Assignment 3 Analysis

November 4, 2018

In [10]:
```
import numpy as np
import pandas as pd
import sys

sys.path.insert(0, '../Assignment4/')
sys.path.insert(0, '../Assignment3/')
sys.path.insert(0, '../Assignment2/')
sys.path.insert(0, '../Assignment1/')
sys.path.insert(0, '../')

from assignment2 import get_stations, get_alameda_county_points, get_station_weights
from weighted_avg import calc_inv_weighted_avg
```

The first thing we had to do for Assignment 3 is to pull all the stations within 10 miles of any grid point in Alameda County and then get all the weather data for the relevant time periods from those stations. We noticed that we can find out which station weather data came from based on the ID column in the weather data and station data. Once we figured this out, we pulled both datasets into pandas DataFrames and merged them on the ID of both DataFrames.

In [11]:
```
def get_weather_data(points = get_alameda_county_points(), max_distance = 10):
    """Gets weather data for a given set of grid points

    args:
        points: list of grid points (lat, lon) (default alameda county grid)
        max_distance: max distance to search around each grid point (default 10)
    return:
        pd.DataFrame: dataframe with data from weather and stations
    """

    # load data from csv
    stations = pd.DataFrame(get_stations(points, max_distance))
    weather = pd.read_csv('Assignment3/data/ca_weather_data.csv')

    # merge data on the ID
    merged = stations.merge(weather, on='ID', how='inner')
    return merged
```

```
Loading formatted geocoded file...
```

Once we retrieved the data for the relevant time periods, we had to limit our search to meet Ranson's criteria for inclusion in each year. However, to do this, we needed to clean our data so that we would be able to access the year attribute. Additionally, we need some properties like year, month, and day that we used later in the analysis. To speed up computation, we removed all rows that weren't pertinent to our analysis (eg. PRCP data).

```
In [13]: def clean_data(df):
             """ Cleans data from original dataframe

             args:
                 df: dataframe of data
             return:
                 df: cleaned dataframe
             """

             # clean the date data
             df['date'] = pd.to_datetime(df['YEARMONTHDAY'].astype(str), format='%Y%m%d')
             df['year'] = df['date'].dt.year
             df['month'] = df['date'].dt.month
             df['day'] = df['date'].dt.day

             # only keep temperature rows
             df = df[df['ELEMENT'] != 'PRCP']
             return df
```

Ranson's criteria for inclusion was to only include data points for a particular (weather station, year, element) tuple that had atleast 100 records that matched it. To do this, we counted all the records in the data set grouped by the (weather station, year, element). Once we had this data, we filtered the rows to include only (weather station, year, element) tuples that had atleast 100 records. With this new dataset, we were able to get the intersection of the new dataset and our original dataset to get the records we wanted. We accomplished this by doing an inner join on the tables on the (weather station, year, element) columns.

```
In [12]: def filter_ranson_criteria(df):
             """Filters the rows based on Ranson's criteria which states
             to only include year, station, element combinations that have
             more than 100 points

             args:
                 df: dataframe of data
             return:
                 df: dataframe of data with only ranson criteria satisfying rows
             """

             yearly_count = df.groupby(['NAME','year', 'ELEMENT']).count()
             new_df = yearly_count.reset_index()
             new_df = new_df[new_df['ID'] >= 100][['NAME', 'year', 'ELEMENT']]
             return new_df.merge(df, how='inner', on=['NAME', 'year', 'ELEMENT'])
```

Then, we had to run our bias correction algorithm. The bias correction algorithm was given to us in Lab 8. We implemented this by first initializing all of our parameters. Then we ran a while loop that would continue until the convergence criteria is met. The convergence criteria will be discussed later in this description.

