BTS | Barcelona Technology School

Advanced Data Analysis

Master in Big Data Solutions 2020-2021

Ankit Tewari
Course Instructor, Advanced Data Analysis (2020-21)
ankit.tewari@bts.tech

Contents

Today's topics at a glance

Spatial Weights

What do we mean by Spatial Similiarity?

Spatial weights are a key component in any cross-sectional analysis of spatial dependence. They are an essential element in the construction of spatial autocorrelation statistics, and provide the means to create spatially explicit variables, such as spatially lagged variables and spatially smoothed rates.

Formally, the weights express the neighbour structure between the observations as a **n x n** matrix **W** in which the elements $\mathbf{w_{i,j}}$ of the matrix are the spatial weights:

Spatial Weights

What do we mean by Spatial Similiarity?

Spatial weights are a key component in any cross-sectional analysis of spatial dependence. They are an essential element in the construction of spatial autocorrelation statistics, and provide the means to create spatially explicit variables, such as spatially lagged variables and spatially smoothed rates.

Formally, the weights express the neighbour structure between the observations as a **n x n** matrix **W** in which the elements $\mathbf{w_{i,j}}$ of the matrix are the spatial weights:

Spatial Autocorrelation

What do we mean by Spatial Similiarity?

Visual inspection of the map pattern for the prices allows us to search for spatial structure. If the spatial distribution of the prices was random, then we should not see any clustering of similar values on the map. However, our visual system is drawn to the darker clusters in the south west as well as the centre, and a concentration of the lighter hues (lower prices) in the north central and south east.

Our brains are very powerful pattern recognition machines. However, sometimes they can be too powerful and lead us to detect false positives, or patterns where there are no statistical patterns. This is a particular concern when dealing with visualization of irregular polygons of differing sizes and shapes.

Spatial Autocorrelation

What do we mean by Spatial Similiarity?

Visual inspection of the map pattern for the prices allows us to search for spatial structure. If the spatial distribution of the prices was random, then we should not see any clustering of similar values on the map. However, our visual system is drawn to the darker clusters in the south west as well as the centre, and a concentration of the lighter hues (lower prices) in the north central and south east.

The concept of spatial autocorrelation relates to the combination of two types of similarity:

1. Spatial similarity: We use spatial weights to formalize the notion of spatial similarity. There are many ways to define spatial weights and queen contiguity is one of such ways;
2. Attribute similarity: So the spatial weight between neighbourhoods i and j indicates if the two are neighbours (i.e., geographically similar). What we also need is a measure of attribute similarity to pair up with this concept of spatial similarity. The spatial lag is a derived variable that accomplishes this for us. For neighbourhood i-the spatial lag is defined as:

$$y_i = \sum_j w_{i,j} y_j$$

Although there are many different measures of spatial autocorrelation, they all combine these two types of similarity into a summary measure.

Spatial Similarity

What do we mean by Spatial Similarity?

The concept of spatial autocorrelation relates to the combination of two types of similarity:

1. Spatial similarity: We use spatial weights to formalize the notion of spatial similarity. There are many ways to define spatial weights and queen contiguity is one of such ways;

```
wq =  lp.Queen.from_dataframe(df)
wq.transform = 'r'
```

2. Attribute similarity: So the spatial weight between neighbourhoods i and j indicates if the two are neighbours (i.e., geographically similar). What we also need is a measure of attribute similarity to pair up with this concept of spatial similarity. The spatial lag is a derived variable that accomplishes this for us.

```
y = df['median_pri']
ylag = lp.lag_spatial(wq, y)
```

```
ylag
```

