Explore weather trends

September 16, 2020

Leon de Jong

1 Extracting data from SQL files

Extracting all data from the global_data schema:

`SELECT * FROM global_data`

I want to examine the city data for the last to city's I've lived in. Therefore I need to find the cities available in the city_list database.

'SELECT * FROM city_list WHERE country = 'Netherlands' OR country = "Denmark";

The only available cities in The Netherlands and Denmark are *Amsterdam* and *Copenhagen*. Therefore I shall extract the data for Amsterdam and Copenhagen from the city_data schema using:

`SELECT * FROM city_data WHERE city = 'Copenhagen' OR city='Amsterdam';`

Both resulting .csv files are stored locally and names *globale_data.csv* and *city_data.csv* respectively.

1.1 Preparing the Jupyter Notebook to read the data and do a first exploratory data analysis

As I have taken an introductory course in Python, I decided to analyse the data in Python utilising pandas and numpy for data handling and calculations and bokeh for plotting.

[125]:
```
import pandas as pd
import numpy as np
from bokeh.plotting import figure, show, output_file
from bokeh.io import output_notebook # allows for bokeh plots to be rendered␣
↪directly in the jupyter notebook
from bokeh.io import export_png # allows the export of bokeh plots to .png in␣
↪order to be included in the pdf
```

1.2 Importing the CSV files and preparing pandas for analysis

The two csv files are imported in pandas. Because I am evaluating two cities, the data from *city_data.csv* needs to be split into data for Amsterdam and Copenhagen. Therefore two new pandas are created.

```
[26]: #import city_data.csv
city = pd.read_csv(r"C:\Users\lpede\OneDrive\Data Analyst Nano Degree\Project 1␣
 ↪Weather data\city_data.csv")
#extract Amsterdam data from city
city_amsterdam = city[city['city']  == "Amsterdam"]
#extract Copenhagen data from city
city_copenhagen = city[city['city']  == "Copenhagen"]
#import global_data.csv
world = pd.read_csv(r"C:\Users\lpede\OneDrive\Data Analyst Nano Degree\Project 1␣
 ↪Weather data\global_data.csv")
```

```
[27]: # testing wether all pandas are imported correctly

# city data
print(city.head())
# Amsterdam data
print(city_amsterdam.head())
# Copenhagen data
print(city_copenhagen.head())
# World temperature data
print(world.head())
```

```
   year       city      country  avg_temp
0  1743  Amsterdam  Netherlands      7.43
1  1744  Amsterdam  Netherlands     10.31
2  1745  Amsterdam  Netherlands      3.06
3  1746  Amsterdam  Netherlands       NaN
4  1747  Amsterdam  Netherlands       NaN
   year       city      country  avg_temp
0  1743  Amsterdam  Netherlands      7.43
1  1744  Amsterdam  Netherlands     10.31
2  1745  Amsterdam  Netherlands      3.06
3  1746  Amsterdam  Netherlands       NaN
4  1747  Amsterdam  Netherlands       NaN
     year        city  country  avg_temp
271  1743  Copenhagen  Denmark      6.37
272  1744  Copenhagen  Denmark      9.29
273  1745  Copenhagen  Denmark      0.09
274  1746  Copenhagen  Denmark       NaN
275  1747  Copenhagen  Denmark       NaN
   year  avg_temp
0  1750      8.72
1  1751      7.98
2  1752      5.78
3  1753      8.39
4  1754      8.47
```

The data seems to have been imported correctly. The data for cities has been split in order to perform specific analysis for Amsterdam and Copenhagen. The next step is to evaluate the data

1.3 Initial analysis

Utlising the info command e.g. `city.info()` some basic characteristics of the data will be given that gives a first idea of the quality and quantity of the data.

[29]:
```
city.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 542 entries, 0 to 541
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      542 non-null    int64
 1   city      542 non-null    object
 2   country   542 non-null    object
 3   avg_temp  534 non-null    float64
dtypes: float64(1), int64(1), object(2)
memory usage: 17.1+ KB
```

We see that the city database contains 542 lines, with 8 missing values for average temperatures.