We first identify a randomly selected reference station. Next, we follow the algorithm to get all the data where the stations intersect and sum them in the way the equation mentions to. We calculate the station intersection by filtering the original dataframe by the station name and then merging the two datasets on the date. We use an inner join so that we only get rows that intersect. In this way, we are able to determine all the data points from both the station and the reference. This data along with the number of intersected rows allows us to update the intercept value for that station.

```
In [16]: def station_intersection(station1_name, station2_name, df):
             """Gets the intersection of rows between station1 and station2

             args:
                 station1_name: name of station 1
                 station2_name: name of station 2
                 df: dataframe of data
             return:
                 int: length of the merged data
                 df: dataframe of intersected data
             """

             # find the days that the stations intersect
             station1_rows = df[df['NAME'] == station1_name]
             station2_rows = df[df['NAME'] == station2_name]
             merged = station1_rows.merge(station2_rows, on='date', how='inner')
             return len(merged['date']), merged
```

We repeat this procedure till convergence. Our convergence criteria was to ensure that every (2 * # of stations) time we run the algorithm, the difference between the original intercepts calculated aren't 'too different' from the iteration (2 * # of stations) before the current iteration. We define 'too different' as having a loss less than 0.000001.

```
In [17]: def bias_correction(df):
             """Adjusts the data by doing a bias correction

             args:
                 df: dataframe to run bias correction on
             return:
                 df: dataframe with bias corrected data
             """

             stations = df['NAME'].unique()

             # initialize intercepts
             intercepts = [0 for _ in range(len(stations))]
```

```
old_intercepts = intercepts[:]
count = 0

# iterate till convergence
while True:

    # pick random point in reference
    reference_idx = np.random.randint(0, len(stations))
    reference_station = stations[reference_idx]

    # iterate through all stations to update intercept
    for station_idx in range(len(stations)):
        station = stations[station_idx]

        # get intersection of stations
        n, k = station_intersection(reference_station, station, df)

        # calculate sum for all intersected rows
        curr_sum = 0.0
        for _, row in k.iterrows():
            station1_data = row['DATA VALUE_x']
            station2_data = row['DATA VALUE_y']
            curr_sum += station1_data + intercepts[reference_idx] - (station2_data

        # update the stations intercept
        if n != 0:
            intercepts[station_idx] = intercepts[station_idx] + curr_sum / n

    if count % (2 * len(stations)) == 0 and count != 0:
        # check if it converges by calculating loss
        loss = np.sum((np.array(intercepts) - np.array(old_intercepts)) ** 2)
        old_intercepts = intercepts[:]
        # set convergence criteria
        if loss < 0.000001:
            break
    count += 1

# map station name to intercept value
bias = dict(zip(stations, intercepts))

# get latitude and longitude of stations
lat_lon = []
for station in stations:
    data = df[df['NAME'] == station].iloc[0]
    latitude, longitude = data['LATITUDE'], data['LONGITUDE']
    lat_lon.append((latitude, longitude))

# get station weights
```

```
    weights = np.array(calc_inv_weighted_avg(get_alameda_county_points(), lat_lon))
    weights = weights * np.array(intercepts) / np.sum(weights)

    # calculate C
    C = dict(zip(stations, weights))

    # apply formula to calculate adjusted data
    df['adjusted_data'] = df.apply(lambda x: x['DATA VALUE'] + bias[x['NAME']] - C[x['N

    return df
```

The final step of this algorithm is to bin the data. With some help from stackoverflow, we were able to find an easy way to do this. Essentially, we ran our entire algorithms through the above functions and grouped the data set by year and month and then binned the bias corrected data into bins [0, 300, 10] for each of the elements we were interested in.

```
In [ ]: def main(elements=['TMAX', 'TMIN'], points = get_alameda_county_points()):
            """Bins adjusted data for each element. Binning idea received
            by: https://stackoverflow.com/questions/34317149/pandas-groupby-with-bin-counts

            args:
                elements: specifies elements to calculate data for (default ['TMAX', 'TMIN'])
            return:
                list: returns list of binned data as dataframe
            """

            df = filter_ranson_criteria(clean_data(get_weather_data(points)))
            result = []
            for element in elements:
                filtered = df[df['ELEMENT'] == element]
                corrected = bias_correction(filtered)

                # average data from the same day
                grouped = corrected.groupby(['year', 'month', 'day']).aggregate({'adjusted_data'

                # bin the data
                final = grouped.groupby(['year', 'month', pd.cut(grouped['adjusted_data'], range
                result.append(final)
            return result
```