

# Data Analyst Project: Explore Weather Trends

## Data Analyst Nanodegree Program

This document is intended to show the findings of weather trends by analyzing temperature data. With the temperature data gathered for centuries and assistance of appropriate analysis tools, we can make the change of worldwide temperature become visible and comprehensible.

### Outline of this project:

1. Extract temperature data from SQL database in Project Workspace. Export query results to CSV files.
2. Load CSV files into any tool that can perform data analysis. For this project this can be Microsoft Excel or Google Sheet, but we will be using a Python program for this.
3. Use Python program to calculate moving average from avg\_temp column in global and city level data.
4. Use Python program to create a line chart from computed moving average data.
5. From the line chart, try to discover the similarities and differences between the variation of time series data of global and local city, as well as overall trends.

## 1. Extract the data

Go to Udacity's Project Workspace and execute the following SQL queries to fetch average temperature data by year. Remember to export query results to individual CSV files by clicking the "Download CSV" button.

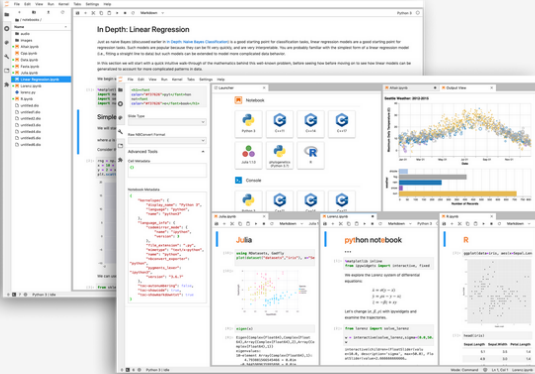
	Tasks	SQL Queries
A.	Extract the average global temperatures by year.	SELECT * FROM global_data ORDER BY year
B.	Extract the average temperatures for each city by year.	SELECT * FROM city_data ORDER BY country, city, year
C.	Extract the list of cities and countries.	SELECT * FROM city_list ORDER BY country, city

If you are only interested in specific country/city combinations, you can modify the SQL query which reads the **city\_data** table to make it fetch only a small subset of city level data by adding a WHERE clause. But here we are dumping the whole **city\_data** table, because it can go with the analysis tool we built in this project better.

## 2. Load CSV files into Python Program

Starting from this step, we will be working in a JupyterLab environment offered by Project Jupyter for free. Instead of creating a Python script, we will be using a Jupyter Notebook (submitted with this document) to assist us in loading CSV files and line chart creation.


To create a temporary Jupyter server on the fly, first you will need to visit [Project Jupyter official website](#) and click the link "Try it in your browser" in the page section of "JupyterLab".



## JupyterLab: Jupyter's Next-Generation Notebook Interface

JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data. JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning. JupyterLab is extensible and modular: write plugins that add new components and integrate with existing ones.

[Try it in your browser](#) [Install JupyterLab](#)



Then on the next page, you should click on the banner of “Try JupyterLab”.

## Try Jupyter


You can try Jupyter out right now, without installing anything. Select an example below and you will get a temporary Jupyter server just for you, running on [mybinder.org](https://mybinder.org). If you like it, you can [install Jupyter](#) yourself.

### Try Classic Notebook




A tutorial introducing basic features of Jupyter notebooks and the IPython kernel using the classic Jupyter Notebook interface.

### Try JupyterLab



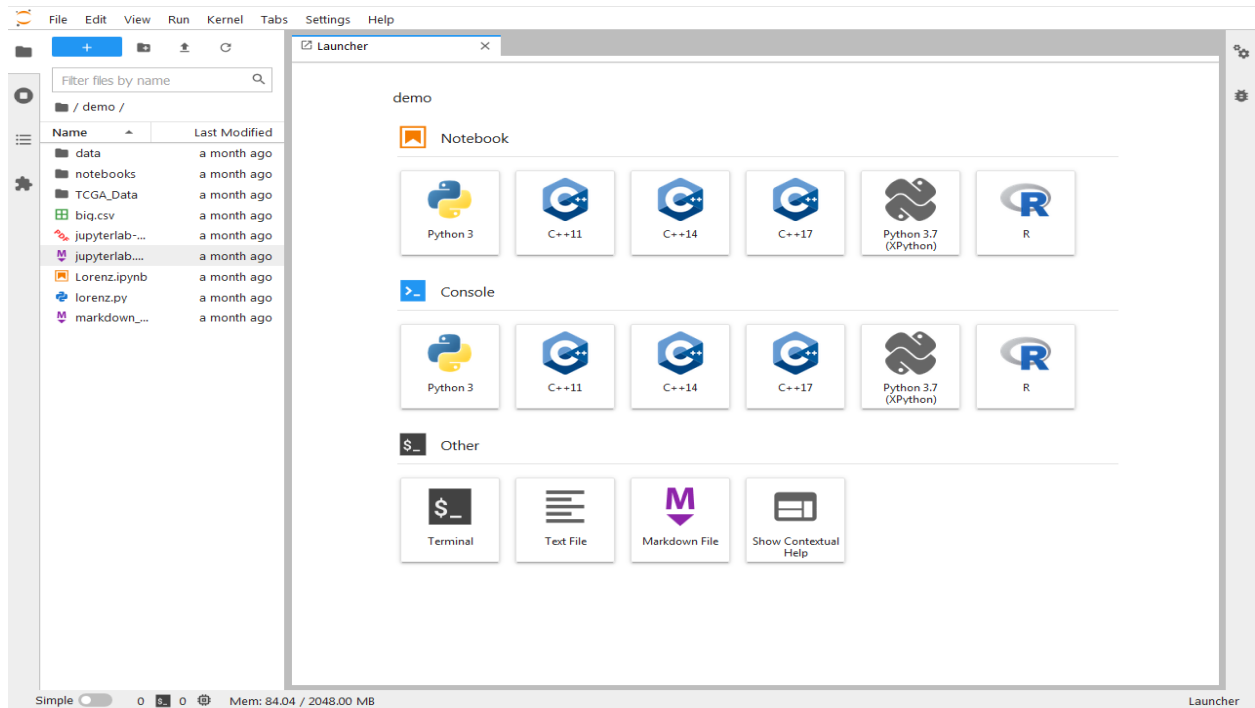
JupyterLab is the new interface for Jupyter notebooks and is ready for general use. Give it a try!

