

# Cities Fingerprints

Andres PDLR

**"" A city's physical structures, once established, may remain in place for more than 150 years ""**

**What**

# JOURNAL OF THE ROYAL SOCIETY INTERFACE

Home Content Information for About us Sign up Submit



## • A typology of street patterns

Rémi Louf, Marc Barthelemy

Published 8 October 2014. DOI: 10.1098/rsif.2014.0924

Article Figures & Data Info & Metrics eLetters

### Abstract

We propose a quantitative method to classify cities according to their street pattern. We use the conditional probability distribution of shape factor of blocks with a given area and define what could constitute the ‘fingerprint’ of a city. Using a simple hierarchical clustering method, these fingerprints can then serve as a basis for a typology of cities. We apply this method to a set of 131 cities in the world, and at an intermediate level of the dendrogram, we observe four large families of cities characterized by different abundances of blocks of a certain area and shape. At a lower level of the classification, we find that most European cities and American cities in our sample fall in their own sub-category, highlighting quantitatively the differences between the typical layouts of cities in both regions. We also show with the example of New York and its different boroughs, that the fingerprint of a city can be seen as the sum of the ones characterizing the different neighbourhoods inside a city. This method provides a quantitative comparison of urban street patterns, which could be helpful for a better understanding of the causes and mechanisms behind their distinct shapes.





## A typology of street patterns

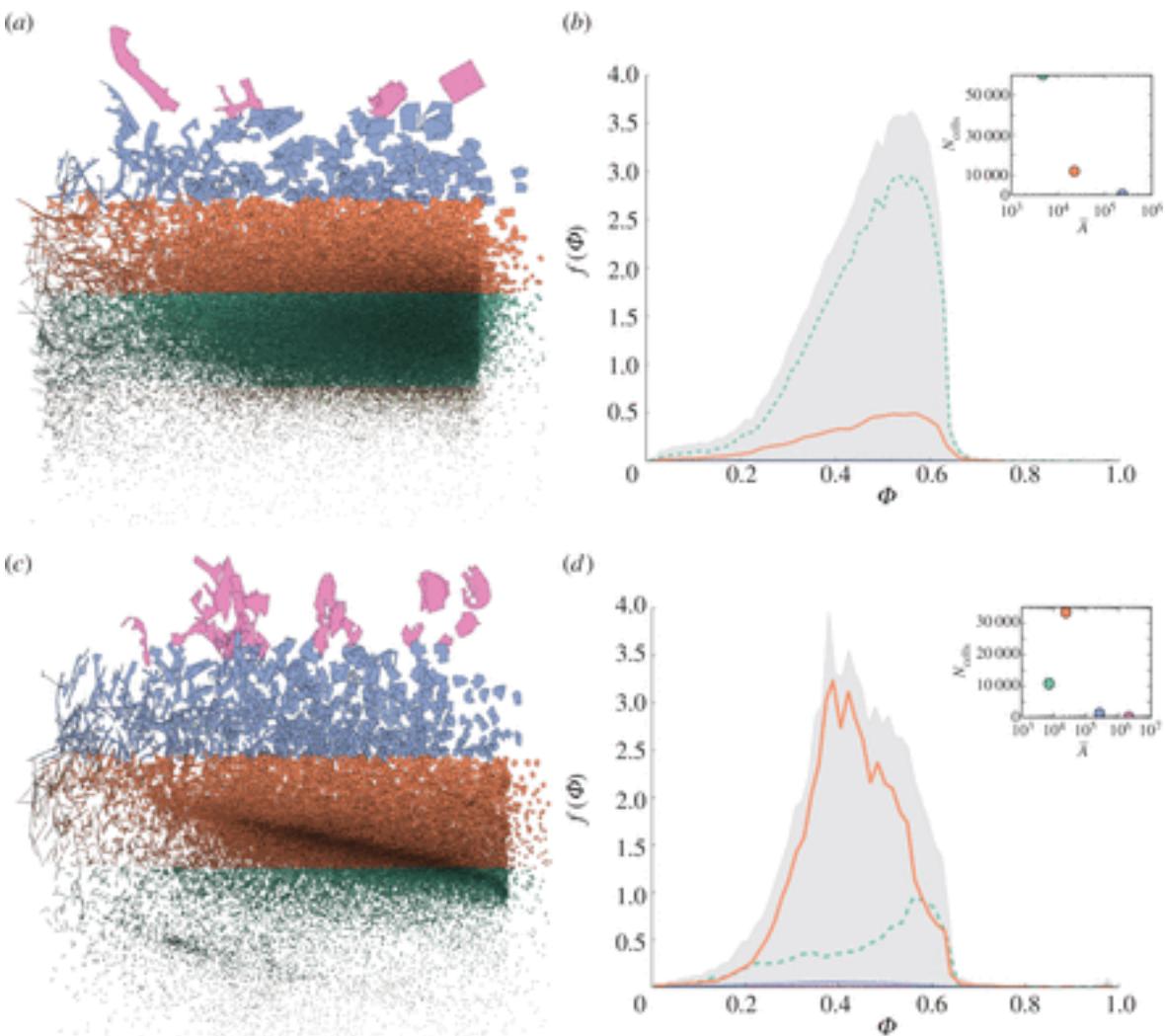
Rémi Louf, Marc Barthelemy

Published 8 October 2014. DOI: 10.1098/rsif.2014.0924

[Article](#) [Figures & Data](#) [Info & Metrics](#) [eLetters](#)

### Abstract

We propose a quantitative method to classify cities according to their street pattern. We use the conditional probability distribution of shape factor of blocks with a given area and define what could constitute the ‘fingerprint’ of a city. Using a simple hierarchical clustering method, these fingerprints can then serve as a basis for a typology of cities. We apply this method to a set of 131 cities in the world, and at an intermediate level of the dendrogram, we observe four large families of cities characterized by different abundances of blocks of a certain area and shape. At a lower level of the classification, we find that most European cities and American cities in our sample fall in their own sub-category, highlighting quantitatively the differences between the typical layouts of cities in both regions. We also show with the example of New York and its different boroughs, that the fingerprint of a city can be seen as the sum of the ones characterizing the different neighbourhoods inside a city. This method provides a quantitative comparison of urban street patterns, which could be helpful for a better understanding of the causes and mechanisms behind their distinct shapes.



*Why*

- Science of cities
- Urban economics
- ~~Rely~~ in data provided from governments
  - Street patterns
  - Satellite slums detection

