Mapping Geographic data in Python

July 19, 2020

1 Import relavent packages

[1]:
```
import geopandas
from shapely.geometry import Point
import missingno as msn
import numpy as np
import pandas as pd
import shapefile as shp
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
# Initializing visual set
%matplotlib inline
```

Read the shape file from the computer

[3]:
```
shp_path = '/Users/admin/Downloads/aqueductglobalmaps21shp/
 ↪aqueduct_global_dl_20150409.shp'
sf = shp.Reader(shp_path)
```

[4]:
```
# type of file
type(sf)
```

[4]:
```
shapefile.Reader
```

Converting shapefile data on Pandas dataframe

[5]:
```
def read_shapefile(sf):
    """
    Read a shapefile into a Pandas dataframe with a 'coords'
    column holding the geometry information. This uses the pyshp
    package
    """
    fields = [x[0] for x in sf.fields][1:]
    records = sf.records()
    shps = [s.points for s in sf.shapes()]
    df = pd.DataFrame(columns=fields, data=records)
```

```
    df = df.assign(coords=shps)
    return df
```

[6]:
```
df = read_shapefile(sf)
df.shape
```

[6]:
```
(25010, 61)
```

[7]:
```
# making sure its a data frame
type(df)
```

[7]:
```
pandas.core.frame.DataFrame
```

[8]:
```
# random sample of 5 rows
df.sample(5)
```

[8]:
```
          GU  Shape_Leng    Shape_Area  BasinID    COUNTRY BASIN_NAME  \
5055    5056    0.007339  1.184325e-06     7711    Iceland
7136    7137   11.667353  2.032439e+00     1321      Egypt
21769  21770    0.010160  4.200627e-08     8539     Israel
16251  16252    1.305385  6.903196e-02    14407  Argentina      BAKER
924      925    2.676787  1.538498e-01    14103      Chile

        WITHDRAWAL    CONSUMPTIO            BA        BWS  …    W_CHEM  \
5055          28.0  2.848425e+00  5.922040e+05   0.000047  …  1.562862
7136    61859280.0  4.219933e+07  2.399529e+06  25.779759  …  1.785501
21769  526159296.0  3.738975e+08  1.007784e+08   5.220955  …  2.177283
16251   21839344.0  9.510263e+06  1.197106e+10   0.001824  …  0.717912
924    118388104.0  8.689599e+07  3.439762e+08   0.344175  …  2.302866

        W_POWER    W_MINE  W_OILGAS DEF_PQUANT     W_AGR  W_FOODBV     W_TEX  \
5055   0.639715  1.183392  1.788941   0.302920  0.395617  1.631812  0.688262
7136   3.026709  1.743568  1.102963   4.057425  3.150637  2.242610  2.579887
21769  2.831866  2.518652  1.050201   4.090933  2.977737  3.325252  3.491459
16251  0.789685  0.258541  0.415354   0.316861  0.463264  0.369810  0.282449
924    2.805554  2.444083  1.772900   2.855752  2.642259  2.416381  2.472154

                       OWR_cat  \
5055              Low risk (0-1)
7136   Medium to high risk (2-3)
21769            High risk (3-4)
16251             Low risk (0-1)
924    Medium to high risk (2-3)

                                                  coords
5055   [(-17.56103401221418, 65.97343675230383), (-17…
7136   [(28.118118132590382, 27.491568184661105), (28…
```

```
21769  [(34.258611286300265, 31.184144184787783), (34…
16251  [(-71.33692591055285, -46.158983564088146), (-…
924    [(-71.82992005322433, -34.45243532493754), (-7…

[5 rows x 61 columns]
```

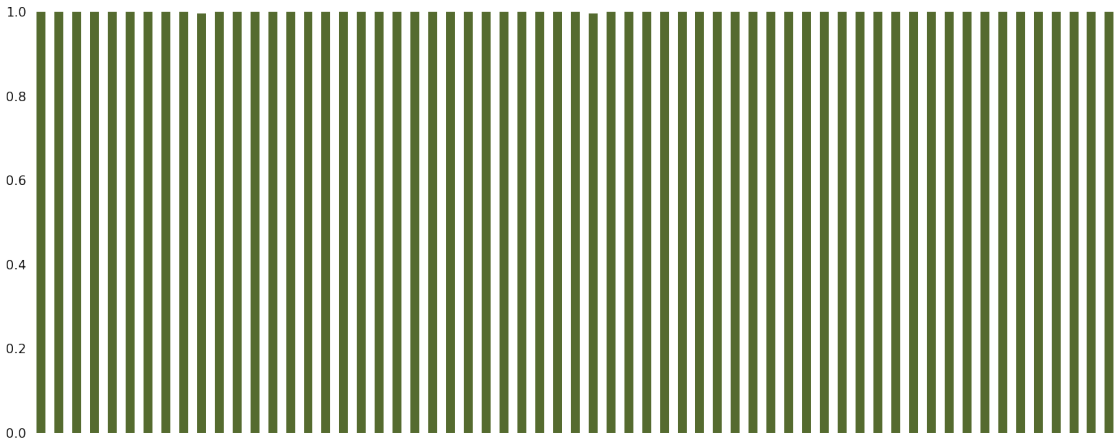
Exploratory data analysis: 1. Checking the information, data type 2. Any missing value 3. Statistical data