### Try Jupyter with Julia

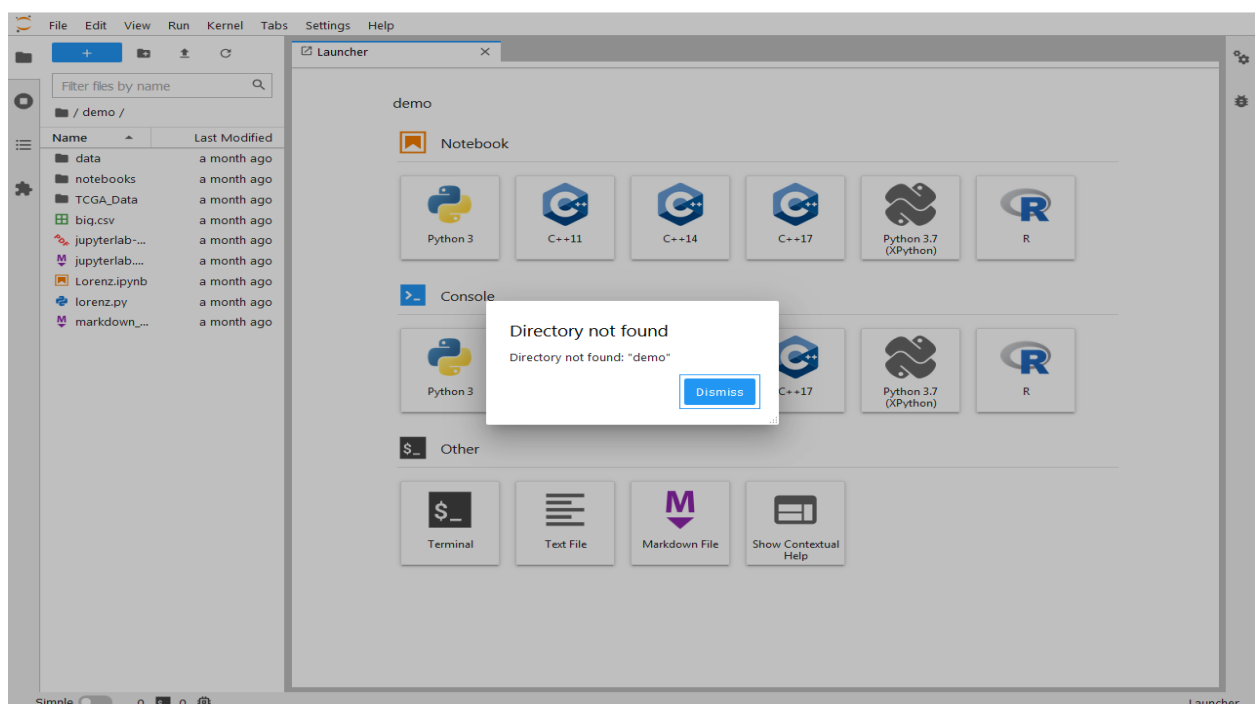


A basic example of using Jupyter with Julia.

By doing so, you will be redirected to a short-term Jupyter server for your own use.



For anyone who needs to do some Python works, this can serve as a handy, instant workspace (no need to install and configure various softwares on your computer), but remember to save your work periodically by downloading files to your own computer, since the server will terminate itself in less than a few idling hours. Your browser window/tab may look like the following picture if your short-term Jupyter server has been terminated.



If this happens to you, you can close this browser window/tab and follow the steps described before to create another temporary Jupyter server again.

Now, with a personal server ready-to-use, you can either begin your development work, or resume your unfinished work by uploading files from your computer to this server.

Loading a CSV file into a Python program is analogous to opening a CSV file in Microsoft Excel/Google Sheet. The main difference is while it usually takes manual work to perform analysis when using spreadsheet applications, there is almost no manual work needed when you offload most of the tasks to a written program.

In this project we will use the Pandas library in our Python program, and create DataFrame objects to hold data in CSV files. DataFrames can also be helpful when we need to do some data manipulations afterwards. Following is the code snippet to load CSV files.

```
import pandas as pd

# Load exported CSV files into Pandas DataFrames
df_city_list = pd.read_csv('results_city_list.csv')
df_raw_global = pd.read_csv('results_global_data.csv')
df_raw_city_all = pd.read_csv('results_city_data.csv')
```

Because we are going to create a line chart which contains both global and local city data, it would be desirable if we can merge both global data and local city data into a single entity. When using Excel/Google Sheet, we would put global and local data in the same sheet and create a line chart based on data in this sheet. The same tasks should also be accomplished when we use a Python program instead.