# How



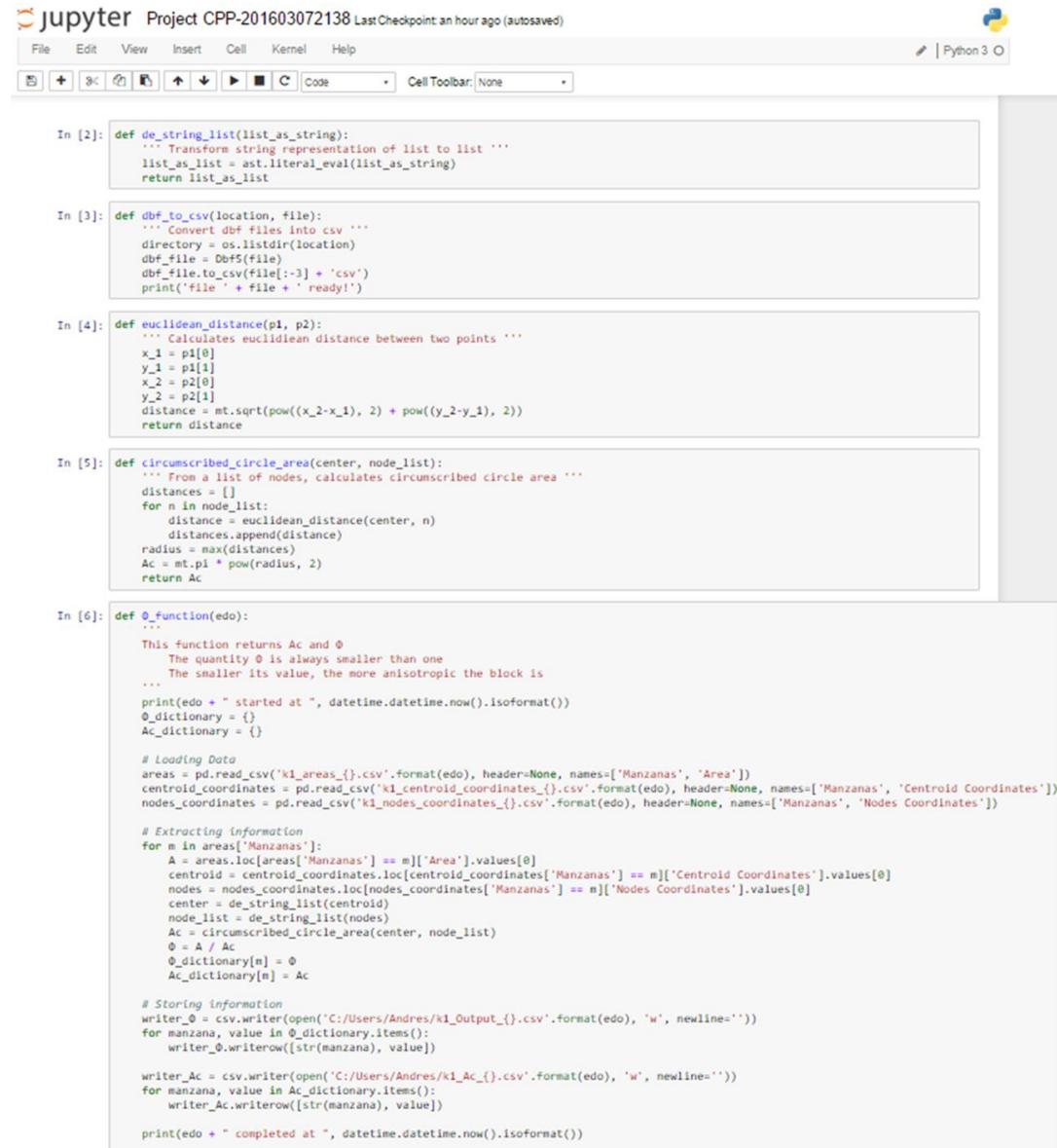
Python Console - QGis

## Two modules:

### GIS Analysis

```
86 def extract_areas(edo):
87     ''' Extracts the area of a polygon '''
88     areas = {}
89     path = 'C:/Users/Andres/geo_mex_2015/{}/{}.format(edo, edo) + "_manzanas_urbanas.shp"
90     manzanas_urbanas_layer = iface.addVectorLayer(path, edo + "_manzanas_urbanas", "ogr")
91     manzanas_urbanas_features = manzanas_urbanas_layer.getFeatures()
92     for feature in manzanas_urbanas_features:
93         manzana_ID = feature['CVEGEO']
94         geometry = feature.geometry()
95         m_area = geometry.area()
96         if manzana_ID in areas.keys():
97             areas[manzana_ID].append(m_area)
98         else:
99             areas[manzana_ID] = m_area
100    writer = csv.writer(open('C:/Users/Andres/k1_areas_{}.csv'.format(edo), 'wb'))
101    for key, value in areas.items():
102        writer.writerow([str(key), value])
103
104
105
106
107
108 def extract_centroid_coordinates(edo):
109     ''' Extracts centroids coordinates of a polygon layer '''
110     centroids_coordinates = {}
111     path = 'C:/Users/Andres/geo_mex_2015/{}/{}.format(edo, edo) + "_manzanas_urbanas_centroids.shp"
112     centroids_layer = iface.addVectorLayer(path, "{}_centroids_layer".format(edo), "ogr")
113     centroid_features = centroids_layer.getFeatures()
114     for feature in centroid_features:
115         manzana_ID = feature['CVEGEO']
116         geometry = feature.geometry()
117         x = geometry.asPoint().x()
118         y = geometry.asPoint().y()
119         if manzana_ID in centroids_coordinates.keys():
120             centroids_coordinates[manzana_ID].append((x, y))
121         else:
122             centroids_coordinates[manzana_ID] = (x, y)
123     writer = csv.writer(open('C:/Users/Andres/k1_centroid_coordinates_{}.csv'.format(edo), 'wb'))
124     for key, value in centroids_coordinates.items():
125         writer.writerow([str(key), value])
126
```

