Podstawy programowania

Biblioteki analizy danych: Shapely, Psycopg2, Pandas

Inne Biblioteki GeoPythona

- Shapely
- Psypog2
- Pandas i Geopandas

Biblioteka shapely

Biblioteka służąca do przetwarzania i analizowania obiektów geometrycznych w formacie well-known text

```
from shapely.geometry import Polygon
from shapely.geometry import Point
from shapely.geometry import LineString

polygon = Polygon([(0, 0), (0, 1), (1, 1),(1, 0)])
polygon.area #obliczenie powierzchni
polygon.length #długość obwiedni
polygon.bounds #granice obwiedni
polygon.geom_type
Point(0,0).distance(Point(1,1)) #odległość
line = LineString([(0, 0), (1, 1)])
line.length
```

Shapely: analiza geometrii

```
import ogr, shapely.wkt

ds=ogr.Open(r'C:/GIS',1)
lyr=ds.GetLayer('gminy_wielkopolska')
lyr.SetAttributeFilter("jpt_nazwa_ = 'Poznań'")
feat = lyr.GetNextFeature()

geom = feat.GetGeometryRef()
wkt = geom.ExportToWkt()
#załadowanie do obiektu shapely
outline = shapely.wkt.loads(wkt)
#odczyt właściwości
print (outline.area, outline.length)
print(outline.bounds, outline.geom_type)
print (outline.centroid.x, outline.centroid.y)
```

Shapely – funkcja interpolate

```
>>> from shapely.geometry import Point, LineString
>>> line = LineString([(0, 0), (1, 1)])
>>> print( line)
LINESTRING (0 0, 1 1)
>>> point=Point(0,1)
>>> print(point)
POINT (0 1)
>>> pt_interpolate=line.interpolate(line.project(point))
>>> print (pt_interpolate)
POINT (0.5 0.5)
>>> point.distance(pt_interpolate)
0.7071067811865476
>>> pt_025=line.interpolate(0.25)
>>> print( pt_025)
POINT (0.1767766952966369 0.1767766952966369)
```

Biblioteka Psycopg2: import shapefile do bazy danych

```
import ogr, psycopg2
#psycopg2 –biblioteka przeznaczona do komunikacji z bazą danych PostgreSQL
#nawiązanie połączenia z bazą danych
connection = psycopg2.connect(database="wielkopolska", user="postgres", password="haslo")
#utworzenie obiektu cursor służącego do wykonywania zapytań sql
cursor = connection.cursor()
#wysłanie zapytania kasowania tabeli
cursor.execute("DROP TABLE IF EXISTS gis.lasy")
#wysłanie zapytania utworzenie nowej tabeli
cursor.execute("CREATE TABLE gis.lasy (" +
    "gid SERIAL PRIMARY KEY," +
    "typ VARCHAR," +
    "nazwa VARCHAR," +
    "geom geometry(multipolygon,2180))")
#wysłanie zapytania nałożenie indeksu przestrzennego
cursor.execute("CREATE INDEX lasy_index ON gis.lasy USING GIST(geom)")
```

Biblioteka Psycopg2: import shapefile do bazy danych c.d.

```
#odczyt warstwy shapefile
ds = ogr.Open(r'C:/GIS/osm_landuse',1)
layer = ds.GetLayer('landuse')
layer.SetAttributeFilter("fclass='forest'")

for feat in layer:
    fclass = feat.GetField("fclass")
    name = feat.GetField("name")
    geometry = feat.GetGeometryRef()
    wkt = geometry.ExportToWktgeometrii do postaci wkt
    #wysłanie zapytania wstawianie danych do ()#przepisanie tabeli
    cursor.execute("INSERT INTO gis.lasy (typ, nazwa, geom) VALUES (%s, %s,
    ST_Multi(ST_GeomFromText(%s,2180)))", (fclass, name, wkt))

#zatwierdzenie zmian w bazie danych
connection.commit()
```

Biblioteka Psycopg2: odczyt z bazy danych PostGIS

```
import ogr, psycopg2

#nawiązanie połączenia
connection = psycopg2.connect(database="wielkopolska", user="postgres",
    password="haslo")
cursor = connection.cursor()

#wykonanie zapytania
cursor.execute("SELECT gid, name, ST_Area(geom) as area FROM gis.landuse
    ORDER BY name LIMIT 10")
#wyświetlenie wyników
for gid, name, area in cursor:
    print(gid, name, area)
```

