Database Queries and Interactive Visualizations with Python

Haoran Jia

2023-10-22

Table of contents

In this tutorial, we’ll work with sqlite3, plotly, and pandas on a climate dataset. By the end, you should be comfortable with creating a SQLite database using Python, querying with SQL, and crafting customized graphs with Plotly.

**Note:** For the successful rendering of the interactive plots, run the following.

import plotly.io as pio  
pio.renderers.default="iframe"

### 0.1 Data Import

First, we need to read in three required tables with pandas, and then we will be creating a database based on these tables.

import pandas as pd  
  
# Read from files  
stations = pd.read\_csv("station-metadata.csv")  
temps = pd.read\_csv('temps\_stacked.csv')  
countries = pd.read\_csv('countries.csv')

Let’s take a look at each dataset.

stations.head()

|  | ID | LATITUDE | LONGITUDE | STNELEV | NAME |
| --- | --- | --- | --- | --- | --- |
| 0 | ACW00011604 | 57.7667 | 11.8667 | 18.0 | SAVE |
| 1 | AE000041196 | 25.3330 | 55.5170 | 34.0 | SHARJAH\_INTER\_AIRP |
| 2 | AEM00041184 | 25.6170 | 55.9330 | 31.0 | RAS\_AL\_KHAIMAH\_INTE |
| 3 | AEM00041194 | 25.2550 | 55.3640 | 10.4 | DUBAI\_INTL |
| 4 | AEM00041216 | 24.4300 | 54.4700 | 3.0 | ABU\_DHABI\_BATEEN\_AIR |

temps.head()

|  | ID | Year | Month | Temp |
| --- | --- | --- | --- | --- |
| 0 | ACW00011604 | 1961 | 1 | -0.89 |
| 1 | ACW00011604 | 1961 | 2 | 2.36 |
| 2 | ACW00011604 | 1961 | 3 | 4.72 |
| 3 | ACW00011604 | 1961 | 4 | 7.73 |
| 4 | ACW00011604 | 1961 | 5 | 11.28 |

countries.head()

|  | FIPS 10-4 | ISO 3166 | Name |
| --- | --- | --- | --- |
| 0 | AF | AF | Afghanistan |
| 1 | AX | - | Akrotiri |
| 2 | AL | AL | Albania |
| 3 | AG | DZ | Algeria |
| 4 | AQ | AS | American Samoa |

Notice that there is no country names in stations. We may want to add an extra column named country to it for future convenience. We can do this by merging stations and countries using the two-letter ID. As a good practice, all operations with dataframe will be enclosed in a function.

def prepare\_df(stations):  
 """  
 Prepare the stations DataFrame by merging it with the countries DataFrame and renaming columns.  
  
 This function extracts the 'FIPS 10-4' code from the 'ID' column of the stations DataFrame,  
 merges the stations DataFrame with the countries DataFrame based on the 'FIPS 10-4' code,  
 drops unnecessary columns, and renames the 'Name' column to 'Country'.  
  
 Parameters:  
 - stations (pd.DataFrame): The stations dataset.  
  
 Returns:  
 - pd.DataFrame: A modified version of the stations DataFrame after merging and renaming operations.  
 """  
  
 # Extract the 'FIPS 10-4' code from the 'ID' column  
 stations['FIPS 10-4'] = stations['ID'].str[:2]  
  
 # Merge the stations DataFrame with the countries DataFrame based on the 'FIPS 10-4' code  
 stations = pd.merge(stations, countries, on='FIPS 10-4')  
  
 # Drop unnecessary columns  
 stations = stations.drop(['FIPS 10-4', 'ISO 3166'], axis=1)  
  
 # Rename the 'Name' column to 'Country'  
 stations.rename(columns={'Name': 'Country'}, inplace=True)  
  
 return stations  
  
stations = prepare\_df(stations)

### 0.2 Creating Databases

In this section, we are going to create a database with three tables we have: temperatures, stations, and countries.

import sqlite3  
  
# Create database  
conn = sqlite3.connect("temps.db")

We observe that temps is extremely large, so we may want to add it to the database by chunks.We can write a function to do this.

def read\_csv\_to\_database(csv\_file\_path, database\_connection):  
 # Read segmented table into database  
 df\_iter = pd.read\_csv(csv\_file\_path, chunksize=100000)  
 for df in df\_iter:  
 df.to\_sql("temperatures", database\_connection, if\_exists="append", index=False)  
  
csv\_file\_path = 'temps\_stacked.csv'  
read\_csv\_to\_database(csv\_file\_path, conn)

Add other two datasets to the database.