### Data Analysis



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, View, Insert, Cell, Kernel, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Run, and Cell.
- Cell Toolbar:** Set to "None".
- Code Cells:** There are six code cells labeled In [2] through In [6].
- In [2]:**

```
def de_string_list(list_as_string):
    """ Transform string representation of list to list """
    list_as_list = ast.literal_eval(list_as_string)
    return list_as_list
```
- In [3]:**

```
def dbf_to_csv(location, file):
    """ Convert dbf files into csv """
    directory = os.listdir(location)
    dbf_file = Dbf5(file)
    dbf_file.to_csv(file[:-3] + 'csv')
    print('file ' + file + ' ready!')
```
- In [4]:**

```
def euclidean_distance(p1, p2):
    """ Calculates euclidean distance between two points """
    x_1 = p1[0]
    y_1 = p1[1]
    x_2 = p2[0]
    y_2 = p2[1]
    distance = mt.sqrt(pow((x_2-x_1), 2) + pow((y_2-y_1), 2))
    return distance
```
- In [5]:**

```
def circumscribed_circle_area(center, node_list):
    """ From a list of nodes, calculates circumscribed circle area """
    distances = []
    for n in node_list:
        distance = euclidean_distance(center, n)
        distances.append(distance)
    radius = max(distances)
    Ac = mt.pi * pow(radius, 2)
    return Ac
```
- In [6]:**

```
def _function(edo):
    """
    This function returns Ac and Ø
    The quantity Ø is always smaller than one
    The smaller its value, the more anisotropic the block is
    ...
    print(edo + " started at ", datetime.datetime.now().isoformat())
    Ø_dictionary = {}
    Ac_dictionary = {}

    # Loading Data
    areas = pd.read_csv('k1_areas_{}.csv'.format(edo), header=None, names=['Manzanas', 'Area'])
    centroid_coordinates = pd.read_csv('k1_centroid_coordinates_{}.csv'.format(edo), header=None, names=['Manzanas', 'Centroid Coordinates'])
    nodes_coordinates = pd.read_csv('k1_nodes_coordinates_{}.csv'.format(edo), header=None, names=['Manzanas', 'Nodes Coordinates'])

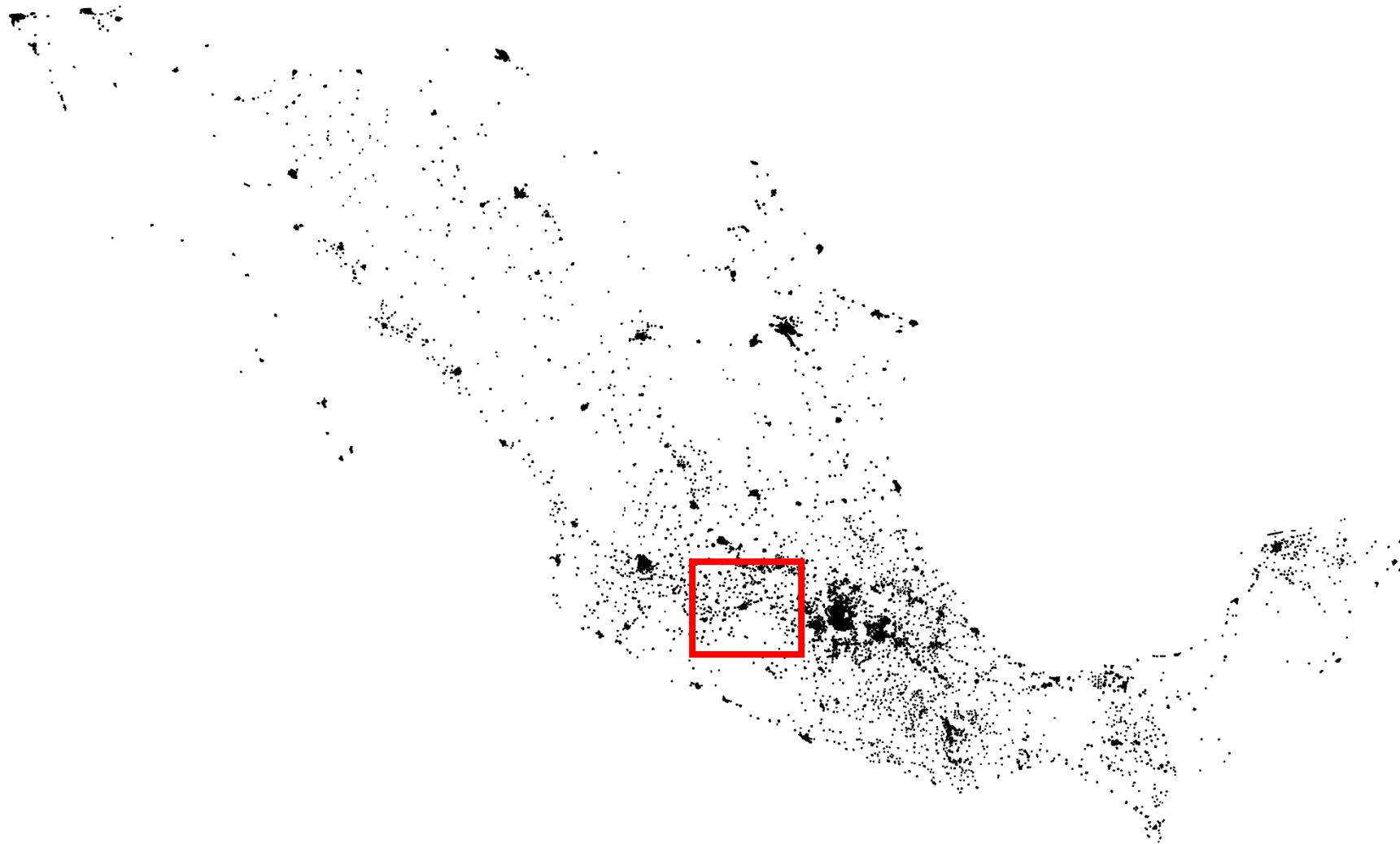
    # Extracting Information
    for m in areas['Manzanas']:
        A = areas.loc[areas['Manzanas'] == m]['Area'].values[0]
        centroid = centroid_coordinates.loc[centroid_coordinates['Manzanas'] == m]['Centroid Coordinates'].values[0]
        nodes = nodes_coordinates.loc[nodes_coordinates['Manzanas'] == m]['Nodes Coordinates'].values[0]
        center = de_string_list(centroid)
        node_list = de_string_list(nodes)
        Ac = circumscribed_circle_area(center, node_list)
        Ø = A / Ac
        Ø_dictionary[m] = Ø
        Ac_dictionary[m] = Ac

    # Storing Information
    writer_Ø = csv.writer(open('C:/Users/Andres/k1_Output_{}.csv'.format(edo), 'w', newline=''))
    for manzana, value in Ø_dictionary.items():
        writer_Ø.writerow([str(manzana), value])

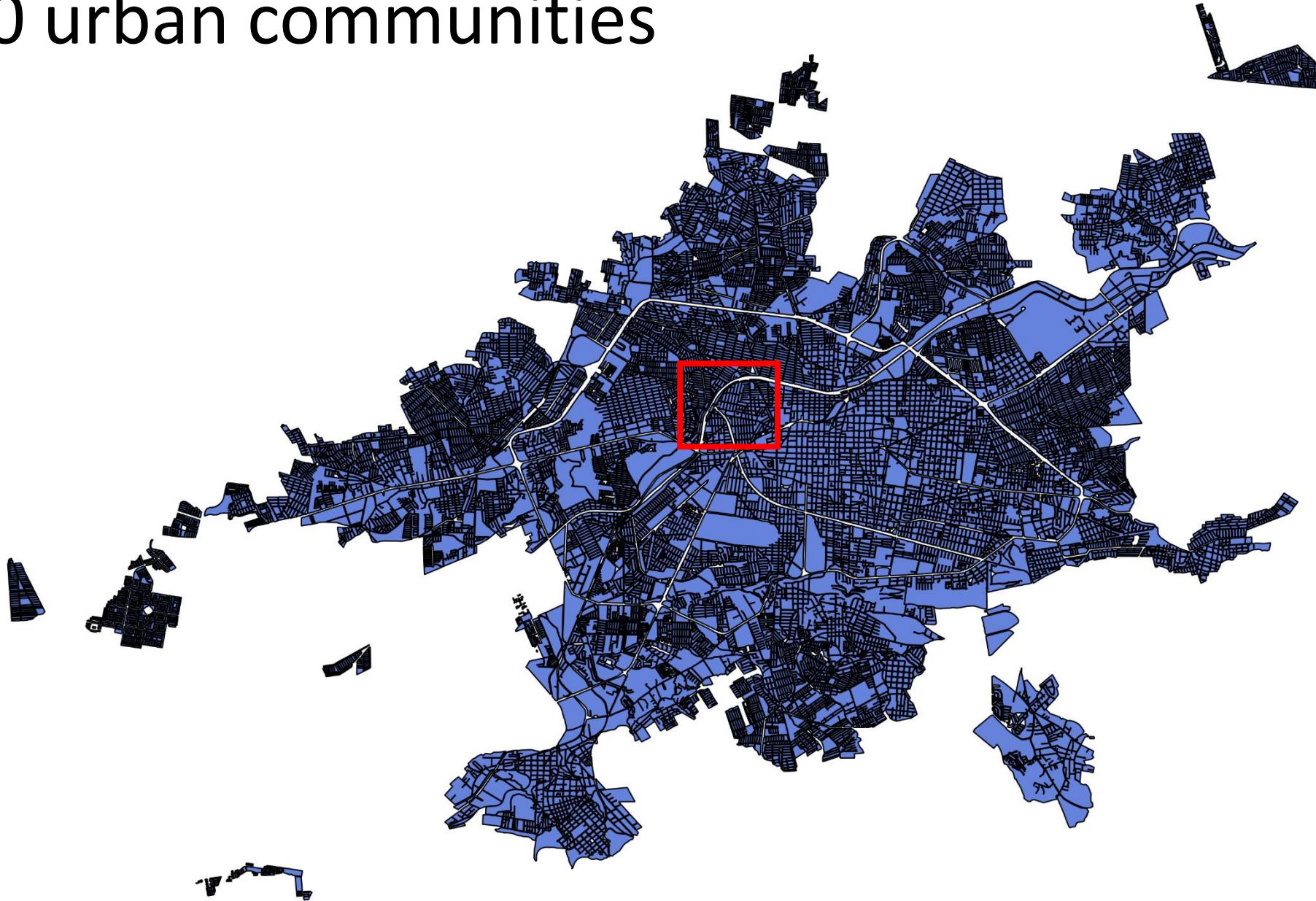
    writer_Ac = csv.writer(open('C:/Users/Andres/k1_Ac_{}.csv'.format(edo), 'w', newline=''))
    for manzana, value in Ac_dictionary.items():
        writer_Ac.writerow([str(manzana), value])

    print(edo + " completed at ", datetime.datetime.now().isoformat())
```