First we rename the column “avg\_temp” in both global and local data:

```
# Rename column to distinguish global avg_temp from local avg_temp
df_raw_global.columns = ['year', 'avg_temp_global']
df_raw_city_all.columns = ['year', 'city', 'country', 'avg_temp_local']
```

Then we extract a small subset of city level data:

```
# Extract only rows containing selected city and country from the whole
city data
df_selected_city = df_raw_city_all.loc[(df_raw_city_all['city'] == 'Rio De
Janeiro') & (df_raw_city_all['country'] == 'Brazil')]
```

Lastly we combine global and local city data into a single DataFrame (along with some data manipulations):

```
# Combine global data and local city data into a single DataFrame
# Use inner join to align year range of global data with city level data
df_combined = pd.merge(df_raw_global, df_selected_city, how="inner",
on='year')

# Reset DataFrame index (row number)
df_combined.reset_index(drop=True, inplace=True)
```

### 3. Calculate Moving Average

Essentially the way to calculate moving average of average temperature data is identical to the instructions provided in the course (refer to “Moving Averages” in the project). However, there are some details that we may need to pay attention to.

1. Notice the cell position where the first calculated moving average populates itself. This example picture in the course shows the 7-Day MA starts from C8, the same row as the end of 7-Day window (B2:B8). This makes sense because we would need at least 7 daily data points to calculate an initial 7-Day MA value.

<i>fx</i>	=AVERAGE(B2:B8)			
	A	B	C	
1	Date	Sales	7-Day MA	
2	1/1/2009	7,855		
3	1/2/2009	12,329		
4	1/3/2009	11,617		
5	1/4/2009	7,684		
6	1/5/2009	8,448		
7	1/6/2009	8,022	9,195 ×	
8	1/7/2009	8,410	=AVERAGE(B2:B8)	
9	1/8/2009	7,850		

2. The example provided in the course does not show the situation when empty/missing/NULL values occur. Ideally there should be no vacant cell in our gathered data, but actually some cities in the **city\_data** table do have NULL avg\_temp values. We will take a look at how Excel/Google Sheet handles this when calculating moving averages.

In the picture below we can see that Rio De Janeiro in Brazil has no data point during 1844-1850.

	A	B	C	D
1	year	city	country	avg_temp
2	1832	Rio De Janeiro	Brazil	23.05
3	1833	Rio De Janeiro	Brazil	24.11
4	1834	Rio De Janeiro	Brazil	23.27
5	1835	Rio De Janeiro	Brazil	22.73
6	1836	Rio De Janeiro	Brazil	22.91
7	1837	Rio De Janeiro	Brazil	22.29
8	1838	Rio De Janeiro	Brazil	22.81
9	1839	Rio De Janeiro	Brazil	22.54
10	1840	Rio De Janeiro	Brazil	23.32
11	1841	Rio De Janeiro	Brazil	22.97
12	1842	Rio De Janeiro	Brazil	23.41
13	1843	Rio De Janeiro	Brazil	23.55
14	1844	Rio De Janeiro	Brazil	
15	1845	Rio De Janeiro	Brazil	
16	1846	Rio De Janeiro	Brazil	
17	1847	Rio De Janeiro	Brazil	
18	1848	Rio De Janeiro	Brazil	
19	1849	Rio De Janeiro	Brazil	
20	1850	Rio De Janeiro	Brazil	
21	1851	Rio De Janeiro	Brazil	23.53
22	1852	Rio De Janeiro	Brazil	23.74
23	1853	Rio De Janeiro	Brazil	23.76
24	1854	Rio De Janeiro	Brazil	23.83
25	1855	Rio De Janeiro	Brazil	23.89
26	1856	Rio De Janeiro	Brazil	22.83
27	1857	Rio De Janeiro	Brazil	23.46
28	1858	Rio De Janeiro	Brazil	22.37
29	1859	Rio De Janeiro	Brazil	23.04
30	1860	Rio De Janeiro	Brazil	23.91

As we already know, to calculate a moving average of window size n, there should be exactly n data points filled in the window, but things become tricky when there are missing values in the window. Let's see what will happen when we use the same method to calculate moving average for this dataset.

	A	B	C	D	E	F
1	year	city	country	avg_temp	10-Year MA	
2	1832	Rio De Janeiro	Brazil	23.05		
3	1833	Rio De Janeiro	Brazil	24.11		
4	1834	Rio De Janeiro	Brazil	23.27		
5	1835	Rio De Janeiro	Brazil	22.73		
6	1836	Rio De Janeiro	Brazil	22.91		
7	1837	Rio De Janeiro	Brazil	22.29		
8	1838	Rio De Janeiro	Brazil	22.81		
9	1839	Rio De Janeiro	Brazil	22.54		
10	1840	Rio De Janeiro	Brazil	23.32		
11	1841	Rio De Janeiro	Brazil	22.97	=AVERAGE(D2:D11)	
12	1842	Rio De Janeiro	Brazil	23.41	23.036	
13	1843	Rio De Janeiro	Brazil	23.55	22.98	

This picture shows the calculation result of 10-Year MA when the window is filled with data points. Now, compare this with the next picture, which the window contains missing values.