[9]:
```
# Checking the information, data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25010 entries, 0 to 25009
Data columns (total 61 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   GU          25010 non-null  int64
 1   Shape_Leng  25010 non-null  float64
 2   Shape_Area  25010 non-null  float64
 3   BasinID     25010 non-null  int64
 4   COUNTRY     25010 non-null  object
 5   BASIN_NAME  25010 non-null  object
 6   WITHDRAWAL  25010 non-null  float64
 7   CONSUMPTIO  25010 non-null  float64
 8   BA          25010 non-null  float64
 9   BWS         24916 non-null  float64
 10  BWS_s       25010 non-null  float64
 11  BWS_cat     25010 non-null  object
 12  WSV         25010 non-null  float64
 13  WSV_s       25010 non-null  float64
 14  WSV_cat     25010 non-null  object
 15  SV          25010 non-null  float64
 16  SV_s        25010 non-null  float64
 17  SV_cat      25010 non-null  object
 18  HFO         25010 non-null  float64
 19  HFO_s       25010 non-null  float64
 20  HFO_cat     25010 non-null  object
 21  DRO         25010 non-null  float64
 22  DRO_s       25010 non-null  float64
 23  DRO_cat     25010 non-null  object
 24  BT          25010 non-null  float64
 25  STOR        25010 non-null  float64
 26  STOR_s      25010 non-null  float64
 27  STOR_cat    25010 non-null  object
 28  GW          25010 non-null  float64
 29  GW_s        25010 non-null  float64
 30  GW_cat      25010 non-null  object
```

```
 31  WRI          24916 non-null  float64
 32  WRI_s        25010 non-null  float64
 33  WRI_cat      25010 non-null  object
 34  ECO_S        25010 non-null  float64
 35  ECO_S_s      25010 non-null  float64
 36  ECO_S_cat    25010 non-null  object
 37  MC           25010 non-null  float64
 38  MC_s         25010 non-null  float64
 39  MC_cat       25010 non-null  object
 40  ECO_V        25010 non-null  float64
 41  ECO_V_s      25010 non-null  float64
 42  ECO_V_cat    25010 non-null  object
 43  WCG          25010 non-null  float64
 44  WCG_s        25010 non-null  float64
 45  WCG_cat      25010 non-null  object
 46  DEF_PQUAL    25010 non-null  float64
 47  DEF_REGREP   25010 non-null  float64
 48  W_SEMICO     25010 non-null  float64
 49  DEFAULT      25010 non-null  float64
 50  W_CONSTR     25010 non-null  float64
 51  W_CHEM       25010 non-null  float64
 52  W_POWER      25010 non-null  float64
 53  W_MINE       25010 non-null  float64
 54  W_OILGAS     25010 non-null  float64
 55  DEF_PQUANT   25010 non-null  float64
 56  W_AGR        25010 non-null  float64
 57  W_FOODBV     25010 non-null  float64
 58  W_TEX        25010 non-null  float64
 59  OWR_cat      25010 non-null  object
 60  coords       25010 non-null  object
dtypes: float64(43), int64(2), object(16)
memory usage: 11.6+ MB
```

[10]:
```
# checking for any missing value
msn.bar(df, color='darkolivegreen');
```



[11]:
```
# statistical data
df.describe()
```

[11]:

	GU	Shape_Leng	Shape_Area	BasinID	WITHDRAWAL	\
count	25010.000000	25010.000000	2.501000e+04	25010.000000	2.501000e+04	
mean	12505.500000	4.657028	6.085516e-01	7342.814994	4.771135e+08	
std	7219.909452	8.493537	4.349143e+00	4446.593705	2.146302e+09	
min	1.000000	0.000095	3.018234e-10	-1.000000	-3.276700e+04	
25%	6253.250000	1.204026	2.519207e-02	3514.250000	5.748678e+05	
50%	12505.500000	2.792722	1.398187e-01	7128.000000	1.560863e+07	
75%	18757.750000	5.953255	5.310626e-01	11261.750000	1.729721e+08	
max	25010.000000	1004.848961	6.595916e+02	15006.000000	5.340514e+10	