# 1,456,596 urban blocks



+ 4,500 urban communities





# Output

$A$  = Area of the block

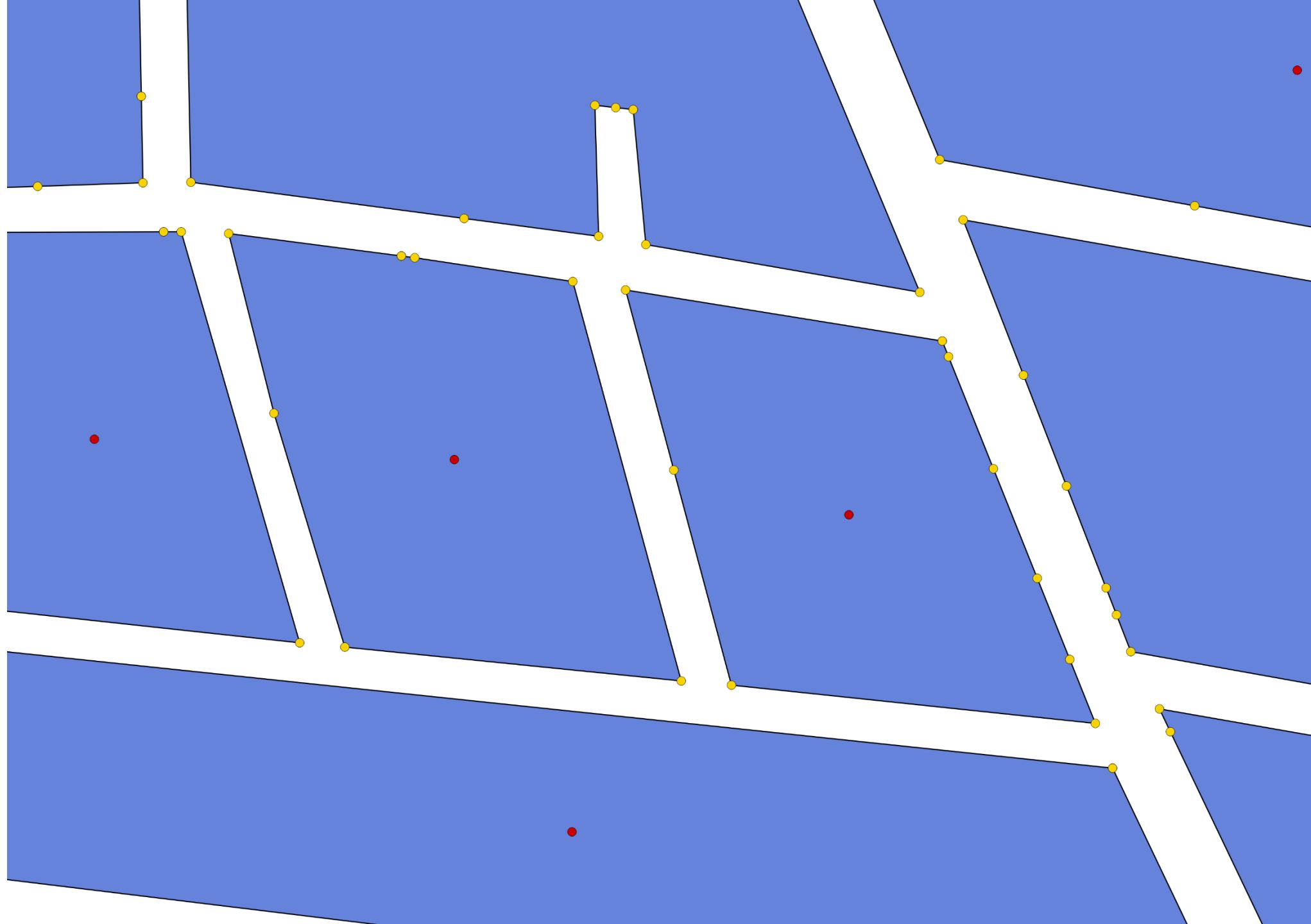
$AC$  = Area of the circumscribed circle  $C$

