# Exploring Weather Trends

Udacity Data Analysis Nanodegree

Term 1 Project 1

Jesse Fredrickson

7/14/18

## 1. Methods:

Below I describe the steps I took to retrieve and analyze the data, and to present my findings

### 1.1. SQL Query:

The raw data is stored on a SQL server hosted by Udacity in the form of 3 databases:

- city_list - This contains a list of cities and countries in the database.
- city_data - This contains the average temperatures for each city by year (ºC).
- global_data - This contains the average global temperatures by year (ºC).

I first queried the city_list database to see what cities were available in the United States. I retrieved all columns and sorted by city to get results in alphabetical order. In all SQL queries I ran, I retrieved all columns – each database is relatively small and there was no reason to restrict my query.

```
SELECT *
FROM city_list
WHERE country = 'United States'
ORDER BY city
```

I live in Somerville MA, and as expected, the closest city is Boston MA. I next queried the city_data database to retrieve all weather data for Boston. I sorted the output by year, although I could also have done this in the analysis step. It should be noted that there were some missing values here. I then exported the returned data to a .csv file.

```
SELECT *
FROM city_data
WHERE city = 'Boston' OR city = 'boston'
ORDER BY year
```

Finally, I queried the global_data database to get weather data from across the globe. Again I ordered it by year and exported the returned data to a .csv file.

```
SELECT *
FROM global_data
ORDER BY year
```

### 1.2. Python analysis:

Now that I had temperature data for Boston and the globe in the form of csv files, I needed to import the data into a tool to analyze it. I opted to use Python for this. My source code is appended at the end of this document for review.

I used the pandas library to load the data from each .csv file and combine them into a single DataFrame object, indexed by year. I then created two extra columns to show the rolling mean of the data in a 10-year window in order to smooth the curves. This is accomplished by the following line of code (ex: boston):

```
df['boston_rolling_temp'] = df['avg_temp'].rolling(10, min_periods=10).mean()
```

This line of code is defining a new column "boston_rolling_temp" in the primary DataFrame "df" which is generated from the "avg_temp" column and generated by the .mean() method called on the window indicated by the .rolling() method. Calling the .rolling() method on the avg_temp column allows me to perform an action on a moving window of columns values – I've defined the window to be 10 rows, which corresponds to 10 years. I have explicitly instructed the program to only return a value if there are 10 non-missing values in the window, which effectively means that the rolling average will not exist whenever there is a missing value in the avg_temp column, and for 9 rows after. This is not necessary as the .mean() method will scale based on how many missing values are in the window, but it can result in a marginally better trendline in the next step since outliers will be mitigated by the maximum of 10 other points every time.

Next, I removed all rows which had missing data for either rolling average – this was to ensure that I could calculate trendlines in the next step. I used the numpy library to perform 3$^{rd}$-degree linear regression on each rolling mean, and stored the values output by each trendline on another pair of columns. Finally, I calculated the coefficient of determination ($R^2$) for each trendline to see how well they fit the real (rolling mean) data.

### 1.3. Visualization

I used the matplotlib.pyplot library to create a simple line graph to visualize the data. Note that I restricted the y-axis values to better show detail in the graphs – ultimately this is probably unnecessary because the rolling mean smoothed outliers which were skewing the y-axis range, but I left it in. I plotted the local and global rolling means along with each of their trendlines. On the legend, I printed line labels along with the coefficients of determination for the fit lines. An $R^2$ value indicates that a trendline perfectly accounts for variations in the data; in this case, both trendlines account for >70% of variation in the data. Calculating a higher order polynomial fit increases the coefficient of determination with diminishing returns, but does make the overall trend a little harder to understand/draw conclusion from. See figure 1 below.