	CONSUMPTIO	BA	BWS	BWS_s	WSV	\
count	2.501000e+04	2.501000e+04	24916.000000	25010.000000	25010.000000	
mean	2.513469e+08	2.521857e+10	-29.669867	-0.866664	-163.216090	
std	1.329732e+09	2.126521e+11	1671.665388	293.035887	2310.802793	
min	-3.276700e+04	-3.276700e+04	-32767.000000	-32767.000000	-32767.000000	
25%	8.334966e+04	9.147408e+07	0.000806	0.000000	0.270642	
50%	3.655522e+06	7.835471e+08	0.020030	0.080421	0.404305	
75%	6.788413e+07	5.075966e+09	0.202641	4.370467	0.673232	
max	3.564753e+10	6.050806e+12	190209.541317	5.000000	7.937254	

	…	DEFAULT	W_CONSTR	W_CHEM	W_POWER	\
count	…	25010.000000	25010.000000	25010.000000	25010.000000	
mean	…	-0.588408	-0.666400	-0.549585	-0.802877	
std	…	293.032715	293.032241	293.032186	293.030118	
min	…	-32767.000000	-32767.000000	-32767.000000	-32767.000000	
25%	…	1.089194	0.951097	1.395527	1.105113	
50%	…	1.858734	1.631046	1.972526	1.579172	
75%	…	2.912897	3.038578	2.667577	2.484798	

```
max    …      5.000000      5.000000      5.000000      5.000000

             W_MINE      W_OILGAS    DEF_PQUANT         W_AGR      W_FOODBV  \
count  25010.000000  25010.000000  25010.000000  25010.000000  25010.000000
mean      -0.501776     -0.647819     -0.684594     -0.755804     -0.529295
std      293.033053    293.031874    293.032914    293.031029    293.032550
min   -32767.000000 -32767.000000 -32767.000000 -32767.000000 -32767.000000
25%        1.333581      1.121818      0.748313      1.005495      1.377493
50%        2.133368      1.815041      1.495203      1.615565      2.038912
75%        2.837052      2.706945      3.244840      2.690274      2.686892
max        5.000000      5.000000      5.000000      5.000000      5.000000

              W_TEX
count  25010.000000
mean      -0.699455
std      293.031570
min   -32767.000000
25%        1.010364
50%        1.745756
75%        2.762228
max        5.000000

[8 rows x 45 columns]
```

Creating a Data Frame with limited data from df

[12]:
```
frame = pd.DataFrame({
    'Shape_Leng': df['Shape_Leng'],
    'Shape_Area': df['Shape_Area'],
    'COUNTRY': df['COUNTRY'],
    'BWS': df['BWS'],
    'BWS_s': df['BWS_s'],
    'BWS_cat': df['BWS_cat'],
    'coords': df['coords']
    }
)

frame.head()
```

[12]:
```
   Shape_Leng  Shape_Area        COUNTRY       BWS  BWS_s        BWS_cat  \
0    0.559986    0.005929  Guinea Bissau  0.032105    0.0  1. Low (<10%)
1    2.272372    0.099605  Guinea Bissau  0.002884    0.0  1. Low (<10%)
2    0.610379    0.017086         Guinea  0.003614    0.0  1. Low (<10%)
3    5.384966    0.636739  Guinea Bissau  0.003614    0.0  1. Low (<10%)
4    1.815190    0.055072  Guinea Bissau  0.003726    0.0  1. Low (<10%)

                                              coords
```

```
0  [(-14.752295656190142, 12.618836812527263), (-…
1  [(-16.137677267978745, 12.29409573338586), (-1…
2  [(-13.562381566112606, 12.667425473965409), (-…
3  [(-13.732176994830752, 12.578342493768957), (-…
4  [(-15.72857010960513, 11.971260070346545), (-1…
```

