Blog Post 1

In this blog post, I will create interactiee graphics using NOAA climate data.

1. Data Prep - Creating a Database

First, its easiest to import all your necessary packages

In [1]:
```
import sqlite3
import pandas as pd
import numpy as np
from plotly import express as px
from sklearn.linear_model import LinearRegression
```

Next, use the pandas package to turn your csvs into a dataframe that is easily manipulated.

In [2]:
```
temps = pd.read_csv("temps.csv")
cntry = pd.read_csv("countries.csv")
statn = pd.read_csv("station-metadata.csv")
```

Good data is that which is plenty and is easy to work with. We have the plenty O' data thing down now, so lets make it easy to work with :)))

The stations' dataset can lose the "STNELEV" column because we have no use for it. Also we are going to create a new column called "FIPS 10-4" because the FIPS 10-4 code is simply the first two letters of each station's ID. We will use the FIPS 10-4 code shortly.

In [3]:
```
statn = statn.drop(["STNELEV"], axis = 1)
statn["FIPS 10-4"] = statn["ID"].str[0:2]
statn
```

Out[3]:

	ID	LATITUDE	LONGITUDE	NAME	FIPS 10-4
0	ACW00011604	57.7667	11.8667	SAVE	AC
1	AE000041196	25.3330	55.5170	SHARJAH_INTER_AIRP	AE
2	AEM00041184	25.6170	55.9330	RAS_AL_KHAIMAH_INTE	AE
3	AEM00041194	25.2550	55.3640	DUBAI_INTL	AE
4	AEM00041216	24.4300	54.4700	ABU_DHABI_BATEEN_AIR	AE
...	...	...	...	...	...
27580	ZI000067983	-20.2000	32.6160	CHIPINGE	ZI
27581	ZI000067991	-22.2170	30.0000	BEITBRIDGE	ZI
27582	ZIXLT371333	-17.8300	31.0200	HARARE_BELVEDERE	ZI
27583	ZIXLT443557	-18.9800	32.4500	GRAND_REEF	ZI
27584	ZIXLT622116	-19.4300	29.7500	GWELO	ZI

27585 rows × 5 columns

Again, we drop unecessary columns and do a little renaming to make our table more clear.

In [4]:
```
cntry = cntry.drop(["ISO 3166"], axis = 1)
cntry = cntry.rename(columns = {"Name"  : "Country"})
cntry
```

Out[4]:

	FIPS 10-4	Country
0	AF	Afghanistan
1	AX	Akrotiri
2	AL	Albania
3	AG	Algeria
4	AQ	American Samoa
...	...	...
274	-	World
275	YM	Yemen
276	-	Zaire
277	ZA	Zambia
278	ZI	Zimbabwe

279 rows × 2 columns

The Temps dataset has a little more involved procedure called stacking, so I would rather comment the code step by step below.

In [5]:
```
# Stacking makes data visualization easy and logical

#first we seperate any Columns that would not stack well
#we choose ID so all of IDs are placed nicely together
#we choose Year to order them chronologically
temps = temps.set_index(keys=["ID", "Year"])
temps = temps.stack()

#this will recover Month and Temp columns, making them nice and neat
temps = temps.reset_index()

#some renaming and cleaning up to make more legible
temps = temps.rename(columns = {"level_2"  : "Month" , 0 : "Temp"})
temps["Month"] = temps["Month"].str[5:].astype(int)
temps["Temp"] = temps["Temp"] / 100
temps
```

Out[5]:

	ID	Year	Month	Temp

	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
...	...	...	...	...
13992657	ZIXLT622116	1970	8	15.40
13992658	ZIXLT622116	1970	9	20.40
13992659	ZIXLT622116	1970	10	20.30
13992660	ZIXLT622116	1970	11	21.30
13992661	ZIXLT622116	1970	12	21.50

TA-DAA we now have three clean dataframes! Now... we upload into the mainframe

Ok well not actually, but we will create our own SQL database to upload our three dataframes for easy and quicker acess.

In [6]:
```
#this line creates an SQL database that is much easier to index
conn = sqlite3.connect("NOAA.db")

#following lines upload our dataframes, one at a time.
cntry.to_sql("countries", conn, if_exists='append', index=False)
temps.to_sql("temperatures", conn, if_exists='append', index=False)
statn.to_sql("stations", conn, if_exists='append', index=False)
```

```
C:\Anaconda3\lib\site-packages\pandas\core\generic.py:2872: UserWarning: The s
paces in these column names will not be changed. In pandas versions < 0.14, sp
aces were converted to underscores.
  sql.to_sql(
```

Alright, now to check that our data is there...

In [7]:
```
cursor = conn.cursor()
cursor.execute("SELECT name FROM sqlite_master WHERE type='table'")
print(cursor.fetchall())
```

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

Great! It seems our databases have turned into three seperate tables within our database! Now just close off the connection for safety's sake.

