Lab Package 1.2

Friday, August 20, 2021 7:34 PM

You can do the lab package on the lab computer, Google Colaboratory (https://research.google.com/colaboratory/faq.html), or your own computer

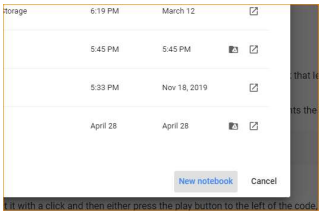
On the lab computer: Anaconda > Jupyter Notebook or ArcGIS > Jupyter Notebook
At Google Colaboratory: You need to have a google account first, and then https://colab.research.google.com/notebooks/
On your own computer: install Anaconda (https://www.anaconda.com/products/individual) and Anaconda > Jupyter Notebook

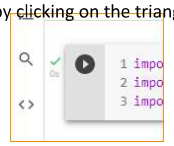
General python workflows for spatial data science:

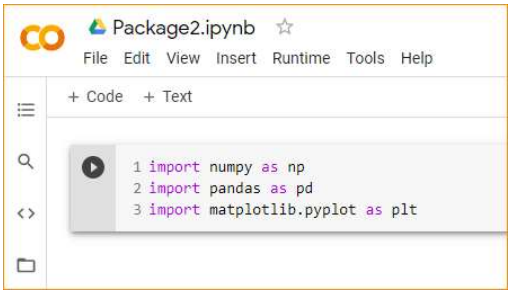
1. Import libraries
2. Open data
3. Check what the data entail: rows and columns
4. Inspect data for missing data or outliers
5. Explore data for correlation
6. Analyze data or develop model
7. Inspect the findings
8. Save the figures and findings

Essential Python libraries for Spatial Data Science:
numpy: mathematical functions
pandas: data wrangling and table manipulations
Matplotlib: data visualization
Seaborn: data visualization (built on Matplotlib with more functions and prettier settings)
plotly_express: simple syntax for data visualization
Scikit: machine learning
Re: regular expression (pattern matching and detection)
Geopandas: pandas with geospatial components
Fiona: read and write geospatial files. Work with GDAL
Folium: to use web base map
Pygmap: plot against google map
descartes: used by Geopandas for dilation (buffering) and erosion (shrinking)
GDAL/OGR: translator for raster and vector geospatial data formats
RSGISLib: remote sensing tools for raster processing (including object segmentation and classification -- GEOBIA)
Pyproj: project and transform georeferencing systems and perform geodetic calculations and distances for a given datum
Ipyleaflet: create interactive map

1. Login to your google account
2. Then go to https://colab.research.google.com/ and then New Notebook at the lower right corner
3. Change the notebook name to Package2
4. Import the necessary libraries as the image below and run the cell by clicking on the triangle icon or use shift+enter







5. The colab libraries do not include geospatial libraries. We need to install these libraries first before we can import them to the notebook. Note that Collab requires the installation for every runtime.

```
!apt install Geopandas
# install important gopython libraries
!apt install gdal-bin python-gdal python3-gdal
# Install rtree - Geopandas requirment
!apt install python3-rtree
# Install Geopandas
!pip install git+git://github.com/geopandas/geopandas.git
# Install descartes - Geopandas requirment
!pip install descartes
```

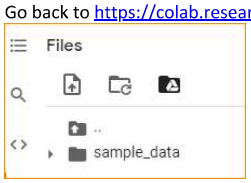
6. Save in your google drive:
   - File > Save (to save in a ColabNotebook folder on your google drive)
   - File > Save and pin revision (allow you to check revision history)

7. Then import these libraries

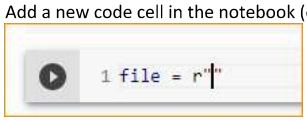
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
% matplotlib inline

import geopandas as gpd
import descartes
import fiona
from shapely.geometry import Point, Polygon
```

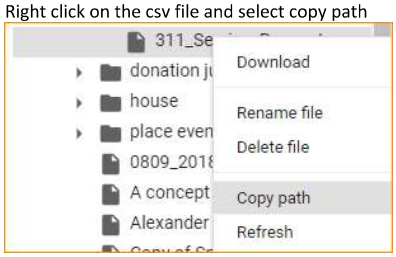
7. Open the csv file (311_Service_Requests_October_1__2020_to_Present.csv) we used in Package1.
   a. Open a new browser window and go to https://drive.google.com/
   b. Create a new folder: Package2, and then upload the csv file to the folder
   c. Go back to https://colab.research.google.com and Mount the google drive



   d. Add a new code cell in the notebook (either by clicking on +Code or Ctrl+m+b)



   e. Navigate to Package2 folder
   f. Right click on the csv file and select copy path



   g. Go back to the cell and enter ctrl+v to paste the path to the space between the quotation mark and use pandas read_csv function to read the file and convert the file to a data frame (df).

