Project Report



Name: Iracema Lopes Semedo

Course: Data Analysis for Marketing

Date: 01/09/2021

Index

1. GitHub URL
2. Abstract
3. Introduction
4. Dataset Inspection and Cleaning
   1. Import\_dataset function
   2. Dataset Inspection
   3. Dataset Cleaning

5. Problem statement and Implementation Process

6. Results and Insights

7. Other Python concepts demonstrated

* 1. Dictionaries, Lists, Merge dataframes
  2. Python Loops (iterrows)

1. **GitHub URL**

<https://github.com/IracemaLopes/UCDPA_2021_Iracema>

Committed files:

* dataset\_inspection.py
* dataset\_cleaning.py
* dataset\_elaboration.py
* import\_dataset.py (function)
* merge\_dataframe.py
* europe\_and\_north\_america.py
* python\_loop.py

The meaning of all these files will be explained in the subsequent sections.

**Important note:**

**It has not been possible to upload the cvs file into the folder, as it resulted too large for github to be uploaded. Thus, a “readme” file has been added including the link of the dataset to be downloaded prior to run any code.**

1. **Abstract**

A dataset containing all the temperature of the major cities in the word has been chosen for this project to demonstrate the effects of global warming, which can be quantified in first order by measuring the temperature of these states over time. The major Python concepts and modules have been used to accomplish the research, as will be seen in next sections. A function called *“import\_dataset.py”* has been created to import the dataset in all the python files used along the project. A file *“dataset\_inspection.py”* has been created to inspect the dataset and derive its characteristics. The file *“dataset\_cleaning.py”* serves to clean the dataset by removing duplicates, correcting 0 or NaN’s and correcting some incorrect year typos. Finally, the file *“dataset\_elaboration.py”* is the main project file, where dataset\_inspection.py and dataset\_cleaning.py have been recalled, and further elaboration and plots have been performed. Results show that Africa outperforms the other regions in terms of yearly average temperature, although there are some peaks for the Middle East region that also are meaningful, placing it to the second place. All the others are well below Africa and Middle East, with the lowest being Europe. Also, there is a slight increase in all regions’ yearly temperatures, confirming that global warming is a real and always worsening phenomenon. As expected, by comparing the hottest and coldest city in the world in descending order we can see that Africa and South/Central America & Caribbean’s monthly average temperature is quite constant, while for other regions the monthly average temperature changes. Other three scripts called *“merge\_dataframe.py”*, *“europe\_and\_north\_america.py”* and *“python\_loops.py”* have been created to demonstrate other Python concepts separately, that have not been possible to include in the project itself.

1. **Introduction**

The dataset chosen for this project can be found [here](https://www.kaggle.com/sudalairajkumar/daily-temperature-of-major-cities). Global warming is the long-term heating of Earth’s climate system observed since the pre-industrial period (between 1850 and 1900) due to human activities, primarily fossil fuel burning, which increases heat-trapping greenhouse gas levels in Earth’s atmosphere. It has been demonstrated by direct temperature measurements and by measurements of various effects of the warming. The term is frequently used interchangeably with the term climate change, though the latter refers to both human- and naturally produced warming and the effects it has on our planet. It is most measured as the average increase in Earth’s global surface temperature.

**Important note:**

**To run any of the following scripts/functions, please install all the required packages before: Matplotlib, Seaborn, Bokeh, Folium, Pandas. To do this, go to File 🡪 Settings and Add the package under the Python Interpreter section:**

A screenshot of a computer

Description automatically generated with medium confidence

1. **Dataset: Inspection and Cleaning**

* 1. **Import\_dataset (file: import\_dataset.py)**

A function *“import\_dataset.py”* has been created to import the dataset in all the python file used along the project. It has been committed in the GitHub folder and it is accessible.

The function’s code is below:

import pandas as pd  
def import\_dataset(path):  
 df = pd.read\_csv("city\_temperature.csv", low\_memory=False, na\_values=[-99])  
 print("starting dataset")  
 print(df)  
 return df

It has been decided to include the print of the dataset into the function, as this is one possible reliable method to see the function working at the beginning of the code.

* 1. **Dataset Inspection (file: dataset\_inspection.py)**

Firstly, a file “dataframe\_inspection.py” was created to inspect the downloaded CSV. The file is committed on the GitHub folder and can be accessed.

