Manipulação e análise de dados com o Postgis - II

Tópicos Avançados de Bases de Dados

Geometrias PostGIS

```
CREATE TABLE geometries (name varchar, geom geometry);

INSERT INTO geometries VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
  ('PolygonWithHole', 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))'),
  ('Collection', 'GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))');
```

```
SELECT name, ST_AsText(geom) FROM geometries;
```

- **ST_GeometryType(geometry)** returns the type of the geometry
- **ST_NDims(geometry)** returns the number of dimensions of the geometry
- **ST_SRID(geometry)** returns the spatial reference identifier number of the geometry

```
SELECT name, ST_GeometryType(geom), ST_NDims(geom), ST_SRID(geom)
  FROM geometries;
```

name	st_geometrytype	st_ndims	st_srid
Point	ST_Point	2	0
Polygon	ST_Polygon	2	0
PolygonWithHole	ST_Polygon	2	0
Collection	ST_GeometryCollection	2	0
Linestring	ST_LineString	2	0

Points

A spatial **point** represents a single location on the Earth. This point is represented by a single coordinate (including either 2-, 3- or 4-dimensions).
Points are used to represent objects when the exact details, such as shape and size, are not important at the target scale. For example, cities on a map of the world can be described as points, while a map of a single state might represent cities as polygons.

```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name = 'Point';
```

```
POINT(0 0)
```

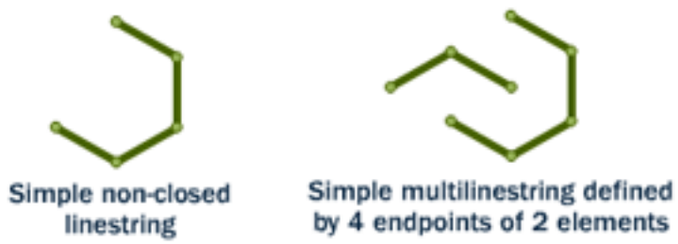
Some of the specific spatial functions for working with points are:

- **ST_X(geometry)** returns the X ordinate
- **ST_Y(geometry)** returns the Y ordinate

So, we can read the ordinates from a point like this:

```
SELECT ST_X(geom), ST_Y(geom)
  FROM geometries
  WHERE name = 'Point';
```

Linestrings



A **linestring** is a path between locations. It takes the form of an ordered series of two or more points. Roads and rivers are typically represented as linestrings. A linestring is said to be **closed** if it starts and ends on the same point. It is said to be **simple** if it does not cross or touch itself (except at its endpoints if it is closed). A linestring can be both **closed** and **simple**.

The following SQL query will return the geometry associated with one linestring (in the **ST_AsText** column).

```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name = 'Linestring';
LINESTRING(0 0, 1 1, 2 1, 2 2)
```

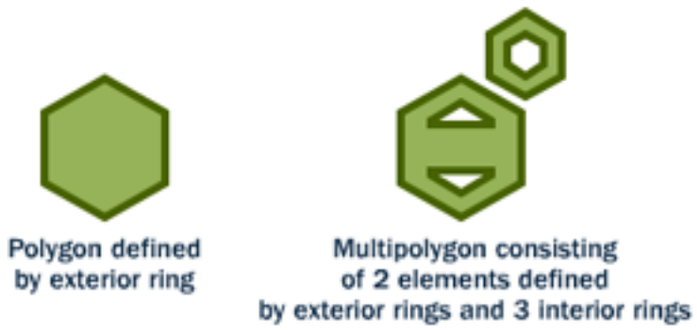
Some of the specific spatial functions for working with linestrings are:

- **ST_Length(geometry)** returns the length of the linestring
- **ST_StartPoint(geometry)** returns the first coordinate as a point
- **ST_EndPoint(geometry)** returns the last coordinate as a point
- **ST_NPoints(geometry)** returns the number of coordinates in the linestring

So, the length of our linestring is:

```
SELECT ST_Length(geom)
  FROM geometries
  WHERE name = 'Linestring';
```

Polygons



A polygon is a representation of an area. The outer boundary of the polygon is represented by a ring.
This ring is a linestring that is both closed and simple as defined above.
Holes within the polygon are also represented by rings.
Polygons are used to represent objects whose size and shape are important.
City limits, parks, building footprints or bodies of water are all commonly represented as polygons when the scale is sufficiently high to see their area. Roads and rivers can sometimes be represented as polygons.