```
[3]  1 file = r"/content/drive/MyDrive/Package2/311_Service_Requests_October_1__2020_to_Present.csv"
     2 df = pd.read_csv(file)
```

To read excel file,

```
1 file = r'/content/drive/MyDrive/Package2/Top Service Request Types.xlsx'
2 TopService = pd.read_excel(file)
```

Data inspection and engineering with df data frame.

df	--- Display the data frame df
df.head()	--- List the first 10 rows of the data
df.tail()	--- List the last 10 rows of the data
from google.colab import data_table data_table.DataTable(df, num_rows_per_page=20) --- display the dataframe df as an interactive table	
df.columns	--- List the columns of the data frame
df.Address	--- List the column named Address
df.iloc[1:10]	--- List rows 1 to 9
df.iloc[:,1:10]	--- List rows 1 to 9
df.describe()	--- Describe the data frame
df.info()	--- List column information
df.dtypes	- -- List data type for each column
df = df.convert_dtypes() --- convert object data type to string or numeric data	
df['Service Request Type'] = df['Service Request Type'].astype('string') -- try this if convert_dtype does not change an object type to a string type	
df[['Address', 'Service Request Type', 'Outcome']] --- List all rows and columns 1 to 9	
Len(list) --- how many elements in the list	
df.rename(columns={'ERT (Estimated Response Time)':'ERT', 'Overall Service Request Due Date':'Overall Due Date'}) --- Rename columns	
df[['Lat_Long Location']] = df[['Lat_Long Location']].astype(str) --- Change the data type to "string"	
df[['Latitude', 'Longitude']] = df['Lat_Long Location'].str.split(",",expand=True) --- split a column to two based on ','	

1.
a.
b.
c.
2.
a.
b.
c.
d.
3.
a.

```
df.Latitude = [x.split('(')[1] for x in df.Latitude]   --- retain everything after '(' in a string
df.Latitude = pd.to_numeric(df.Latitude, errors = 'coerce')    --- Convert a string column to a float column

df['Service Request Type'].unique()   --- List unique values of a column named 'Service Request Type'
df['Service Request Type'].nunique()    --- the number of unique values in the column 'Service Request Type'
df['Service Request Type'].value_counts(ascending=True)  --- List the occurrences for each unique value
df['Service Request Type'] = df['Service Request Type'].str.capitalize()
                                                --- capitalize the first letter only
df['Service Request Type'] = [x.strip() for x in df['Service Request Type']]
                                                 --- remove extra spaces between words
df['Service Request Type'] = [x.split(' -')[0] for x in df['Service Request Type']]
                                                --- only take the description before ' -'
ServiceType = df['Service Request Type'].unique()  --- save the unique values of service request types to a numpy
array
ServiceType = ServiceType.tolist()               --- convert the unique values of service request types to a list
ServiceType.sort()                               --- sort the values for easy inspection
dfTopService = pd.merge(df, TopService, how='left', left_on='Service Request Type',
                                                 right_on='Service_Request_Type_Simple')
        ---- Join two dataframes based on common values in both dataframes but every row in the left dataframe will
retain (left join) with the right dataframe on 'Service Request Type" to match the left dataframe on 'Service Request
Type Simple'

gdf = gpd.GeoDataFrame(dfTopService, geometry=gpd.points_from_xy(dfTopService.Longitude, dfTopService.Latitude))
 ---- convert a dataframe to a point gdf
gdf.crs   --- find the coordinate reference system of the gdf
gdf = gdf.set_crs(epsg=4326)  --- set the gdf coordinate reference system to wgs84 (most of latitude and longitude data likely be in wgs84)
gdf = gdf.to_crs(epsg=xxxx)  --- reproject the gdf to a new coordinate reference system (before any spatial analysis)
file = r'/content/drive/MyDrive/Package2/DallasTract2020.shp.zip'
DallasTracts = gpd.read_file(file)     ---- read a shapefile to a geodataframe
fig, ax = plt.subplots(figsize=(12,8))
DallasTracts.plot(ax=ax, column='sf_totalpo')
gdf.plot(ax=ax, color='red', markersize=0.1) ---- plot and make sure that geodataframes lined up before spatial
analysis
plt.tight_layout()
plt.show()
```

i. To show two geodataframes on a display and save the display to an image file

Spatial Analysis

```
CallsTracts = gpd.sjoin(gdf, DallasTracts, op='within')    --- spatial join
match = set(CallsTracts.index)
all = set(gdf.index)
unmatch = list(all - match)
len(unmatch)

gdf.reset_index(inplace=True)
list1 = []
list2 = []
for item in unmatch:
  Polygon_index = DallasTracts.distance(gdf.iloc[item]['geometry']).sort_values().index[0]
  list1.append(item)
  list2.append(DallasTracts.iloc[Polygon_index]['id'])

data = {'index': list1, 'id': list2}
unmatchDF = pd.DataFrame(data)

unmatchCalls = unmatchDF.merge(gdf, how='left', on='index')
--- select the columns useful for analysis, combine records from both dataframes and check if any missing data