The first step is to call the function previously created. The piece of code to do this is below:

import import\_dataset as id  
df=id.import\_dataset("city\_temperature.csv")

and the printout is:

Graphical user interface, text

Description automatically generated

We can see that:

* The dataset has 2906327 rows and 8 columns which are not all displayed.
* We got a low memory warning which can be solved by adding low\_memory=False to the read and used **na\_values** parameter to replace -99 in AvgTemperature column as per below:

df=pd.read\_csv("city\_temperature.csv",low\_memory=False, na\_values=[-99])

Using the .head()method we can see that it returns the first few rows (the “head” of the DataFrame).

print(df.head())

from the printout we can see all columns that we could not see before.

Graphical user interface, text, application

Description automatically generated

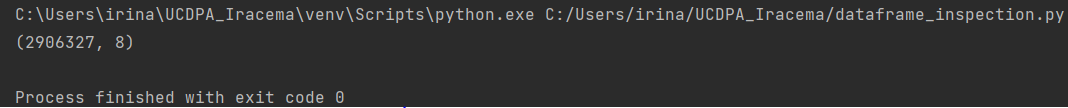
Using the .info() method it will show information on each of the columns, such as the data type and number of missing values.

Text

Description automatically generated

Using the .shape() method it returns the number of rows and columns of the DataFrame.

print(df.shape)



Using the .describe() method we can calculate a few summary statistics for each column.

print(df.describe())

Text

Description automatically generated

Using the. isna().sum() method it counts the number of missing values in each columns.

print(df.isna().sum())

Text

Description automatically generated

Using the .isna() method on dataframe we get Booleans for every single value indicating whether the value is missing or not.

print(df.isna())

Graphical user interface

Description automatically generated with low confidence

* 1. **Dataset Cleaning (file: dataset\_cleaning.py)**

Firstly, the column State has been cancelled as it has missing data.

print("we have cancelled the column State because it only contained Nans")  
df.drop('State',axis='columns', inplace=True)  
cleaned\_df=df  
print(cleaned\_df)

Graphical user interface

Description automatically generated with medium confidence

It is worth to check the year range, to see that the dataset goes from 1995 to 2020

print(cleaned\_df.Year.value\_counts().index)

A picture containing text, electronics, close

Description automatically generated

We can immediately understand that there are two extra years values, 201 and 200 which represent typos. Since 201 could either be 2010 or 2001 as a typo (the same for 200 which might have even more possible combinations), the best thing is to remove these data. Thus, we counted how many values there are for each year to understand how many values there are for 201 and 200.

print(cleaned\_df.Year.value\_counts())

Table

Description automatically generated

Then, we removed 201 and 200 years also because in total they count 440 values which is a small number compared to the initial total number.

cleaned\_df=cleaned\_df[cleaned\_df.Year>1990]  
print(cleaned\_df.Year.value\_counts())

A picture containing table

Description automatically generated

By looking for missing values, we can now see that AvgTemperature has 79232 missing data.

print(cleaned\_df.isna().sum())

Text

Description automatically generated with medium confidence

It would be a mistake to replace these temperature data with all 0s and it would be bad also to remove them, as compared to the previous data we removed, they now are quite a lot, thus a lot of data would be lost. One possible way to fix these missing data could be replacing them with the temperature of the previous day. This would be a realistic assumption, as we saw in the dataset that it contains all the temperatures day by day: assuming the missing temperature for a given day is like the day before not only is fair but will not create spikes in the data.

cleaned\_df["AvgTemperature"].fillna(method="ffill", inplace=True)  
print(cleaned\_df.isna().sum())

Text

Description automatically generated

As we can see, there are no more missing values.

1. Problem statement and Implementation Process (dataset\_elaboration.py)

After having analysed and cleaned the dataset as shown in the previous sections, let’s now get to the problem we want to solve, and derive some insights from the dataset we previously elaborated.

Global warming is the long-term heating of Earth’s climate system observed since the pre-industrial period (between 1850 and 1900) due to human activities, primarily fossil fuel burning, which increases heat-trapping greenhouse gas levels in Earth’s atmosphere. It has been demonstrated by direct temperature measurements and by measurements of various effects of the warming. For this reason, the dataset that I choose for this project is appropriate for this kind of analysis. I would like to show how the temperatures for each state varied over time, what is the average over the years included in the dataset, and the average temperature year by year, and derive useful insights from these elaborations.

This paragraph will only explain the code used to perform this analysis, while next paragraph will deal with results.

As one could be not familiar with Fahrenheit temperature, it has been decided to convert the temperature from Fahrenheit to Celsius searching in Google the appropriate formula to perform this conversion.