10	1840	Rio De Janeiro	Brazil	23.32		
11	1841	Rio De Janeiro	Brazil	22.97	23	
12	1842	Rio De Janeiro	Brazil	23.41	23.036	
13	1843	Rio De Janeiro	Brazil	23.55	22.98	
14	1844	Rio De Janeiro	Brazil		22.94777778	
15	1845	Rio De Janeiro	Brazil		22.975	
16	1846	Rio De Janeiro	Brazil		22.98428571	
17	1847	Rio De Janeiro	Brazil		23.1	
18	1848	Rio De Janeiro	Brazil		23.158	
19	1849	Rio De Janeiro	Brazil		23.31	
20	1850	Rio De Janeiro	Brazil		=AVERAGE(D11:D20)	
21	1851	Rio De Janeiro	Brazil	23.53	23.49666667	

Further inspection reveals that the moving average value , 23.31, is derived from adding up all available data points, and then divide by number of available data points (in this case would be  $(22.97+23.41+23.55) / 3 = 23.31$ ).



In other words, the AVERAGE() function in Google Sheet (and Excel, too) will still proceed with calculation of moving average as long as the window is not completely filled with empty values, and will adjust its calculation accordingly by evaluating only non-empty values.

We will make our Python program to imitate the behaviors of Google Sheet explained before, and use it to calculate moving averages in our stead.

```
# Calculate 10-year MA of avg_temp and append these computed columns to
DataFrame
ma_window_size = 10
df_combined['{}_year_ma_global'.format(ma_window_size)] =
df_combined.rolling(window=ma_window_size,
min_periods=1).avg_temp_global.mean()
df_combined['{}_year_ma_local'.format(ma_window_size)] =
df_combined.rolling(window=ma_window_size,
min_periods=1).avg_temp_local.mean()

# Delete MA values which do not conform to window size (MA values that
shouldn't be generated when using Excel/Google Sheet)
for i in range(0, (ma_window_size-1), 1):
    df_combined.loc[i, ['10_year_ma_global', '10_year_ma_local']] = None
```

At this moment we have our DataFrame ready for the task of line chart creation.

## 4. Line Chart Creation

There are plenty of libraries for creating visualizations in Python. Some of these can even integrate with Pandas as a plotting backend. In this project, either Matplotlib or Bokeh would be a good choice, since they are already installed on temporary Jupyter servers.

As Matplotlib becomes the default plotting backend of Pandas, a line chart can be easily created by writing only 1 line of code:

```
import matplotlib
import matplotlib.pyplot as plt

# Create a line plot with both global and city level data displayed
df_combined.plot(x='year', y=['10_year_ma_global', '10_year_ma_local'],
color=['blue', 'red'], kind='line', figsize=(12,8))
```

But in order to make the line chart suits our needs, more codes should be added to customize the chart itself:

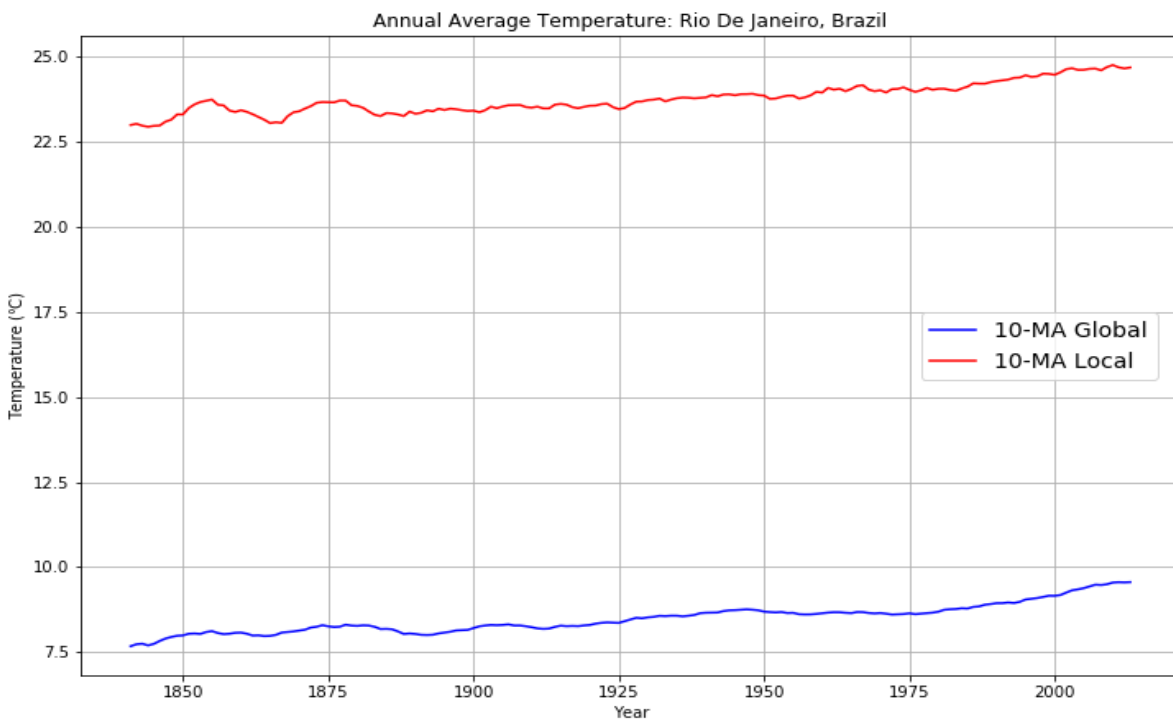
```
# Customization of the line plot
plt.title('Annual Average Temperature: {}, {}'.format('Rio De Janeiro',
'Brazil'))
plt.xlabel('Year')
plt.ylabel('Temperature (°C)')
plt.grid()
plt.legend(['10-MA Global', '10-MA Local'], loc='best',
bbox_to_anchor=(1.0, 0.6), borderaxespad=1, fontsize=14)
```

