Analysis

December 7, 2018

1 Group Assignment Analysis — Group 8

This file contains the analysis for all group assignments regarding Ranson (2014) on Climate and Crime. All parts are in sequential order, accompanied by code, explanation, and visualization.

1.1 Environment

Environment configuration files are located within the folder named assignmentX. Please use the environment file from the latest assignment folder to run this notebook.

1.2 Miscellaneous Information

- The service provided by Nominatim is sometimes unstable. If you encountered network-related errors such as "geopy.exc.GeocoderTimedOut: Service timed out", try running the same code again.
- The implementation for parts uses multi-threading to improve performance. The names of these multi-threading functions will end with _parallel. If your computer does not support that, you should be able to find the non-multithreading version of the function under the same name (without _parallel).
- We used google map for several visualization tasks. Please do not release the API key to any third party or to the public.

1.3 Setup

In [1]:
```
%reload_ext autoreload
%autoreload 2
import sys
import gmaps
import pandas as pd
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt
from matplotlib.colors import to_hex
gmaps.configure(api_key='AIzaSyBTHDv-ei4m_2CqFFHEBwGx_rg27q1L3aA')
np.random.seed(100)
pd.set_option('display.max_rows', 20)

if sys.version_info[0] < 3: raise Exception("Must be using Python 3")
```

---

2 Group Assignment 1

- Write, document, and test code that takes a collection of values, (lat, long) pairs intended to represent weather stations, and finds the inverse-distance weighted average value to another given set of (lat, long) points intended to represent grid points within a county. This is to replicate Ranson's calculation of the daily temperature in a county. The code should do something sensible if any distance is zero.

2.1 Analysis

For this assignment, we first created a set of random grid points and 20 random stations to compute the average inverse distance. We then follow the same procedure mentioned in the paper to compute the weight for each station.

In addition, in order to handle the issues caused by any potential zero distances, we added a lower bound of MIN_DISTANCE = 1e-6 for all distance values. This means for any distance smaller than this value, we will use 1e-6 as the distance during computation.

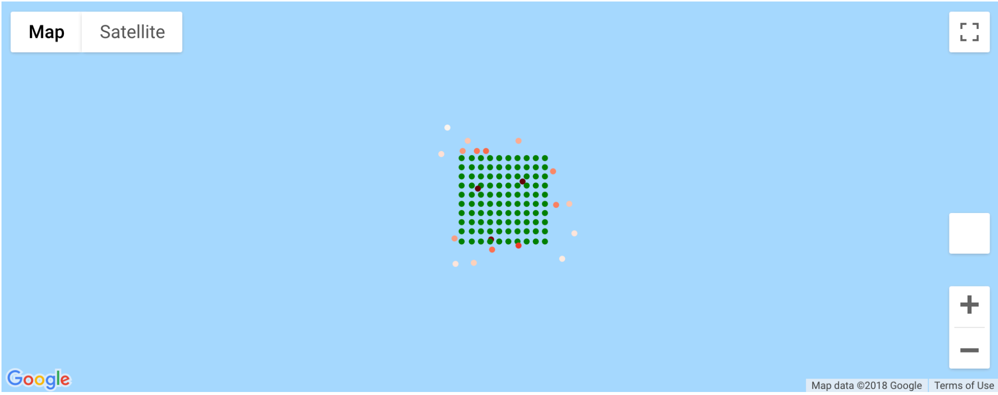
```
In [2]: from assignment1.code import *
```

```
In [3]: # Generate random grid and random stations
        grid       = generate_random_grid()
        stations   = generate_random_stations(20)

        # Compute distance
        distances = compute_average_inverse_distance(grid, stations)
```

```
In [4]: pd.Series(distances)
```

```
Out[4]: 0      0.045195
        1      0.028332
        2      0.018247
        3      0.028001
        4      0.033993
        5      0.021859
        6      0.025622
        7      0.018226
        8      0.026402
        9      0.029401
        10     0.021768
        11     0.030442
        12     0.043982
        13     0.020856
        14     0.018939
        15     0.045502
```



map

```
16    0.029809
17    0.017297
18    0.015659
19    0.024313
dtype: float64
```

The following diagram is a visualization of the result from the code above. All grid points are shown in green and all stations are shown in red. The color of the stations depends on their average inverse distance. The darker the color is, the closer the stations are to the grid.

```
In [5]: # Visualize the result
        fig = gmaps.figure()
        fig.add_layer(gmaps.symbol_layer(grid, fill_color='green',
                                         stroke_color='green', scale=2))
        station_colors = list(map(lambda distance: to_hex(cm.Reds(
            (distance-min(distances))/(max(distances)-min(distances))),
            keep_alpha=False), distances))
        fig.add_layer(gmaps.symbol_layer(stations, fill_color=station_colors,
                                         stroke_color=station_colors, scale=2))
        fig
```

```
Figure(layout=FigureLayout(height='420px'))
```

---

3 Group Assignment 2

- Construct a grid of (lat, long) points within Alameda county, separated by approximately 5 miles. The first point should be at (37.905098, -122.272225), near Summit Reservoir.
- Write code to identify all weather stations within x miles of Alameda County.
- Identify all weather stations within 10 miles (_not_ Ranson's 50 miles) of any of the grid points in Alameda county, and find the weighted average inverse distance from each station

to the points in the county grid. Your code for finding the stations should take the distance range as an input parameter (i.e., your code should let you find all stations within 5 miles or 50 miles, too).

3.1 Analysis

```
In [6]: from assignment2.code import *
```

The first task in the Assignment is to generate a grid of latitude and longitude coordinates of 5 mile spacing over Alameda County, beginning at the specified start point at Summir Reservoir.

We began by finding the rectangular bounding box of the county by using the “boundingbox” key in geopy’s location dictionary. Then, we create a grid of candidate points originating from Summit Reservoir that extend to a rectangular bounding box of the county. We ensured that each pair of vertically and horizontally adjacent points in the candidate grid are exactly 5 miles from each other using ulitity functions that we wrote, which use Newton’s method to find solutions of any given function. Next, we filtered the candidate grid based on whether each candidate point was within Alameda. This was easily done using the “county” key in one of geopy’s location dictionaries.

```
In [7]: # Generate all points within the rectangle bound of the county
        grid = generate_grid_within_rectangle_bounds("Alameda County",
                                                      start_point=(37.905098, -122.272225),
                                                      spacing=5)

        # Filter out points that are actually outside the county
        grid = filter_grid_parallel(grid, "Alameda County")
```

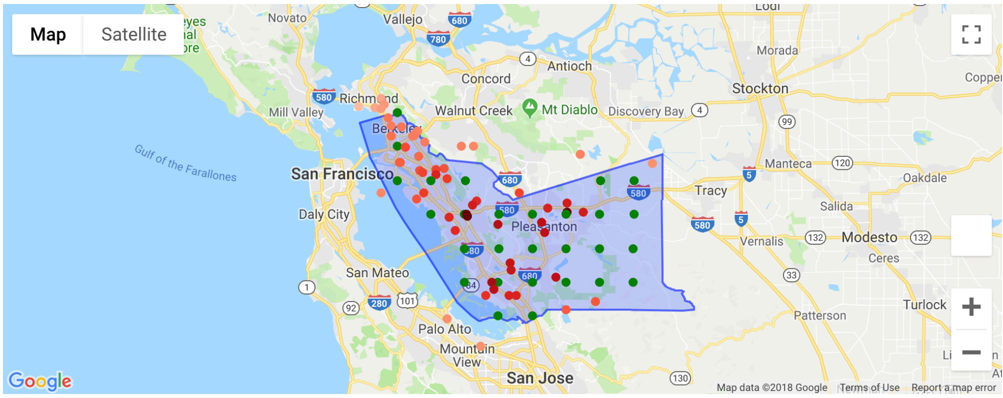
Next, we loaded the weather station station data from the stations_ca.csv file and wrote a function that would help in determining which stations would be valid stations. We iteratively computed the distance from all stations to all points in the grid and kept the station if it is within 10 miles from any grid point.

```
In [8]: # Load all stations
        stations_raw = load_stations()

        # Find all weather stations within 10 miles from any grid point of Alameda County
        stations = filter_stations_parallel(stations_raw, grid, 10)

        # Compute average inverse distance
        distances = compute_average_inverse_distance(
            grid, convert_stations_to_coordinates(stations))
```

The following diagram provided a visual representation of Alameda’s grid points in green and valid weather stations in varying shades of red. The darker the shade of red, the more central the weather station is to the entire county. In addition, the geographical border of Alameda County is shown in blue.



map

```
In [9]: # Visualize the result
        fig = gmaps.figure()
        # Plot all points in the grid
        fig.add_layer(gmaps.symbol_layer(grid, fill_color='green',
                                         stroke_color='green', scale=3))
        # Plot all stations, colored by weighted average distance
        colors = list(map(lambda distance: to_hex(cm.Reds(
            (distance-sorted(distances)[1])/(sorted(distances)[-2]-sorted(distances)[1])+0.3),
                                                      keep_alpha=False), distances))
        fig.add_layer(gmaps.symbol_layer(convert_stations_to_coordinates(stations),
                                         fill_color=colors, stroke_color=colors, scale=3))
        # Plot the border of the county
        fig.add_layer(gmaps.drawing_layer(
            features=[gmaps.Polygon(get_county_polygon_border("Alameda County"),
                                    stroke_color='blue', fill_color='blue')],
            show_controls=False
        ))
        fig
```

```
Figure(layout=FigureLayout(height='420px'))
```

3.1.1 Extra Credit - Compute distance of stations using polygon border

The following part is for extra credit and computed distances of stations using the polygon border of Alameda County.