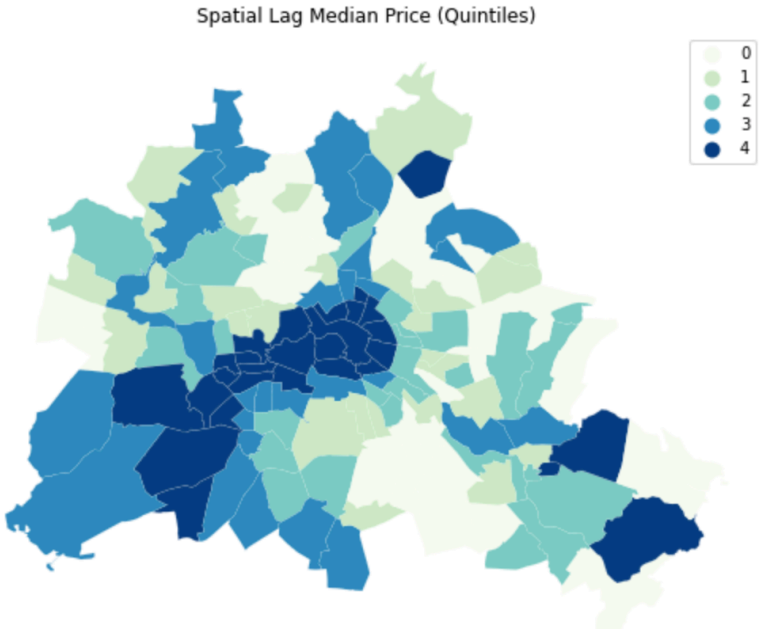
```
array([45.2       , 52.625     , 45.75      , 32.5       , 63.5       ,
       42.        , 45.625     , 44.14285714, 43.33333333, 38.75      ,
       41.5       , 50.8       , 36.6875    , 54.36363636, 54.375     ,
       38.92857143, 38.125     , 50.9       , 35.6875    , 59.66666667,
       46.875     , 46.92857143, 49.58333333, 47.25      , 53.25      ,
       40.57142857, 37.66666667, 37.14285714, 40.75      , 41.5       ,
       45.9       , 35.3       , 47.9375    , 47.33333333, 40.        ,
       44.        , 58.3       , 53.16666667, 41.1459854 , 43.75      ,
       51.625     , 52.3       , 50.5       , 46.91666667, 47.        ,
       38.125     , 35.33333333, 48.83333333, 46.6       , 43.125     ,
       39.95498783, 41.33333333, 42.        , 44.43248175, 55.66666667,
       46.2       , 47.33333333, 49.84124088, 47.93248175, 42.92857143,
       43.4       , 40.78571429, 37.42857143, 32.75      , 45.57142857,
       51.25      , 44.        , 33.33333333, 33.25      , 42.        ,
       40.        , 34.8       , 43.57142857, 41.75      , 43.85714286,
       39.14285714, 42.25      , 47.21428571, 46.        , 51.2       ,
```

Spatial Autocorrelation

Plotting Spatial Lags

```
from mapclassify import Quantiles
ylagq5 = Quantiles(ylag, k=5)
```

```
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=ylagq5.yb).plot(column='cl', categorical=True, \
        k=5, cmap='GnBu', linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.title("Spatial Lag Median Price (Quintiles)")

plt.show()
```

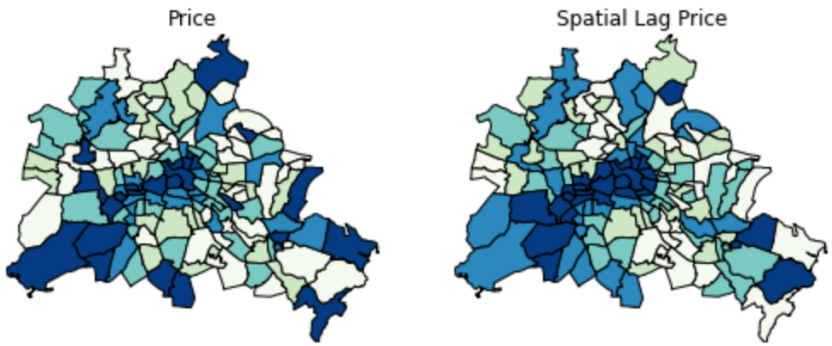


Spatial Autocorrelation

Price vs Spatial Lagged Price

However, we still have the challenge of visually associating the value of the prices in a neighbourhood with the value of the spatial lag of values for the focal unit. The latter is a weighted average of homicide rates in the focal county's neighbourhood.

```
df['lag_median_pri'] = ylag
f,ax = plt.subplots(1,2,figsize=(2.16*4,4))
df.plot(column='median_pri', ax=ax[0], edgecolor='k',
        scheme="quantiles",  k=5, cmap='GnBu')
ax[0].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[0].set_title("Price")
df.plot(column='lag_median_pri', ax=ax[1], edgecolor='k',
        scheme='quantiles', cmap='GnBu', k=5)
ax[1].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[1].set_title("Spatial Lag Price")
ax[0].axis('off')
ax[1].axis('off')
plt.show()
```



Spatial Autocorrelation

Global Spatial Correlation

We begin with a simple case where the variable under consideration is binary. This is useful to unpack the logic of spatial autocorrelation tests. So even though our attribute is a continuously valued one, we will convert it to a binary case to illustrate the key concepts:

```
y.median()
```