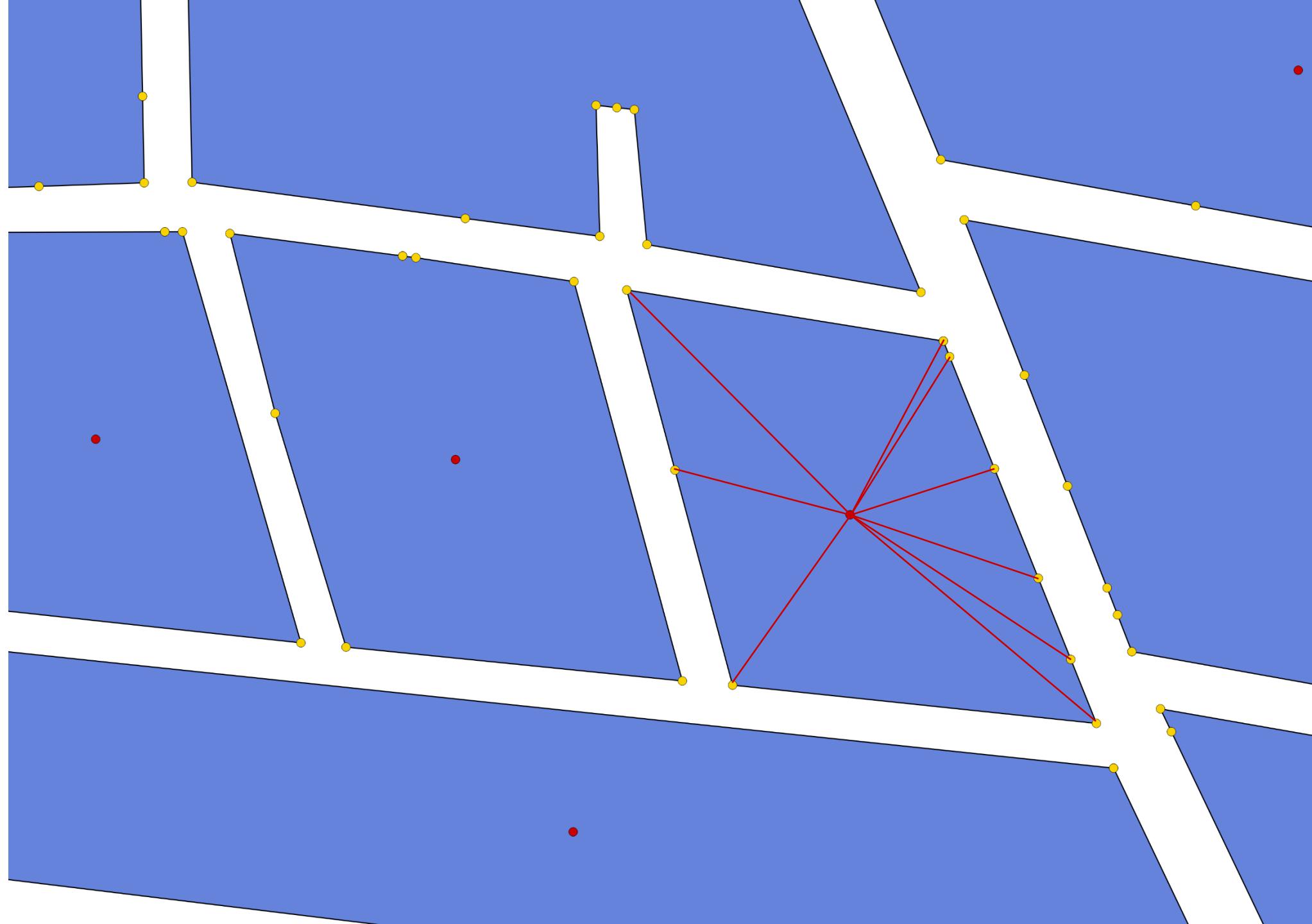
$\Phi$  =  $A/AC$

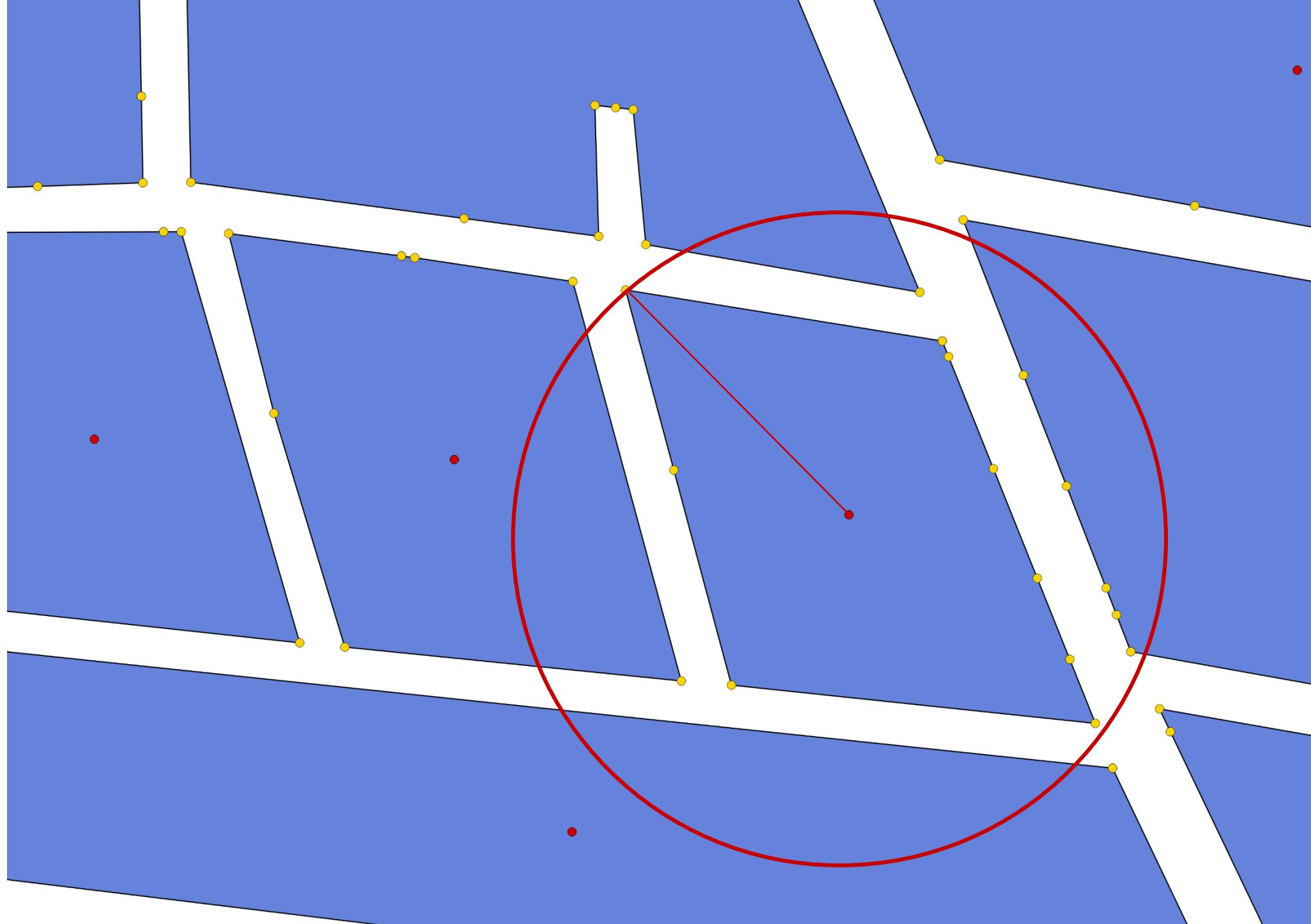
$P(\Phi)$  = Distribution

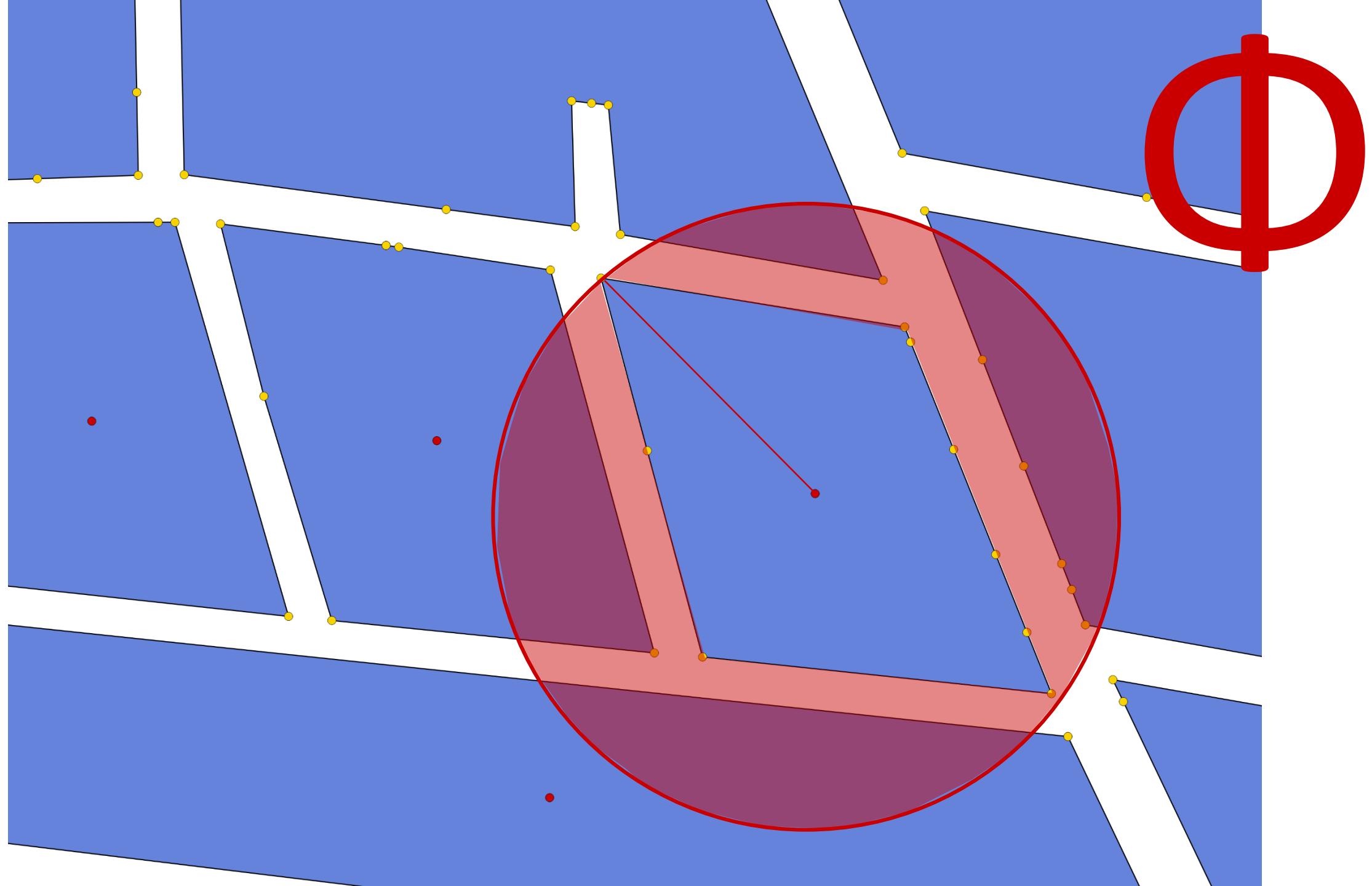












```
In [62]: dataframe_all.head()
```

Out[62]:

	Manzanas	Area	$\Phi$	Ac
0	0100100014325009	1.514624e-06	0.159168	9.515873e-06
1	0100100014325008	1.833420e-07	0.224965	8.149784e-07
2	0100100014325005	8.994317e-07	0.187952	4.785435e-06
3	0100100014325004	8.021103e-07	0.196058	4.091192e-06
4	0100100014325007	6.534422e-07	0.303527	2.152833e-06

```
In [48]: # Number of blocks in Mexico  
len(dataframe_all)
```

Out[48]: 1456596

Municipio



Hierarchical ID

State

Manzana (*block*)

0.0000010

0.0000008

0.0000006

0.0000004

0.0000002

0.0000000

Area

0.0

0.2

0.4

0.6

0.8

1.0

$\Phi$

### Distribution of Area vs $\Phi$

Aguascalientes (ags)



0.0000010

0.0000008

0.0000006

0.0000004

0.0000002

0.0000000

Area

0.0

0.2

0.4

0.6

0.8

1.0

$\Phi$

Distribution of Area vs  $\Phi$

Mexico City (df)