First, we excluded many stations in California on the basis that they were not in the rectangle bounding box used in step 1. Then, we excluded the rest by computing the polygon boarder for Alameda County and checked whether a station was within the polygon or within 5 miles from it. Notice that this interpretation of being within 5 miles differs from that of Ranson, who determined that a station was within x miles if it was within x milea of a gridpoint.

We retrieve the polygon border using the "polygon_geojson" parameter from Nominatim (see Nominatim's wiki for more information). The polygon border is given as a list of turning-point coordinates on the polygon border. We connected these points into lines and checked if each station is either within the polygon or is within 5 miles from any of these border lines. The algorithm has the following two parts:

- **Check if a point is within the polygon**: For any point within a polygon, if we draw a line toward arbitary direction, it should cut the border line an odd number of times. Therefore, in our program, we try to draw a line rightwards for each station and count the number of times that its intersection with the border line is on the border lne segment (i.e. cuts the border line).
- **Check if a point is 5 miles aways from any border line segment**: For any point within 5 miles from a border line, if we try to draw a circle of radius 5 miles centered at the given point, the circle should cut the border line segment at least once. The algorithm will first check if the two endpoints of the border line are within the circle by computing their distances; if not, the algorithm will try to find a perpendicular line from the given point to the border line and compute the distance to check if it is less than or equal to 5 miles.

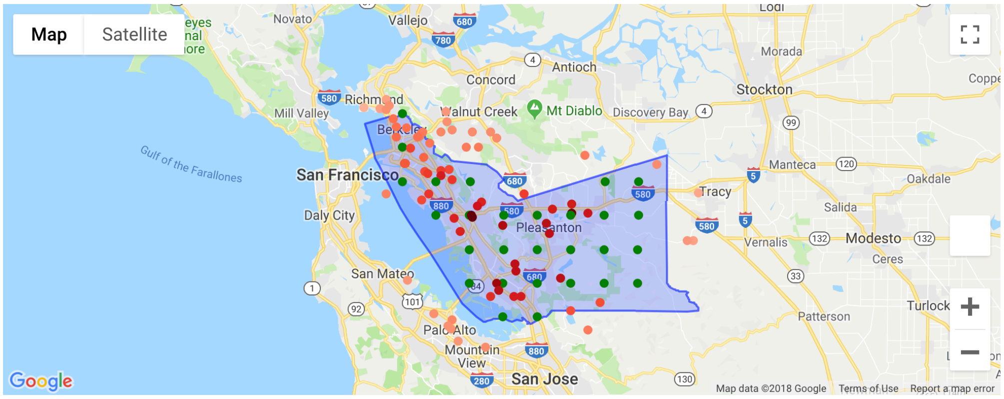
All valid stations should satisfy either of the two conditions above. Once we obtained the list of valid weather stations, we use the function written in Assignment 1 called compute_average_inverse_distance to find the weight of the weather stations with different distances from Alameda County.

In [10]:
```
# Find all weather stations within 10 miles from polygon border of Alameda County
stations_ec  = filter_stations_ec_parallel(stations_raw, "Alameda County", 5)

# Compute average inverse distance
distances_ec = compute_average_inverse_distance(
    grid, convert_stations_to_coordinates(stations_ec))
```

The following diagram provided a visual representation of Alameda's grid points in green and valid weather stations in varying shades of red. The darker the shade of red, the more central the weather station is to the entire county. In addition, the geographical border of Alameda County is shown in blue.

In [11]:
```
# Visualize the result
fig = gmaps.figure()
# Plot all points in the grid
fig.add_layer(gmaps.symbol_layer(grid, fill_color='green',
                                 stroke_color='green', scale=3))
# Plot all stations, colored by weighted average distance
colors = list(map(lambda distance: to_hex(cm.Reds(
    (distance-sorted(distances_ec)[1])/(sorted(distances_ec)[-2]-sorted(distances_ec)
                                          keep_alpha=False), distances_ec))
fig.add_layer(gmaps.symbol_layer(convert_stations_to_coordinates(stations_ec),
                                 fill_color=colors, stroke_color=colors, scale=3))
# Plot the border of the county
fig.add_layer(gmaps.drawing_layer(
    features=[gmaps.Polygon(get_county_polygon_border("Alameda County"),
                            stroke_color='blue', fill_color='blue')],
```



map

```
        show_controls=False
    ))
    fig
```

```
Figure(layout=FigureLayout(height='420px'))
```

---

4 Group Assignment 3

- retrieve the weather data for the relevant time periods for stations within 10 miles of any grid point in Alameda County
- identify the stations that meet Ranson's criteria for inclusion in each year
- calculate the "bias" adjustment for each weather station and for the county
- bin the averaged adjusted temperature data, aggregate it by month using the categories Ranson used

---

4.1 Analysis

The first part of this assignment was to retrieve the weather data for relevant time periods (1980 to 2009) for all stations within 10 miles of Alameda County. We loaded this data from the given file and retrieved that which was relevant, including temperature and precipitation data. We converted both of these to the appropriate units, Fahrenheit and mm, respectively.

In [12]:
```python
from assignment3.code import *
```

In [13]:
```python
# Retrieve the weather data for the relevant time periods
weather_data       = load_weather_data(stations, start_year=1980, end_year=2009)

# Extract temperature and precipitation data, and convert to corresponding units
temperature_data   = get_temperature_data(weather_data)
precipitation_data = get_precipitation_data(weather_data)
```

The weather data from each station are taken from GHCN Daily. Ranson placed complete trust in their data filtering criteria, which included checks for duplicated months and unrealistic streaks of a constant temperature. Instead of repeating this filtration, it will suffice to perform a particular sanity check. Our function filter_weather_data checks that for any given temperature report, the reported maximum is greater than the reported minimum. If this condition is not met by any maximum and minimum pair, we then deduce that one or both of the data values must be wrong and remove both of them. We have found that a few max-min pairs were removed under this check.

In [14]:
```
# Filter weather data asserting that reported maximum temperature must be greater tha
# minimum temperature for all day
weather_data = filter_weather_data(weather_data)
```

In [15]:
```
# Identify the stations that meet Ranson's criteria for inclusion in each year
valid_stations_id  = find_valid_stations_id_each_year(weather_data)

pd.Series(valid_stations_id)
```

Out[15]:
```
1980    {USC00047414, USW00023244, USC00046336, USW000...
1981    {USC00047414, USW00023244, USC00046336, USC000...
1982    {USC00047414, USW00023244, USC00046336, USC000...
1983    {USC00047414, USW00023244, USC00046336, USW000...
1984    {USC00047414, USW00023244, USC00046336, USW000...
1985    {USC00047414, USW00023244, USC00046336, USW000...
1986    {USC00047414, USW00023244, USC00046336, USW000...
1987    {USC00047414, USW00023244, USC00046336, USC000...
1988    {USC00047414, USW00023244, USC00046336, USC000...
1989    {USC00047414, USW00023244, USC00046336, USC000...
                              ...
2000    {USC00047414, USW00023285, USR0000COKS, USC000...
2001    {USC00047414, USW00023285, USR0000COKS, USC000...
2002    {USC00047414, USW00023285, USR0000COKS, USC000...
2003    {USC00047414, USW00023285, USR0000COKS, USC000...
2004    {USC00047414, USW00023285, USR0000COKS, USC000...
2005    {USC00047414, USW00023285, USR0000COKS, USC000...
2006    {USC00047414, USW00023285, USR0000COKS, USC000...
2007    {USC00047414, USW00023285, USR0000COKS, USC000...
2008    {USC00047414, US1CAAL0001, USW00023285, USR000...
2009    {USC00047414, US1CAAL0001, USW00023285, USR000...
Length: 30, dtype: object
```

Next, we calculated and applied the “bias” adjustment for each weather station for both temperature and precipitation data.