Graphical user interface, text, application, email

Description automatically generated

The code used to perform this conversion is as per below:

print("converting the dataset containing Fahrenheit temperatures in Celsius")  
cleaned\_df["celsius"]=(cleaned\_df["AvgTemperature"] - 32) \* 5/9  
print(cleaned\_df.head())

In order to check the average temperature of regions between 1995 and 2020, the groupby() function has been used to calculate summary statistic along with the mean() method and sorting the column “Celsius” in descending order.

print(cleaned\_df[['Region','celsius']].groupby('Region').mean().sort\_values(by='celsius', ascending=False))

Next, we checked the yearly average of regions, using the groupby() function along with the mean() method. Then the reset\_index() is used to reset the index after the groupby as the groupby() function split the data into groups based on some criteria like *mean,* *median, value\_counts,* etc. Finally, we printed the head()method which returns the first few rows.

df\_region = cleaned\_df[['Region','Year','celsius']].groupby(['Region','Year']).mean().reset\_index()  
print(df\_region.head())

After checking the yearly average of regions, it has been decided to use Seaborn which is a powerful python library for creating data visualizations. Few previous attempts done using Matplotlib did not give a pretty plot for the data.

sns.relplot(x="Year", y="celsius", data=df\_region, kind="line",style="Region", hue="Region")  
plt.show()

To compare the average temperature between 1995 and 2019 (ignoring 2020 as 2020 is not completed in the dataset), the groupby() function along with the mean() method has been used, and the reset\_index() to reset the index after the groupby.

avg\_temp=cleaned\_df[cleaned\_df.Year.isin([1995,2019])][['Region','Year','celsius']].groupby(['Region','Year']).mean().reset\_index()  
print(avg\_temp)

As we noticed 2020 column does not contain reliable data, and to use another Seaborn feature, it has been decided to use catplot() function with Seaborn. This function provides access to several axes-level functions that show the relationship between a numerical and one or more categorical variables using one of several visual representations. As you can see, we choose the bar plot and set the hue parameter equals to dataframe column in this case “Region”. Then Seaborn will automatically color each bar and will add a legend to the plot automatically.

sns.catplot(x="Year", y="celsius", data=avg\_temp, kind="bar", hue="Region")  
plt.show()

Then, we wanted to do another comparison calculating the average monthly temperature of regions

df\_region2 = cleaned\_df[['Region','Month','celsius']].groupby(['Region','Month']).mean().reset\_index()  
print(df\_region2)

and to have a better understanding we also plotted the average monthly temperature of regions using relplot() function with Seaborn.

sns.relplot(x="Month", y="celsius", data=cleaned\_df, kind="line", style="Region", hue="Region", markers=True)

Also, it is interesting to compare the hottest and coldest city in the world in descending order.

df\_city=cleaned\_df[['City','celsius','Country']].groupby(['City','Country']).mean().sort\_values(by='celsius',ascending=False).reset\_index()

Let’s now apply some of the Python concepts we did not apply till now. We want to produce a bar plot with Matplotlib to show average temperatures over 1995 till 2020, for just two states, let’s say Europe and North America. To do this we need to work with NumPy arrays, which we did not apply up to now.

First, let’s subset “Europe” region from the dataframe:

europe\_temperatures=df\_region[(df\_region["Region"]=="Europe")]

and let’s create the labels for the “x” axis, which are the years:

labels=[1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]

Then, we need to take only the “Celsius” column from “europe\_temperature” dataframe:

eu\_temp=europe\_temperatures[["celsius"]]

if we print this out, it is possible to see that there are two problems:

* There are numerical indexes still present.
* The type of eu\_temp is a dataframe; it is possible to verify it with the instruction:

print(type(eu\_temp))

We know that Matplotlib works with NumPy arrays, so we need to convert it using the function “np.to\_numpy()” as follows:

eu\_temp=eu\_temp.to\_numpy()

Still, there is a problem. By doing print(eu\_temp) you can figure out that the result would be something like this:

Text

Description automatically generated with medium confidence

This means we need to transpose the NumPy, and then take just the first element.

eu\_temp=np.transpose(eu\_temp)  
eu\_temp=eu\_temp[0]

Let’s see how it looks like now:

print("fourth print")  
print(eu\_temp)  
print(eu\_temp.shape)

Text

Description automatically generated

Now it is fine, and Matplotlib can take it.

The same procedure has been applied for the North America region:

x = np.arange(len(labels)) # the label locations  
width = 0.35 # the width of the bars  
fig, ax = plt.subplots()  
rects1 = ax.bar(x - width/2, eu\_temp, width, label='Europe')  
rects2 = ax.bar(x + width/2, n\_am\_temp, width, label='North America')

Now we can write the code to plot these two NumPy arrays created. To get a pretty print of the results, we also insert some labels and legend.