Then we can show the chart in Jupyter Notebook.

```
# (Optional) Save the line plot to a file
plt.savefig("10-year-MA_{}_{}.png".format('Brazil', 'Rio De Janeiro'))

# Show the line plot in Jupyter Notebook
plt.show()
```

Below is a sample line chart created by Matplotlib.



The drawback of using Matplotlib in Jupyter Notebook is that it can only create static plots (there are many interactive plot samples of Matplotlib on the Internet, but apparently these samples do not work under JupyterLab environments). It would be simpler to use another library if we want to create an interactive line chart, and that's where Bokeh kicked in. Notice that Bokeh can also be configured as a Pandas plotting backend, but a separate installation of another Python package, Pandas-Bokeh, is required. For this project we would just create plots directly with Bokeh library.

Importing Bokeh library requires more steps to take care of:

```
from bokeh.io import output_file, output_notebook, show
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource
from bokeh.models.tools import CrosshairTool, HoverTool
```

Since we are not using Bokeh as a Pandas plotting backend, we have to feed our Pandas DataFrame to ColumnDataSource object in order to use it as a data source of Bokeh plot. Other than that, the logic behind the scenes is similar to creating a plot with Matplotlib.

```
# Use Pandas DataFrame as a Bokeh data source
source = ColumnDataSource(df_combined)

# Define the size of Bokeh plot
p = figure(plot_height=600, plot_width=800)

# Create a line plot with both global and city level data displayed
p.line(x='year', y='10_year_ma_global', color='blue', source=source,
       legend_label='10-MA Global')
p.line(x='year', y='10_year_ma_local', color='red', source=source,
       legend_label='10-MA Local')
```

And customizations are similar, too.

```
# Customization of the line plot
p.title.text = 'Annual Average Temperature: {}, {}'.format('Rio De
Janeiro', 'Brazil')
p.xaxis.axis_label = 'Year'
p.yaxis.axis_label = 'Temperature (°C)'
p.legend.location = "right"
```

And the similarity between the two libraries ends here. Next we add some interactive features which make it easier to visualize temperature data.

```
# Create a tooltip to display labels and values when hovering over a plot

hover = HoverTool()

hover.tooltips=[
    ('Year', '@year'),
    ('10-MA Global', '@10_year_ma_global'),
    ('10-MA Local', '@10_year_ma_local')
]

# Use mode 'vline' instead of 'mouse' (default)
hover.mode='vline'

# Use CrosshairTool to enable a sliding-bar like user experience
crosshair = CrosshairTool()

# Add these defined tools to Bokeh plot
p.add_tools(hover, crosshair)
```