# Read other two tables into database  
stations.to\_sql("stations", conn, if\_exists = "replace", index = False)  
countries.to\_sql("countries", conn, if\_exists = "replace", index = False)

279

Always remember to close the dataset connection when you are not using.

conn.close()

### 0.3 A simple Query Function

Connect to the database before using, and then create a cursor to interact with it.

# connect to database  
conn = sqlite3.connect("temps.db")

Let’s first check all three tables are in the database.

# Create a cursor object to interact with the SQLite database  
cursor = conn.cursor()  
  
# Execute an SQL query to fetch the names of all tables in the database  
cursor.execute("SELECT name FROM sqlite\_master WHERE type='table'")  
  
# Print the names of the tables  
print(cursor.fetchall())  
  
# Close the database connection  
conn.close()

[('temperatures',), ('stations',), ('countries',)]

With the data, we can start to ask our first question: ***how does the average yearly change in temperature vary within a given country?***

We may create a map scatter plot where color of dots will represent temperature. But before doing that, we may want to retrieve necessary data frist.

We are going to do this with a query function called query\_climate\_database which accepts four arguments:

* country, a string giving the name of a country for which data should be returned.
* year\_begin and year\_end, two integers giving the earliest and latest years for which should be returned.
* month, an integer giving the month of the year for which should be returned.

The return value is a Pandas dataframe of temperature readings for the specified country, in the specified date range, in the specified month of the year.

def query\_climate\_database(country, year\_begin, year\_end, month):  
 """  
 Fetch climate data for a specified country, year range, and month from an SQLite database.  
  
 This function queries temperature data based on a specified country, year range, and month  
 from an SQLite database and returns the data as a pandas DataFrame.  
  
 Parameters:  
 - country (str): The name of the country for which data is to be fetched.  
 - year\_begin (int): The starting year for the data range.  
 - year\_end (int): The ending year for the data range.  
 - month (int): The month for which data is to be fetched.  
  
 Returns:  
 - pd.DataFrame: A DataFrame containing temperature data for the specified country, year range, and month.  
  
 Example:  
 >>> df = query\_climate\_database('India', 2000, 2020, 1)  
 Fetches temperature data for India for the month of January between the years 2000 and 2020.  
 """  
  
 # Connect to the SQLite database  
 conn = sqlite3.connect("temps.db")  
 cursor = conn.cursor()  
  
 # Execute the SQL query to fetch temperature data  
 cursor.execute(f"SELECT s.NAME, s.LATITUDE, s.LONGITUDE, s.Country, t.Year, t.Month, t.Temp \  
 FROM temperatures t JOIN stations s ON s.ID = t.ID \  
 WHERE s.Country = '{country}' AND Year >= {year\_begin} AND Year <= {year\_end} AND t.Month = {month}")  
  
 # Convert the fetched data to a pandas DataFrame  
 result\_df = pd.DataFrame(cursor.fetchall(), columns=['Name', 'Latitude', 'Longitude',  
 'Country', 'Year', 'Month', 'Temp'])  
  
 # Close the database connection  
 conn.close()  
  
 return result\_df

Let’s see an example usage.

# Example output  
query\_climate\_database(country = "India",  
 year\_begin = 1980,  
 year\_end = 2020,  
 month = 1)

|  | Name | Latitude | Longitude | Country | Year | Month | Temp |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | PBO\_ANANTAPUR | 14.583 | 77.633 | India | 1980 | 1 | 23.48 |
| 1 | PBO\_ANANTAPUR | 14.583 | 77.633 | India | 1980 | 1 | 23.48 |
| 2 | PBO\_ANANTAPUR | 14.583 | 77.633 | India | 1980 | 1 | 23.48 |
| 3 | PBO\_ANANTAPUR | 14.583 | 77.633 | India | 1980 | 1 | 23.48 |
| 4 | PBO\_ANANTAPUR | 14.583 | 77.633 | India | 1980 | 1 | 23.48 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 15755 | DARJEELING | 27.050 | 88.270 | India | 1997 | 1 | 5.70 |
| 15756 | DARJEELING | 27.050 | 88.270 | India | 1997 | 1 | 5.70 |
| 15757 | DARJEELING | 27.050 | 88.270 | India | 1997 | 1 | 5.70 |
| 15758 | DARJEELING | 27.050 | 88.270 | India | 1997 | 1 | 5.70 |
| 15759 | DARJEELING | 27.050 | 88.270 | India | 1997 | 1 | 5.70 |