In [16]:
```
# Calculate the "bias" adjustment for each weather station
temperature_bias    = compute_station_bias(temperature_data)
precipitation_bias = compute_station_bias(precipitation_data)

# Apply the "bias" adjustment
```

```
temperature_data   = apply_station_bias(temperature_data, temperature_bias)
precipitation_data = apply_station_bias(precipitation_data, precipitation_bias)
```

We then computed the corresponding inverse distance weights for each station.

```
In [17]: station_weights = get_station_weights(stations, distances)
```

The next step we took was to bin and aggregate the data. We did this by constructing 11 bins (representing the 11 different 10-degree temperature ranges) and having each bin contain the per-month count for the relevant year (01/1980 to 12/2009). For a given bin and year-month, there is a count for the number of days in the month that belong to that particular bin. The data is aggregated for temperature and precipitation by weighting the data for each relevant station of that particular bin and year-month combination by using each station's inverse distance weight and then taking the average of all of those weighted values. These averages are then put into their corresponding bins and represent the distribution of counts across bins by year-month.

```
In [18]: # Each element in the list represents a bin
         # Each bin is a list of counts on a particular month of a particular year

         # Bins for temperature
         bins_count_for_T = aggregate_temperature_data(temperature_data, station_weights)

         # Bins for precipitation
         bins_count_for_P = aggregate_precipitation_data(precipitation_data, station_weights)
```

The following tables show the distribution for the bin counts for temperature and precipitation, respectively:

```
In [19]: pd.DataFrame(temperature_data).ID.unique()

Out[19]: array(['USR0000CLVR', 'USC00049001', 'USC00047414', 'USC00043244',
                'USW00093228', 'USC00040693', 'USR0000CMLR', 'USR0000CTRA',
                'USC00047661', 'USC00044997', 'USW00023230', 'USC00049185',
                'USR0000COKN', 'USW00023239', 'USR0000COKS', 'USW00023244',
                'USC00046144', 'USW00023285', 'USR0000CRSP', 'USC00046336',
                'USR0000CCLV'], dtype=object)

In [20]: # Display temperature bins
         month_year = [str(year) + '-' + str(month) for year in range(1980,2010) for month in i
         df = pd.DataFrame(bins_count_for_T)
         df = df.transpose()
         df.loc[:,'Year_Month'] = month_year
         df.set_index('Year_Month', inplace = True)
         df.columns = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70-80',
         df

Out[20]:             0-10  10-20  20-30  30-40  40-50  50-60  60-70  70-80  80-90  \
         Year_Month
         1980-1         0      0      0      0      0     26      5      0      0
```

```
1980-2          0     0     0     0     0     4    25     0     0
1980-3          0     0     0     0     0     2    26     3     0
1980-4          0     0     0     0     0     4    18     7     1
1980-5          0     0     0     0     0     0    23     8     0
1980-6          0     0     0     0     0     0    14    13     3
1980-7          0     0     0     0     0     0     5    16    10
1980-8          0     0     0     0     0     0     2    27     1
1980-9          0     0     0     0     0     0     3    21     5
1980-10         0     0     0     0     0     0     8    16     4
...           ...   ...   ...   ...   ...   ...   ...   ...   ...
2009-3          0     0     0     0     0    15    13     3     0
2009-4          0     0     0     0     0    10    12     4     2
2009-5          0     0     0     0     0     1     7    20     1
2009-6          0     0     0     0     0     0    14     7     7
2009-7          0     0     0     0     0     0     1    20     8
2009-8          0     0     0     0     0     0     0    13    15
2009-9          0     0     0     0     0     0     1    11    11
2009-10         0     0     0     0     0     0    18    12     1
2009-11         0     0     0     0     0     8    19     3     0
2009-12         0     0     0     0     5    24     2     0     0

            90-100  100-110
Year_Month
1980-1           0        0
1980-2           0        0
1980-3           0        0
1980-4           0        0
1980-5           0        0
1980-6           0        0
1980-7           0        0
1980-8           1        0
1980-9           1        0
1980-10          3        0
...            ...      ...
2009-3           0        0
2009-4           2        0
2009-5           2        0
2009-6           2        0
2009-7           2        0
2009-8           3        0
2009-9           7        0
2009-10          0        0
2009-11          0        0
2009-12          0        0

[360 rows x 11 columns]
```

In [21]:
```
# Display precipitation bins
```

```
df = pd.DataFrame(bins_count_for_P)
df = df.transpose()
df.loc[:,'Year_Month'] = month_year
df.set_index('Year_Month', inplace = True)
df.columns = ['<1', '1-5', '5-15', '15-29', '>29']
df
```

Out[21]:

```
            <1  1-5  5-15  15-29  >29
Year_Month
1980-1       0    0    21     10    0
1980-2       0    0     2     27    0
1980-3       0    0     2     29    0
1980-4       0    0     1     29    0
1980-5       0    0     0     31    0
1980-6       0    0     0     28    2
1980-7       0    0     0     24    7
1980-8       0    0     0     30    1
1980-9       0    0     0     26    4
1980-10      0    0     0     26    5
...         ..  ...   ...    ...  ...
2009-3       0    0    13     18    0
2009-4       0    0     8     20    2
2009-5       0    0     1     28    2
2009-6       0    0     0     26    4
2009-7       0    0     0     22    9
2009-8       0    0     0     22    9
2009-9       0    0     0     18   12
2009-10      0    0     0     31    0
2009-11      0    0     7     23    0
2009-12      0    0    28      3    0

[360 rows x 5 columns]
```

---

5 Group Assignment 4

- split Alameda county into two pieces along the eastern edges of zipcodes 94552 and 94539. Consider all zipcodes within Alameda county that are in or west of either of those zipcodes to be West Alameda and all zipcodes in Alameda that are east of those two zipcodes to be East Alameda. Repeat what you did in group assignments (2) and (3) for East Alameda and West Alameda separately (but using the same grid of points–the original gridpoints in Alameda that are in East Alameda form the grid for East Alameda, and the original gridpoints in Alameda that are in West Alameda form the grid for West Alameda).

5.1 Analysis

5.1.1 Split grid

For this assignment, we first generated the grid of Alameda County, making sure to filter out all points outside the county boundary. We then split the grid into two along the eastern edges of the zipcodes 94552 and 94539, so that all points are either in West or East Alameda.

```
In [22]: from assignment4.code import *
```

```
In [23]: # Generate all points within the rectangle bound of the county
         grid = generate_grid_within_rectangle_bounds("Alameda County",
                                                       start_point=(37.905098, -122.272225),
                                                       spacing=5)

         # Filter out points that are actually outside the county
         grid = filter_grid_parallel(grid, "Alameda County")
```

```
In [24]: east_zipcodes = [94568, 94588, 94566, 94550, 94586, 94551, 94514]
         east_grid, west_grid = split_grid(grid, east_zipcodes)
```

The following is a visualization of this grid split, with West Alameda in red and East Alameda in green.

```
In [25]: # Visualize the result
         fig = gmaps.figure()
         # Plot all points in the grid
         fig.add_layer(gmaps.symbol_layer(east_grid, fill_color='green', stroke_color='green',
         fig.add_layer(gmaps.symbol_layer(west_grid, fill_color='red', stroke_color='red', scal
         # Plot the border of the county
         fig.add_layer(gmaps.drawing_layer(
             features=[gmaps.Polygon(get_county_polygon_border("Alameda County"), stroke_color=
             show_controls=False
         ))
         fig
```

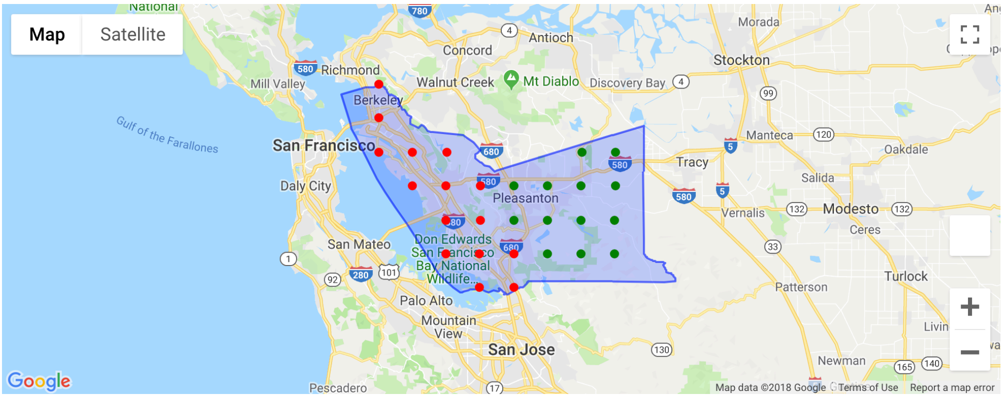
```
Figure(layout=FigureLayout(height='420px'))
```

5.1.2 Load Stations

We loaded the Alameda County weather station data and determined which stations are in West Alameda and which stations are in East Alameda. For those stations, we computed the average inverse distances.

```
In [26]: # Load all stations
         stations_raw = load_stations()

         # Find all weather stations within East Alameda
```



map

```
east_stations = filter_stations_parallel(stations_raw, east_grid, 10)

# Find all weather stations within West Alameda
west_stations = filter_stations_parallel(stations_raw, west_grid, 10)
```

```
In [27]: # Compute average inverse distance
         east_distances = compute_average_inverse_distance(
             east_grid, convert_stations_to_coordinates(east_stations))

         west_distances = compute_average_inverse_distance(
             west_grid, convert_stations_to_coordinates(west_stations))
```