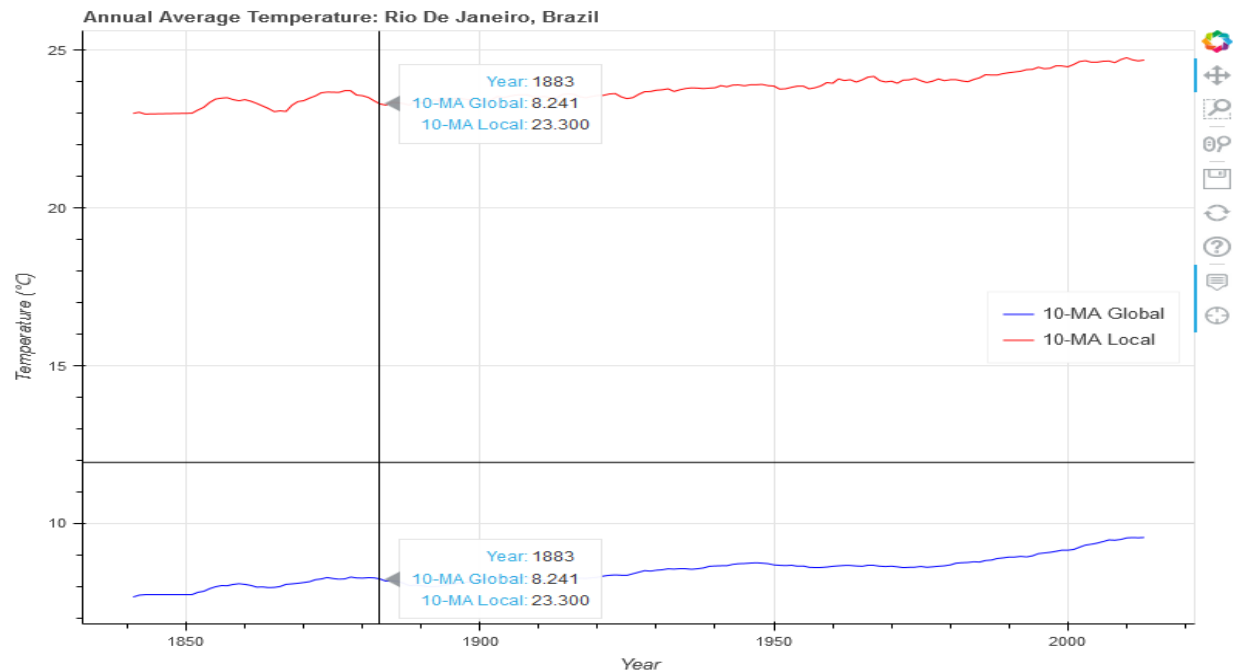
Once again, it is time to show the chart in Jupyter Notebook.

```
# (Optional) Save the line plot to a file
output_file('10-year-MA_{}_{}.html'.format('Brazil', 'Rio De Janeiro'))

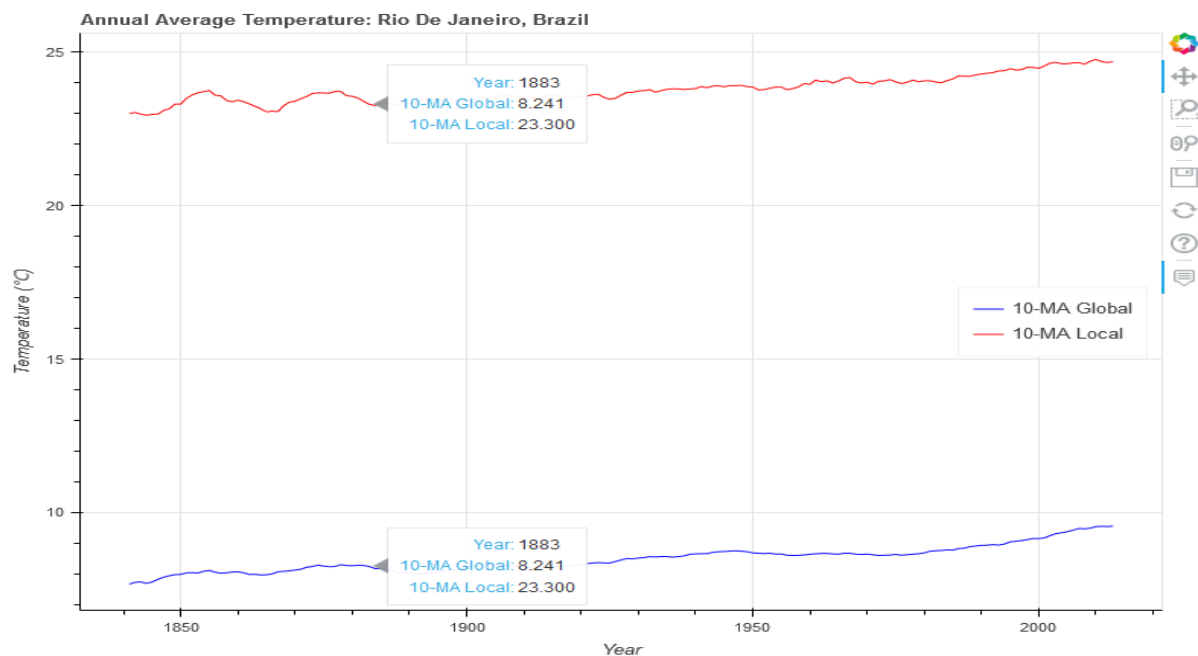
# Enable Jupyter Notebook to display the line plot
output_notebook()

# Show the line plot in Jupyter Notebook
show(p)
```

Below is a sample line chart created by Bokeh.



From the picture above we can see the chart has 2 interactive features enabled. First it will use a tooltip to show current data values when you hover your mouse cursor over the chart. The 'vline' mode also makes the mouse cursor to slide through the entire chart at ease. Secondly, the crosshair cursor makes it easier to observe all the ups and downs and go over all parts of the line chart. As a comparison, below is the same chart with this feature disabled. The position of cursor cannot be captured in the picture.



In this document we will use charts created by Matplotlib to finish this project. Charts created by Bokeh should be used in conjunction with JupyterLab to have a real feeling of interactive plots powered by Bokeh.