### 0.4 Visualization 1: Geographic Scatter Plot for Yearly Temperature Increases

Before moving, let’s first compute a simple estimate of the **year-over-year average change in temperature** in each month at each station to add more fun. For this, we’ll use our old friend, linear regression. We’ll use the statistical fact that, when regressing Temp against Year, the coefficient of Year will be an estimate of the yearly change in Temp.

from sklearn.linear\_model import LinearRegression  
  
def coef(data\_group):  
 """  
 Calculate the coefficient of the linear regression model for temperature against year.  
  
 This function fits a linear regression model using the "Year" as the independent variable  
 and "Temp" as the dependent variable. It returns the coefficient of the "Year",  
 which represents the rate of change of temperature with respect to the year.  
  
 Parameters:  
 - data\_group: A pandas grouped object.  
  
 Returns:  
 - float: The coefficient of the "Year" in the linear regression model.  
 """  
  
 # Extract x and y variables from data group  
 x = data\_group[["Year"]]  
 y = data\_group["Temp"]  
  
 # Initialize and fit the linear regression model  
 LR = LinearRegression()  
 LR.fit(x, y)  
  
 # Return the coefficient of the "Year"  
 return LR.coef\_[0]

Now we are going to write a function called temperature\_coefficient\_plot to visualize the data we fetched from the database. This function will accept five explicit arguments, and an undetermined number of keyword arguments.

country, year\_begin, year\_end, and month are as in the previous part. min\_obs, the minimum required number of years of data for any given station. Only data for stations with at least min\_obs years worth of data in the specified month would be plotted; the others would be filtered out. We will use df.transform() plus filtering to achieve this task. \*\*kwargs is an additional keyword arguments passed to px.scatter\_mapbox(). These can be used to control the colormap used, the mapbox style, etc.

The output of this function would be an interactive geographic scatterplot, constructed using Plotly Express, with a point for each station, such that the color of the point reflects an estimate of the yearly change in temperature during the specified month and time period at that station.

from plotly import express as px  
  
# Define a color map for the plot  
color\_map = px.colors.diverging.RdGy\_r  
# Create a mapping for creating titles  
month\_mapping = {  
 1: "January",  
 2: "February",  
 3: "March",  
 4: "April",  
 5: "May",  
 6: "June",  
 7: "July",  
 8: "August",  
 9: "September",  
 10: "October",  
 11: "November",  
 12: "December"  
}  
  
def temperature\_coefficient\_plot(country, year\_begin, year\_end, month, min\_obs, \*\*kwargs):  
 """  
 Generate a scatter map plot showing the estimated yearly increase in temperature for various stations.  
  
 This function queries temperature data for a specified country, month, and year range.  
 It then calculates the coefficient of the linear regression model for temperature against year  
 for each station and plots the stations on a map. The color of each station represents the  
 estimated yearly increase in temperature.  
  
 Parameters:  
 - country (str): The name of the country for which the plot is to be generated.  
 - year\_begin (int): The starting year for the data range.  
 - year\_end (int): The ending year for the data range.  
 - month (int): The month for which the plot is to be generated.  
 - min\_obs (int): Minimum number of observations required for a station to be included in the plot.  
 - \*\*kwargs: Additional keyword arguments for the plotly express scatter\_mapbox function.  
  
 Returns:  
 - plotly.graph\_objs.\_figure.Figure: A Plotly figure object representing the scatter map plot.  
  
 Example:  
 >>> fig = temperature\_coefficient\_plot("India", 1980, 2020, 1, min\_obs=10, zoom=3, center={'lat':23,'lon':80})  
 Generates a scatter map plot for India for the month of January between the years 1980 and 2020.  
 """  
  
 # Query the climate database for the specified parameters  
 temp\_df = query\_climate\_database(country, year\_begin, year\_end, month)  
  
 # Filter stations based on the minimum number of observations  
 temp\_df['min\_obs'] = temp\_df.groupby('Name')['Year'].transform('nunique')  
 temp\_df = temp\_df[temp\_df['min\_obs'] >= min\_obs]  
  