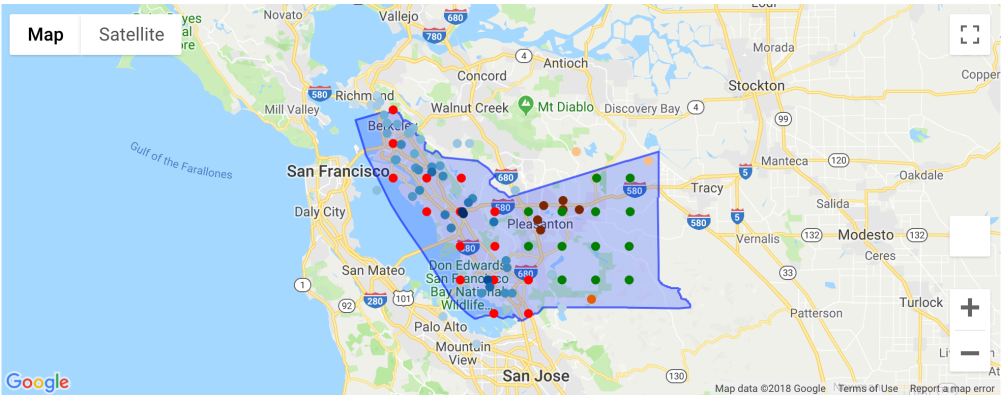
The following is a visualization of the stations in Alameda County based on whether they are in West or East Alameda, overlayed on the West and East Alameda grid split from the previous visualization. Here, the weather stations in West Alameda are shown in blue and those in East Alameda are shown in maroon.

```
In [28]: # Visualize the result
         fig = gmaps.figure()

         # Plot all points in the grid
         fig.add_layer(gmaps.symbol_layer(east_grid, fill_color='green', stroke_color='green',
         fig.add_layer(gmaps.symbol_layer(west_grid, fill_color='red', stroke_color='red', scal

         # Plot all stations, colored by weighted average distance
         east_colors = list(map(lambda distance: to_hex(cm.Oranges(
         (distance-sorted(east_distances)[1])/(sorted(east_distances)[-2]-sorted(east_distance
                                              keep_alpha=False), east_distances))
         fig.add_layer(gmaps.symbol_layer(convert_stations_to_coordinates(east_stations),
                                     fill_color=east_colors, stroke_color=east_colors, scale=

         west_colors = list(map(lambda distance: to_hex(cm.Blues(
             (distance-sorted(west_distances)[1])/(sorted(west_distances)[-2]-sorted(west_dist
                                                  keep_alpha=False), west_distances))
```



map

```
fig.add_layer(gmaps.symbol_layer(convert_stations_to_coordinates(west_stations),
                                 fill_color=west_colors, stroke_color=west_colors, sca
# Plot the border of the county
fig.add_layer(gmaps.drawing_layer(
    features=[gmaps.Polygon(get_county_polygon_border("Alameda County"),
                            stroke_color='blue', fill_color='blue')],
    show_controls=False
))
fig
```

```
Figure(layout=FigureLayout(height='420px'))
```

5.1.3 Retrieve weather data

We retrieved the weather data for both West and East Alameda for the relevant time periods (1980 to 2009). This weather data includes temperature and precipitation data.

In [29]:
```
# Retrieve the weather data of east stations for the relevant time periods
east_weather_data         = load_weather_data(east_stations, start_year=1980, end_year=
east_temperature_data   = get_temperature_data(east_weather_data)
east_precipitation_data = get_precipitation_data(east_weather_data)
```

In [30]:
```
# Retrieve the weather data of west stations for the relevant time periods
west_weather_data         = load_weather_data(west_stations, start_year=1980, end_year=
west_temperature_data   = get_temperature_data(west_weather_data)
west_precipitation_data = get_precipitation_data(west_weather_data)
```

5.1.4 Filter by Ranson's criteria

As explained by our analysis for Assignment3, we filtered the weather data for the west and east stations based on Ranson's criteria for inclusion.

In [31]:
```python
# Filter east weather data based on Ranson's criteria for inclusion
east_weather_data = filter_weather_data(east_weather_data)

# Identify the east stations that meet Ranson's criteria for inclusion in each year
east_valid_stations_id  = find_valid_stations_id_each_year(east_weather_data)

pd.Series(east_valid_stations_id)
```

Out[31]:
```
1980                              {USC00044997, USC00049001}
1981                              {USC00044997, USC00049001}
1982                              {USC00044997, USC00049001}
1983                              {USC00044997, USC00049001}
1984                              {USC00044997, USC00049001}
1985                              {USC00044997, USC00049001}
1986                              {USC00044997, USC00049001}
1987                              {USC00044997, USC00049001}
1988                              {USC00044997, USC00049001}
1989                              {USC00044997, USC00049001}
                                 ...
2000    {USR0000CLVR, USC00049001, USC00044997, USW000...
2001    {USR0000CLVR, USC00049001, USC00044997, USW000...
2002    {USR0000CLVR, USC00049001, USC00044997, USW000...
2003    {USR0000CLVR, USC00049001, USC00044997, USW000...
2004    {USR0000CLVR, USC00049001, USC00044997, USW000...
2005    {USR0000CLVR, USC00049001, USC00044997, USW000...
2006    {USR0000CLVR, USC00049001, USC00044997, USW000...
2007    {USR0000CLVR, USC00049001, USC00044997, USW000...
2008    {US1CAAL0004, USC00049001, USC00044997, USW000...
2009    {US1CAAL0004, USC00049001, USC00044997, USW000...
Length: 30, dtype: object
```

In [32]:
```python
# Filter west weather data based on Ranson's criteria for inclusion
west_weather_data = filter_weather_data(west_weather_data)

# Identify the west stations that meet Ranson's criteria for inclusion in each year
west_valid_stations_id  = find_valid_stations_id_each_year(west_weather_data)

pd.Series(west_valid_stations_id)
```

Out[32]:
```
1980    {USC00047414, USW00023244, USC00046336, USW000...
1981    {USC00047414, USW00023244, USC00046336, USC000...
1982    {USC00047414, USW00023244, USC00046336, USC000...
1983    {USC00047414, USW00023244, USC00046336, USW000...
1984    {USC00047414, USW00023244, USC00046336, USW000...
1985    {USC00047414, USW00023244, USC00046336, USW000...
1986    {USC00047414, USW00023244, USC00046336, USW000...
1987    {USC00047414, USW00023244, USC00046336, USC000...
1988    {USC00047414, USW00023244, USC00046336, USC000...
```

```
1989    {USC00047414, USW00023244, USC00046336, USC000...
                              ...
2000    {USC00047414, USW00023244, USC00046336, USR000...
2001    {USC00047414, USW00023244, USC00046336, USR000...
2002    {USC00047414, USW00023244, USC00046336, USR000...
2003    {USC00047414, USW00023244, USC00046336, USR000...
2004    {USC00047414, USW00023244, USC00046336, USR000...
2005    {USC00047414, USW00023244, USC00046336, USR000...
2006    {USC00047414, USW00023244, USC00046336, USR000...
2007    {USC00047414, USW00023244, USC00046336, USR000...
2008    {USC00047414, US1CAAL0001, USW00023244, USR000...
2009    {USC00047414, US1CAAL0001, USW00023244, USR000...
Length: 30, dtype: object
```

5.1.5 Compute Bias

We then calculated the “bias” adjustment for each weather station in West and East Alameda. This was done separately for temperature and precipitation for each of the two regions.

```
In [33]: # Calculate the "bias" adjustment for each weather station in the East Alameda.
         east_temperature_bias   = compute_station_bias(east_temperature_data)
         east_precipitation_bias = compute_station_bias(east_precipitation_data)

         # Apply the "bias" adjustment
         east_temperature_bias   = apply_station_bias(east_temperature_data, east_temperature_b
         east_precipitation_bias = apply_station_bias(east_precipitation_data, east_precipitati
```

```
In [34]: # Calculate the "bias" adjustment for each weather station in the West Alameda.
         west_temperature_bias   = compute_station_bias(west_temperature_data)
         west_precipitation_bias = compute_station_bias(west_precipitation_data)

         # Apply the "bias" adjustment
         west_temperature_bias   = apply_station_bias(west_temperature_data, west_temperature_b
         west_precipitation_bias = apply_station_bias(west_precipitation_data, west_precipitati
```

5.1.6 Get stations’ corresponding inverse weight

We calculated the inverse weights for the stations in both West and East Alameda.

```
In [35]: # Compute the inverse weight for both stations in East Alameda and those in West Alam
         east_station_weights = get_station_weights(east_stations, east_distances)
         west_station_weights = get_station_weights(west_stations, west_distances)
```

5.1.7 Bin and Aggregate Data

Our next step was to bin and aggregate the data, using the same procedure we explained in our Assignment3 analysis. The result is a number of bins containing counts for temperature and precipitation data in West Alameda and East Alameda organized by temperature and year-month.

```
In [36]: # Each element in the list represents a bin
         # Each bin is a list of counts on a particular month of a particular year

         # Bins for temperature
         east_bins_count_for_T = aggregate_temperature_data(east_temperature_data, east_statio

         # Bins for precipitation
         east_bins_count_for_P = aggregate_precipitation_data(east_precipitation_data, east_sta
```

```
In [37]: # Each element in the list represents a bin
         # Each bin is a list of counts on a particular month of a particular year

         # Bins for temperature
         west_bins_count_for_T = aggregate_temperature_data(west_temperature_data, west_statio

         # Bins for precipitation
         west_bins_count_for_P = aggregate_precipitation_data(west_precipitation_data, west_sta
```

