title: "PIC16B HW1:Data Wrangling and Visualization" author: "Zhe Shen" date: "2023-1-28" categories: [PIC16B HW]

HW1:Data Wrangling and Visualization

In this assignment, we will use NOAA climate data to create several interesting interactive data graphs and discover several interesting phenomena.

1. Create a Database

First, we need to create a database with three tables: **temperatures**, **stations**, and **countries** and keep these as three separate tables in your database.This step will help us extract or find data faster and more efficiently by storing it in a database.

import package

In [1]:

```
import pandas as pd
from plotly import express as px
import sqlite3
import calendar
import numpy as np
from sklearn.linear_model import LinearRegression
```

connect local sqlite

In [2]:

```
conn = sqlite3.connect("temps.db")
```

load temperatures data

In [3]:

```
df = pd.read_csv("temps_stacked.csv")
df.to_sql("temperatures", conn, index=False, if_exists='replace')
```

load countries data

In [4]:

```
df = pd.read_csv("countries.csv")
df.to_sql("countries", conn, index=False, if_exists='replace')
```

```
C:\Users\shenz\anaconda3\lib\site-packages\pandas\core\generic.py:2779: UserWarning: The spaces in these column names will not
be changed. In pandas versions < 0.14, spaces were converted to underscores.
  sql.to_sql(
```

load stations data

In [5]:

```
df = pd.read_csv("station-metadata.csv")
df.to_sql("stations", conn, index=False, if_exists='replace')
```

2. Write a Query Function

In second part， In order to check the temperature of a particular country for a particular month at a particular time， we will write a Python function called **query_climate_database()** which accepts four arguments:

- country, a string giving the name of a **country** for which we specified.
- **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 we specified.

The return value of this function would be a Pandas dataframe of temperature readings for the specified country, in the specified year range, in the specified month of the year. This dataframe include columns as follows:

- The station name.
- The latitude and longitude of the station.
- Thethe country name in which the station is located.

- The year which we specified.
- The month which we specified.
- The average temperature at the specified station during the specified year and month.

In [6]:

```
def query_climate_database(country, year_begin, year_end, month):
    # sql command
    cmd = \
        """
        SELECT S.NAME, S.LATITUDE, S.LONGITUDE, C.Name AS Country, T.Year, T.Month, T.Temp
        FROM temperatures T
        INNER JOIN stations S ON T.id = S.id
        INNER JOIN countries C ON SUBSTRING(S.id,1,2) = C.'FIPS 10-4'
        WHERE C.Name='{}' AND T.Year>={} AND T.Year<={} AND T.Month={}
        """.format(country, year_begin, year_end, month)
    # call pandas sql query function
    dread = pd.read_sql_query(cmd, conn)
    return dread
```

Test Case

Query the temperature records of all stations in India from 1980 to January 2020

In [7]:

```
country_temps = query_climate_database(country="India", year_begin=1980, year_end=2020, month=1)
country_temps
```

Out[7]:

	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	1981	1	24.57
2	PBO_ANANTAPUR	14.583	77.633	India	1982	1	24.19
3	PBO_ANANTAPUR	14.583	77.633	India	1983	1	23.51
4	PBO_ANANTAPUR	14.583	77.633	India	1984	1	24.81
...	...	...	...	...	...	...	...
3147	DARJEELING	27.050	88.270	India	1983	1	5.10
3148	DARJEELING	27.050	88.270	India	1986	1	6.90
3149	DARJEELING	27.050	88.270	India	1994	1	8.10
3150	DARJEELING	27.050	88.270	India	1995	1	5.60
3151	DARJEELING	27.050	88.270	India	1997	1	5.70

3152 rows × 7 columns

3. Write a Geographic Scatter Function for Yearly Temperature Increases

In this part, we will explore a more complex issue by create a function to create visualizations:

How does the average yearly change in temperature vary within a given country?

We create and call this function as **temperature_coefficient_plot()**. This function will accept five explicit arguments, and an undetermined number of keyword arguments. Especially worth mentioning are the following arguments:

- **min_obs**, the minimum required number of years of data for any given station.
- ****kwargs**, 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. We can see 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. This change value is calculated by a linear regression model.