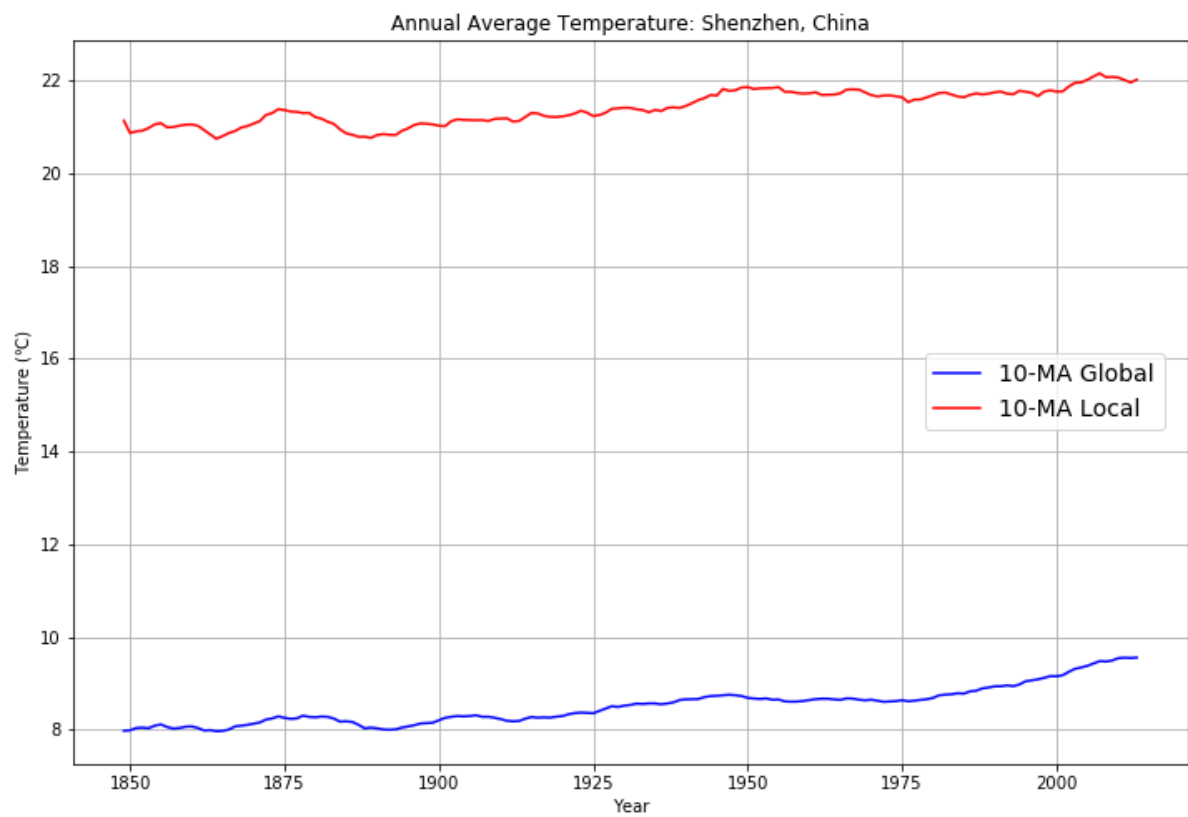
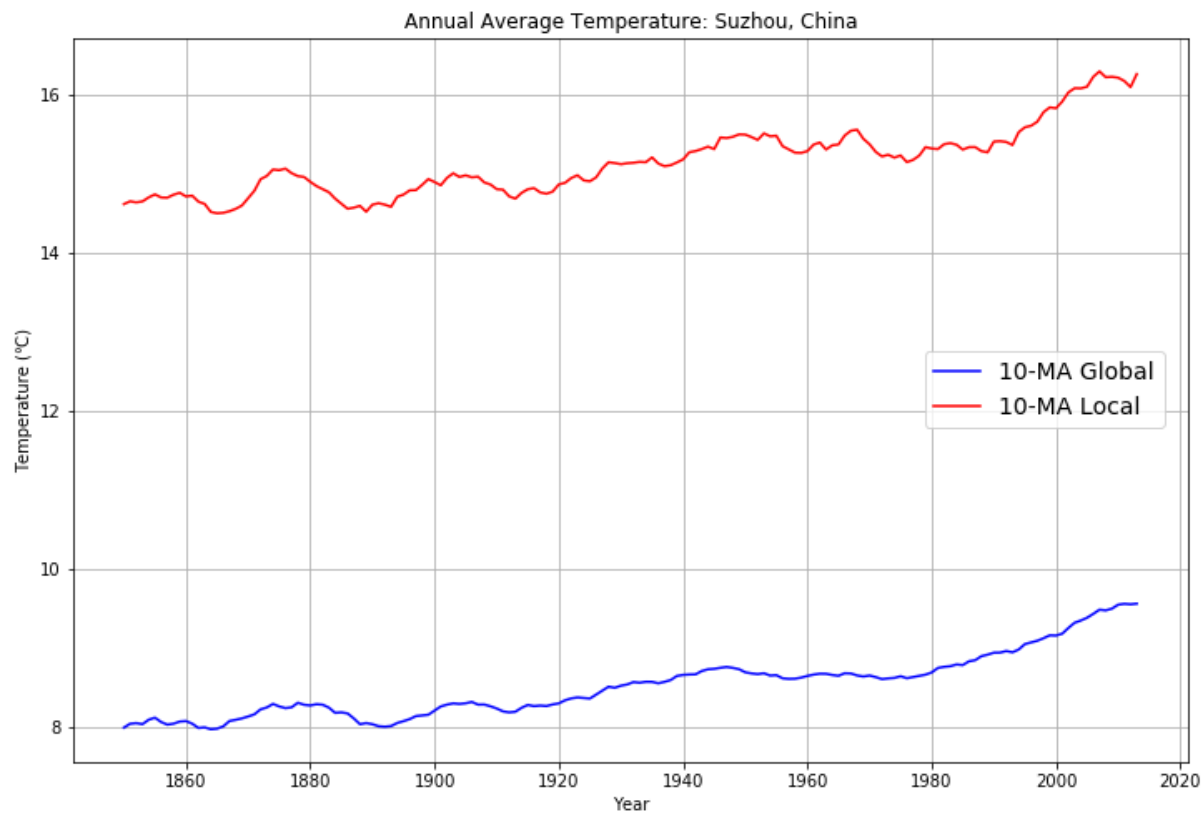
## 5. Observations