Biblioteka Pandas

Biblioteka wprowadza nowe typy danych:

- Serie
- DataFrame

Serie

Seria to jednowymiarowa tablica zawierająca sekwencję wartości indeksowanych liczbami lub dowolnymi wartościami

```
>>> import pandas as pd
>>> obj = pd.Series([4, 7, -5, 3])
>>> obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
```

Serie

```
>>> S = pd.Series([11, 28, 72, 3, 5, 8])
>>> S
0  11
1  28
2  72
3   3
4   5
5   8
dtype: int64
>>> print(S.index)
RangeIndex(start=0, stop=6, step=1)
>>> print(S.values)
[11 28 72  3  5  8]
#Serie są podobne do tablic numpy
>>> import numpy as np
>>> X = np.array([11, 28, 72, 3, 5, 8])
>>> print(X)
[11 28 72  3  5  8]
```

Serie

```
>>> fruits = ['apples', 'oranges', 'cherries', 'pears']
>>> quantities = [20, 33, 52, 10]
>>> S = pd.Series(quantities, index=fruits)
>>> S
apples      20
oranges     33
cherries    52
pears       10
dtype: int64
```

W efekcie powstała seria indeksowana nazwami w przeciwieństwie do numpy

Takie same serie można do siebie dodawać:

```
>>> S2 = pd.Series([17, 13, 31, 32], index=fruits)
>>> print(S + S2)
apples      37
oranges     46
cherries    83
pears       42
dtype: int64
>>>
```

Serie

- Dodawanie serii z różnymi indeksami

```
>>> fruits = ['peaches', 'oranges', 'cherries', 'pears']
>>> fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']
>>> S = pd.Series([20, 33, 52, 10], index=fruits)
>>> S2 = pd.Series([17, 13, 31, 32], index=fruits2)
>>> print(S + S2)
cherries       83.0
oranges        46.0
peaches         NaN
pears         42.0
raspberries     NaN
dtype: float64
```

Obliczenia w seriach

```
>>> obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
>>> obj2[obj2 > 0]
d    4
b    7
c    3
dtype: int64
>>> obj2 * 2
d     8
b    14
a   -10
c     6
dtype: int64
>>> import numpy as np
>>> np.exp(obj2)
d      54.598150
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

Zamiana słowników w serie

```
>>> cities = {"London":   8615246,
        "Berlin":   3562166,
        "Madrid":   3165235,
        "Rome":     2874038,
        "Paris":    2273305,
        "Vienna":   1805681,
        "Bucharest":1803425,
        "Hamburg":  1760433,
        "Budapest": 1754000,
        "Warsaw":   1740119,
        "Barcelona":1602386,
        "Munich":   1493900,
        "Milan":    1350680}

>>> city_series = pd.Series(cities)
>>> print(city_series)
Barcelona    1602386
Berlin       3562166
Bucharest    1803425
Budapest     1754000
Hamburg      1760433
London       8615246
Madrid       3165235
Milan        1350680
Munich       1493900
Paris        2273305
Rome         2874038
Vienna       1805681
Warsaw       1740119
dtype: int64
```

DataFrame

- Podstawą utworzenia typu DataFrame jest możliwość pracy na danych w postaci Arkuszów jak w Excelu
- Każda kolumna może przechowywać różne wartości jednego typu, natomiast kolumny mogą być różnych typów
- Dataframe jest połączeniem serii będących kolumnami

Tworzenie DataFrame z serii

```
>>> years = range(2014, 2018)
>>> shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index=years)
>>> shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index=years)
>>> shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index=years)
>>> shops_df = pd.concat([shop1, shop2, shop3], axis=1)
>>> shops_df
                0        1        2
2014  2409.14  1203.45  3412.12
2015  2941.01  3441.62  3491.16
2016  3496.83  3007.83  3457.19
2017   3119.55  3619.53  1963.10

#Dodanie nazw do kolumn
>>> cities = ["Zürich", "Winterthur", "Freiburg"]
>>> shops_df.columns = cities
>>> shops_df
     Zürich  Winterthur  Freiburg
2014  2409.14    1203.45   3412.12
2015  2941.01    3441.62   3491.16
2016  3496.83    3007.83   3457.19
2017  3119.55    3619.53   1963.10
```

Tworzenie DataFrame ze słownika zawierającego tablice