The following SQL query will return the geometry associated with one linestring (in the **ST_AsText** column).

```
SELECT ST_AsText(geom)
  FROM geometries
  WHERE name LIKE 'Polygon%';
```

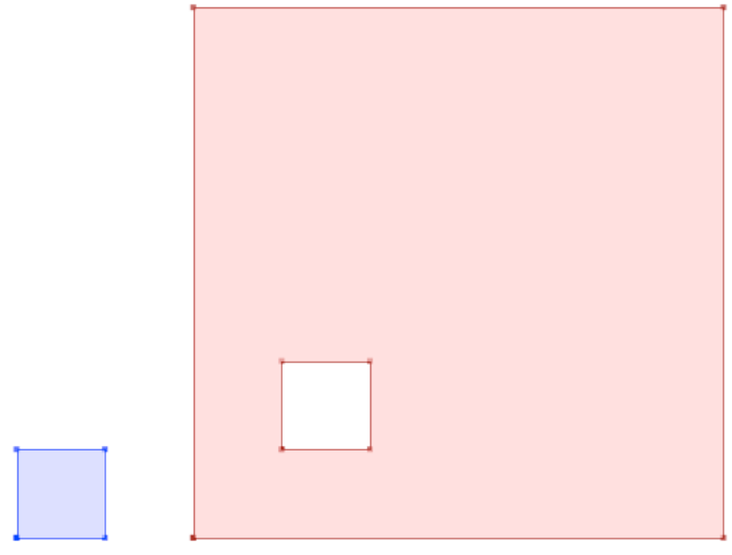
Polygons

```
POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))
POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))
```

The first polygon has only one ring. The second one has an interior “hole”. Most graphics systems include the concept of a “polygon”, but GIS systems are relatively unique in allowing polygons to explicitly have holes.

Some of the specific spatial functions for working with polygons are:

- **ST_Area(geometry)** returns the area of the polygons
- **ST_NRings(geometry)** returns the number of rings (usually 1, more of there are holes)
- **ST_ExteriorRing(geometry)** returns the outer ring as a linestring
- **ST_InteriorRingN(geometry,n)** returns a specified interior ring as a linestring
- **ST_Perimeter(geometry)** returns the length of all the rings