Restricting the dataframe only to India

[13]:
```
frame = frame[frame.COUNTRY=='India']
frame.head()
```

[13]:
```
       Shape_Leng  Shape_Area COUNTRY       BWS     BWS_s  \
11562    3.946209    0.291442   India  0.076408  0.611787
11563    3.389910    0.221784   India  1.307863  4.709139
11564    2.409330    0.106535   India  0.039209  0.000000
11565    2.350081    0.083753   India  1.656910  5.000000
11566    1.706614    0.065695   India  1.229345  4.619817

                      BWS_cat  \
11562             1. Low (<10%)
11563  5. Extremely high (>80%)
11564             1. Low (<10%)
11565  5. Extremely high (>80%)
11566  5. Extremely high (>80%)

                                                  coords
11562  [(75.49198144560114, 13.137811681128767), (75…
11563  [(79.89782121424452, 14.036834472408373), (79…
11564  [(75.64703294173268, 12.485170073197082), (75…
11565  [(79.84928219448398, 13.400194365845095), (79…
11566  [(80.32177685948642, 13.4209251806206), (80.31…
```

[14]:
```
# Re-indexing the data frame
frame = frame.reset_index(drop=True)
frame.head()
```

[14]:
```
   Shape_Leng  Shape_Area COUNTRY       BWS     BWS_s  \
0    3.946209    0.291442   India  0.076408  0.611787
1    3.389910    0.221784   India  1.307863  4.709139
2    2.409330    0.106535   India  0.039209  0.000000
3    2.350081    0.083753   India  1.656910  5.000000
4    1.706614    0.065695   India  1.229345  4.619817

                     BWS_cat                                          coords
0             1. Low (<10%)  [(75.49198144560114, 13.137811681128767), (75…
1  5. Extremely high (>80%)  [(79.89782121424452, 14.036834472408373), (79…
2             1. Low (<10%)  [(75.64703294173268, 12.485170073197082), (75…
```

```
3  5. Extremely high (>80%)  [(79.84928219448398, 13.400194365845095), (79…
4  5. Extremely high (>80%)  [(80.32177685948642, 13.4209251806206), (80.31…
```

[15]:
```
# length of frame.coords[0]
len(frame.coords[0])
```

[15]:
```
260
```

[16]:
```
len(frame.coords)
```

[16]:
```
598
```

Creating a data frame called 'new_frame' to which I append all the coordinates from the previous data frame 'frame'. This new data frame has two columns for now, namely, latitude and longitude. The coordinates are thus easier to work with as compared with the 'coords' column in the previous data frame 'frame'.

[17]:
```
new_frame = pd.DataFrame(columns=['latitude', 'longitude'])
for i in range(0,len(frame.coords)):
    new_frame = new_frame.append(pd.DataFrame(frame.coords[i],␣
 ↪columns=['latitude', 'longitude']))
new_frame
```

[17]:

	latitude	longitude
0	75.491981	13.137812
1	75.493501	13.128840
2	75.502347	13.131266
3	75.503865	13.122294
4	75.512711	13.124719
..	…	…
197	87.768816	25.027541
198	87.767734	25.036567
199	87.826170	25.044089
200	87.824014	25.062143
201	87.833755	25.063394

[125749 rows x 2 columns]

[18]:
```
new_frame['coordinates'] = new_frame[['longitude', 'latitude']].values.tolist()
new_frame.head()
```

[18]:

	latitude	longitude	coordinates
0	75.491981	13.137812	[13.137811681128767, 75.49198144560114]
1	75.493501	13.128840	[13.128839945471611, 75.49350102736446]
2	75.502347	13.131266	[13.131266129292953, 75.50234673742602]
3	75.503865	13.122294	[13.12229389361272, 75.50386533443168]
4	75.512711	13.124719	[13.124719004542897, 75.51271075311291]

[19]:
```
# checking the type of the coordinates column
type(new_frame['coordinates'])
```

[19]:
```
pandas.core.series.Series
```

[20]:
```
# Change the coordinates to a geoPoint
new_frame['coordinates'] = new_frame['coordinates'].apply(Point)
new_frame.head()
```

[20]:

	latitude	longitude	coordinates
0	75.491981	13.137812	POINT (13.13781168112877 75.49198144560114)
1	75.493501	13.128840	POINT (13.12883994547161 75.49350102736446)
2	75.502347	13.131266	POINT (13.13126612929295 75.50234673742602)
3	75.503865	13.122294	POINT (13.12229389361272 75.50386533443168)
4	75.512711	13.124719	POINT (13.1247190045429 75.51271075311291)

Converting the data frame 'newFrame' to a Geo frame

[21]:
```
# Convert the count df to geodf
new_frame = geopandas.GeoDataFrame(new_frame, geometry='coordinates')
new_frame.head()
```

[21]:

	latitude	longitude	coordinates
0	75.491981	13.137812	POINT (13.13781 75.49198)
1	75.493501	13.128840	POINT (13.12884 75.49350)
2	75.502347	13.131266	POINT (13.13127 75.50235)
3	75.503865	13.122294	POINT (13.12229 75.50387)
4	75.512711	13.124719	POINT (13.12472 75.51271)