5.1.8 Display Result

The following are tables displaying our results. The first table contains the bin counts for East Alameda temperature data. The second table is bin counts for West Alameda temperature data. The third displays East Alameda precipitation data bin counts. The final table is bin counts for West Alameda precipitation data.

```
In [38]: month_year = [str(year) + '-' + str(month) for year in range(1980,2010) for month in r

         # East temperature bins
         east_df_T = pd.DataFrame(east_bins_count_for_T)
         east_df_T = east_df_T.transpose()
         east_df_T.loc[:,'Year_Month'] = month_year
         east_df_T.set_index('Year_Month', inplace = True)
         east_df_T.columns = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70

         east_df_T.head()
```

```
Out[38]:            0-10  10-20  20-30  30-40  40-50  50-60  60-70  70-80  80-90  \
         Year_Month
         1980-1         0      0      0      0      8     20      3      0      0
         1980-2         0      0      0      0      0     20      9      0      0
         1980-3         0      0      0      0      0     11     19      1      0
         1980-4         0      0      0      0      0      6     13      8      3
         1980-5         0      0      0      0      0      2     13     12      3

                    90-100  100-110
         Year_Month
         1980-1          0        0
         1980-2          0        0
         1980-3          0        0
```

```
1980-4           0         0
1980-5           1         0
```

```
In [39]: # West temperature bins
         west_df_T = pd.DataFrame(west_bins_count_for_T)
         west_df_T = west_df_T.transpose()
         west_df_T.loc[:,'Year_Month'] = month_year
         west_df_T.set_index('Year_Month', inplace = True)
         west_df_T.columns = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70

         west_df_T.head()
```

```
Out[39]:             0-10  10-20  20-30  30-40  40-50  50-60  60-70  70-80  80-90  \
         Year_Month
         1980-1         0      0      0      0      0     22      9      0      0
         1980-2         0      0      0      0      0      3     26      0      0
         1980-3         0      0      0      0      0      2     26      3      0
         1980-4         0      0      0      0      0      2     22      5      1
         1980-5         0      0      0      0      0      2     24      5      0

                     90-100  100-110
         Year_Month
         1980-1           0        0
         1980-2           0        0
         1980-3           0        0
         1980-4           0        0
         1980-5           0        0
```

```
In [40]: # East precipitation bins
         east_df_P = pd.DataFrame(east_bins_count_for_P)
         east_df_P = east_df_P.transpose()
         east_df_P.loc[:,'Year_Month'] = month_year
         east_df_P.set_index('Year_Month', inplace = True)
         east_df_P.columns = ['<1', '1-5', '5-15', '15-29', '>29']

         east_df_P.head()
```

```
Out[40]:             <1  1-5  5-15  15-29  >29
         Year_Month
         1980-1       0    0    28      3    0
         1980-2       0    0    16     13    0
         1980-3       0    0     6     25    0
         1980-4       0    0     4     26    0
         1980-5       0    0     1     27    3
```

```
In [41]: # West precipitation bins
         west_df_P = pd.DataFrame(west_bins_count_for_P)
         west_df_P = west_df_P.transpose()
         west_df_P.loc[:,'Year_Month'] = month_year
```

```
west_df_P.set_index('Year_Month', inplace = True)
west_df_P.columns = ['<1', '1-5', '5-15', '15-29', '>29']

west_df_P.head()
```

Out[41]:

	<1	1-5	5-15	15-29	>29
Year_Month					
1980-1	0	0	22	9	0
1980-2	0	0	3	26	0
1980-3	0	0	2	29	0
1980-4	0	0	2	28	0
1980-5	0	0	1	30	0

---

6 Group Assignment 5

Consider weather data from the HCN Berkeley station (ID: USC00040693) and the HCN Livermore station (ID: USC00044997) for the time period covered by Ranson's work. + Bin the maximum temperature data, separately for the two stations, using the categories Ranson used. + Devise and implement a stratified permutation test for the hypothesis that the two cities have "the same weather." Formulate the hypothesis as a generalized *two-sample problem*, i.e., ask whether differences (between the cities) in the number of days of each month in which the maximum temperature is in each bin could reasonably be attributed to chance, if the maximum temperatures had been a single population of numbers randomly split across the two cities. - What did you stratify on? Why is that a good choice? Why stratify at all? - Combine results across strata using Fisher's combining function - Can you use the chi-square distribution to calibrate the test? Why or why not? - Discuss how to take into account simulation uncertainty in estimating the overall p-value. - Discuss what this means for Ranson's approach.

6.1 Approach

NullHypothesis: The temperatures in Oakland and Livermore are the same. That is, their maximum and minimum temperatures come from the same underlying distribution.

In order to test this, we must perform the following tasks:

1. Get the data from the stations for the days on which both stations report
2. Create 1000 permutations of these data and then for each permutation:
    - separate the data by month
    - for each month, bin the temperatures using the same bins as before

    - get a p-value for each month using a chi-square contingency table
    - combine the p-values using the Fisher combining function to get the Fisher combination statistic
3. Calculate the percentage of the permutation cases (including the unpermuted data) with a Fisher combination statistic greater than the Fisher combination statistic for the unpermuted data

6.2 Answers to the Other Questions

- What did you stratify on? Why is that a good choice?
  - We will stratify on month. This is a natural and sensible choice for stratification because Ranson binned and analyzed weather patterns according to month
- Why stratify at all?
  - In this context, stratification is a logical choice because we know that weather patterns change according to season. The weather is hotter in the Summer and colder in the Winter. Thus, pooling all temperature data together as if it were generated from a single distribution would be overlooking this general knowledge.
- Can you use the chi-square distribution to calibrate the test?
  - Yes, we can. The purpose and assumptions of a chi square are met because we will be working with categorical data. Additionally, added independent Chi Squares have additive degrees of freedom.
- Combine results across strata using Fisher's combining function
  - This function is not difficult to create at all. It simply requires implementing the following $f(\mathbf{x}) = -2\sum_{k=1}^{n} ln(x_k)$, where $\mathbf{x}$ is an n dimensional vector of p-values.

6.3 Analysis

6.3.1 Retrieve and bin maximum temperature data for Livermore and Oakland stations separately

```
In [42]: from assignment5.code import *
         pd.set_option('display.max_rows', 10)
```

```
In [43]: # Assign Station IDs
         liv_id, oak_id = 'USC00044997', 'USC00040693'

         # Retrieve the weather data for the relevant time periods
         weather_data = load_weather_data([{'ID': liv_id}, {'ID': oak_id}],
                                          start_year=1980, end_year=2009)

         # Extract and bin max temperature
         binned_temperature = bin_temperature_data(get_temperature_data(weather_data))

         # Get binned max temperature for each station
         liv_temp_daily = binned_temperature[liv_id]
         oak_temp_daily = binned_temperature[oak_id]
```

6.3.2 Sanity check

```
In [44]: # Assert the data covers every day from 1980 to 2009 (though some of them might be NA
         assert (len(set(liv_temp_daily.keys())) ==
           (pd.Timestamp(2009, 12, 31) - pd.Timestamp(1980, 1, 1)).days + 1)
```

```
assert (len(set(oak_temp_daily.keys())) ==
  (pd.Timestamp(2009, 12, 31) - pd.Timestamp(1980, 1, 1)).days + 1)
```

```
In [45]: # Find all days for which at least one of the stations does not have a reported tempe
         df = pd.DataFrame({'Livermore':pd.Series(liv_temp_daily),
                            'Oakland':pd.Series(oak_temp_daily)})
         df[df.isna().any(axis=1)]
```

```
Out[45]:          Livermore Oakland
         19800725       NaN   70-80
         19820806       NaN   80-90
         19821101     60-70     NaN
         19821102     60-70     NaN
         19821103     60-70     NaN
         ...            ...     ...
         20090124     50-60     NaN
         20090125     50-60     NaN
         20090126     50-60     NaN
         20090131     70-80     NaN
         20090201     60-70     NaN

         [1647 rows x 2 columns]
```

From the result above, we can see that there are 1647 out of 10958 total days from 1980 to 2019 for which at least one of the stations does not have a reported temperature value. These days will be removed in future analyses.

```
In [46]: liv_temp_daily, oak_temp_daily = remove_na_from_data(liv_temp_daily, oak_temp_daily)

         len(liv_temp_daily), len(oak_temp_daily)
```

```
Out[46]: (9311, 9311)
```

6.3.3 Stratified Permutation Test

Now that we have assembled the daily maximum temperatures and removed days on which any data is missing for either the Oakland weather station or the Livermore station, we will proceed with the test.

We will proceed with the test by computing the p-values associated with the chi-square contingency tables for each month. Then, we will combine the p-values into a single statistic using Fisher's combining function.

```
In [47]: # Aggregate into monthly data
         liv_temp_monthly = aggregate_bin_temperature_by_month(liv_temp_daily)
         oak_temp_monthly = aggregate_bin_temperature_by_month(oak_temp_daily)

         # Get the p-values for each month
         p_values = compute_p_values(liv_temp_monthly, oak_temp_monthly)
```

```
# Calculate the statistic associated with the Null Hypothesis using Fisher's combinin
original_stat = fisher_combine(p_values)
```

In [48]:
```
round(original_stat,2)
```

Out[48]:
```
4014.64
```

As displayed above, the statistic that is obtained through combining p-values of the monthy tests via Fisher's Combining function is 4014.64.

Next, using a cryptographically secure PRNG from the cryptorandom package, we wrote a function that will make random permutations of the Oakland station's data and the Livermore station's data. A permutation of the temperature data means swapping daily maximum temperatures in Livermore and Oakland on each day with probability $p = \frac{1}{2}$.

In [49]:
```
# Perform the permutation test

# Make permutations and get the Fisher Combination Statisitc for each permutation
stats = permutation_test(liv_temp_daily, oak_temp_daily, reps=1000)
```

In [50]:
```
# Identify all statistics that exceed the original Fisher statistic
stats = np.asarray(stats)
good = stats[stats > original_stat]
```

We take the p_value of the test to be
$\hat{p} = \frac{\#\left\{k \geq 0\colon F(\pi_k(x_0)) \geq F(\pi_0(x_0))\right\}}{n+1}$
where:
x_0: the original data
n: the number of permutations
pi_0: the identity permutation on the symmetry group
pi_k: a random permutation of the symmetry group
F: the function thaat computes p-values and returns Fisher's combining function statistic

In [51]:
```
p_value = float(1+len(good)) / (1+len(stats))
round(p_value,4)
```

Out[51]:
```
0.001
```

As displayed above, the p-value for the randomized test is far below the 5% significance level. Thus, the test strongly rejects the Null Hypothesis, and we can conclude that there are months for which the temperatures in Oakland and Livermore are significantly different.