Collections

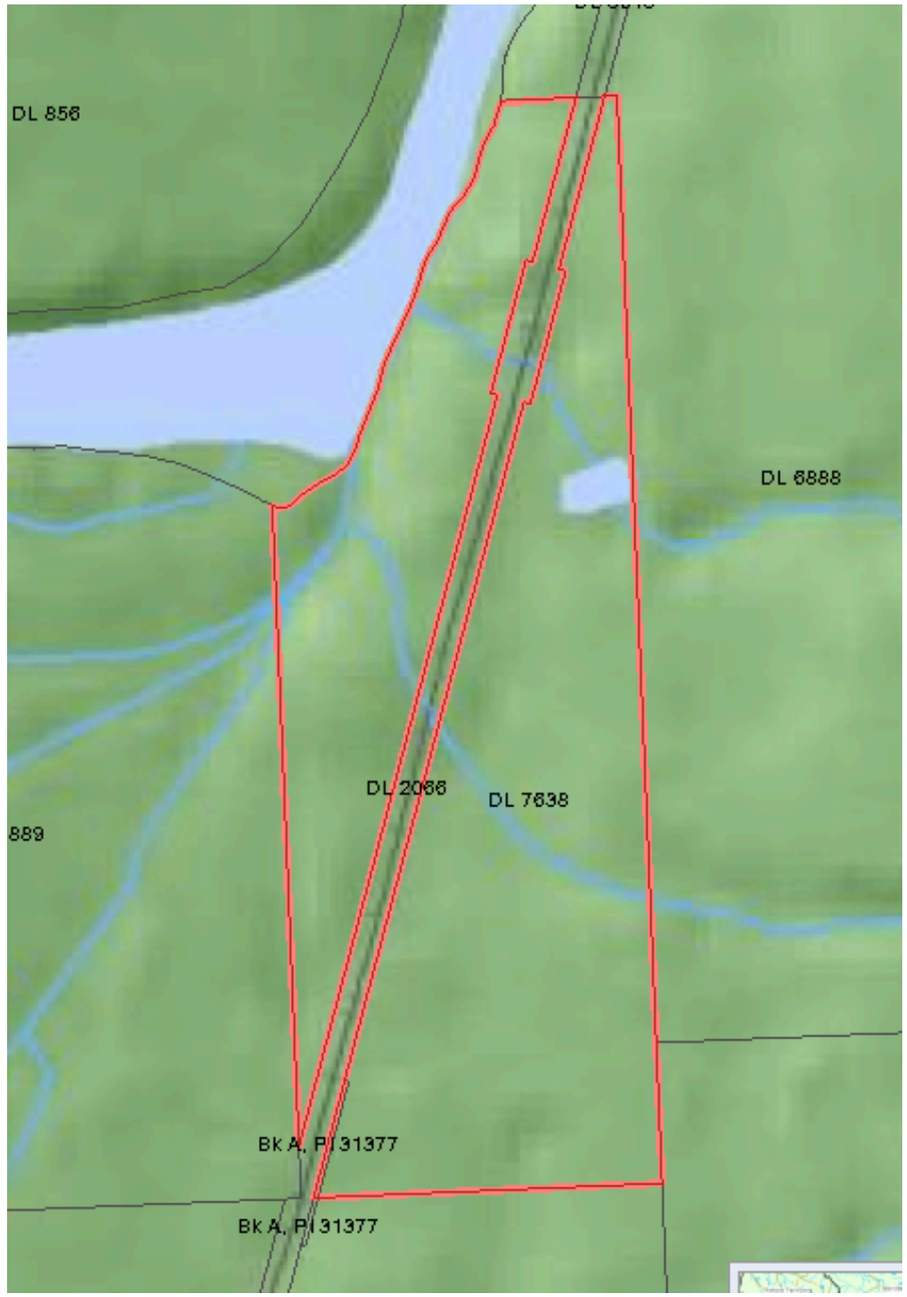
There are four collection types, which group multiple simple geometries into sets.

- **MultiPoint**, a collection of points
- **MultiLineString**, a collection of linestrings
- **MultiPolygon**, a collection of polygons
- **GeometryCollection**, a heterogeneous collection of any geometry (including other collections)

Collections are another concept that shows up in GIS software more than in generic graphics software.

They are useful for directly modeling real world objects as spatial objects.

For example, how to model a lot that is split by a right-of-way?
As a **MultiPolygon**, with a part on either side of the right-of-way.



Collections

Our example collection contains a polygon and a point:

```
SELECT name, ST_AsText(geom)
  FROM geometries
  WHERE name = 'Collection';
```

```
GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))
```

Some of the specific spatial functions for working with collections are:

- **ST_NumGeometries(geometry)** returns the number of parts in the collection
- **ST_GeometryN(geometry,n)** returns the specified part
- **ST_Area(geometry)** returns the total area of all polygonal parts
- **ST_Length(geometry)** returns the total length of all linear parts

Nova tabela de polígonos

http://www.dgterritorio.pt/cartografia_e_geodesia/cartografia/carta_administrativa_oficial_de_portugal_caop/caop__download_/carta_administrativa_oficial_de_portugal___versao_2018/