 # Calculate the coefficient for each station and merge it into the main dataframe  
 coefs = temp\_df.groupby(["Name"]).apply(coef)  
 coefs = coefs.reset\_index()  
 temp\_df = pd.merge(temp\_df, coefs, on='Name')  
 temp\_df.rename(columns={0: 'Estimate'}, inplace=True)  
 temp\_df['Estimate'] = temp\_df['Estimate'].round(4) # Round the estimates for better visualization  
 max\_abs\_estimate = temp\_df['Estimate'].abs().max() # Determine the maximum absolute estimate for color scaling  
  
 # Create the scatter map plot using plotly express  
 fig = px.scatter\_mapbox(temp\_df,  
 hover\_name="Name",  
 lat="Latitude",  
 lon="Longitude",  
 color='Estimate',  
 range\_color=[-max\_abs\_estimate, max\_abs\_estimate],  
 \*\*kwargs)  
  
 # Update the layout and title of the plot  
 fig.update\_layout(title\_text=f"""Estimate of yearly increase in temperature in {month\_mapping[month]} for stations in {country} between {year\_begin} to {year\_end}""")  
  
 return fig

Let’s try generating a scatter map plot of temperature for India in January between the years 1980 and 2020.

# Example usage: Generate and display the plot for India  
fig = temperature\_coefficient\_plot("India", 1980, 2020, 1, min\_obs=10,  
 zoom=3,  
 center={'lat':23,'lon':80},  
 mapbox\_style="carto-positron",  
 color\_continuous\_scale=color\_map)  
fig.show()

Unable to display output for mime type(s): text/html

From this graph we can see that the estimate in temperature increase mostly appears near the border lines of India, while inland temperature is expected to drop.

To generalize the fucntion, let’s test with an another example.

fig = temperature\_coefficient\_plot("China", 1960, 1980, 10, min\_obs = 10,  
 zoom = 2.5,  
 center = {'lat':40,'lon':110},  
 mapbox\_style="carto-positron",  
 color\_continuous\_scale=color\_map)  
fig.show()

Unable to display output for mime type(s): text/html

The graphs shows that the stations mostly centered at the South Eastern side of China, while the largest estimate in temperature increase appears in far North and West.

### 0.5 Visualization 2: Line Plot for Yearly and Monthly Average Temperature Change

In this section, we want to ask: ***how does the average temperature in certain country changes over time?***

To answer this question, we may create a line plot where x axis is date and y axis is the average temperature.

Again, we are going to start with a query function called query\_avg\_temp\_by\_country which accepts three arguments: country, year\_begin, and year\_end as in the previous part.

The return value is a Pandas dataframe of temperature readings for the specified country, in the specified date range, in the specified month of the year.

def query\_avg\_temp\_by\_country(country, year\_begin=1901, year\_end=2020):  
 """  
 Fetch average temperature data for a specified country from an SQLite database.  
  
 This function queries temperature data based on a specified country and a year range  
 from an SQLite database and returns the data as a pandas DataFrame.  
  
 Parameters:  
 - country (str): The name of the country for which data is to be fetched.  
 - year\_begin (int, optional): The starting year for the data range. Default is 1901.  
 - year\_end (int, optional): The ending year for the data range. Default is 2020.  
  
 Returns:  
 - pd.DataFrame: A DataFrame containing temperature data for the specified country and year range.  
  
 Example:  
 >>> df = query\_avg\_temp\_by\_country('India', year\_begin=2000, year\_end=2010)  
 Fetches temperature data for India between the years 2000 and 2010.  
 """  
  
 # Connect to the SQLite database  
 conn = sqlite3.connect("temps.db")  
 cursor = conn.cursor()  
  
 # Execute the SQL query to fetch temperature data  
 cursor.execute(f"SELECT s.Country, t.Year, t.Month, t.Temp \  
 FROM temperatures t JOIN stations s ON s.ID = t.ID \  
 WHERE s.Country = '{country}' AND t.Year >= {year\_begin} AND t.Year <= {year\_end}")  
  
 # Convert the fetched data to a pandas DataFrame  
 result\_df = pd.DataFrame(cursor.fetchall(), columns=['Country', 'Year', 'Month', 'Temp'])  
  
 # Close the database connection  
 conn.close()  
  
 return result\_df

Then similarly, we will create a user-friendly function called temp\_trend\_lineplot to help us with the plotting. The function is supposed to takes in country, year\_begin, and year\_end as in the previous part, and returns a line plot show the monthly and yearly change in average temperature.

But beside the baseline, we can have some addition to the interactive plot. We will create two seperate figures for monthly and yearly average temperature, and adds bottons to control the displayed time period.