```
>>>cities = {"name": ["London", "Berlin", "Madrid", "Rome", "Paris",
    "Vienna", "Bucharest", "Hamburg", "Budapest", "Warsaw", "Barcelona",
    "Munich", "Milan"],
"population": [8615246, 3562166, 3165235, 2874038, 2273305, 1805681,
    1803425, 1760433, 1754000, 1740119, 1602386, 1493900, 1350680],
"country": ["England", "Germany", "Spain", "Italy", "France", "Austria",
    "Romania", "Germany", "Hungary", "Poland", "Spain", "Germany", "Italy"]}

>>>city_frame = pd.DataFrame(cities)
```

DataFrame ze słownika

```
>>> city_frame
```

	country	name	population
0	England	London	8615246
1	Germany	Berlin	3562166
2	Spain	Madrid	3165235
3	Italy	Rome	2874038
4	France	Paris	2273305
5	Austria	Vienna	1805681
6	Romania	Bucharest	1803425
7	Germany	Hamburg	1760433
8	Hungary	Budapest	1754000
9	Poland	Warsaw	1740119
10	Spain	Barcelona	1602386
11	Germany	Munich	1493900
12	Italy	Milan	1350680

Ustawienie indeksu wierszy

city_frame.set_index("country", inplace=True)

country	name	population
England	London	8615246
Germany	Berlin	3562166
Spain	Madrid	3165235
Italy	Rome	2874038
France	Paris	2273305
Austria	Vienna	1805681
Romania	Bucharest	1803425
Germany	Hamburg	1760433
Hungary	Budapest	1754000
Poland	Warsaw	1740119
Spain	Barcelona	1602386
Germany	Munich	1493900
Italy	Milan	1350680

Dodanie nowej kolumny z listy

```
>>> area = [1572, 891.85, 605.77, 1285,
      105.4, 414.6, 228, 755,
      525.2, 517, 101.9, 310.4,
      181.8]
>>> city_frame["area"] = area
```

country	name	population	area
England	London	8615246	1572.00
Germany	Berlin	3562166	891.85
Spain	Madrid	3165235	605.77
Italy	Rome	2874038	1285.00
France	Paris	2273305	105.40
Austria	Vienna	1805681	414.60
Romania	Bucharest	1803425	228.00
Germany	Hamburg	1760433	755.00
Hungary	Budapest	1754000	525.20
Poland	Warsaw	1740119	517.00
Spain	Barcelona	1602386	101.90
Germany	Munich	1493900	310.40
Italy	Milan	1350680	181.80

Dostęp do kolumn

Kolumna DataFrame jest serią

```
>>> s=city_frame["area"]
```

```
name
London       1572.00
Berlin        891.85
Madrid        605.77
Rome         1285.00
Paris         105.40
Vienna        414.60
Bucharest     228.00
Hamburg       755.00
Budapest      525.20
Warsaw        517.00
Barcelona     101.90
Munich        310.40
Milan         181.80
Name: area, dtype: float64
```

Przykład II : DataFrame

```
>>>cities = {"name": ["London", "Berlin", "Madrid", "Rome", "Paris", "Vienna", "Bucharest",
    "Hamburg", "Budapest", "Warsaw", "Barcelona", "Munich", "Milan"],
"population": [8615246, 3562166, 3165235, 2874038, 2273305, 1805681, 1803425, 1760433,
    1754000, 1740119, 1602386, 1493900, 1350680],
"area" : [1572, 891.85, 605.77, 1285, 105.4, 414.6, 228, 755, 525.2, 517, 101.9, 310.4, 181.8] }
>>>city_frame = pd.DataFrame(cities,
                columns=["population", "area"],
                index=cities["name"])
>>> print(city_frame)
```

	population	area
London	8615246	1572.00
Berlin	3562166	891.85
Madrid	3165235	605.77
Rome	2874038	1285.00
Paris	2273305	105.40
Vienna	1805681	414.60
Bucharest	1803425	228.00
Hamburg	1760433	755.00
Budapest	1754000	525.20
Warsaw	1740119	517.00
Barcelona	1602386	101.90
Munich	1493900	310.40
Milan	1350680	181.80

Dodanie kolumny do DataFrame