Note: We abstain from attaching a 95% confidence bound on the p-value because it is unecessary. Indeed, the computation of the p-value above is motivated by the goal of approximating the number of permutations of the data that yield a Fisher combinataion statistic greater than or equal to the original Fisher combination statistic. However, it has been proven that $\hat{p}$ is a conservative p-value for a randomized test. Viewing each permutation (including the original set of data) as independent and identically distributed elements of the orbit of the original data enables this fine result. Thus, the p-value is a valid p-value for a randomized test and returns high significance.

What does this mean for Ranson's appraoch?

Ranson's approach and analysis were based on the underlying assumption that the weather for different cities within a particular county is the same. However, our p-value and output above show that for particular months, the weather for Oakland and Livermore, both cities within Alameda County, are significantly different. Given that we have found one of Ranson's underlying assumptions to be invalid, we cannot support the validity of Ranson's overall approach. Since we have found his approach to be invalid, the conclusions he makes should be taken with a grain of salt, as they are based on false assumptions.

---

7 Group Assignment 6

- Fit the Poisson regression model to the data for all of Alameda County, and for the two pieces of Alameda county separately. Fit the separate estimates simultaneously, including dummy variables for all of Alameda county (treat Alameda County as a whole the way Ranson treated states; East and West Alameda are the two counties in the State of Alameda).
  - **Hint.** If some covariate has the same value in both parts of Alameda in every month (e.g., the number of days with maximum temperature below 10F), do not include it in the model: the corresponding parameter is not identifiable, and the estimation problem will be unstable.
  - **Hint.** `statsmodels` has a GLM function similar to that of R, and has an R-style language for writing formulae
- Devise and perform a permutation test to check whether the two pieces of Alameda county are consistent with a single model.
  - Explain the particular randomization you are using, its assumptions, and your justification for using it as the null hypothesis
  - Try using a cryptographic quality PRNG to simulate random permutations; if you run into computational bottlenecks, it is OK to use Python's default PRNG instead.
  - **Hint.** One way to ask whether the relationship between weather and crime is different in the two parts of Alameda is to check whether fitting two separate models fits the crime data "surprisingly better" than if the relationship were the same everywhere. The complication is that fitting two models will always fit somewhat better than a single model, because there are more parameters. Consider some combined measure of the fit of the two models to their corresponding data, e.g., RMS error. That's the test statistic. Now, randomly split the data into two pieces. For instance, take the two sets of monthly binned weather data and crime data, and toss a fair coin to decide whether East Alameda gets its original data, or the data from West Alameda (West gets whichever East didn't get. The randomization needs to keep the weather and crime for a given month together, so we allocate each entire month of weather and crime data to one location or to the other.) Fit two models to these randomized data, and calculate the RMS error. Repeat many times. If the RMS error of the two models to the randomly assigned data is typically much larger than it is for the original data, that's evidence that the relationship between weather and crime is truly different in the two parts of the county.

7.1 Approach

We need a table of the following form:

$YearMonth$	$CrimeCount$	T^1	$\ldots$	T^{11}	P^1	$\ldots$	P^5
$1980,01$	$C_{1980,01}$	$T^1_{1980,01}$	$\ldots$	$T^{11}_{1980,01}$	$P^1_{1980,01}$	$\ldots$	$P^5_{1980,01}$
$\ldots$							
y,m	$C_{y,m}$	$T^1_{y,m}$	$\ldots$	$T^{11}_{1980,01}$	$P^1_{y,m}$	$\ldots$	$P^5_{y,m}$
$\ldots$							
$2009,12$	$C_{2009,12}$	$T^1_{2009,12}$	$\ldots$	$T^{11}_{2009,12}$	$P^1_{2009,12}$	$\ldots$	$P^5_{2009,12}$

Recall that there are 11 temperature bins and 5 precipitation bins.
$T^j_{y,m}$ is the number of days in month m of year y with temperature in bin j. $j = 1,2,\ldots,11$
$P^k_{y,m}$ is the number of days in month m of year y with precipitation in bin k $k = 1,2,\ldots,5$
We must convert the table above into a table of this form:

$YearMonth$	$CrimeCount$	T^{1^*}	$\ldots$	T^{11^*}	P^{1^*}	$\ldots$	P^{5^*}
$1980,01$	$C_{1980,01}$	$T^{1^*}_{1980,01}$	$\ldots$	$T^{11^*}_{1980,01}$	$P^{11}_{1980,01}$	$\ldots$	$P^{5^*}_{1980,01}$
$\ldots$							
y,m	$C_{y,m}$	$T^1_{y,m} + T^1_{y,m-1}$	$\ldots$	$T^{11}_{y,m} + T^{11}_{y,m-1}$	$P^1_{y,m} + P^1_{y,m-1}$	$\ldots$	$P^5_{y,m} + P^5_{y,m-1}$
$\ldots$							
$2009,12$	$C_{2009,12}$	$T^1_{2009,12} + T^1_{2009,11}$		$T^{11}_{2009,12} + T^{11}_{2009,11}$	$P^1_{2009,12} + P^1_{2009,11}$		$P^5_{2009,12} + P^5_{2009,11}$

Notice that $T_{y,m}^{j*} = T_{y,m}^{j} + T_{y,m-1}^{j}$, and $P_{y,m}^{k*} = P_{y,m}^{k} + P_{y,m-1}^{k}$

7.2 Poisson Regression

$$C_{y,m} \sim Poisson(X_{y,m})$$

where,

$$log(X_{y,m}) = \sum_{j=1}^{11} \alpha_j T^{j^*}_{y,m} + \sum_{k=1}^{5} \beta_k P^{k^*}_{y,m} + \theta_{cy} + \phi_{sm}$$

Note that θ_{cy} represents the county-by-year fixed effect of weather on crime, and ϕ_{sm} represents the state-by-month fixed effect of weather on crime.

We will estimate the Poisson Regression coefficients using Maximum Likelihood Estimation.

7.3 Answers to the Other Questions

- Explain the particular randomization you are using, its assumptions, and your justification for using it as the null hypothesis.
  - Because Ranson modeled crime and weather as being related by month, we will be using a by-month randomization method. Since we are assuming with our null hypothesis that the crime and weather data from East and West Alameda are consistent with a single model, if we randomly swap their data by month, there should not be a

significant impact on the test statistic RMS. This particular monthly randomization and the subsequent RMS calculations allow us to determine whether or not there is evidence that the relationship between crime and weather is truly different for East Alameda and West Alameda.

7.4 Analysis

```
In [52]: from assignment6.code import *
```

```
In [53]: # Load all crime data from file
         crime_data_df = load_crime_data(1980, 2009)
```

```
In [54]: # Some of the crime reports don not have an associated zip code and therefore will be
         crime_name = ['murder', 'manslaughter', 'rape', 'aggravated_assault',
                       'simple_assault', 'robbery', 'burglary', 'larceny',
                       'vehicle_theft']
         crime_data_df.loc[crime_data_df['zip_code'] == 0, crime_name].sum()
```

```
Out[54]: murder                   0
         manslaughter             0
         rape                     0
         aggravated_assault     241
         simple_assault           3
         robbery                 11
         burglary              2494
         larceny                685
         vehicle_theft           15
         dtype: int64
```

```
In [55]: # Make sure the number of invalid data entries is negligible compared to the full data
         print('Total Number of Reports:', crime_data_df.shape[0])
         crime_data_df.loc[:, crime_name].sum()
```

```
Total Number of Reports: 9648
```

```
Out[55]: murder                   4465
         manslaughter               47
         rape                    18575
         aggravated_assault     176561
         simple_assault         412711
         robbery                155666
         burglary               529107
         larceny               1466589
         vehicle_theft          350861
         dtype: int64
```

```
In [56]: # Convert crime data from pd.DataFrame to a vector of counts of each month for each c
         crime_data_dict = generate_crime_dict(crime_data_df)
```

```
# Example: The Y value for burglary crime.
crime_data_dict["burglary"]
```

```
Out[56]: array([2647, 2332, 2454, 2294, 2411, 2325, 2354, 2547, 2307, 2277, 2255,
                2769, 2891, 2750, 2874, 2643, 2689, 2315, 2319, 2319, 2311, 2404,
                2534, 2582, 2293, 2231, 2294, 2160, 2119, 2121, 2154, 2170, 2220,
                2237, 2234, 2375, 2554, 2030, 1970, 1753, 1794, 1857, 2061, 2119,
                2095, 2027, 2042, 2306, 2221, 2078, 1947, 1928, 2103, 2128, 2146,
                2093, 1940, 2102, 2230, 2331, 2377, 2120, 2271, 1992, 1975, 1816,
                2003, 2061, 1986, 2099, 1977, 2284, 2304, 1968, 2077, 2012, 1879,
                1871, 2122, 2019, 2045, 1923, 2004, 2168, 2236, 2044, 2066, 1843,
                1864, 1851, 1844, 1699, 1569, 1768, 1771, 1844, 1856, 1829, 1912,
                1605, 1808, 1766, 1860, 1847, 1876, 1939, 2084, 1926, 1946, 1658,
                1917, 1822, 1893, 1752, 2062, 1926, 1679, 1696, 1488, 1472, 1686,
                1464, 1528, 1288, 1367, 1402, 1747, 1632, 1560, 1583, 1505, 1443,
                1581, 1423, 1643, 1635, 1674, 1565, 1823, 1763, 1836, 1801, 1767,
                1914, 1782, 1546, 1791, 1546, 1658, 1637, 1710, 1696, 1436, 1592,
                1612, 1474, 1702, 1511, 1856, 1558, 1614, 1568, 1642, 1863, 1629,
                1730, 1529, 1637, 1564, 1352, 1454, 1432, 1468, 1356, 1447, 1596,
                1435, 1406, 1352, 1416,  873,  818,  806,  676,  740,  748,  781,
                 766,  716,  772,  774,  812, 1462, 1268, 1263, 1232, 1205, 1119,
                1188, 1228, 1170, 1118, 1054, 1073, 1078, 1159, 1121, 1233, 1195,
                1221, 1292, 1216, 1189, 1229, 1087, 1232, 1228, 1183, 1275, 1151,
                1326, 1228, 1218, 1283, 1250, 1108,  966, 1283, 1100,  986, 1136,
                1033,  845,  910, 1054,  862,  653,  873,  919, 1534, 1108,  843,
                 960,  815,  923,  730,  803,  775,  675,  796,  841,  757,  931,
                 767,  776,  806,  840,  794, 1014,  875,  882, 1089, 1075, 1186,
                1330,  945,  909,  855,  956, 1017, 1042,  978,  918,  973,  877,
                 976,  847,  845,  745,  883,  934,  863,  752,  726,  863, 1013,
                1412, 2175,  646,  595, 1077, 1215, 1199,  990, 1003, 1004,  989,
                1029, 1049, 1028, 1151, 1079, 1188, 1114, 1086, 1116, 1041, 1089,
                1106, 1127, 1220, 1176, 1131,  934, 1136, 1097, 1181, 1090, 1059,
                1096, 1029, 1017,  975, 1100, 1045,  854,  865,  955, 1161, 1071,
                1060, 1020, 1025, 1053,  959,  997,  991,  790,  843,  980,  960,
                 904,  932, 1057,  979, 1037, 1019, 1033,  936,  923, 1048,  939,
                 923,  937,  986,  961,  882,  917,  949, 1077], dtype=int32)
```

