(object):
    def __init__(self, lat, long, numZones):
        "Assumes lat and long ar lists."
        self.lat = lat
        self.long = long
        self.numZones = numZones
        self.zones = []
        self.scalar = 100000
        self.hits = {}

    def generate_zones(self):
        latRange = self.get_lat_range()
        longRange = self.get_long_range()
        latStep = round(len(latRange)/float(self.numZones))
        longStep = round(len(longRange)/float(self.numZones))

        for i in  range(latRange[0], latRange[-1], latStep):
            for j in range(longRange[0], longRange[-1], longStep):
                self.zones.append(((i, i + latStep), (j,j + longStep)))

    def get_lat_range(self):
        return range(round(self.scalar*(min(self.lat) - 0.05)), round(self.scalar*(max(self.lat) + 0.05)))

    def get_long_range(self):
        return range(round(self.scalar*(min(self.long) - 0.05)), round(self.scalar*(max(self.long) + 0.05)))

    def is_crime_in_zone(self, crimeLoc, zoneNum):
        if math.floor(self.scalar*crimeLoc[0]) in range(self.zones[zoneNum][0][0],self.zones[zoneNum][0][1]) and math.floor(sel
            if zoneNum not in self.hits.keys():
                self.hits[zoneNum] = [crimeLoc]
            else:
                self.hits[zoneNum].append(crimeLoc)

    def report_crimes(self, crimeLocs):
        for crimeLoc in crimeLocs:
            for zoneNum in range(self.numZones**2):
                self.is_crime_in_zone(crimeLoc, zoneNum)

def style(numCrimes, mean):
    if numCrimes > (2/3)*mean:
        return ('ro', 'High Crime Density')
    elif numCrimes > mean/3:
        return ('yo', 'Medium Crime Density')
    else:
        return ('go', 'Low Crime Density')

def plotCrimes(Map):
    myMap = Map
    mean = 90
    plt.figure('Chicago Crime Heat Map')
    for area in myMap.hits.values():
        numCrimes = len(area)
        styles = style(numCrimes, mean)
        lats = [coord[0] for coord in area]
        longs = [coord[1] for coord in area]
        plt.plot(longs, lats, styles[0], label = styles[1], alpha = .1)
    plt.xlabel('Longitud')
    plt.ylabel('Latitud')
    plt.title('Chicago Crime Heat Map')
```

All Streets with +50 Vehicular Crimes in the past year

{'S PAULINA ST': 53, 'S SANGAMON ST': 52, 'W MADISON ST': 72, 'S INDIANA AVE': 84, 'W JACKSON BLVD': 60, 'W MONROE ST': 64, 'S WENTWORTH AVE': 59, 'S STATE ST': 103, 'S PULASKI RD': 52, 'S WABASH AVE': 73, 'S CALUMET AVE': 64, 'W WASHINGTON BLVD': 61, 'S MICHIGAN AVE': 141, 'S LOWE AVE': 59, 'S DR MARTIN LUTHER KING JR DR': 88, 'S ASHLAND AVE': 74, 'S COTTAGE GROVE AVE': 54, 'S HALSTED ST': 83, 'S CARPENTER ST': 51, 'S MORGAN ST': 63, 'W ADAMS ST': 60, 'S PRAIRIE AVE': 73, 'S LAFLIN ST': 51}




```
"""finds all crime related to that category"""
#use 'VEHICLE' for chicago
def crime_source_finder(loc,category):
    field_list = loc_to_info[loc][0]
    source = loc_to_info[loc][2]
    category_colnums = loc_to_info[loc][3]
    categorical_crimes = []
    with open(source) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')
        line_count = 0
        found = False
        for row in csv_reader:
            column_count = 0
            row_info = []
            for colnum in category_colnums:
                if category in row[colnum]:
                    found = True
            if found == True:
                for colnum in category_colnums:
                    if 'VIN' in row[colnum] or 'REGISTRATION' in row[colnum] or 'MOTOR' in row[colnum] or 'TITLE' in row[colnum]:
                        found = False
            if found == True:
                for column in row:
                    if column_count in field_list:
                        row_info.append(column)
                    column_count += 1
                categorical_crimes.append(row_info)
            found = False
        line_count += 1
    return categorical_crimes
```

*code used is very similar to on used to tally up deer data

Future Considerations

- Code wise:
  - Consolidate all code into single language (all python ideally)
  - Filter data more accurately
  - Clustering
- Generally:
  - Seems to have a company(Location Inc) that has consolidated a lot of these police reports (the source for the neighborhoodscout website)
  - Collaborate with existing companies that have compiled data
  - Gentrification can cause changes in larger cities
  - Some are hard lines whereas others blur into more dangerous
  - Need to be careful with existing maps because some may highlight a lot of crime but may be more petty theft (i.e. Cambridge because college towns)

Summary

Review

- Animal Crossings
  - Equation
  - Mapped animal collisions
  - Filtered data to collisions
- Crime and Unsafe Conditions
  - Mapped out relevant vehicle crimes using data
  - Correctly correlates to safe and dangerous neighborhoods
  - Consolidate more data
- We’re excited to join you for the summer!


We're so excited to join you in the summer!!