In [8]:

```
def temperature_coefficient_plot(country, year_begin, year_end, month, min_obs, zoom, mapbox_style, **kwargs):
    df = query_climate_database(country, year_begin, year_end, month)
    df['count'] = df.groupby('NAME')['Year'].transform('count')
    df = df[(df['count'] >= min_obs)]
    df['Estimated Yearly Increase(° C)'] = df.groupby('NAME')['Temp'].transform('std')
    df.round({'Estimated Yearly Increase(° C)':4})

    max_val = abs(df['Estimated Yearly Increase(° C)'].max())
    min_val = abs(df['Estimated Yearly Increase(° C)'].min())
    if min_val > max_val:
        max_val = min_val
    # set title use args
    title = 'Estimates of yearly increase in temperature in {} for stations in {}, years {} - {}'.format(calendar.month_name[month], country,

    fig = px.scatter_mapbox(df,
                        lat = "LATITUDE",
                        lon = "LONGITUDE",
                        hover_name = "NAME",
                        color = "Estimated Yearly Increase(° C)",
                        title = title,
                        zoom = zoom,
                        opacity = 0.5,
                        height = 500,
                        mapbox_style = mapbox_style,
                        range_color  = [max_val * -1, max_val],
                        **kwargs)

    fig.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
    return fig
```

In [9]:

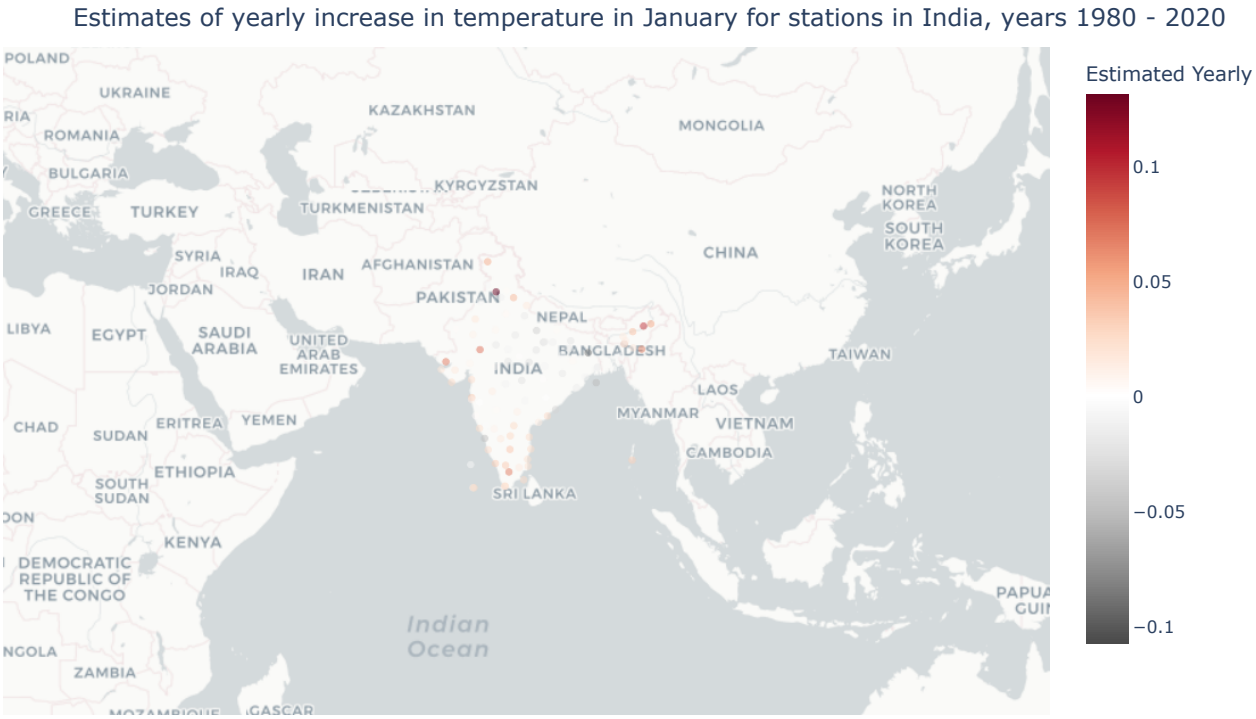
```
def temperature_coefficient_plot(country, year_begin, year_end, month, min_obs, zoom, mapbox_style, **kwargs):

    # compute the first coefficient of a linear regression model at that station
    def get_rate_from_linear(data):

        # Fits and applies linear regression to data.
        linear = LinearRegression()
        linear.fit(np.array(data['Year']).reshape(-1, 1), data['Temp'])
        rate = linear.coef_[0]
        return round(rate, 4)

    # get data by call query_climate_database function
    df = query_climate_database(country, year_begin, year_end, month)
    # count the number of records in each station
    df['count'] = df.groupby('NAME')['Year'].transform('count')
    # filtered less than min obs station
    df = df[(df['count'] >= min_obs)]
    # set title
    title = 'Estimates of yearly increase in temperature in {} for stations in {}, years {} - {}'.format(calendar.month_name[month], country,
    # get rate use linear regression
    df1 = df.groupby(['NAME', 'Month']).apply(get_rate_from_linear).reset_index()

    # merge two dataframe
    merged_data = df.merge(df1, how='left', on=['NAME', 'Month'])
    # rename the increase rate
    merged_data.rename(columns={0: 'Estimated Yearly Increase(° C)'}, inplace=True)

    # get the max value of rate to set range color
    max_val = abs(merged_data['Estimated Yearly Increase(° C)'].max())
    min_val = abs(merged_data['Estimated Yearly Increase(° C)'].min())
    if min_val > max_val:
        max_val = min_val

    # remove duplicate rows
    norepeat_df = merged_data.drop_duplicates(subset=['LATITUDE', 'LONGITUDE', 'NAME', 'Estimated Yearly Increase(° C)'], keep='first')
    # get a figuare to return
    fig = px.scatter_mapbox(norepeat_df,
                        lat = "LATITUDE",
                        lon = "LONGITUDE",
                        hover_name = "NAME",
                        color = "Estimated Yearly Increase(° C)",
                        title = title,
                        zoom = zoom,
                        opacity = 0.5,
                        height = 500,
                        mapbox_style = mapbox_style,
                        range_color  = [max_val * -1, max_val],
                        **kwargs)

    fig.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
    return fig
```