In [8]:
```
conn.close()
```

2. Writing A Query Function

Now to write a query function for our data. This function will return pandas dataframe of

temperature readings for the specified country, in the specified date range, in the specified month of the year.

In [14]:
```
def query_climate_database(country, year_begin, year_end, month):
    '''
    This function will return pandas dataframe of temperature readings for the
    specified country, in the specified date range, in the specified month of

    @param country: a string giving the name of a country for which data shoul
    @param year_begin and year_end: two integers giving the earliest and lates
                                    which should be returned
    @param month: an integer giving the month of the year for which should be

    return joined: a Pandas dataframe of temperature readings for the specifie
                   in the specified date range, in the specified month of the
    '''

    conn = sqlite3.connect("NOAA.db")
    cursor = conn.cursor()

    #we use sqlite to join all of the important columns of our tables togethe
    #the constant columns id and FIPS 10-4 (see I told you we'd use it soon.)
    cmd = \
    """
    SELECT S.NAME, S.LATITUDE, S.LONGITUDE, C.Country, T.Year, T.Month, T.Temp
    FROM stations S
    LEFT JOIN temperatures T on S.ID = T.ID, countries C on C.'FIPS 10-4' = S
    WHERE C.Country = '{0}' AND T.Year>= {1} AND T.Year<= {2} AND T.Month = {
    """
    cmd = cmd.format(country, year_begin, year_end, month)
    joined = pd.read_sql_query(cmd, conn)
    conn.close()
    return joined
```

As you can see, our function works perfectly!

In [15]:
```
query_climate_database(country = "India",
                       year_begin = 1980,
                       year_end = 2020,
                       month = 1)
```

Out[15]:

	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

	NAME	LATITUDE	LONGITUDE	Country	Year	Month	Temp
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

3. Write a Geographic Scatter Function for Yearly Temperature Increases

Now we want to find the year-over-year average change in temperature. How can we do that you ask? A little mathmatical model called Linear Regression

In [16]:

```
def coef(data_group):
    '''
    We are going to use this function to model a regression of the change in t
    against the years to find a coefficient over the yearly change in temperat

    @param data_group: the group of data that you want to model your regressio

    return LR.coef_[0]: the coefficient of regression
    '''
    x = data_group[["Year"]] # 2 brackets because X should be a df
    y = data_group["Temp"]   # 1 bracket because y should be a series
    LR = LinearRegression()
    LR.fit(x, y)
    return LR.coef_[0]
```

We are going to incorporate the coefficients with the original queried dataframe in a function that will soon create a nice pretty picture... Just follow my comments and I'll explain.

In [17]:

```
def temperature_coefficient_plot(country, year_begin, year_end, month, min_obs
    '''
    This function will create 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 s

    @param country: a string giving the name of a country for which data shoul
    @param year_begin and year_end: two integers giving the earliest and lates
                                    which should be returned
    @param month: an integer giving the month of the year for which should be
    @param min_obs: an integer stating the minimum required number of years o
                    for any given station
    @param **kwargs: additional keyword arguments passed to px.scatter_mapbox

    return a plotly express scatterplot of user specified variables.
    '''

    #creates a queried database of the specified variables (so computer mustn
    queri = query_climate_database(country, year_begin, year_end, month)

    #will create a new dataframe will the coeficcients of each station over ti
    coefs = queri.groupby(["NAME", "Month"]).apply(coef)
    coefs = coefs.reset_index()
    coefs = coefs.rename(columns = {0  : "Estimated Yearly Increase (°C)"})
    coefs["Estimated Yearly Increase (°C)"] = round(coefs["Estimated Yearly In

    #merges the two together to be easily plottable
    queri = pd.merge(queri, coefs, on=['NAME','Month'])

    #makes sure that Only data for stations with at least min_obs years worth
    #in the specified month should be plotted; the others should be filtered
    queri = queri.groupby('NAME').filter(lambda x: len(x) >= min_obs)

    #a dictionary to easily change title based on user inputted variables
    monthDict = {1:'Janauary',
                 2:'February',
                 3:'March',
                 4:'April',
                 5:'May',
                 6:'June',
                 7:'July',
                 8:'August',
                 9:'September',
                10:'October',
                11:'November',
                12:'December'}

    #title prompt for the plot
    title = "Estimates of yearly increase in temperature in {} for stations i
    #updating with user inputted variables
    title = title.format(monthDict[month], country, year_begin, year_end)

    #plot
    return px.scatter_mapbox(queri,
                             lon="LONGITUDE",
                             lat="LATITUDE",
                             color="Estimated Yearly Increase (°C)", # which co
```

Now if we did it right it should work...

In [18]:
```
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()
```

Estimates of yearly increase in temperature in Janauary for

Look at that! Our own interactive map!!

In [ ]:
```

```