In the actual inplementation, we need to process the data using groupby() to get the average for a specific time period and to\_datetime function to convert the time into datetime object for plotting.

import plotly.graph\_objects as go  
from plotly.subplots import make\_subplots  
  
def temp\_trend\_lineplot(country, year\_begin=1901, year\_end=2020):  
 """  
 Generate a time series line plot showing the monthly and yearly average temperature for a specified country.  
  
 Parameters:  
 - country (str): The name of the country for which the plot is to be generated.  
 - year\_begin (int, optional): The starting year for the data range. Default is 1901.  
 - year\_end (int, optional): The ending year for the data range. Default is 2020.  
  
 Returns:  
 - A Plotly figure object representing the time series line plot.  
  
 Example:  
 >>> fig = temp\_trend\_lineplot('India')  
 Generates a line plot for India showing monthly and yearly average temperatures.  
 """  
  
 # Fetch average temperature data for the specified country and year range  
 result\_df = query\_avg\_temp\_by\_country(country, year\_begin, year\_end)  
  
 # Calculate monthly average temperatures  
 avgs\_df = result\_df.groupby(['Year', 'Month'])['Temp'].mean().reset\_index()  
 avgs\_df['Date'] = pd.to\_datetime(avgs\_df['Year'].astype(str) + '-' + avgs\_df['Month'].astype(str) + '-01')  
  
 # Calculate yearly average temperatures  
 yearly\_avg = avgs\_df.resample('Y', on='Date').mean()  
  
 # Create subplots for both monthly and yearly averages  
 fig = make\_subplots(rows=2, cols=1,  
 shared\_xaxes=True, vertical\_spacing=0.1,  
 subplot\_titles=('Monthly Average', 'Yearly Average'))  
  
 # Add trace for monthly average temperatures  
 fig.add\_trace(  
 go.Scatter(x=avgs\_df.Date, y=avgs\_df.Temp, name='Monthly Average'),  
 row=1, col=1)  
  
 # Add trace for yearly average temperatures  
 fig.add\_trace(  
 go.Scatter(x=yearly\_avg.index, y=yearly\_avg.Temp, name='Yearly Average'),  
 row=2, col=1)  
  
 # Set the title and layout properties for the plot  
 fig.update\_layout(  
 title\_text=f"Time series average temperature in {country}",  
 height=800  
 )  
  
 # Add a range slider for date selection  
 fig.update\_layout(  
 xaxis=dict(  
 rangeselector=dict(  
 buttons=list([  
 dict(count=6,  
 label="6m",  
 step="month",  
 stepmode="backward"),  
 dict(count=1,  
 label="1y",  
 step="year",  
 stepmode="backward"),  
 dict(count=5,  
 label="5y",  
 step="year",  
 stepmode="backward"),  
 dict(step="all")  
 ])  
 ),  
 rangeslider=dict(  
 visible=False  
 ),  
 type="date"  
 )  
 )  
 return fig

# Example usage: Generate and display the plot for India  
fig = temp\_trend\_lineplot('India')  
fig.show()

Unable to display output for mime type(s): text/html

It seems that the seasonal temperature change are pretty regular in the long term, but in general, the yearly average temperature plot suggests that the temperature in India keeps getting warm from 1901 to recent years, though with some fluctuations.

See an another example.

fig = temp\_trend\_lineplot('United States', year\_begin = 2000, year\_end = 2020)  
fig.show()

Unable to display output for mime type(s): text/html

The seasonal pattern is also clear in the US, but it fluctuates at a much lower level compared with India, from around 0 to 25, while India’s temperature varies from around 16 to 30.

### 0.6 Visualization 3: Stacked Histogram With Box Plot for Temperature Distribution

In this section, we want to answer the question: ***how does the distribution of temperature differ between different countries***?

As usual, the first task is to create a query function to retrieve the data. We will create a function called query\_temp\_by\_country that takes four parameters, one required list-type argument countries, and three optional arguments: month,year\_begin, year\_end. If month is not specified, then it will return all temperature data within the time period. The return would be a DataFrame containing temperature data for the specified countries, month, and year range.

def query\_temp\_by\_country(countries, month=0, year\_begin=1901, year\_end=2020):  
 """  
 Fetch temperature data for specified countries from an SQLite database.  
  
 This function queries temperature data based on a list of countries, a specified month,  
 and a year range from the database. The data is returned as a pandas DataFrame.  
  