# Add some text for labels, title and custom x-axis tick labels, etc.  
ax.set\_ylabel('Temp [C]')  
ax.set\_title('Yearly Average Temperatures in Europe and North America ')  
ax.set\_xticks(x)  
ax.set\_xticklabels(labels)  
ax.legend()  
ax.bar\_label(rects1, padding=3)  
ax.bar\_label(rects2, padding=3)  
fig.tight\_layout()  
plt.show()

We can produce the same plot using the Bokeh library, so let’s apply it too. First, let’s import it:

from bokeh.io import output\_file, show  
from bokeh.plotting import figure

and the example we decided to implement consists of this piece of code.

print("plotting with bokeh average temperature of Europe from 1995 to 2020")  
p=figure(x\_axis\_label="Year", y\_axis\_label="celsius")  
p.line(labels,eu\_temp, color="blue")  
p.circle(labels, eu\_temp, color="blue", size=10,line\_color="white" , legend\_label="Europe")

output\_file('average\_temperature\_europe.html')  
show(p)

One problem with the result of this code is that the legend overlaps with the graph, because the default place where the legend is placed is top right. To modify the position of the legend to the top left we can add this:

# display legend in top left corner (default is top right corner)  
p.legend.location = "top\_left"

Another interesting application from Bokeh is the rows (or columns) of plots. First, we import the row function from the bokeh.layouts module.

from bokeh.layouts import row

Then we pass all the elements as argument to the row in the order we would like them to appear from left to right. In the example below we pass 2 plots, p1, p2 to row. We can see in the resulting output, the 2 plots: p1, p2 are arranged visually in a horizontal row from left to right.

print("plotting with bokeh average temperature of Europe and North America from 1995 to 2020")  
p1=figure(x\_axis\_label="Year", y\_axis\_label="celsius")  
p1.line(labels,eu\_temp, color="blue")  
p1.circle(labels, eu\_temp, color="blue", size=10,line\_color="white" , legend\_label="Europe")  
# display legend in top left corner (default is top right corner)  
p1.legend.location = "top\_left"  
  
p2=figure(x\_axis\_label="Year", y\_axis\_label="celsius")  
p2.line(labels,n\_am\_temp, color="red")  
p2.circle(labels, n\_am\_temp, color="red", size=10,line\_color="white" , legend\_label="North America")  
# display legend in top left corner (default is top right corner)  
p2.legend.location = "top\_left"  
layout=row(p1,p2)  
output\_file('average\_temperature\_europe\_vs\_north\_america.html')  
show(layout)

6. Results and Insights

By checking the yearly average temperatures of the various regions, this is what we get:

Text

Description automatically generated

This is not extremely clear and meaningful, as the print cannot show all the rows that the dataset has. However, we can still see that Africa exhibits the highest average temperature compared with all the other states. As this print does not give too many other insights, let’s go for a plot instead, as also described in the previous sections. This is what we get:

Chart, line chart

Description automatically generated

Apart from the 2020 data which are not meaningful as incomplete, we can still see that Africa outperforms the other regions in terms of yearly average temperature, although there are some peaks for the Middle East region that also are meaningful, placing it to the second place. All the others are well below Africa and Middle East, with the lowest being Europe. The catplot from Seaborn gives me this plot (remember that I ignored the 2020 column as it does not contain reliable data):

Chart, bar chart

Description automatically generated

Note how the graph is a bit more readable now since we got rid of the 2020 column, but also notice how the trend described previously is still confirmed. There is a slight increase in all regions. It seems like global warming is real.

Let’s compare hottest and coldest city in the world in descending order. From below we can see that Africa and South/Central America & Caribbean’s monthly average temperature is quite constant, while for other regions the monthly average temperature changes.

Chart, line chart

Description automatically generated

We also selected two regions (Europe and North America) and compare the yearly average temperature, to apply some other Python concepts, such as Numpy, subsetting, Matplotlib, indexing etc (please recall the previous paragraph for more details of the code).

This is the plot we get:

Chart, bar chart

Description automatically generated

It is a kind of busy plot, as there are a lot of data in it. However, it is quite readable, and it gives us few useful insights:

* In 2016, North America average temperature is the highest (14.6 C) while Europe is on its average.
* While North America exhibits a lot of average temperature spikes with respect to its average (see 1998, 2006, 2012, 2016), Europe is quite steady.
* As we also observed previously, 2020 data are incomplete. The average temperature is well below the average temperature on previous years, and this is because for 2020 data has been inserted till the 13th of May, thus average temperature for the inserted data is low till this period as we expect.