0.0000010

0.0000008

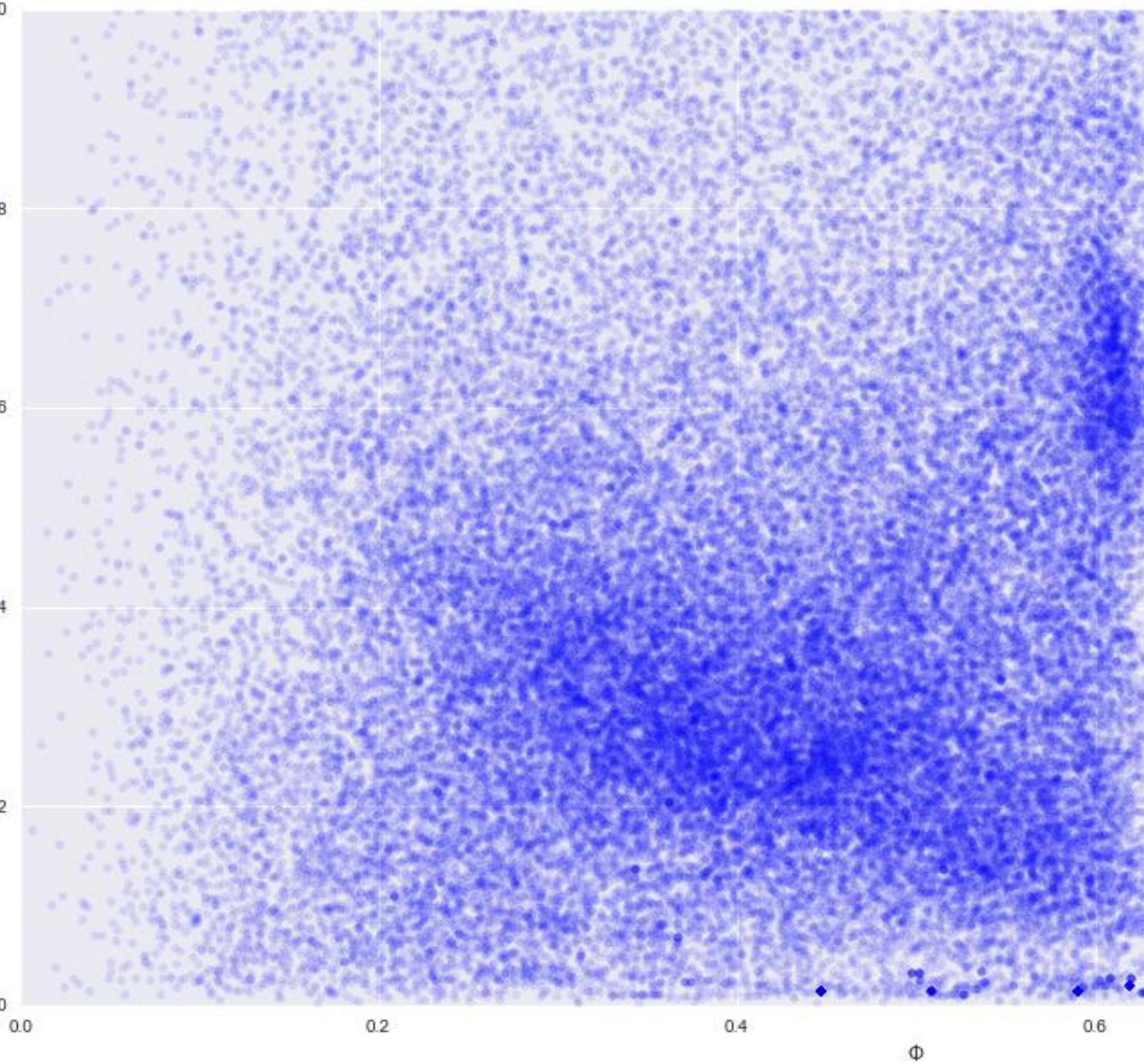
0.0000006

0.0000004

0.0000002

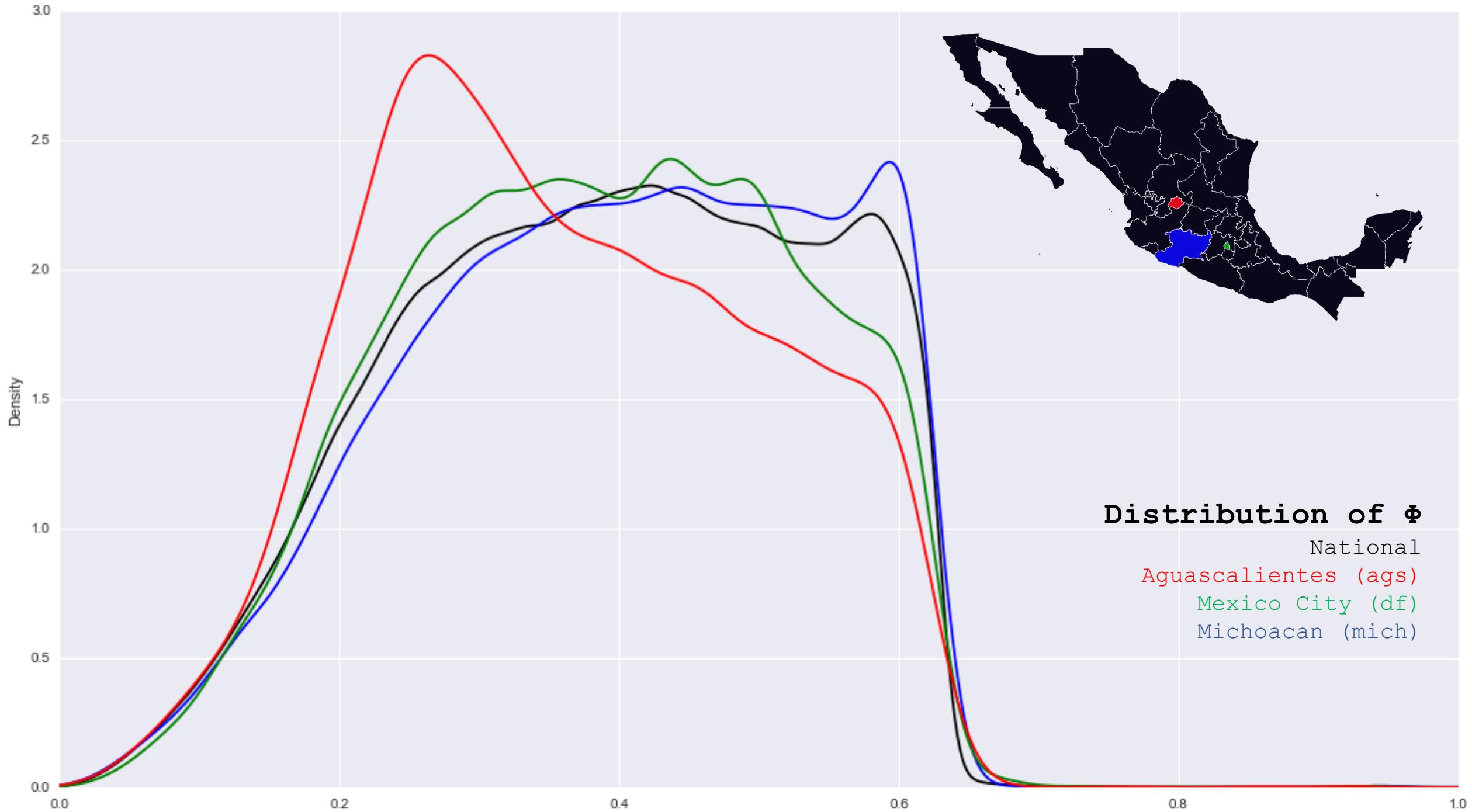
0.0000000

Area



**Distribution of Area vs  $\Phi$**

Michoacan (mich)