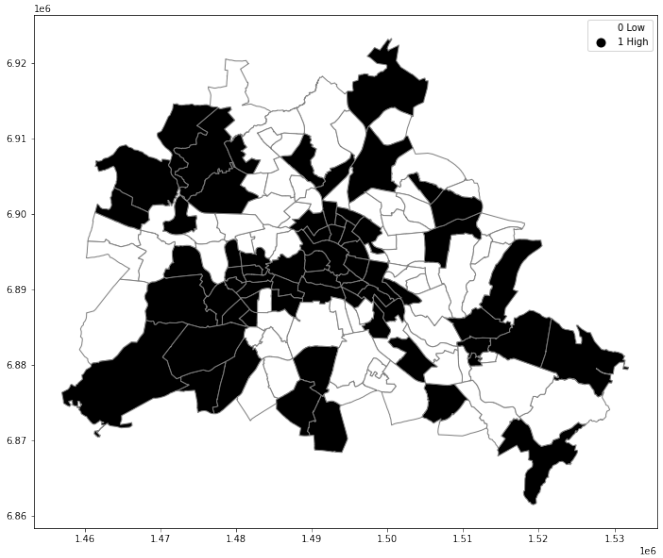
```
42.0
```

```
yb = y > y.median()
sum(yb)
```

```
68
```

We have 68 neighborhoods with list prices above the median and 70 below the median (recall the issue with ties).

```
yb = y > y.median()
labels = ["0 Low", "1 High"]
yb = [labels[i] for i in 1*yb]
df['yb'] = yb
```



Spatial Autocorrelation

Join Counts

One way to formalize a test for spatial autocorrelation in a binary attribute is to consider the so-called joins. A join exists for each neighbour pair of observations, and the joins are reflected in our binary spatial weights object **wq**.

Each unit can take on one of two values "Black" or "White", and so for a given pair of neighbouring locations there are three different types of joins that can arise:
Black Black (BB)
White White (WW)
Black White (or White Black) (BW)

Given that we have 68 Black polygons on our map, what is the number of Black Black (BB) joins we could expect if the process were such that the Black polygons were randomly assigned on the map? This is the logic of join count statistics.

We can use the esda package from PySAL to carry out join count analysis:

Spatial Autocorrelation

Join Counts

One way to formalize a test for spatial autocorrelation in a binary attribute is to consider the so-called joins. A join exists for each neighbour pair of observations, and the joins are reflected in our binary spatial weights object **wq**.

Each unit can take on one of two values "Black" or "White", and so for a given pair of neighbouring locations there are three different types of joins that can arise:
Black Black (BB)
White White (WW)
Black White (or White Black) (BW)

Given that we have 68 Black polygons on our map, what is the number of Black Black (BB) joins we could expect if the process were such that the Black polygons were randomly assigned on the map? This is the logic of join count statistics.

```
import esda
yb = 1 * (y > y.median()) # convert back to binary
wq =  lp.Queen.from_dataframe(df)
wq.transform = 'b'
np.random.seed(12345)
jc = esda.join_counts.Join_Counts(yb, wq)
```

The resulting object stores the observed counts for the different types of joins:

```
jc.bb
```

```
121.0
```

```
jc.ww
```

```
114.0
```

```
jc.bw
```

```
150.0
```

Spatial Autocorrelation

Join Counts

One way to formalize a test for spatial autocorrelation in a binary attribute is to consider the so-called joins. A join exists for each neighbour pair of observations, and the joins are reflected in our binary spatial weights object **wq**.

Each unit can take on one of two values "Black" or "White", and so for a given pair of neighbouring locations there are three different types of joins that can arise:
Black Black (BB)
White White (WW)
Black White (or White Black) (BW)

Given that we have 68 Black polygons on our map, what is the number of Black Black (BB) joins we could expect if the process were such that the Black polygons were randomly assigned on the map? This is the logic of join count statistics.

The critical question for us, is whether this is a departure from what we would expect if the process generating the spatial distribution of the Black polygons were a completely random one?

To answer this, PySAL uses random spatial permutations of the observed attribute values to generate a realization under the null of complete spatial randomness (CSR). This is repeated a large number of times (999 default) to construct a reference distribution to evaluate the statistical significance of our observed counts. The average number of BB joins from the synthetic realizations is:

```
jc.mean_bb
```

```
92.65365365365365
```

which is less than our observed count. The question is whether our observed value is so different from the expectation that we would reject the null of CSR?