Test Case1

Query the annual temperature increase of all stations in India from 1980 to January 2020, each station should have no less than 10 records

In [10]:

```
color_map = px.colors.diverging.RdGy_r # choose a colormap

fig = temperature_coefficient_plot("India", 1980, 2020, 1,
                                   min_obs = 10,
                                   zoom = 2,
                                   mapbox_style="carto-positron",
                                   color_continuous_scale=color_map)

fig.show()
```

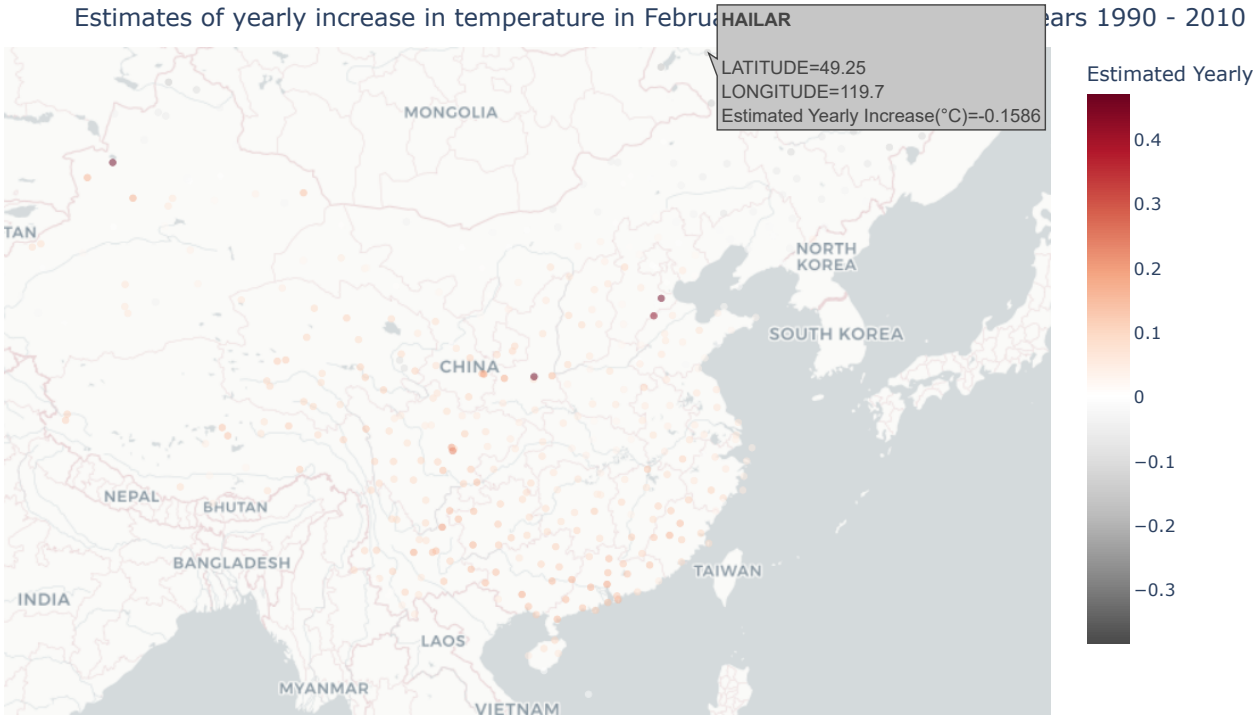


Test Case2

Query the annual temperature increase of all stations in China from 1990 to February 2010, each station should have no less than 5 records

In [11]:

```
color_map = px.colors.diverging.RdGy_r # choose a colormap

fig = temperature_coefficient_plot("China", 1990, 2010, 2,
                                   min_obs = 5,
                                   zoom = 3,
                                   mapbox_style="carto-positron",
                                   color_continuous_scale=color_map)

fig.show()
```



4. Create Two More Interesting Figures

In this part, let's try some somrthing more: We will create more SQL query function and more complex and interesting interactive data visualizations using the same data set.

4.1 One More SQL Query Function about mean temperature

Write a Python function called **query_average_country()** which accepts four arguments:

- **country**, a string giving the name of a country.
- **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.
- **min_temp**, an float giving the value of the year for which should be returned.
- **station_limit**, an integer giving the max number of the stations for which should be returned.

The return value of **query_average_country()** is a Pandas dataframe of average temperature for stations in the specified country, in the specified date range, in the specified month in the specified range of years. This dataframe have columns as follows:

- The station name.
- The latitude of the station.
- The longitude of the station.
- The average temperature for the specified month in the specified range of years.

In [12]:

```
def query_average_country(country, year_begin, year_end, month, min_temp, station_limit):
    cmd = \
        """
        SELECT S.NAME, S.LATITUDE, S.LONGITUDE, ROUND(AVG(T.temp), 4) AS avg_temp
        FROM temperatures T
        INNER JOIN stations S ON T.id = S.id
        INNER JOIN countries C ON SUBSTRING(S.id,1,2) = C.'FIPS 10-4'
        WHERE C.Name='{}' AND T.Year>={} AND T.Year<={} AND T.Month={}
        GROUP BY S.NAME, S.LATITUDE, S.LONGITUDE
        HAVING ROUND(AVG(T.temp), 4) > {}
        ORDER BY ROUND(AVG(T.temp), 4) DESC
        LIMIT {}
        """.format(country, year_begin, year_end, month, min_temp, station_limit)
    df = pd.read_sql_query(cmd, conn)
    return df
```

Test Case

Query the temperature records of all stations in India from 1980 to January 2020, the minimum temperature is 10, and only the top ten stations with the highest average temperature will be returned.

In [13]:

```
country_avg_temps = query_average_country(country="India", year_begin=1980, year_end=2020, month=1, min_temp=10, station_limit=10)
country_avg_temps
```

Out[13]:

	NAME	LATITUDE	LONGITUDE	avg_temp
0	KOZHIKODE	11.250	75.783	28.3778
1	MO_AMINI	11.117	72.733	28.1853
2	THIRUVANANTHAPURAM	8.467	76.950	27.6558
3	MINICOYOBSY	8.300	73.000	27.6415
4	MINICOY	8.300	73.150	27.5959
5	THIRUVANANTHAPURAM	8.483	76.950	27.5607
6	MANGALORE	12.867	74.850	27.4000
7	TRIVANDRUM	8.500	77.000	27.3240
8	MANGALORE_BAJPE	12.917	74.883	26.8050
9	PAMBAN	9.267	79.300	26.3092

4.2 Write a Scatter Function for Monthly Temperature by Year

Write a function called **temperature_year_plot()**. This function should accept one explicit argument.