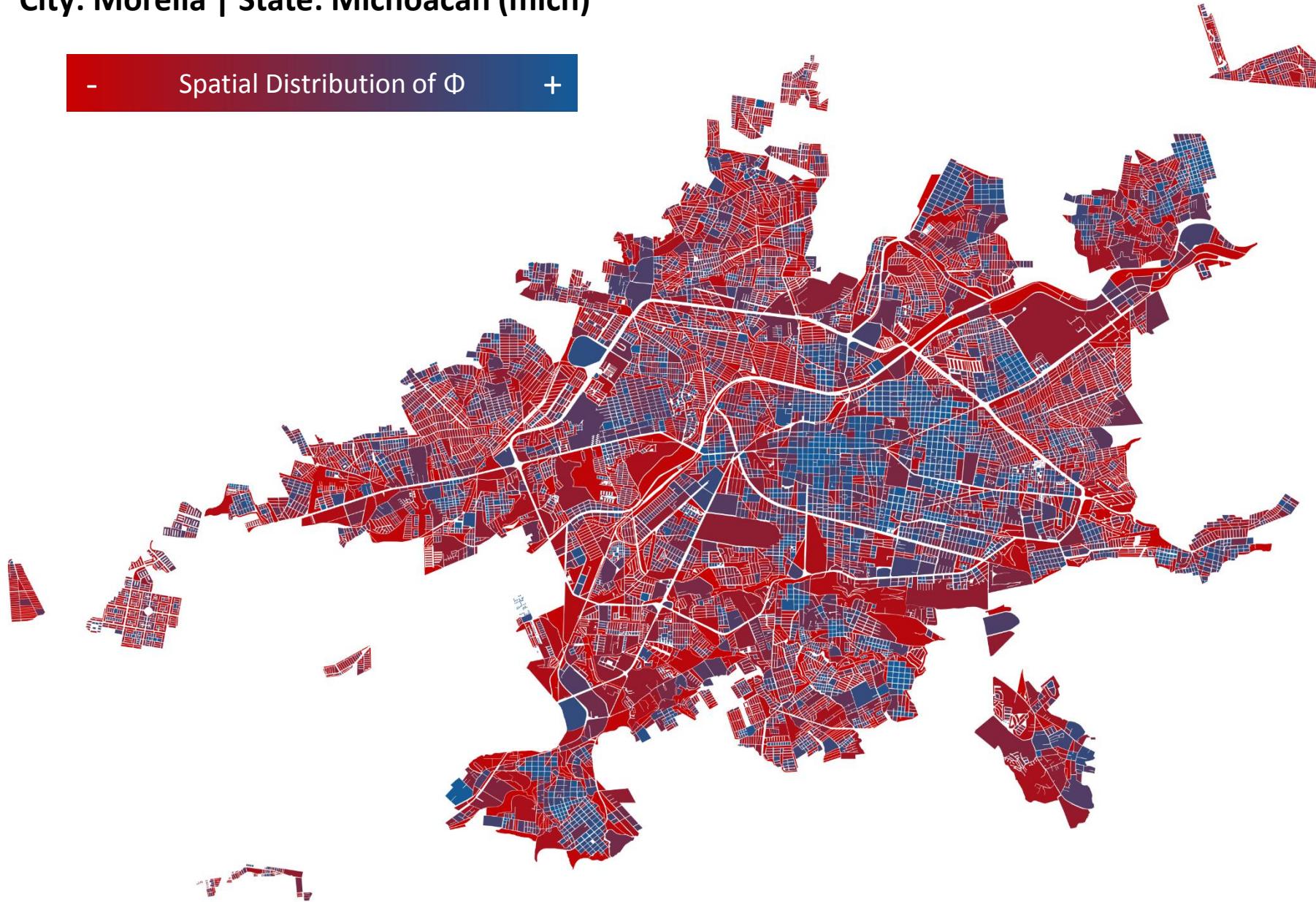


**City: Morelia | State: Michoacan (mich)**

-

Spatial Distribution of  $\Phi$

+

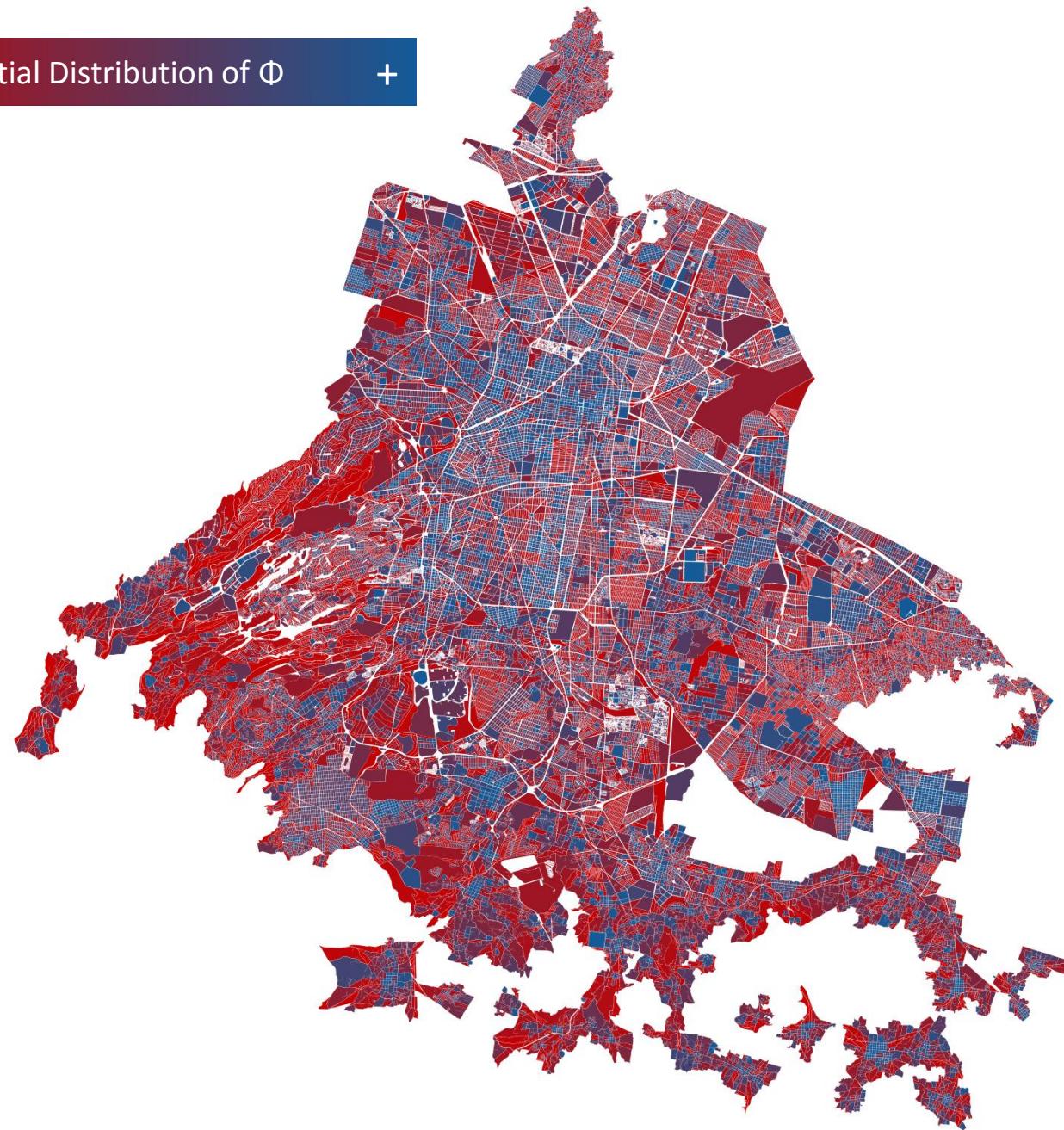


**City: Mexico City | State: Distrito