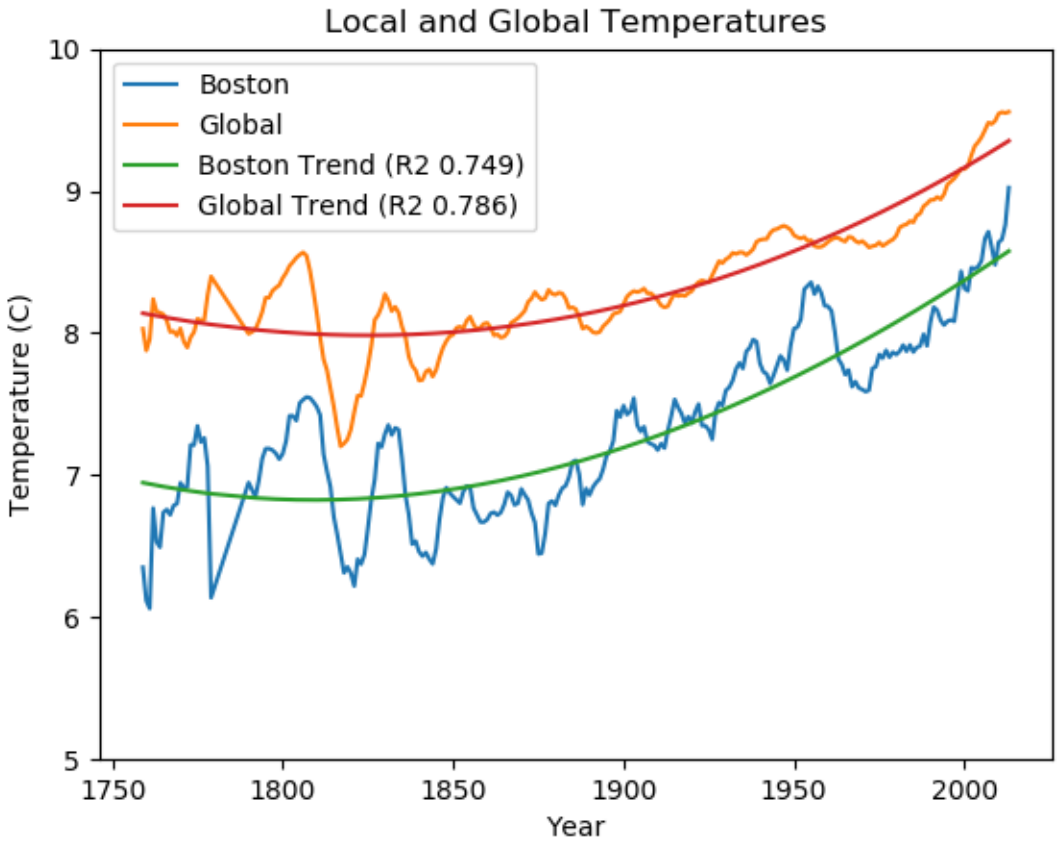


*Figure 1: Boston 10-year rolling average temperature and global 10-year rolling average temperature*

## 2. Observations:

- Boston is on average about 1°C cooler than the global average
    - This is expected as New England is relatively far north of the equator and is known for cool temperatures. The global average temperature includes hot climates closer to the equator.
- Data before 1800 appears much more varied for Boston
    - This is most apparent in the raw (not mean) data, but can still be seen here by looking at the gaps between the peaks and valleys of the Boston data and the Boston trendline. It becomes more consistent through the 1900s, which indicates to me that temperature measuring tools improved in the 1900s and resulted in more accurate readings.
- Both global and Boston temperatures appear to be rising since 1900
    - This is the expected result, as it is commonly accepted that global climate change has led to higher temperatures in the last 100 years especially. The 250-year window here appears to fit well with an exponential curve, but the change is gradual and we likely do not have enough data to confirm that temperatures are rising exponentially.
- Boston in general appears to have more variation than global temp

- Observe that the Boston line appears more jagged than the global line. This is most likely because the global temperature is actually an average of cities around the world as opposed to one location and measurement, and this serves to smooth the data not unlike how I smoothed the data with a rolling curve.

- Boston temperature appears to be growing closer to the global average
  - Although I did not confirm it analytically, the reader can confirm visually that the trendlines are closer together in 2000 than they are in 1800.

# 3. Appendix

## 3.1. t1-p1.py

```
# t1-p1
# Exploring weather trends
# Jesse Fredrickson

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# SQL: search through available cities in city_list
# SELECT *
# FROM city_list
# WHERE country = 'United States'
# ORDER BY city

# SQL: query city_data to get temperatures for Boston
# SELECT *
# FROM city_data
# WHERE city = 'Boston' OR city = 'boston'
# ORDER BY year

# SQL: query global_data to get global temperatures
# SELECT *
# FROM global_data
# ORDER BY year

# read boston and global data as dataframes
pth = 'C:/Users/Jesse/Documents/PyData/DAND_t1_p1/'

df_boston = pd.read_csv(pth + 'boston_weather.csv')
df_global = pd.read_csv(pth + 'global_weather.csv')

# join dataframes by year
df = df_boston.set_index('year').join(df_global.set_index('year'), rsuffix='_global', )

# get rolling mean columns for boston and global weather
df['boston_rolling_temp'] = df['avg_temp'].rolling(10, min_periods=10).mean()
df['global_rolling_temp'] = df['avg_temp_global'].rolling(10, min_periods=10).mean()

# clean missing data
df_clean = df.dropna(subset=['boston_rolling_temp', 'global_rolling_temp'], how='any')

# form linear regression trendlines
trend_bos = np.polyfit(df_clean.index, df_clean['boston_rolling_temp'], 3)
bos_trend_p = np.poly1d(trend_bos)
df_clean['trend_bos'] = bos_trend_p(df_clean.index)

trend_global = np.polyfit(df_clean.index, df_clean['global_rolling_temp'], 3)
global_trend_p = np.poly1d(trend_global)
df_clean['trend_global'] = global_trend_p(df_clean.index)


def r_squared(y, y_fit):
    # returns the coefficient of determination for y and y_fit
    ybar = np.sum(y) / len(y)
    ssreg = np.sum((y_fit - ybar) ** 2)
    sstot = np.sum((y - ybar) ** 2)
    determination = ssreg / sstot

    return determination
```

```
# get R^2 values for the fit lines
boston_r2 = r_squared(df_clean['boston_rolling_temp'], df_clean['trend_bos'])
global_r2 = r_squared(df_clean['global_rolling_temp'], df_clean['trend_global'])

# Plot
df_clean.plot(y=['boston_rolling_temp', 'global_rolling_temp', 'trend_bos', 'trend_global'])
plt.legend(['Boston', 'Global', 'Boston Trend' + ' (R2 {:.3f})'.format(boston_r2),
            'Global Trend' + ' (R2 {:.3f})'.format(global_r2)])
plt.title('Local and Global Temperatures')
plt.ylabel('Temperature (C)')
plt.xlabel('Year')
plt.ylim((5, 10))
plt.show()
```