- **country** a string giving the name of a country for which data should be returned.

The output of this function wouldbe an interactive scatterplot, built using Plotly Express, with a point for each year such that the color of the points reflects the month values for that year. The value at each point highly represents the average temperature across all stations for that year and month.

In [14]:

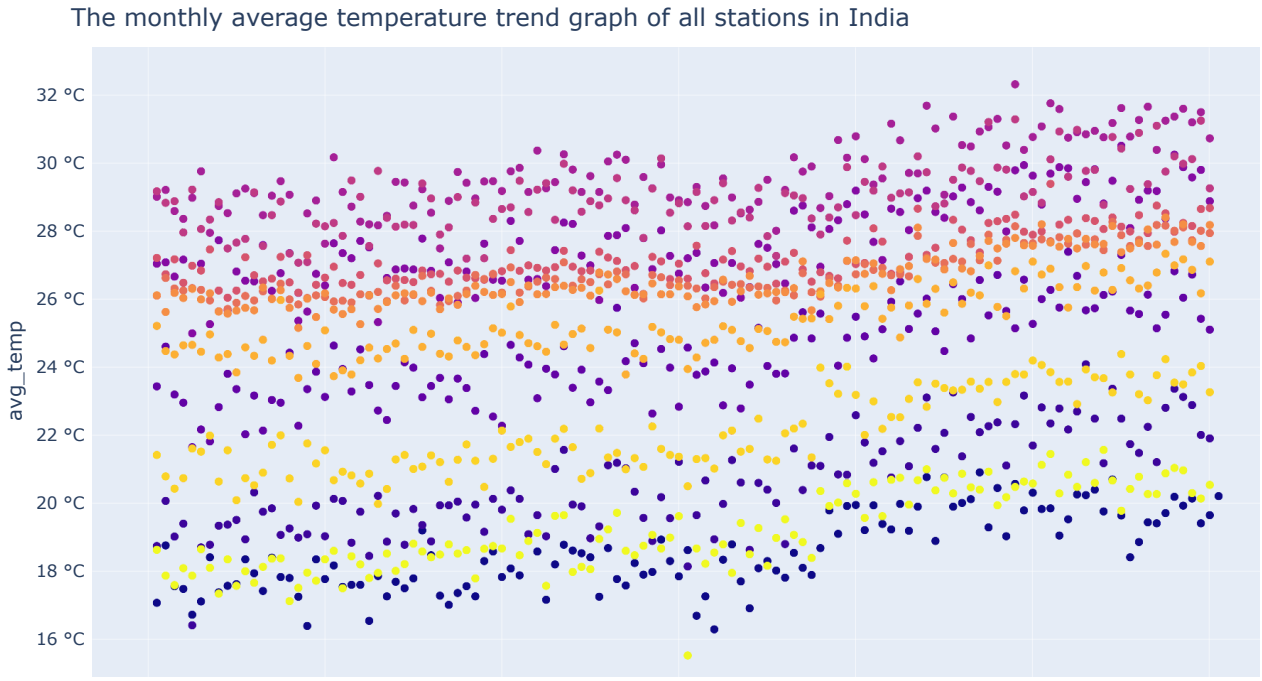
```
def temperature_year_plot(country):
    # sql command
    cmd = \
        """
        SELECT T.Year, T.Month, ROUND(AVG(T.temp), 2) AS avg_temp
        FROM temperatures T
        INNER JOIN stations S ON T.id = S.id
        INNER JOIN countries C ON SUBSTRING(S.id,1,2) = C.'FIPS 10-4'
        WHERE C.Name='{}'
        GROUP BY T.Year, T.Month
        ORDER BY T.Year, T.Month
        """.format(country)
    # set title use args
    title = 'The monthly average temperature trend graph of all stations in {}'.format(country)
    # read data from sqlite
    df = pd.read_sql_query(cmd, conn)
    # set firgure to return
    fig = px.scatter(df,
                     x="Year",
                     y="avg_temp",
                     color="Month",
                     hover_name = "avg_temp",
                     title = title
                    )
    # set figure margin
    fig.update_layout(margin={"r":0,"t":30,"l":0,"b":0})
    fig.layout.yaxis.ticksuffix = ' ° C '
    return fig
```

Test Case

Query the monthly average temperature trend graph of all stations in India in different years.