Federal (df)**

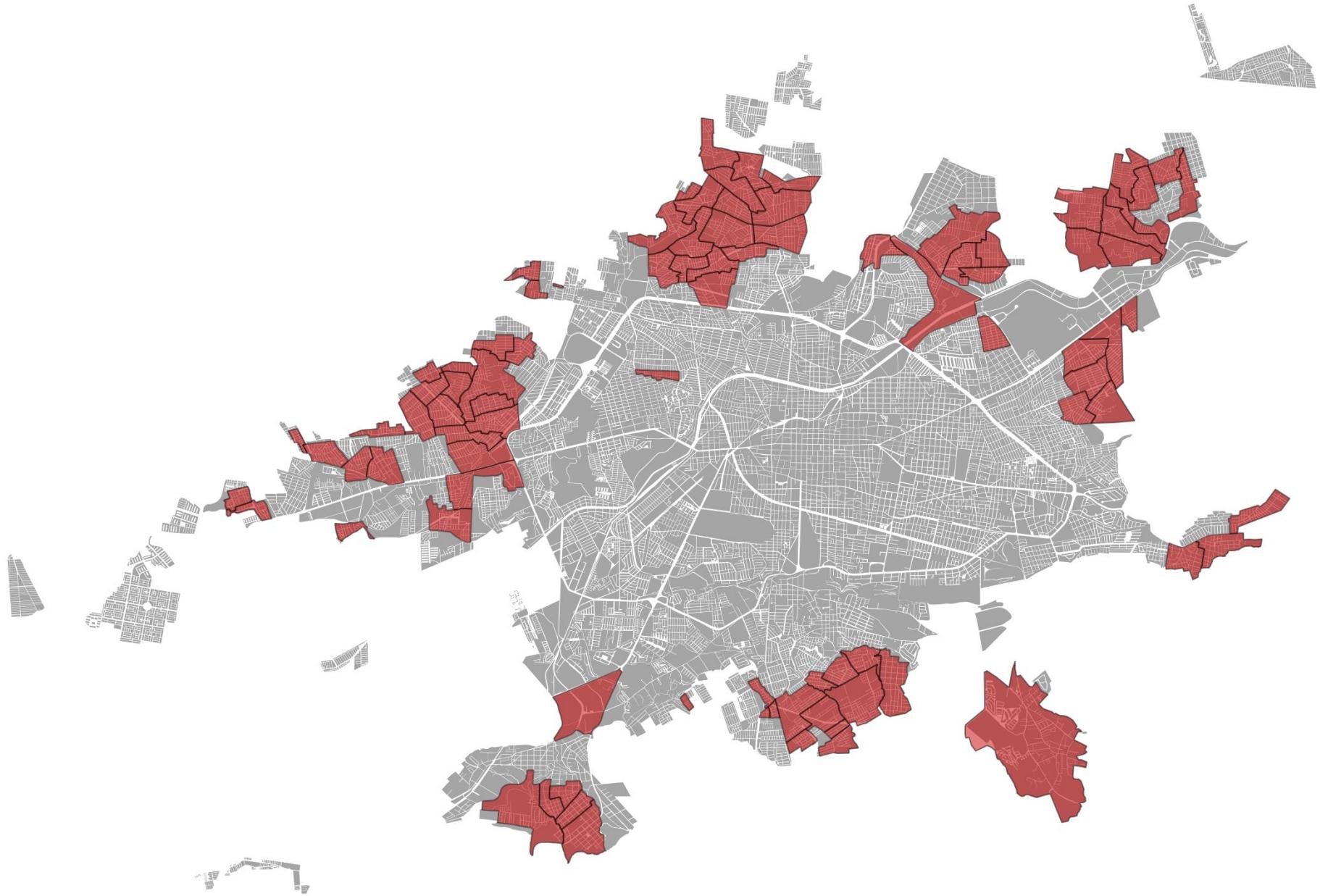
-

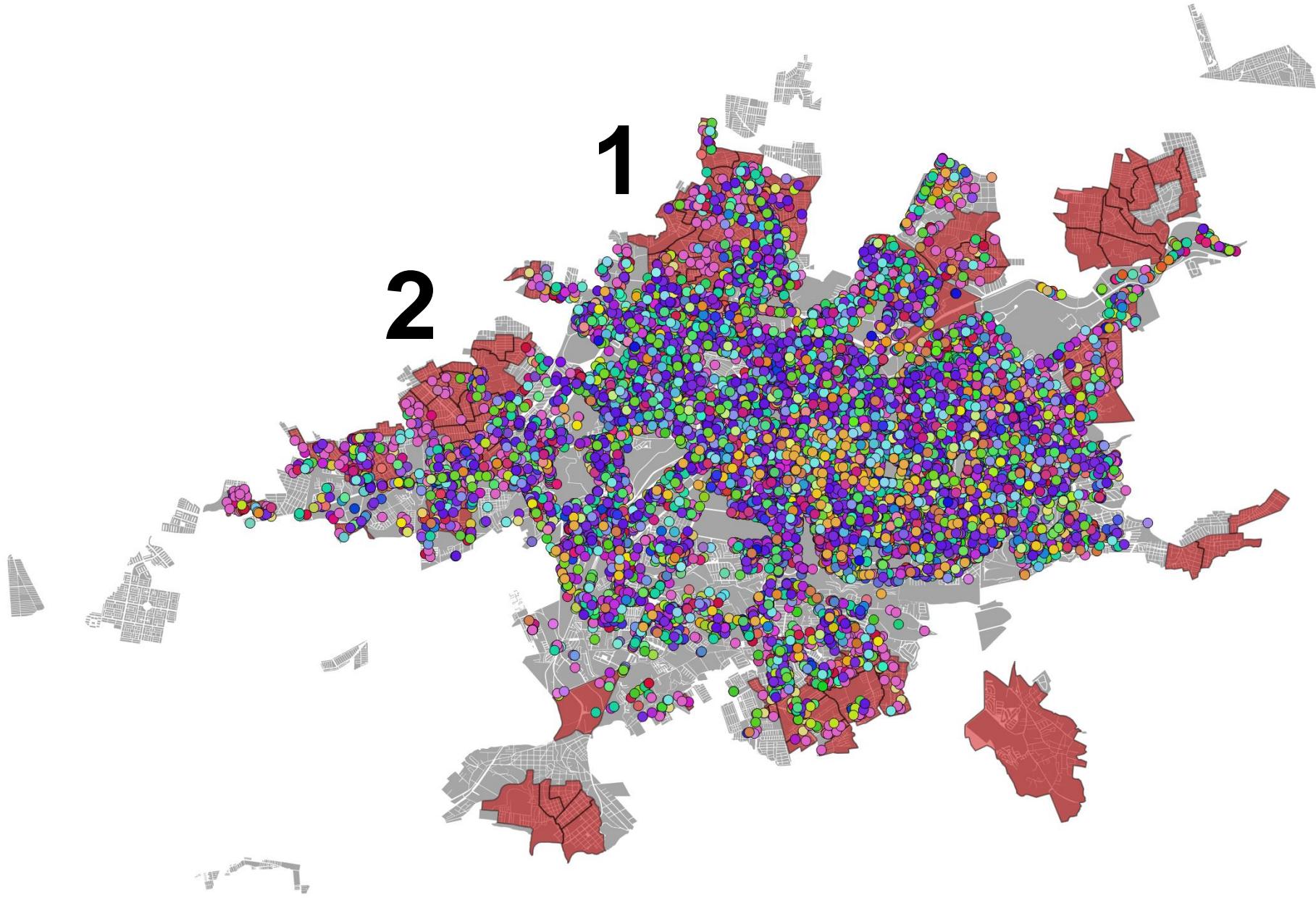
Spatial Distribution of  $\Phi$

+

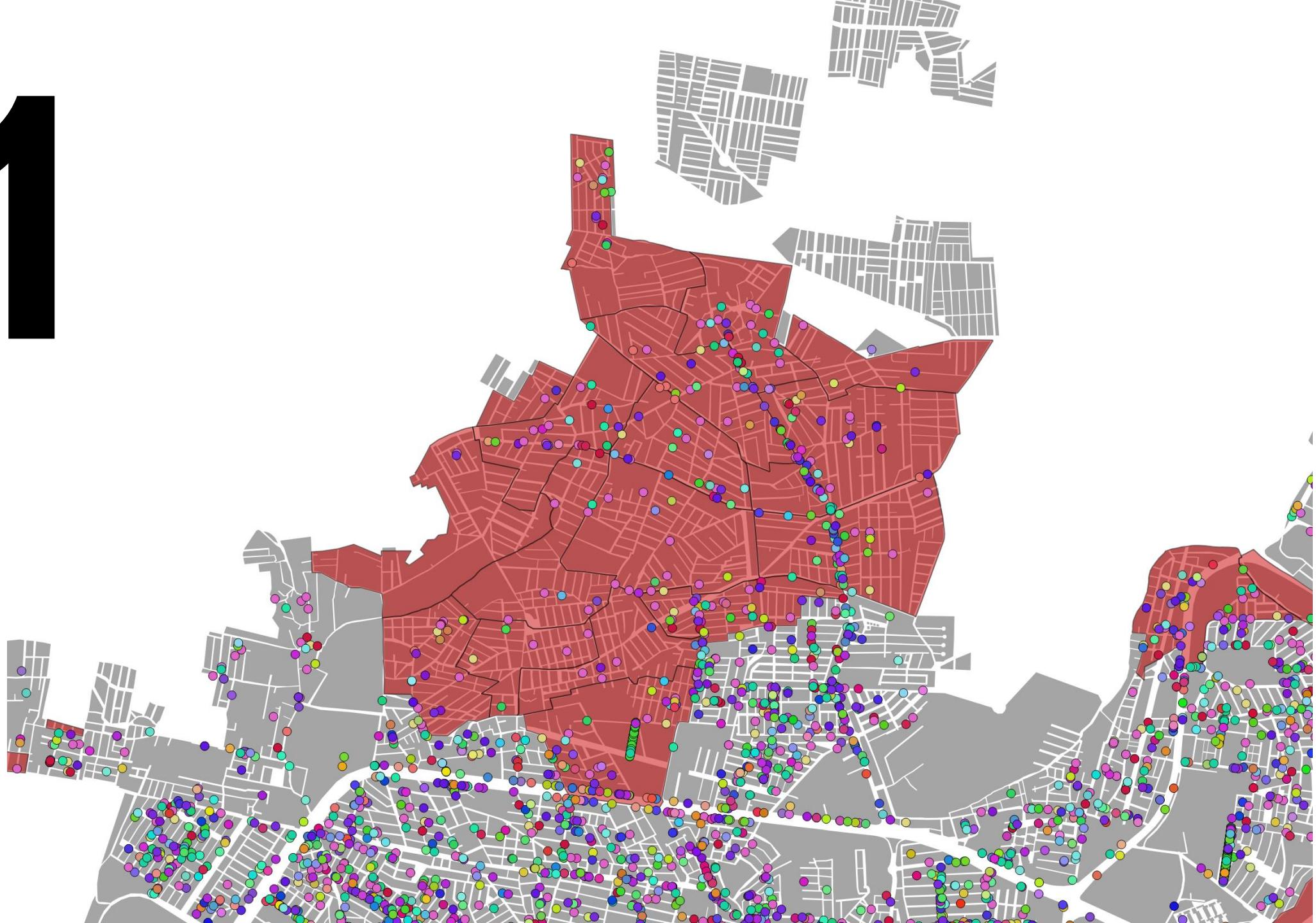


# Next





# 1



# 2