[48]:
```
print("DataFrame info for Amsterdam")
city_amsterdam.info()
print("DataFrame info for Copenhagen")
city_copenhagen.info()
```

```
DataFrame info for Amsterdam
<class 'pandas.core.frame.DataFrame'>
Int64Index: 271 entries, 0 to 270
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      271 non-null    int64
 1   city      271 non-null    object
 2   country   271 non-null    object
 3   avg_temp  267 non-null    float64
dtypes: float64(1), int64(1), object(2)
memory usage: 10.6+ KB
DataFrame info for Copenhagen
<class 'pandas.core.frame.DataFrame'>
Int64Index: 271 entries, 271 to 541
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      271 non-null    int64
 1   city      271 non-null    object
 2   country   271 non-null    object
```

```
 3   avg_temp  267 non-null    float64
dtypes: float64(1), int64(1), object(2)
memory usage: 10.6+ KB
```

Displaying the same information for Amsterdam and Copenhagen respectively shows the missing values are evenly distributed between the two cities with 4 missing values each.

[33]:
```python
world.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 266 entries, 0 to 265
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   year      266 non-null    int64
 1   avg_temp  266 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 4.3 KB
```

We see that the world temperature data has 266 entries, 5 less then the datasets for Amsterdam and Copenhagen. Following this simple analysis there are several questions to be answered

1. What is the range of overlapping years between the world and city datasets
2. What is the best way to deal with the missing values in the city datasets

In order to find out the anwer to question number one, we explore the minimum and maximum values for the city_amsterdam and world data.

[47]:
```python
print("min and max years for Copenhagen data")
print(city_copenhagen["year"].min())
print(city_copenhagen["year"].max())
print("min and max years for Amsterdam data")
print(city_amsterdam["year"].min())
print(city_amsterdam["year"].max())
print("min and max years for world data")
print(world["year"].min())
print(world["year"].max())
```

```
min and max years for Copenhagen data
1743
2013
min and max years for Amsterdam data
1743
2013
min and max years for world data
1750
2015
```

Based on the results above it stands to reason to compare the temperatures between the world data and the data for Amsterdam and Copenhagen respectively between 1750 and 2013 as this is the overlapping window between the datasets. Secondly we deal with the missing values in the Amsterdam and Copenhagen data.

[56]:
```
null_copenhagen = city_copenhagen[city_copenhagen.isna().any(axis=1)]
null_amsterdam = city_amsterdam[city_amsterdam.isna().any(axis=1)]
print("Copenhagen")
print(null_copenhagen)
print("Amsterdam")
print(null_amsterdam)
```

```
Copenhagen
     year        city  country  avg_temp
274  1746  Copenhagen  Denmark       NaN
275  1747  Copenhagen  Denmark       NaN
276  1748  Copenhagen  Denmark       NaN
277  1749  Copenhagen  Denmark       NaN
Amsterdam
   year       city      country  avg_temp
3  1746  Amsterdam  Netherlands       NaN
4  1747  Amsterdam  Netherlands       NaN
5  1748  Amsterdam  Netherlands       NaN
6  1749  Amsterdam  Netherlands       NaN
```

This analysis shows that both Amsterdam and Copenhagen have missing average temperatures for the same years between 1746 and 1749. This makes the deletion of this range from all three datasets the obvious solution. We do this dropping the rows with Nan values.

[76]:
```
#remove rows from amsterdam and copenhagen with indexes as above
ams_new = city_amsterdam.dropna(axis = 0)    #dropping the rows, and having␣
 ↪changes in place generated a copy in place error
cph_new = city_copenhagen.dropna(axis = 0)
print(city_amsterdam.head())
print(city_copenhagen.head())
```

```
   year       city      country  avg_temp
0  1743  Amsterdam  Netherlands      7.43
1  1744  Amsterdam  Netherlands     10.31
2  1745  Amsterdam  Netherlands      3.06
7  1750  Amsterdam  Netherlands     10.04
8  1751  Amsterdam  Netherlands      9.63
     year        city  country  avg_temp
271  1743  Copenhagen  Denmark      6.37
272  1744  Copenhagen  Denmark      9.29
273  1745  Copenhagen  Denmark      0.09
278  1750  Copenhagen  Denmark      8.89
279  1751  Copenhagen  Denmark      8.33
```