```
>>> city_frame['density']=city_frame['population']/city_frame['area']
>>> city_frame
```

	population	area	density
London	8615246	1572.00	5480.436387
Berlin	3562166	891.85	3994.131300
Madrid	3165235	605.77	5225.143206
Rome	2874038	1285.00	2236.605447
Paris	2273305	105.40	21568.358634
Vienna	1805681	414.60	4355.236372
Bucharest	1803425	228.00	7909.758772
Hamburg	1760433	755.00	2331.699338
Budapest	1754000	525.20	3339.680122
Warsaw	1740119	517.00	3365.800774
Barcelona	1602386	101.90	15725.083415
Munich	1493900	310.40	4812.822165
Milan	1350680	181.80	7429.482948

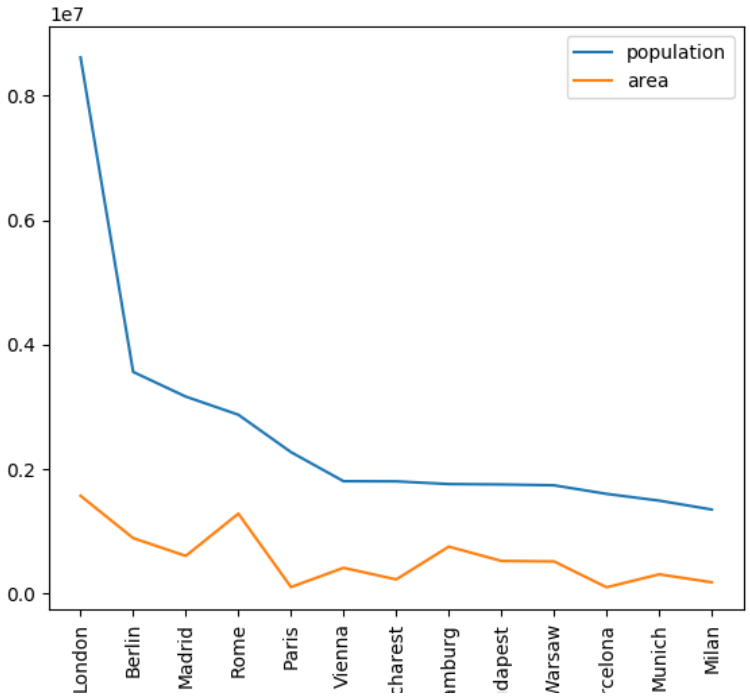
Odczyt wartości

- >>> city_frame.values

```
array([[  9.08821727e+00,   1.20000000e+01,   4.88713579e-02],
       [  2.46278009e-01,   2.00000000e+00,   1.32434562e-03],
       [  1.63085033e+02,   1.37000000e+02,   8.76980245e-01],
       [  5.01244697e+02,   8.00000000e+01,   2.69541410e+00],
       [  1.22170552e+00,   3.30000000e+01,   6.56965011e-03],
       [  2.14093131e+01,   3.70000000e+01,   1.15127332e-01],
       [  1.11063495e+01,   1.06000000e+02,   5.97237463e-02],
       [  9.34994114e+00,   1.00000000e+00,   5.02787627e-02],
       [  3.67497346e+01,   4.00000000e+00,   1.97619553e-01],
       [  3.46469529e+00,   6.00000000e+00,   1.86311966e-02],
       [  4.55218078e-01,   3.00000000e+00,   2.44790864e-03],
       [  8.57256366e+01,   1.28000000e+02,   4.60984606e-01],
       [  2.47586175e-02,   1.00000000e+00,   1.33138020e-04]])
...
```

Wykresy

```
import matplotlib.pyplot as plt
city_frame['population'].plot(use_index=True, xticks=range(len(city_frame.index)))
<matplotlib.axes._subplots.AxesSubplot object at 0x066BBFD0>
>>> plt.show()
```



Przykład III