Now that we have assembled a dictionary of crimes and their counts for all 360 months, we will assemble the data tables as outlined in the "Approach" section. We begin by combining the crime count with the binned weather and precipitation data that were obtained in **Part 3**. The generate_combined_df function carries out this task below.

```
In [57]: crime_weather = generate_combined_df(crime_data_dict, bins_count_for_T, bins_count_fo
```

Below, we assmble the combined crime count data into a dataframe indexed by specific year-month conbinations with columns that represent temperature and precipitation bins.

```
In [58]: month_year = [str(year) + '-' + str(month) for year in range(1980,2010) for month in :
         df = pd.DataFrame(crime_weather)
```

```
df = df.transpose()
df.loc[:,'Year_Month'] = month_year
df.set_index('Year_Month', inplace = True)
T_columns = ['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-70', '70-80', '8
P_columns = ['<1', '1-5', '5-15', '15-29', '>29']
df.columns = ['Total_Crime_Count'] + T_columns + P_columns
df.head()
```

Out[58]:

```
            Total_Crime_Count  0-10  10-20  20-30  30-40  40-50  50-60  60-70  \
Year_Month
1980-1                   9980     0      0      0      0      0     52     10
1980-2                   8962     0      0      0      0      0     30     30
1980-3                   9956     0      0      0      0      0      6     51
1980-4                   9331     0      0      0      0      0      6     44
1980-5                   9839     0      0      0      0      0      4     41

            70-80  80-90  90-100  100-110  <1  1-5  5-15  15-29  >29
Year_Month
1980-1          0      0       0        0   0    0    42     20    0
1980-2          0      0       0        0   0    0    23     37    0
1980-3          3      0       0        0   0    0     4     56    0
1980-4         10      1       0        0   0    0     3     58    0
1980-5         15      1       0        0   0    0     1     60    0
```

In [59]:

```
east_crime, west_crime = split_crime_data(crime_data_df)
```

In [60]:

```
len(west_crime)
```

Out[60]: 8496

Next, we combine the East Alameda and West Alameda crime data with the East and West Alameda weather data, respectively.

In [61]:

```
# Generate the crime count table treating East and West Alameda as separate counties

# Generate Crime Dictionaries for East and West Alameda
east_crime_data_dict = generate_crime_dict(east_crime)
west_crime_data_dict = generate_crime_dict(west_crime)

# Generate East and West crime weather lists
east_crime_weather = generate_combined_df(east_crime_data_dict,
                                          east_bins_count_for_T,
                                          east_bins_count_for_P)

west_crime_weather = generate_combined_df(west_crime_data_dict,
                                          west_bins_count_for_T,
                                          west_bins_count_for_P)
```

Again, we then assemble the East and West crime-weather data into the form of the table outlined in the approach.

```
In [62]: # Assemble East and West Crime-Temperature data frames
         east_df = pd.DataFrame(east_crime_weather)
         east_df = east_df.transpose()
         east_df.loc[:,'Year_Month'] = month_year
         east_df.set_index('Year_Month', inplace = True)
         east_df.columns = ['Total_Crime_Count'] + T_columns + P_columns

         west_df = pd.DataFrame(west_crime_weather)
         west_df = west_df.transpose()
         west_df.loc[:,'Year_Month'] = month_year
         west_df.set_index('Year_Month', inplace = True)
         west_df.columns = ['Total_Crime_Count'] + T_columns + P_columns
         west_df.head()
```

```
Out[62]:             Total_Crime_Count  0-10  10-20  20-30  30-40  40-50  50-60  60-70  \
         Year_Month
         1980-1                   9479     0      0      0      0      0     44     18
         1980-2                   8528     0      0      0      0      0     25     35
         1980-3                   9527     0      0      0      0      0      5     52
         1980-4                   8892     0      0      0      0      0      4     48
         1980-5                   9429     0      0      0      0      0      4     46

                     70-80  80-90  90-100  100-110  <1  1-5  5-15  15-29  >29
         Year_Month
         1980-1          0      0       0        0   0    0    44     18    0
         1980-2          0      0       0        0   0    0    25     35    0
         1980-3          3      0       0        0   0    0     5     55    0
         1980-4          8      1       0        0   0    0     4     57    0
         1980-5         10      1       0        0   0    0     3     58    0
```

Next, in order to perform a Poisson Regression that accounts for fixed county-by-year and state-by-month fixed effects, represented by θ_{cy} and ϕ_{sm}, respectively, we must add one-hot-encodings to all three data frames. Accounting for such fixed effects ensures that the poisson regression is not overlooking natural trends that are known to occur in weather, such as the fact that summer months are generally hotter than winter months and that some months usually have more rain than others.

The information that we will add to each row will have the following form:

YearMonth	*CrimeCount*	*WeatherData*	January (1)	...	Month **m**	...
y, m	$C_{y,m}$	...	0	...	1	...

```
In [63]: # Generate one hot vectors necessary for Poisson regression
         poisson_df = generate_one_hot_encoding_df(df)
         poisson_east_df = generate_one_hot_encoding_df(east_df)
         poisson_west_df = generate_one_hot_encoding_df(west_df)

         poisson_df.head()
```

```
Out[63]:             Total_Crime_Count  0-10  10-20  20-30  30-40  40-50  50-60  60-70  \
         Year_Month
         1980-1                   9980     0      0      0      0      0     52     10
         1980-2                   8962     0      0      0      0      0     30     30
         1980-3                   9956     0      0      0      0      0      6     51
         1980-4                   9331     0      0      0      0      0      6     44
         1980-5                   9839     0      0      0      0      0      4     41

                     70-80  80-90  ...   2000  2001  2002  2003  2004  2005  2006  \
         Year_Month                ...
         1980-1          0      0  ...      0     0     0     0     0     0     0
         1980-2          0      0  ...      0     0     0     0     0     0     0
         1980-3          3      0  ...      0     0     0     0     0     0     0
         1980-4         10      1  ...      0     0     0     0     0     0     0
         1980-5         15      1  ...      0     0     0     0     0     0     0

                     2007  2008  2009
         Year_Month
         1980-1         0     0     0
         1980-2         0     0     0
         1980-3         0     0     0
         1980-4         0     0     0
         1980-5         0     0     0

         [5 rows x 59 columns]
```