CallTracts1 = CallsTracts[['id', 'Service Request Number','Top_Service_Request_Type']]
CallTracts2 = unmatchCalls[['id', 'Service Request Number','Top_Service_Request_Type']]
AllCallTracts = CallTracts1.append(CallTracts2)
AllCallTracts.info()
--- create a table to show how many calls for each service type in each census tract

NumCallsTracts = AllCallTracts.groupby(['id', 'Top_Service_Request_Type'])['Service Request Number'].count().unstack(level=1)
```

Top_Service_Request_Type	Animal	Code	Noise	Others	Parking	Sanitation	Traffic	Water

id								
48085031704	71.0	147.0	1.0	240.0	11.0	236.0	70.0	25.0
48085031706	47.0	68.0	NaN	123.0	9.0	324.0	31.0	14.0
48085031708	56.0	146.0	4.0	199.0	19.0	374.0	57.0	31.0
48085031709	58.0	102.0	NaN	205.0	21.0	269.0	105.0	25.0

9. What ar NumCal

NumCal

--- check the NumCallsTracts. Notice that 'id' is the index. Reset the index to get the 'id' to a column

```
NumCallsTracts.reset_index(inplace=True)
```

10. Enter value

--- identify the dataframe with id and population data
--- add total population in each tract so that we can adjust calls for tract population so that we can compare the calls among tracts equitably

NumC

```
TractPop = DallasTracts[['id', 'sf_totalpo']]
```

11. Challe under now. for th in the

--- Make sure that every tract has a population number (if not, something is wrong since the original data have population in each census tract in Dallas)
--- Some of the service types may have a null value (i.e. NaN) in a census tract. This indicates that some tract has no call for that service type.
--- Need to replace NaN with 0 to calculate proportions. Recall that we assess if a tract has more or less calls than another tract based the ratio between call proportion an

```
NumCallsTracts.fillna(0, inplace=True)
```

15 mi

```
-- join the population data

NumCallsTracts = NumCallsTracts.merge(TractPop, how='left', on='id')
```

call proportion in a tract = total number of calls for a service in the tract / total number of calls in Dallas
population proportion in a tract = total number of people in the tract / total population in Dallas

Ratio between call proportion and population proportion = call proportion in a tract / population proportion in a tract

Hints:

1. Create a list of columns in NumCallsTracts collist = list(df.columns)
2. Loop through each column in the list, construct new column names and
   calculate percentage new_df['ratio_col'] = df['col']/df['col'].sum() -- use different df name in case of mistakes

RatioCallsTracts

	id	ratio_Animal	ratio_Code	ratio_Noise	ratio_Others	ratio_Parking	ratio_Sanitation	ratio_Traffic	ratio_Water	ratio_sf_totalpo
0	48085031704	0.001562	0.001583	0.000497	0.002947	0.000553	0.002814	0.003106	0.000872	0.002641
1	48085031706	0.001034	0.000732	0.000000	0.001695	0.000452	0.003824	0.001376	0.000488	0.001584
2	48085031708	0.001232	0.001572	0.001988	0.002756	0.000955	0.004413	0.002530	0.001082	0.002637
3	48085031709	0.001276	0.001098	0.000000	0.002874	0.001055	0.003151	0.004660	0.000872	0.003110

3. Create a new df based on all proportioncall type columns divided by population proportion column
   Note that the column names remain the same but the dataframe name is different. Something like…

```
PropCallsTracts = RatioCallsTracts
PropCallsTracts = RatioCallsTracts.iloc[:, 1:9].div(RatioCallsTracts.ratio_sf_totalpo, axis=0)
```

PropCallsTracts

	ratio_Animal	ratio_Code	ratio_Noise	ratio_Others	ratio_Parking	ratio_Sanitation	ratio_Traffic	ratio_Water
0	0.591554	0.599347	0.188201	1.116006	0.209352	1.065560	1.176279	0.330281
1	0.652937	0.462283	0.000000	1.069972	0.285605	2.414500	0.868584	0.308397
2	0.467310	0.596205	0.753988	1.045105	0.362176	1.673825	0.959332	0.410192
3	0.410390	0.353178	0.000000	0.924069	0.339420	1.013203	1.498423	0.280490
4	0.925092	0.625641	0.234791	0.793598	0.736047	1.124829	0.482169	0.659270

To replace column names