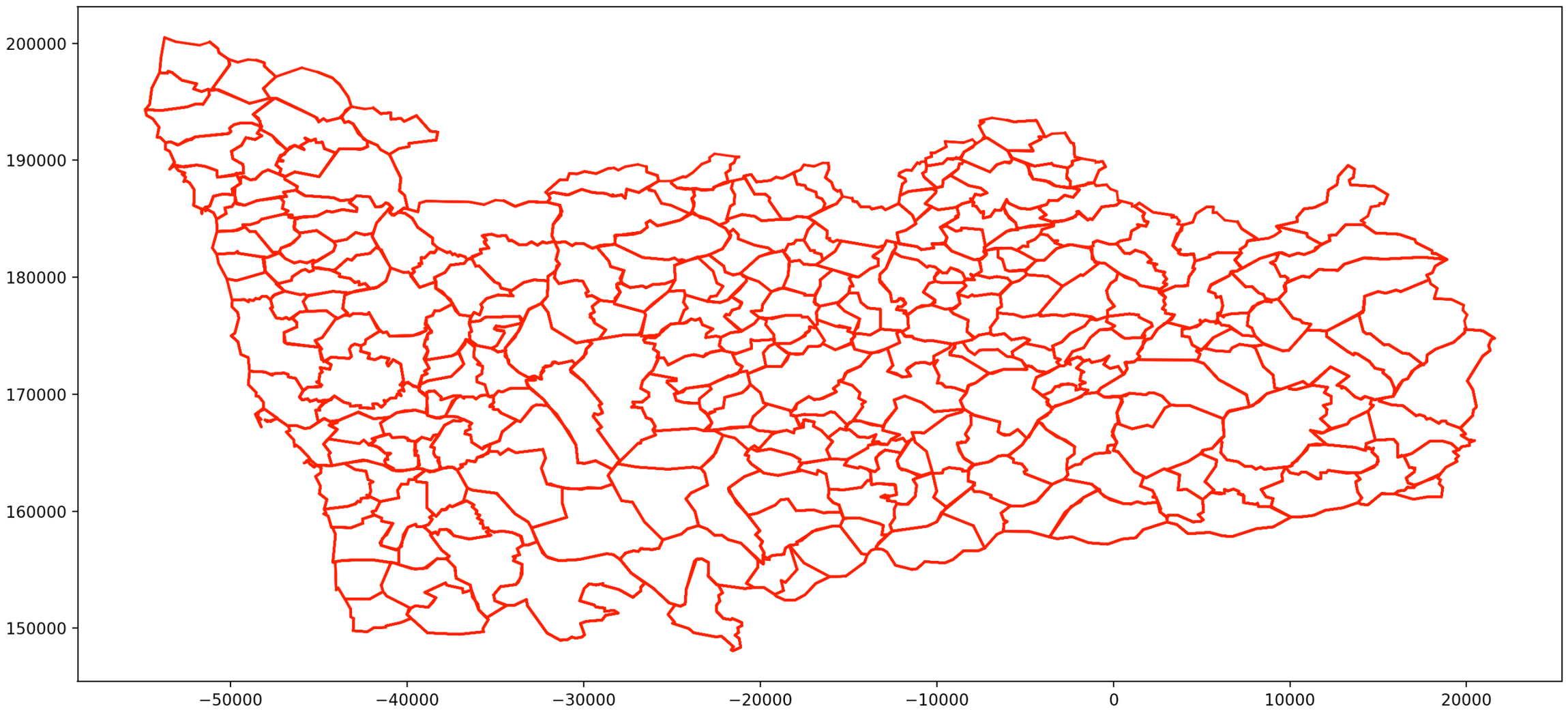
```
                                    Table "public.cont_aad_caop2018"
    Column     |          Type          | Collation | Nullable |                      Default
---------------+------------------------+-----------+----------+-------------------------------------------------
 gid           | integer                |           | not null | nextval('cont_aad_caop2018_gid_seq'::regclass)
 dicofre       | character varying(254) |           |          |
 freguesia     | character varying(254) |           |          |
 concelho      | character varying(254) |           |          |
 distrito      | character varying(254) |           |          |
 taa           | character varying(254) |           |          |
 area_ea_ha    | double precision       |           |          |
 area_t_ha     | double precision       |           |          |
 des_simpli    | character varying(254) |           |          |
 proj_boundary | geometry(Polygon,3763) |           |          |
Indexes:
    "cont_aad_caop2018_pkey" PRIMARY KEY, btree (gid)
    "cont_proj_boundary_idx" gist (proj_boundary)
```