```
c={'residential': 8572563.655673558, 'park': 45521.80777122383,
    'forest': 50124469.74239494, 'meadow': 1110634.9548526453,
    'orchard': 346469.5287002488, 'nature_reserve':
    3674973.4626020906, 'cemetery': 24627.800913020037, 'retail':
    2475.861748448573, 'military': 934994.1140836796, 'industrial':
    2140931.312884256, 'allotments': 908821.7274399129, 'grass':
    122170.55198922203, 'farm': 16308503.311590474}
d={'residential': 128, 'park': 3, 'forest': 80, 'meadow': 106, 'orchard': 6,
    'nature_reserve': 4, 'cemetery': 2, 'retail': 1, 'military': 1, 'industrial':
    37, 'allotments': 12, 'grass': 33, 'farm': 137}


seria_area=pd.Series(c)/100000
seria_count=pd.Series(d)
```

Serie

```
allotments          9.088217
cemetery            0.246278
farm              163.085033
forest            501.244697
grass               1.221706
industrial         21.409313
meadow             11.106350
military            9.349941
nature_reserve     36.749735
orchard             3.464695
park                0.455218
residential        85.725637
retail              0.024759
dtype: float64
```

```
allotments      12
cemetery         2
farm           137
forest          80
grass           33
industrial      37
meadow         106
military         1
nature_reserve   4
orchard          6
park             3
residential    128
retail           1
dtype: int64
```

Utworzenie DataFrame

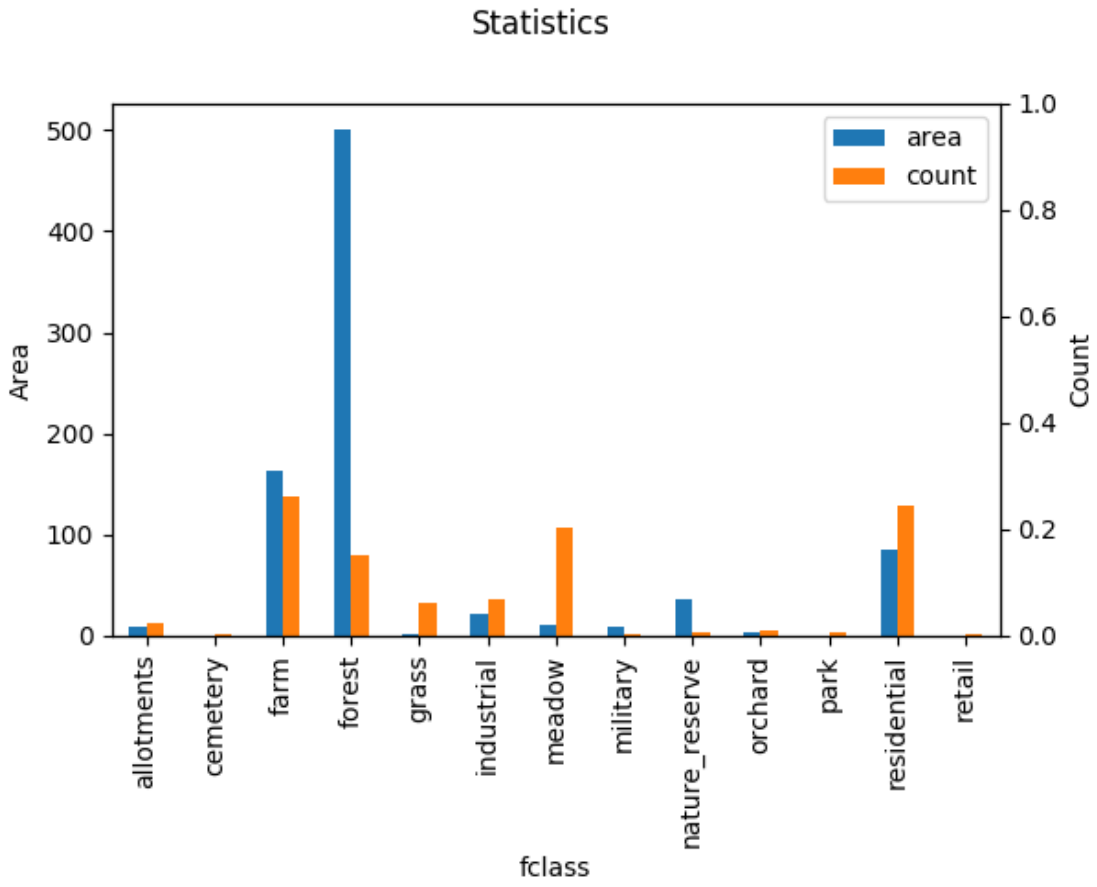
```
data_frame=pd.concat([seria_area,seria_count],axis=1)
data_frame.columns=["area","count"]
```

	area	count
allotments	9.088217	12
cemetery	0.246278	2
farm	163.085033	137
forest	501.244697	80
grass	1.221706	33
industrial	21.409313	37
meadow	11.106350	106
military	9.349941	1
nature_reserve	36.749735	4
orchard	3.464695	6
park	0.455218	3
residential	85.725637	128
retail	0.024759	1