 Parameters:  
 - countries (list of str): A list of country names for which data is to be fetched.  
 - month (int, optional): The month for which data is to be fetched. Default is 0, which fetches data for all months.  
 - year\_begin (int, optional): The starting year for the data range. Default is 1901.  
 - year\_end (int, optional): The ending year for the data range. Default is 2020.  
  
 Returns:  
 - pd.DataFrame: A DataFrame containing temperature data for the specified countries, month, and year range.  
  
 Example:  
 >>> query\_temp\_by\_country(['Zimbabwe', 'USA'], month=1, year\_begin=2000, year\_end=2010)  
 Returns a DataFrame with temperature data for Zimbabwe and USA for the month of January between the years 2000 and 2010.  
 """  
  
 conn = sqlite3.connect("temps.db")  
 cursor = conn.cursor()  
  
 # Create a placeholder for each country in the list  
 placeholders = ', '.join(['?'] \* len(countries))  
  
 # Using parameterized query for different input length  
 query\_str = f"""SELECT t.Year, t.Temp, s.Country, t.Month  
 FROM temperatures t JOIN stations s  
 ON s.ID = t.ID  
 WHERE s.Country IN ({placeholders}) AND t.Year >= ? AND t.Year <= ?"""  
  
 # List of parameters for the query  
 params = countries + [year\_begin, year\_end]  
  
 # Add restrictions to month depending on the input  
 if month > 0:  
 query\_str += ' AND t.Month = ?'  
 params.append(month)  
  
 # Query and get results  
 cursor.execute(query\_str, params)  
 result\_df = pd.DataFrame(cursor.fetchall(), columns=['Year', 'Temp', 'Country', 'Month'])  
 conn.close()  
  
 return result\_df

With the query function, we can start working on the actual plotting function. We will create a function named stacked\_histogram that takes in the same arguments as query\_temp\_by\_country. Since the input is a bit different as previous functions, we hope to remind the user of the data type. In Python Raises...TypeError allows us to do this.

As return, we hope to get a stacked histogram for a list of countries, with their box plots, where the y axis suggest the number of observations, and x axis specifies temperature bins.

def stacked\_histogram(countries, month=0, year\_begin=1901, year\_end=2020):  
 """  
 Create a stacked histogram of temperature data for specified countries.  
  
 This function fetches temperature data based on a list of countries, a specified month,  
 and a year range using the `query\_temp\_by\_country` function. It then creates a stacked histogram  
 using Plotly Express and returns the figure.  
  
 Parameters:  
 - countries (list of str): A list of country names for which the histogram is to be created.  
 - month (int, optional): The month for which the histogram is to be created. Default is 0, which uses data for all months.  
 - year\_begin (int, optional): The starting year for the data range. Default is 1901.  
 - year\_end (int, optional): The ending year for the data range. Default is 2020.  
  
 Returns:  
 - A Plotly figure object representing the stacked histogram.  
  
 Example:  
 >>> fig = stacked\_histogram(['Zimbabwe', 'United States'], month=1, year\_begin=2000, year\_end=2010)  
 Creates and returns a stacked histogram for Zimbabwe and USA for the month of January between the years 2000 and 2010.  
 """  
 if type(countries) != list:  
 raise TypeError('Countries must be a list.')  
  
 # retrieve data from database  
 result\_df = query\_temp\_by\_country(countries, month, year\_begin, year\_end)  
 # plot a stacked histogram  
 fig = px.histogram(result\_df, x="Temp", color="Country", marginal='box')  
 # add a title  
 title\_text = f"Stacked histogram of historical termperature distribution between {year\_begin} and {year\_end}"  
 if month>0:  
 title\_text += f" in {month\_mapping[month]}"  
   
 # Set the title  
 fig.update\_layout(  
 title\_text=title\_text  
 )  
  
 return fig

# Let's try an extreme cases.  
stacked\_histogram(['Zimbabwe','Netherlands'])

Unable to display output for mime type(s): text/html

We can tell that the two countries have different climate patterns since their interquartile range did not overlap. Also, the height of the stacked chart suggest probably these two countries have similar numbers of stations.

# One more example  
stacked\_histogram(['Argentina', 'China','Canada'], month = 10)

Unable to display output for mime type(s): text/html

From this graph, we can tell that Canada have many extreme values below 0, as the outliers. China has the longest interquartile range, suggesting a larger variation in climate patterns within the country.