Error correction utilising the command `city_amsterdam.dropna(axis = 0, inplace = True)` and `city_copenhagen.dropna(axis =0, inplace = True)` generated a copy in place error. Therefore we set new pandas, and reassign them to city_amsterdam and city_copenhagen in the following step

[78]:
```
city_amsterdam = ams_new
city_copenhagen = cph_new
```

1.4 Analysis

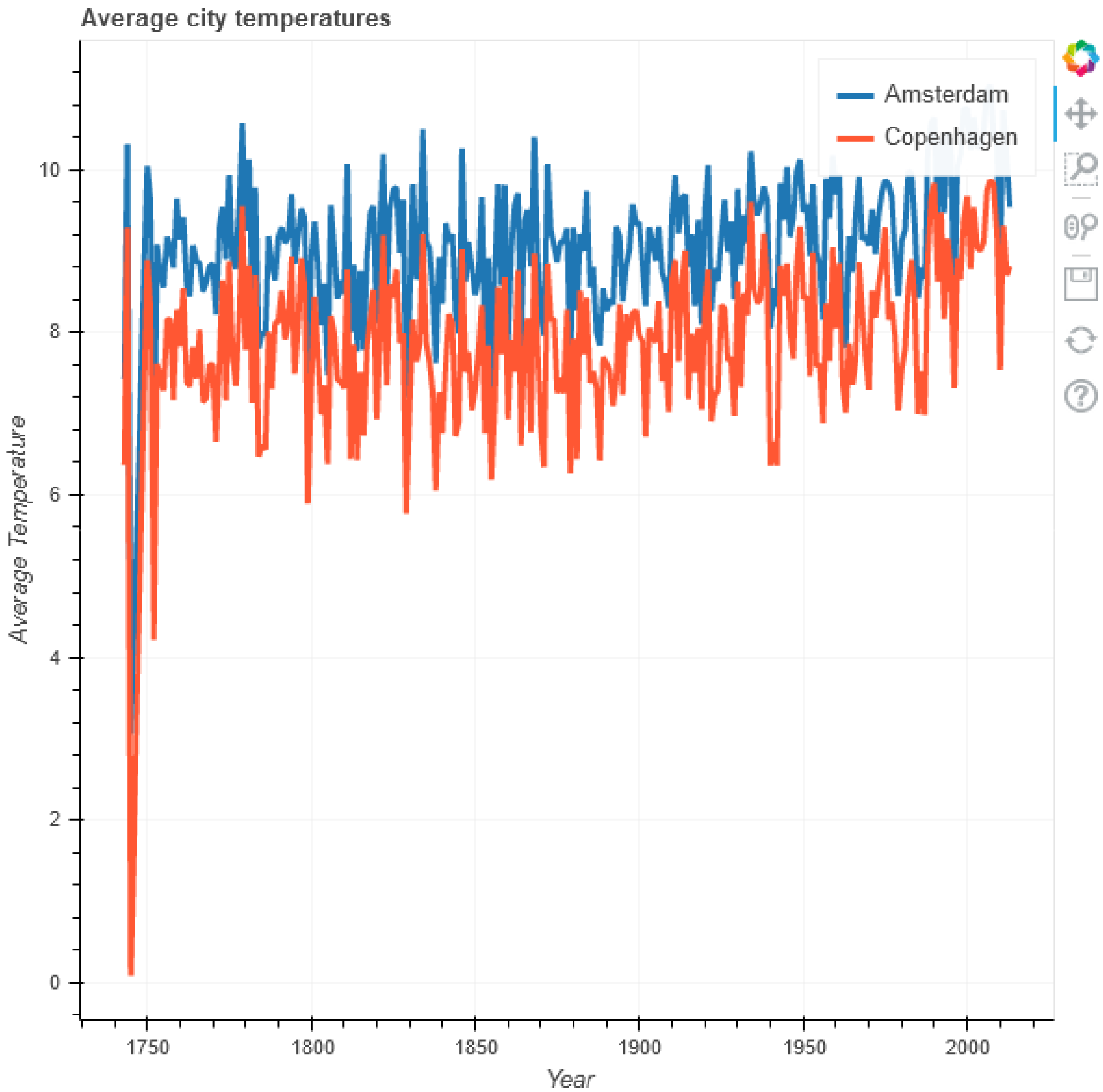
The following shows the calculation of the moving average and a display of it's impact on the readibility of the graphs. The data is presented as line graphs plotted in the same figure, as to make comparisons easier. In order to compare between the raw data and a moving average, first the raw data is plotted using *bokeh*.