```
collist = PropCallsTracts.columns
newlist = ['proportion_'+x.split('_')[1] for x in collist]
PropCallsTracts.columns = newlist
PropCallsTracts.insert(0,'id', RatioCallsTracts.iloc[:,0])
```

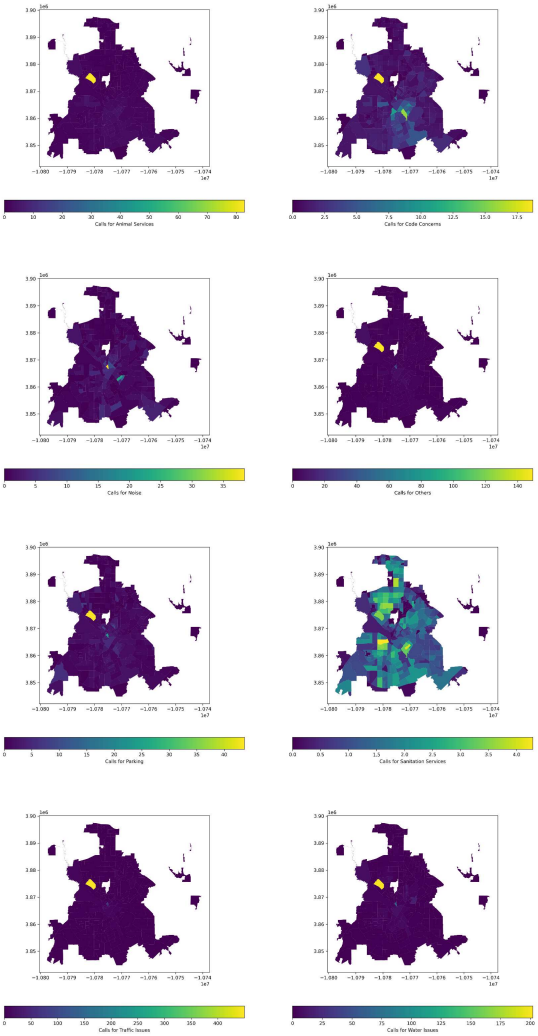
PropCallsTracts

	id	proportion_Animal	proportion_Code	proportion_Noise	proportion_Others	proportion_Parking	proportion_Sanitation	proportion_Traffic	proportion_Water
0	48085031704	0.589536	0.597303	0.187559	1.242743	0.208638	1.052908	1.172267	0.329155
1	48085031706	0.641576	0.454239	0.000000	1.047066	0.280635	2.376414	0.853471	0.303031
2	48085031708	0.465107	0.593394	0.750433	1.030710	0.360469	1.669026	0.954808	0.408258
3	48085031709	0.410990	0.353695	0.000000	0.905891	0.339916	1.024195	1.500616	0.280900

Join the proportional data back to DallasTracts for mapping. Replace NAN to 0 for proportional calls (no calls in these tracts)

```
fig, ax = plt.subplots(nrows=4, ncols=2, figsize=(20,40))
DallasTracts.plot(column='proportion_Animal', ax=ax[0,0], legend=True, legend_kwds={'label':'Calls for Animal Services', 'orientatic
DallasTracts.plot(column='proportion_Code', ax=ax[0,1], legend=True, legend_kwds={'label':'Calls for Code Concerns', 'orientation':
DallasTracts.plot(column='proportion_Noise', ax=ax[1,0], legend=True, legend_kwds={'label':'Calls for Noise', 'orientation':'horizor
DallasTracts.plot(column='proportion_Others', ax=ax[1,1], legend=True, legend_kwds={'label':'Calls for Others', 'orientation':'horiz
DallasTracts.plot(column='proportion_Parking', ax=ax[2,0], legend=True, legend_kwds={'label':'Calls for Parking', 'orientation':'hor
DallasTracts.plot(column='proportion_Sanitation', ax=ax[2,1], legend=True, legend_kwds={'label':'Calls for Sanitation Services', 'or
DallasTracts.plot(column='proportion_Traffic', ax=ax[3,0], legend=True, legend_kwds={'label':'Calls for Traffic Issues', 'orientatic
DallasTracts.plot(column='proportion_Water', ax=ax[3,1], legend=True, legend_kwds={'label':'Calls for Water Issues', 'orientation':
plt.tight_layout

fig.savefig(r'/content/drive/MyDrive/package2/maps.png', dpi=300)
```



Colormaps are available at
https://matplotlib.org/2.0.2/users/colormaps.html

```
-- What is the most common requst in each census tract?
--- Select the columns with counts
DallasTracts.iloc[: , ??:??]

-- reference to the codes in Lab package 1.1 on finding the type with the max count for each tract.
```

```
DallasTracts['MaxType'] = DallasTracts.iloc[: , ??:??].????(axis=?)

DallasTracts.plot(column='MaxType', legend=True, cmap='Set3', figsize=(15, 10))
plt.title('most requested service types in individual tracts')
```