Exemplo: polígonos de freguesias

```
import matplotlib.pyplot as plt
import numpy as np
import psycopg2

def polygon_to_points(polygon_string):
    xs, ys = [],[]
    points = polygon_string[9:-2].split(',')
    for point in points:
        (x,y) = point.split()
        xs.append(float(x))
        ys.append(float(y))
    return xs,ys
```

```
scale=1/30000
conn = psycopg2.connect("dbname=michelferreira user=michelferreira")
cursor_psql = conn.cursor()

# Calculate figure size
sql ="select st_astext(st_envelope(st_collect(st_simplify(proj_boundary,100,FALSE)))) from cont_aad_caop2018 where concelho='PORTO'"
cursor_psql.execute(sql)
results = cursor_psql.fetchall()
row = results[0]
polygon_string = row[0]
xs,ys = polygon_to_points(polygon_string)
width_in_inches = ((max(xs)-min(xs))/0.0254)*1.1
height_in_inches = ((max(ys)-min(ys))/0.0254)*1.1
fig = plt.figure(figsize=(width_in_inches*scale,height_in_inches*scale))


sql = "select st_astext(st_simplify(proj_boundary,100,False)) from cont_aad_caop2018 where distrito in ('PORTO')"
cursor_psql.execute(sql)
results = cursor_psql.fetchall()
for row in results:
    polygon_string = row[0]
    xs, ys = polygon_to_points(polygon_string)
    plt.plot(xs,ys,color='red')
plt.show()
```

Freguesias (distrito Porto)



Exercícios

1. Calcular por freguesia do concelho do Porto, o número de praças de táxi que se localizam em cada uma;

2. Mostrar um mapa com todos os serviços originados dentro do concelho do Porto;

3. Mostrar um mapa com todos os serviços finalizados no distrito do Porto;

4. Considerando um raio de 50 metros à volta de cada praça de táxi, ordene as diferentes praças de táxi descendentemente no número de serviços que emerge em cada uma;

5. Criar uma função que recebe como argumento o identificador de um táxi e mostra um mapa com a seqüência dos trajectos efectuados, com cores diferentes para cada estado (FREE - green, BUSY - red, PICKUP - yellow, PAUSE - grey);