[99]:
```
p = figure(title = "Average city temperatures")
p.grid.grid_line_alpha = 0.3
p.xaxis.axis_label = "Year"
p.yaxis.axis_label = "Average Temperature"
p.line(city_amsterdam["year"], city_amsterdam["avg_temp"], line_width = 3,␣
 ↪legend_label = "Amsterdam")
p.line(city_copenhagen["year"], city_copenhagen["avg_temp"], line_width = 3,␣
 ↪color = "#ff5733", legend_label = "Copenhagen")
```

[99]:
```
GlyphRenderer(id='1762', ...)
```

[92]:
```
output_notebook()
```

[127]:
```
show(p)
export_png(p, filename = "p.png")
```



These lines are prette jagged, therefore a smoothing using a moving average is necessary. We can use the inbuilt panda function

moving average The moving average can be calculated using a in built panda function `df['column name'] = df.iloc[rows,column].rolling(window= x).mean()` We will take a 5 year rolling average to start and see if this smooths out the graph

[114]:
```
city_amsterdam['temp_mov'] = city_amsterdam.iloc[:,3].rolling(window=5).mean()
city_copenhagen['temp_mov'] = city_copenhagen.iloc[:,3].rolling(window=5).mean()
world['temp_mov'] = world.iloc[:,1].rolling(window=5).mean()
```

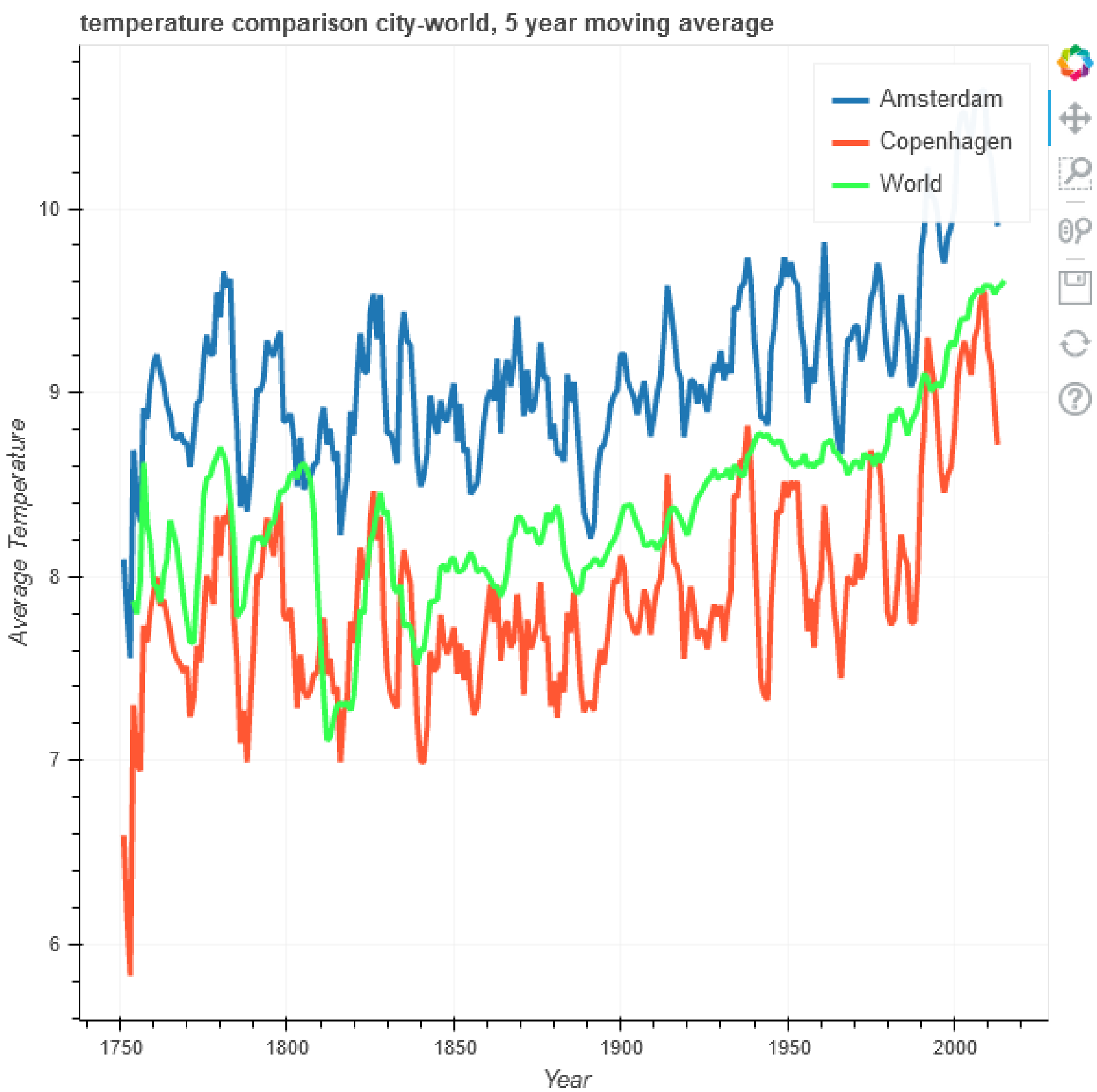
Plotting the new lines using the *temp_mov* parameters in each set