With the crime-weather dataframes in the form as the one above, it is time to perform Poisson regressions on the actual data. Below is the code that performs the first Poisson regressions and displays a summary of the Poisson regression result for Alameda County as a whole:

```
In [64]: # Performing Poisson Regressions
         import statsmodels.api as sm

         # Model for Alameda County
         alameda_model = sm.GLM(np.asarray(poisson_df["Total_Crime_Count"]),
                                poisson_df[poisson_df.columns.difference(["Total_Crime_Count"]
                                family=sm.families.Poisson())
         alameda_results = alameda_model.fit()
         print(alameda_results.summary())

         # Model for East Alameda
         east_model = sm.GLM(np.asarray(poisson_east_df["Total_Crime_Count"]),
                                poisson_east_df[poisson_east_df.columns.difference(["Total_Cri
                                family=sm.families.Poisson())

         # Model for West Alameda
         west_model = sm.GLM(np.asarray(poisson_west_df["Total_Crime_Count"]),
                                poisson_west_df[poisson_west_df.columns.difference(["Total_Cri
                                family=sm.families.Poisson())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  360
Model:                            GLM   Df Residuals:                      309
Model Family:                 Poisson   Df Model:                           50
Link Function:                    log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -11718.
Date:                Fri, 07 Dec 2018   Deviance:                       19517.
Time:                        14:28:23   Pearson chi2:                 2.09e+04
No. Iterations:                   100   Covariance Type:             nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
1              6.6792      0.161     41.542      0.000       6.364       6.994
2              6.5922      0.151     43.713      0.000       6.297       6.888
3              6.6930      0.150     44.480      0.000       6.398       6.988
4              6.6598      0.156     42.580      0.000       6.353       6.966
5              6.6719      0.156     42.710      0.000       6.366       6.978
6              6.6239      0.156     42.417      0.000       6.318       6.930
7              6.6416      0.156     42.507      0.000       6.335       6.948
8              6.6275      0.160     41.440      0.000       6.314       6.941
9              6.5730      0.156     42.039      0.000       6.267       6.879
10             6.6484      0.156     42.548      0.000       6.342       6.955
11             6.6392      0.156     42.477      0.000       6.333       6.945
12             6.7020      0.157     42.736      0.000       6.395       7.009
1980           2.7946      0.063     44.543      0.000       2.672       2.918
1981           2.8133      0.062     45.204      0.000       2.691       2.935
1982           2.7584      0.062     44.247      0.000       2.636       2.881
1983           2.7005      0.062     43.263      0.000       2.578       2.823
1984           2.7098      0.063     43.116      0.000       2.587       2.833
1985           2.7393      0.062     43.989      0.000       2.617       2.861
1986           2.8003      0.062     45.078      0.000       2.679       2.922
1987           2.7818      0.062     44.724      0.000       2.660       2.904
1988           2.8257      0.063     44.950      0.000       2.702       2.949
1989           2.8156      0.062     45.246      0.000       2.694       2.938
1990           2.7309      0.062     43.847      0.000       2.609       2.853
1991           2.8605      0.062     45.857      0.000       2.738       2.983
1992           2.8460      0.063     45.231      0.000       2.723       2.969
1993           2.8527      0.062     45.782      0.000       2.731       2.975
1994           2.8039      0.062     44.962      0.000       2.682       2.926
1995           2.2987      0.062     36.856      0.000       2.176       2.421
1996           2.7373      0.063     43.412      0.000       2.614       2.861
1997           2.7069      0.062     43.439      0.000       2.585       2.829
1998           2.6647      0.062     42.688      0.000       2.542       2.787
1999           2.5282      0.062     40.487      0.000       2.406       2.651
2000           2.4261      0.063     38.521      0.000       2.303       2.550
2001           2.5126      0.062     40.283      0.000       2.390       2.635
2002           2.5450      0.062     40.797      0.000       2.423       2.667
```

```
2003             2.5347       0.062     40.629      0.000       2.412       2.657
2004             2.4973       0.063     39.590      0.000       2.374       2.621
2005             2.4738       0.062     39.645      0.000       2.351       2.596
2006             2.5419       0.062     40.708      0.000       2.419       2.664
2007             2.5220       0.062     40.472      0.000       2.400       2.644
2008             2.4923       0.063     39.612      0.000       2.369       2.616
2009             2.4368       0.062     39.095      0.000       2.315       2.559
0-10          8.312e-18    8.55e-19      9.722      0.000    6.64e-18    9.99e-18
1-5             -0.0078       0.002     -3.513      0.000      -0.012      -0.003
10-20        -2.421e-17    1.03e-18    -23.484      0.000   -2.62e-17   -2.22e-17
100-110          0.0262       0.003      8.357      0.000       0.020       0.032
15-29           -0.0027       0.003     -1.044      0.297      -0.008       0.002
20-30         5.027e-17    7.33e-19     68.550      0.000    4.88e-17    5.17e-17
30-40           -0.0078       0.002     -3.513      0.000      -0.012      -0.003
40-50           -0.0101       0.001     -7.636      0.000      -0.013      -0.008
5-15             0.0018       0.003      0.721      0.471      -0.003       0.007
50-60           -0.0049       0.001     -3.997      0.000      -0.007      -0.002
60-70           -0.0026       0.001     -2.331      0.020      -0.005      -0.000
70-80        -3.975e-05       0.001     -0.035      0.972      -0.002       0.002
80-90           -0.0019       0.001     -1.706      0.088      -0.004       0.000
90-100          -0.0085       0.001     -6.859      0.000      -0.011      -0.006
<1                    0           0        nan        nan           0           0
>29             -0.0010       0.003     -0.397      0.692      -0.006       0.004
==============================================================================


/usr/local/lib/python3.7/site-packages/statsmodels/base/model.py:1100: RuntimeWarning: invalid
  return self.params / self.bse
/usr/local/lib/python3.7/site-packages/scipy/stats/_distn_infrastructure.py:879: RuntimeWarning
  return (self.a < x) & (x < self.b)
/usr/local/lib/python3.7/site-packages/scipy/stats/_distn_infrastructure.py:879: RuntimeWarning
  return (self.a < x) & (x < self.b)
/usr/local/lib/python3.7/site-packages/scipy/stats/_distn_infrastructure.py:1821: RuntimeWarnin
  cond2 = cond0 & (x <= self.a)
```

Now that we have performed Poisson regressions on all of Alameda County, as well as East and West Alameda County speararately, we will now devise and perform a permutation test that tests whether the crime and weather in East and West Alameda are consistent with a single model.

NullHypothesis: In all months from 1980 to 2009, the relationship between crime and weather can be described by the same Poisson Regression model.

The Null hypothesis stated above assumes that weather-crime data for West Alameda and East Alameda occured by the same underlying process. Thus, under this assumption, models trained on monthly permutation of the East and West weather-crime data would not have drastically different overall training RMSEs (specifically very large RMSEs relative to the original RMSE because a lower RMSE indicates a better fit). Thus the permutations will permute the East and West weather-crime data by month and evaluate whether the associated Poisson Regression training RMSEs change drastically.

```
In [65]: # Get original RMS statistics

         east_rms = get_rms(east_model,
                            poisson_east_df[poisson_east_df.columns.difference(["Total_Crime_Co
                            poisson_east_df["Total_Crime_Count"])

         west_rms = get_rms(west_model,
                            poisson_west_df[poisson_west_df.columns.difference(["Total_Crime_Co
                            poisson_west_df["Total_Crime_Count"])

         # Combine the RMS stats into an overall RMS statistic
         original_rms = np.sqrt(0.5*(west_rms**2 + east_rms**2))

In [66]: # Calculate the permutation statistics
         stats = get_permutation_stats(east_df, west_df,"Total_Crime_Count",100)

In [67]: # Calculate the p-value
         stats = np.asarray(stats)
         good = stats[stats < original_stat]
         p_value = float(len(good)+1)/(len(stats)+1)
         round(p_value,4)

Out[67]: 0.0099
```

As shown above, the p-value for the randomized test is below the 5% significance level. Thus, the test rejects the Null Hypothesis, and we can conclude that the relationship between weather and crime is not consistent between East Alameda and West Alameda.

The test above has shown that weather-crime relationships in two halves of Alameda County cannot be described by the same model, and this serves to put some of Ranson's claims severely into question. Ranson assumes that one model can describe the relationship between weather and crime in every county, and thus all further results he achieves on this assumption have questionable validity.

8 Reflection

In this project, we aggregated crime and weather data from Alameda County in the fashion described by Professor Matthew Ranson in his paper "Crime, Weather & Climate Change." We also attempted to reproduce some of his analyses and validate some of his critical assumptions. Ranson performs his analyses and develops a predictive model by county and month. To do so, he makes various assumptions that he fails to verify. Namely, he assumes that weather is effectively the same across all counties, and he assumes that a single Poisson Regression model can adequately describe the relationship between crime and weather throughout a single state.

The first half over our project was devoted to gathering and assembling crime and weather data for Alameda County in a transparent and reproducible fashion. Throughout this process, we discovered that we still had to make choices of our own during the data preparation process. For example, we chose to filter some of the GHCN-Daily weather data that had allegedly already been cleansed because some of the reported maximum temperatures were less than reported minimum temperatures on certain days. Thus, a significant takeaway that we had from this assignment and our consequent analyses is that in order to make work reproducible, it is critically important to carefully document the data engineering choices you make and any underlying assumptions. We also questioned a number of Ranson's choices, including the choice to aggregate weather by month, instead of, for example, by week.

The final half of our project is devoted to testing the validity of Ranson's aforementioned assumptions. Our first permutation test concluded that Oakland and Livermore, despite both being cities in Alameda County, have experienced different temperatures, meaning that their temperatures are inherently different. Thus, the test proved that Ranson's assumption that weather behavior is the same countywide is invalid. Lastly, using another permutation test we showed that it is unreasonable to believe that a Poisson single model can adequately describe the relationship between weather and crime in a single county, let alone a single state. Therefore, because we have proven a few of Ranson's assumptions invalid, this leads us to view his further conclusions with skepticism.