Bokeh result gives this plot:

Graphical user interface, chart

Description automatically generated

As usual, ignore 2020 as it is incomplete.

The rows of plot application examples give as follow:

Graphical user interface, chart

Description automatically generated

As usual, ignore 2020 as it is incomplete.

As a Geospatial application, it has been decided to use the “folium” library to visualize the two compared locations above (Europe and North America) in Google Maps. A new Python script, called *“europe\_and\_north\_america.py”* has been created, and committed in the GitHub folder. This is the code that has been used, which has also been commented to explain the various steps.

import folium  
#location is Canada  
m=folium.Map(location=[59.787000283026565, -111.12920590302417], zoom\_start=12)  
  
#location is Milan  
m2=folium.Map(location=[45.457305121681685, 9.188590225926237], zoom\_start=12)  
  
  
#created marker in Vancouver  
folium.Marker([49.289116777041166, -123.11142033962074],popup='Vancouver',  
 tooltip='Canada Place',  
 icon=folium.Icon(icon='heart', icon\_color='red', color='green')).add\_to(m)  
  
  
#created marker in Duomo di Milano  
folium.Marker([45.464195400558665, 9.191937227026742],popup='Cathedral',  
 tooltip='Duomo di Milano',  
 icon=folium.Icon(icon='camera', icon\_color='red', color='green')).add\_to(m2)  
  
# we added a blue circle to indicate the area we like  
folium.Circle(location =(49.272067602905764, -123.10193315953669),  
 radius=800,  
 popup='Love the area',  
 color='blue',  
 fill=True,  
 fill\_color='blue'  
).add\_to(m)  
  
  
#we added a blue circle to indicate the area we like in Milano city centre  
folium.Circle(location =(45.46382094385277, 9.191974576083203),  
 radius=800,  
 popup='Love the area',  
 color='blue',  
 fill=True,  
 fill\_color='blue'  
).add\_to(m2)  
  
#generate a map in html  
m.save('maps\_Vancouver.html')  
m2.save('map\_Milan.html')

As a result, two html pages were created as an output of the script:



If we open “map\_Milan” this is what we get:

Map

Description automatically generated

while if we open “maps\_Vancouver” we get this:

Map

Description automatically generated

Both of them are as expected.

7.Other Python concepts demonstrated

* 1. Dictionaries, Lists, Merge dataframes

The script *“merge\_dataframe.py”* has been created to demonstrate other concepts we were not able to demonstrate previously.

To merge 2 dataframes, Pandas needs to be imported.

import pandas as pd

Then, to apply some other Python concepts, such as “Dictionary” and “List” two dictionaries that contain a list have been created.

As you can see, these dictionaries have been converted into DataFrames.

#created a dictionaries that contains a list  
region={  
 'Region':['Europe','Africa','Asia','South/Central America','Australia/South Pacific','Middle East'],  
 'Country':['Italy','Angola','Asia','Argentina', 'Australia','Turkey']  
}  
names={  
 'Region':['Europe','Africa','Asia','South/Central America','Australia/South Pacific','Middle East'],  
 'Abbreviation':['IT','AO','JP','AR','AU','TR']

#converted the dictionary into dataframe  
df1=pd.DataFrame(region)  
df2=pd.DataFrame(names)

Finally, to merge these two DataFrames the following code has been used:

# code to merge two dataframes  
df=pd.merge(df1,df2, on='Region')  
print(df)

As you can see from the printout “region” and “names” DataFrames have been merged.

Text

Description automatically generated

* 1. Python Loop (iterrows)

The script *“python\_loop.py”* has been created to demonstrate the iterrows() function.

Firstly, pandas has been imported and a dictionary has been created and then converted in Dataframe.

#created a dictionary  
dict={ 'name':['Lisa','Mark','Joe','Eva'],  
 'degree':['MBA', 'BCA', 'TECH','MATH'],  
 'score': [100,95,89,91]}  
#converted the dictionary into dataframe  
df=pd.DataFrame(dict)

Started iterating through each row of the dataframe.

#iterate through each row of dataframe  
for lab, row in df.iterrows():  
 print(lab)  
 print(row)

As we can see from the printout the iterrows method looks at the dataframe and on each iteration generates two pieces of data: the label of the row and the actual data on the row as a panda series.

Text

Description automatically generated