[121]:
```
p_smooth = figure(title = "temperature comparison city-world, 5 year moving␣
 ↪average")
p_smooth.grid.grid_line_alpha = 0.3
```

```
p_smooth.xaxis.axis_label = "Year"
p_smooth.yaxis.axis_label = "Average Temperature"
p_smooth.line(city_amsterdam["year"], city_amsterdam["temp_mov"], line_width =␣
↪3, legend_label = "Amsterdam")
p_smooth.line(city_copenhagen["year"], city_copenhagen["temp_mov"], line_width =␣
↪3, color = "#ff5733", legend_label = "Copenhagen")
p_smooth.line(world["year"], world["temp_mov"], line_width = 3, color =␣
↪"#33ff4f", legend_label = "World",)
```

[121]: GlyphRenderer(id='2301', ...)

[126]:
```
show(p_smooth)
export_png(p_smooth, filename = "p_smooth.png")
```



1.5 Observations

1.5.1 World data

It is clear that the world data is much more consistent, even with a 5 year moving average applied both the Amsterdam and Copenhagen data show large spikes. The world data shows a clear trend towards higher temperatures starting at +/- 1850. It is also curious that the spikes, present bevore 1850 seem to smooth out. Perhaps this is due to more and better measurements, presumably also from more sources in the modern era.

1.5.2 Amsterdam

The Amsterdam data shows an average temperature consistently above the world average temperature. The dips and highs of the spikes correspond to those in the world and copenhagen data. However, as said before the world data smooths out in the five year average compared to the city data. As a result, the large dip in average tempreature around 1945 in the amsterdam data is not present in the world data. The winter of 1945 is described in history as notoriously cold. Perhaps this winter was a European phenomenon, corroborated by the Copenhagen data, which also shows a large dip around the same time.

1.5.3 Copenhagen

The Copenhagen data shows Copenhagen to be in line with the world data until +/- 1850, after which the rise in temperatures seen in the world data is less in the Copenhagen data. Average temperature in Copenhagen stays consistently below world average temperatures from that point forward. Copenhagen being located relatively far to the north compared to the rest of the world, may account for this. However, Amsterdam being als relatively northerly is consistently above the world average temperature.