```
#generowanie parametrów wykresu
fig, ax = plt.subplots()
fig.suptitle("Statistics")
ax.set_ylabel("Area")
ax.set_xlabel("fclass")
ax2 = ax.twinx()
ax2.set_ylabel("Count")
```

Wykres na podstawie DataFrame

```
data_frame.plot( ax=ax, use_index=True, rot=90, kind="bar",xticks=range(len(data_frame.index)))
plt.show()
```



Dodatkowe obliczenia

data_frame["prcn"]=data_frame["area"]/(clip_geom.GetArea()/10000)*100

	area	count	prcn
allotments	9.088217	12	0.048871
cemetery	0.246278	2	0.001324
farm	163.085033	137	0.876980
forest	501.244697	80	2.695414
grass	1.221706	33	0.006570
industrial	21.409313	37	0.115127
meadow	11.106350	106	0.059724
military	9.349941	1	0.050279
nature_reserve	36.749735	4	0.197620
orchard	3.464695	6	0.018631
park	0.455218	3	0.002448
residential	85.725637	128	0.460985
retail	0.024759	1	0.000133

Obliczenia statystyk

```
>>> data_frame.describe()
```

	area	count	prcn
count	13.000000	13.000000	13.000000
mean	64.859352	42.307692	0.348777
std	139.230125	51.791867	0.748702
min	0.024759	1.000000	0.000133
25%	1.221706	3.000000	0.006570
50%	9.349941	12.000000	0.050279
75%	36.749735	80.000000	0.197620
max	501.244697	137.000000	2.695414

Sumy skumulowane

- >>> data_frame["cumsum_area"]=data_frame["area"].cumsum()

	area	count	prcn	cumsum_area
allotments	9.088217	12	0.048871	9.088217
cemetery	0.246278	2	0.001324	9.334495
farm	163.085033	137	0.876980	172.419528
forest	501.244697	80	2.695414	673.664226
grass	1.221706	33	0.006570	674.885931
industrial	21.409313	37	0.115127	696.295244
meadow	11.106350	106	0.059724	707.401594
military	9.349941	1	0.050279	716.751535
nature_reserve	36.749735	4	0.197620	753.501270
orchard	3.464695	6	0.018631	756.965965
park	0.455218	3	0.002448	757.421183
residential	85.725637	128	0.460985	843.146820
retail	0.024759	1	0.000133	843.171578

Sortowanie DataFrame

- >>> data_frame.sort_values(by='area')

	area	count	prcn	cumsum_area
retail	0.024759	1	0.000133	843.171578
cemetery	0.246278	2	0.001324	9.334495
park	0.455218	3	0.002448	757.421183
grass	1.221706	33	0.006570	674.885931
orchard	3.464695	6	0.018631	756.965965
allotments	9.088217	12	0.048871	9.088217
military	9.349941	1	0.050279	716.751535
meadow	11.106350	106	0.059724	707.401594
industrial	21.409313	37	0.115127	696.295244
nature_reserve	36.749735	4	0.197620	753.501270
residential	85.725637	128	0.460985	843.146820
farm	163.085033	137	0.876980	172.419528
forest	501.244697	80	2.695414	673.664226

Odczyt plików csv do DataFrame

```
csv_fn=r'C:/Users/Robert/Desktop/GIS/GI22.csv'
>>> df = pd.read_csv(csv_fn, sep=',')
>>>df = pd.read_csv(csv_fn, sep=',', index_col=['id'])
>>>df
>>> df
```

id	y	x	z
BAZA	5.815040e+06	6.428067e+06	85.0300
F1	5.815040e+06	6.428064e+06	85.0773
F2	5.815046e+06	6.428066e+06	85.1258
F3	5.815055e+06	6.428065e+06	85.2699
F4	5.815071e+06	6.428069e+06	85.3996
F5	5.815076e+06	6.428068e+06	85.4192
F6	5.815082e+06	6.428062e+06	85.4937
F7	5.815089e+06	6.428069e+06	85.4030
F8	5.815103e+06	6.428069e+06	85.4383
F9	5.815117e+06	6.428079e+06	85.4079