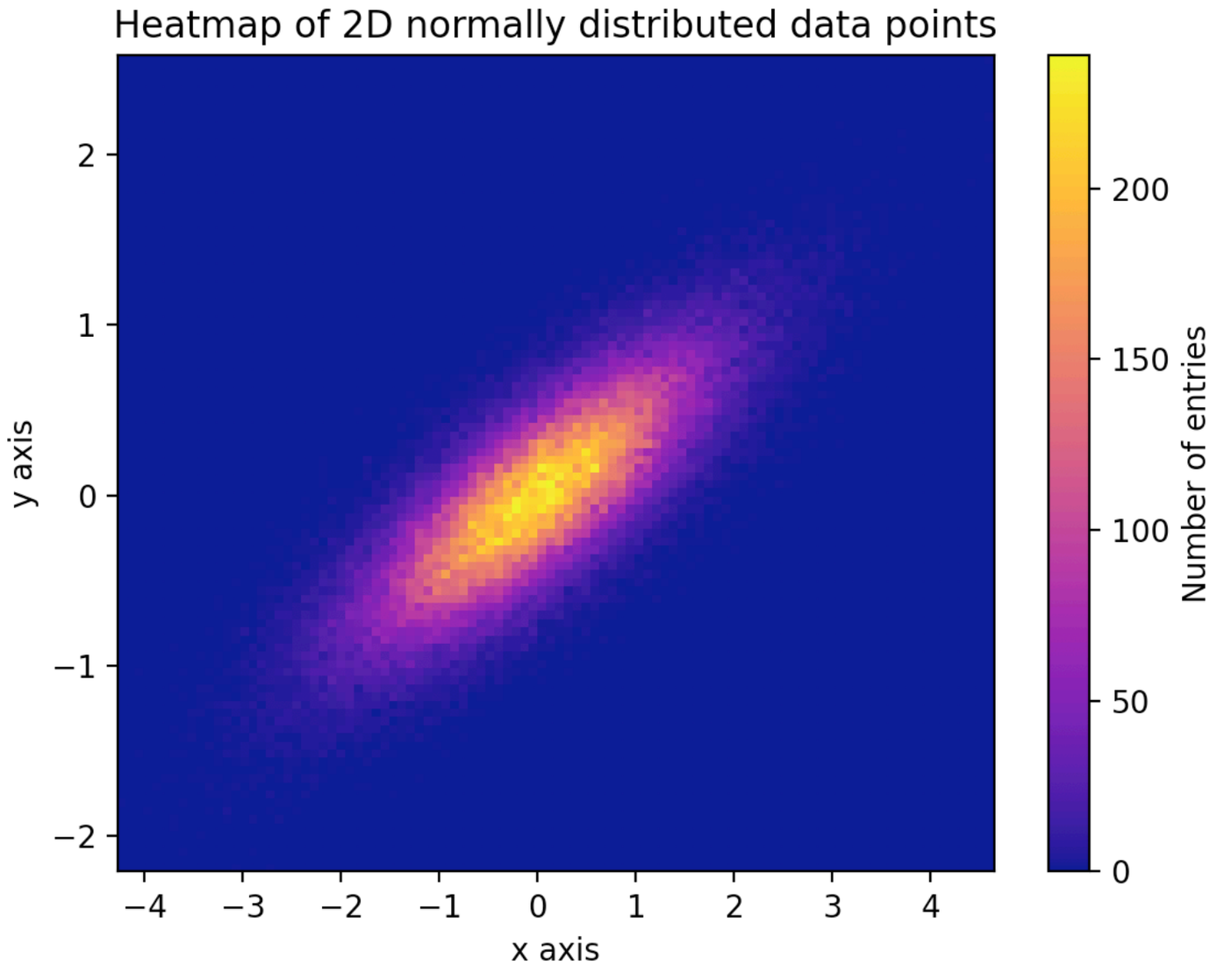
6. Calcular uma matriz Origem/Destino entre freguesia do Porto, com o número de viagens entre cada uma;

7. Criar uma função que representa o gráfico de velocidade para um determinado trajecto.

8. Para os serviços que se iniciam no concelho do Porto, calcular a percentagem dos que terminam no próprio concelho, e as percentagens de outros concelhos onde também terminam;

9. Criar um mapa com todos os trajectos que intersectam uma freguesia do concelho do Porto, mostrando a parte dentro do concelho do Porto, filtrando a sequência de pontos nas Linestrings, de forma a eliminar velocidade irrealistas.

Análise e visualização com Heatmaps



Exemplo heatmap

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Define numbers of generated data points and bins per axis.
N_numbers = 100000
N_bins = 100
# set random seed
np.random.seed(0)
# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
        mean=[0.0, 0.0],      # mean
        cov=[[1.0, 0.4],
             [0.4, 0.25]],    # covariance matrix
        size=N_numbers
        ).T                   # transpose to get columns
# Construct 2D histogram from data using the 'plasma' colormap
print(len(x),len(y))
print(x)
print(y)
plt.hist2d(x, y, bins=N_bins, normed=False, cmap='plasma')
# Plot a colorbar with label.
cb = plt.colorbar()
cb.set_label('Number of entries')
# Add title and labels to plot.
plt.title('Heatmap of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')
# Show the plot.
plt.show()
```

Heatmap para serviços iniciados no Porto