[22]:
```
# making sure that 'newFrame' is now a geo data frame
type(new_frame)
```

[22]:
```
geopandas.geodataframe.GeoDataFrame
```

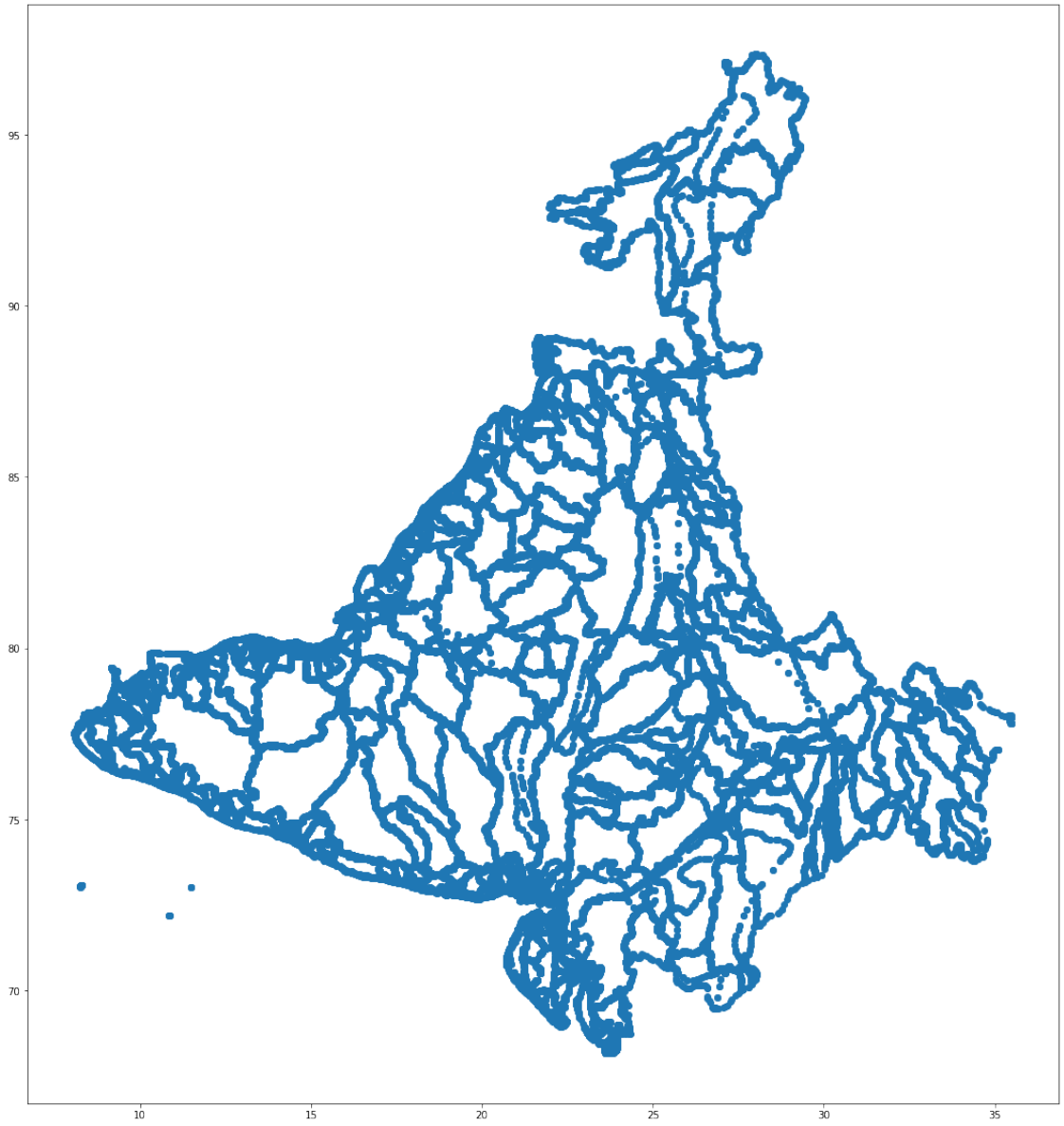
[23]:
```
# checking the type of the coordinates column
type(new_frame['coordinates'])
```

[23]:
```
geopandas.geoseries.GeoSeries
```

2 Visualization

Now that we have successfully converted the data frame into a geo data frame, which contains coordinates that correspond to different locations in India, we can plot them and see what we get!

[24]:
```
new_frame.plot(figsize=(30,20));
```



Woops!!! It's India but upside down! Well this is good progress.