In [15]:

```
fig = temperature_year_plot(country="India")
fig.show()
```



4.3 Write a LineChart Function for Countries by Month

Write a function called **temperature_month_line()**. This function should accept two explicit arguments.

- **year** an integer giving the specified year for which should be returned.
- **countries_list** a str list giving the countries to be returned.

The output of this function should be an interactive line chart, built using Plotly Express, with a point for each month that reflects the average temperature for that month in that country.

In [16]:

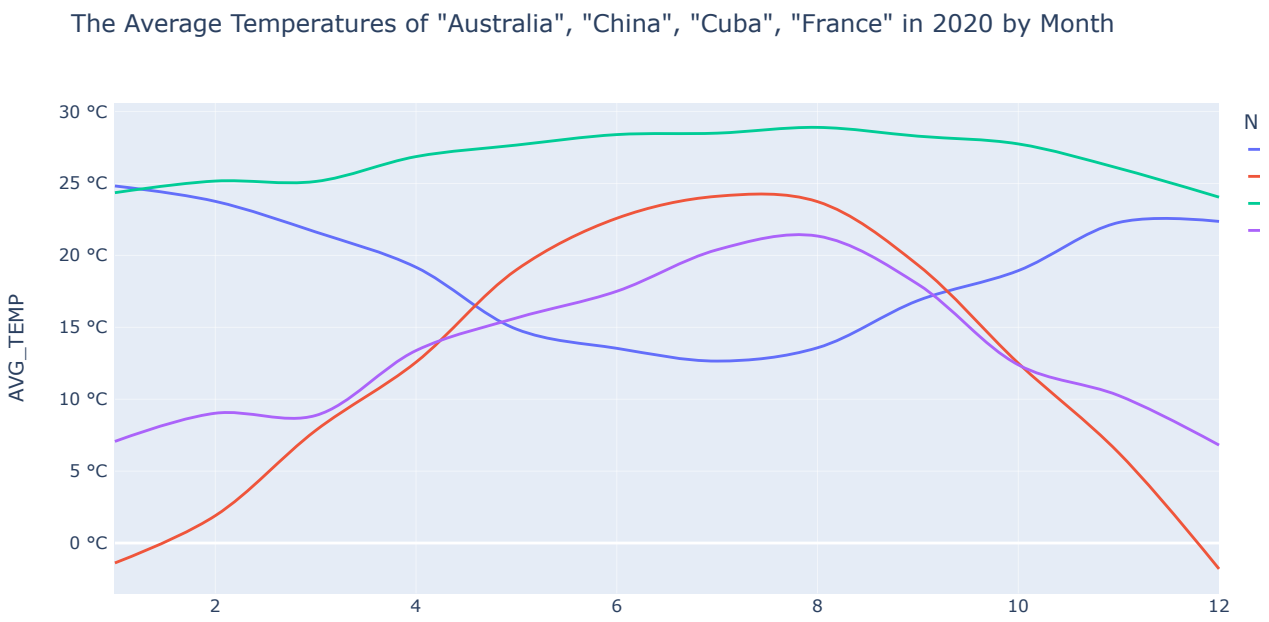
```
def temperature_month_line(year, countries_list):
    # get country string by extract country list
    countries = ', '.join([f'"{country}"' for country in countries_list])
    # sql command
    cmd = \
        """
        SELECT C.Name, T.Month, ROUND(AVG(T.temp), 4) AS AVG_TEMP
        FROM temperatures T
        INNER JOIN stations S ON T.id = S.id
        INNER JOIN countries C ON SUBSTRING(S.id,1,2) = C.'FIPS 10-4'
        WHERE T.Year={} AND C.Name IN ({})
        GROUP BY C.Name, T.Month
        """.format(year, countries)
    # set title use args
    title = 'The Average Temperatures of {} in {} by Month'.format(countries, year)
    # get data
    df = pd.read_sql_query(cmd, conn)

    fig = px.line(df, x="Month", y="AVG_TEMP", title = title, color="Name", line_shape="spline")
    fig.layout.yaxis.ticksuffix = ' ° C '
    return fig
```

Test Case

Query the average monthly temperatures in Australia, China, Cuba, France in 2020.

In [17]:

```
fig = temperature_month_line(year=2020, countries_list=['Australia', 'China', 'Cuba', 'France'])
fig.show()
```



5. Close Database Connection

In [18]:

```
conn.close()
```