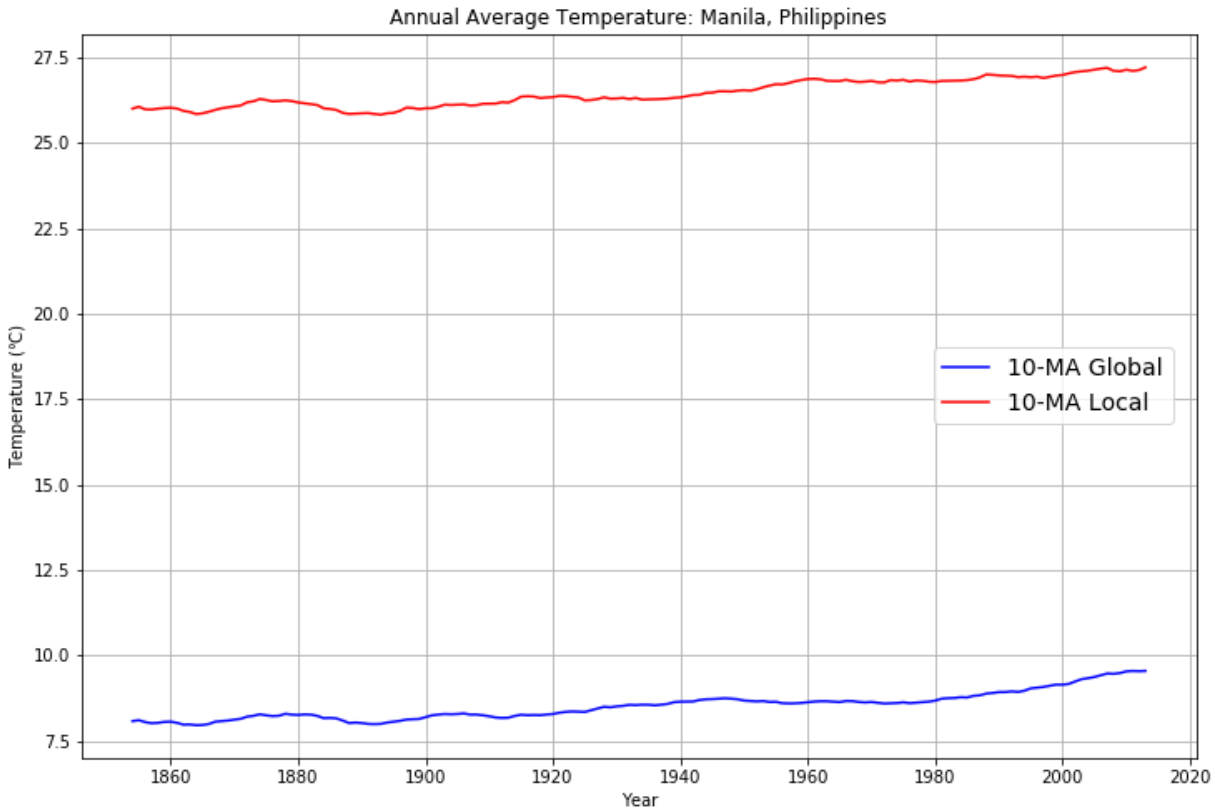
As a Udacity student, to find the closest big city to where I live (Taipei, Taiwan), I will use the [Distance Calculator](#) offered by [timeanddate.com](http://timeanddate.com) to find candidate cities that fit this condition. Based on the calculation results, below is the list of top 3 cities that have the shortest distance to Taipei:

1. Suzhou, China (698 km)
2. Shenzhen, China (810 km)
3. Manila, Philippines (1161 km)

We will try to observe temperature trends in all three cities to get a more accurate view of the region weather surrounding Taiwan. When looking at line charts of the three cities, keep an eye on the following aspects:

- Identify the overall direction of the line chart. Is it heading up, down, or neither up nor down?
- Assume we see an overall direction in the chart. Is it possible to tell at which point in time does this direction become obvious?
- Both global and city level temperatures began to head toward their respective directions at some points in the past. Did they happen in consecutive past years, or was there a noticeable latency between them?





Following is a list of facts validated by inspecting the line charts of the three cities:

1. Over the years, all the three cities have average temperatures higher than the global average. However, we may see different results in cities located at higher latitudes.
2. There is a region in the line charts that resembles “the bottom of a valley” when we look at city level and global data during 1888-1893. From the region we see both the two cities in China started to head up from 1889, Manila in Philippines from 1893, and global data (aligned with selected cities) from 1892. After these points in time, the average temperature never fell back to a value lower than these years.
3. During 1893-1923, the average temperature of all three cities in the region rises at a rate higher than the global average. While Manila gained +0.52, Shenzhen +0.587, Suzhou +0.461, the global average only gained +0.362 in the thirty-year period.
4. The overall trend of the global data shows the world is getting hotter, and the three cities in the region also follow the same trend over time. This can be confirmed by either looking at individual line charts of global and three cities, or calculating the differences of temperatures between the start and the end of each dataset.

This concludes the project.