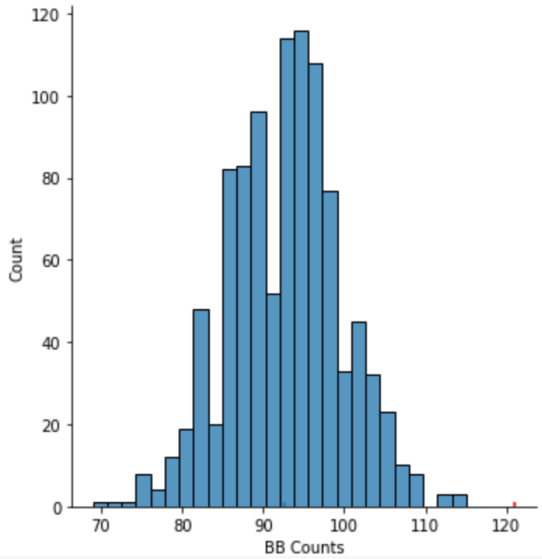
Spatial Autocorrelation

Join Counts

The density portrays the distribution of the BB counts, with the black vertical line indicating the mean BB count from the synthetic realizations and the red line the observed BB count for our prices. Clearly our observed value is extremely high. A pseudo p-value summarizes this:

```
import seaborn as sbn
sbn.displot(jc.sim_bb)
plt.vlines(jc.bb, 0, 1, color='r')
plt.vlines(jc.mean_bb, 0,1)
plt.xlabel('BB Counts')
```

```
Text(0.5, 6.79999999999999, 'BB Counts')
```




```
jc.p_sim_bb
```

```
0.001
```

Since this is below conventional significance levels, we would reject the null of complete spatial randomness in favour of spatial autocorrelation in market prices.

Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

Moran's Scatter Plot

In addition to the Global autocorrelation statistics, PySAL has many local autocorrelation statistics. Let's compute a local Moran statistic for the same d

```
np.random.seed(12345)
import esda
```

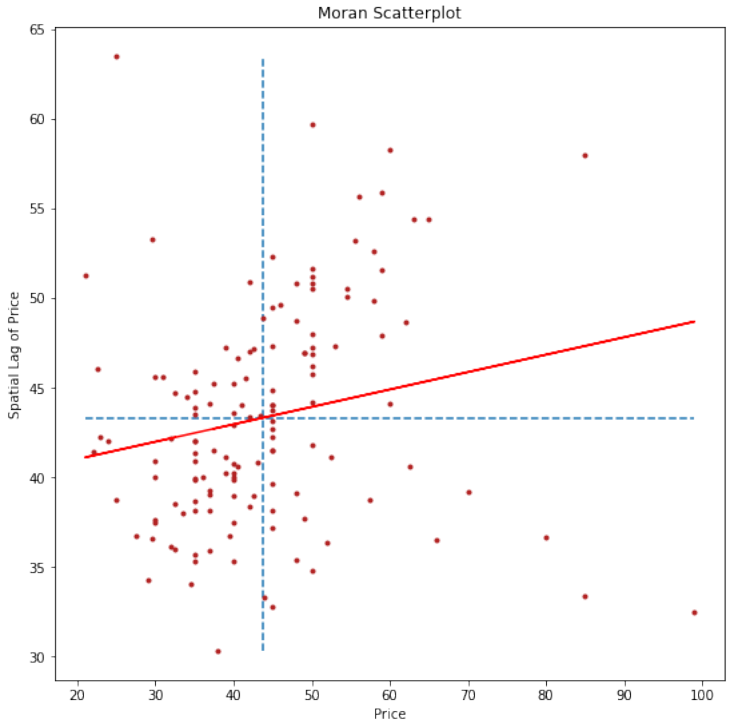
```
wq.transform = 'r'
lag_price = lp.lag_spatial(wq, df['median_pri'])
```

```
price = df['median_pri']
b, a = np.polyfit(price, lag_price, 1)
f, ax = plt.subplots(1, figsize=(9, 9))

plt.plot(price, lag_price, '.', color='firebrick')

 # dashed vert at mean of the price
plt.vlines(price.mean(), lag_price.min(), lag_price.max(), linestyle='--')
 # dashed horizontal at mean of lagged price
plt.hlines(lag_price.mean(), price.min(), price.max(), linestyle='--')

# red line of best fit using global I as slope
plt.plot(price, a + b*price, 'r')
plt.title('Moran Scatterplot')
plt.ylabel('Spatial Lag of Price')
plt.xlabel('Price')
plt.show()
```

Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

Moran's Scatter Plot

In addition to the Global autocorrelation statistics, PySAL has many local autocorrelation statistics. Let's compute a local Moran statistic for the same d



Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

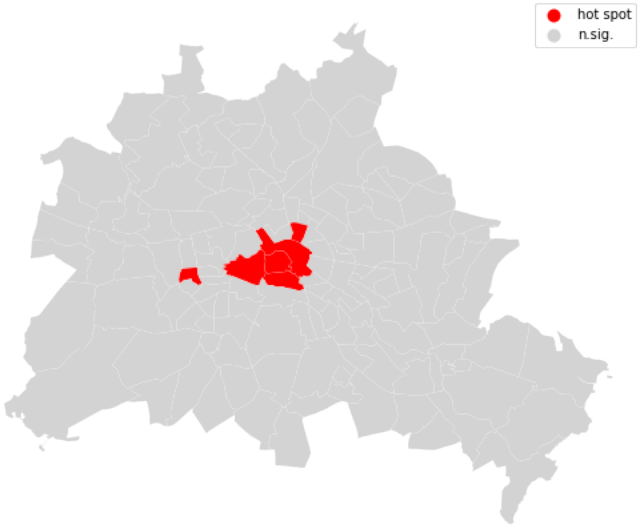
What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
sig = li.p_sim < 0.05
hotspot = sig * li.q==1
coldspot = sig * li.q==3
doughnut = sig * li.q==2
diamond = sig * li.q==4
```

```
spots = ['n.sig.', 'hot spot']
labels = [spots[i] for i in hotspot*1]
```

```
df = df
from matplotlib import colors
hmap = colors.ListedColormap(['red', 'lightgrey'])
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=labels).plot(column='cl', categorical=True, \
        k=2, cmap=hmap, linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.show()
```



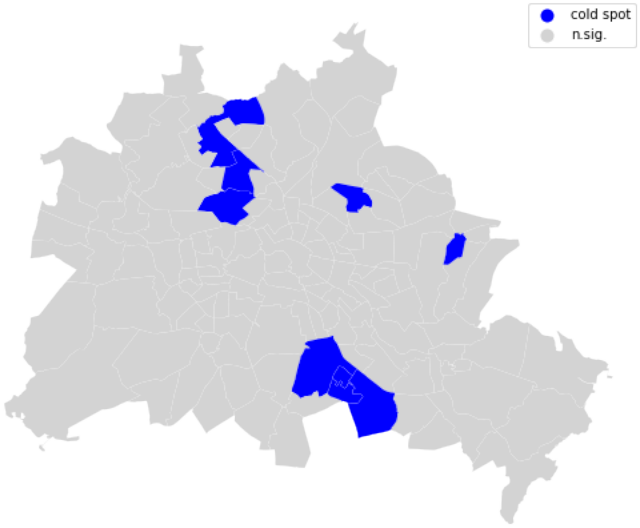
Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
spots = ['n.sig.', 'cold spot']
labels = [spots[i] for i in coldspot*1]
```

```
df = df
from matplotlib import colors
hmap = colors.ListedColormap(['blue', 'lightgrey'])
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=labels).plot(column='cl', categorical=True, \
        k=2, cmap=hmap, linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.show()
```



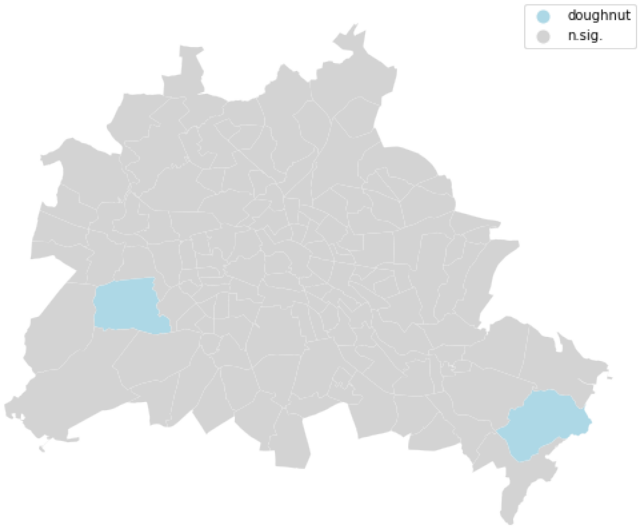
Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
spots = ['n.sig.', 'doughnut']
labels = [spots[i] for i in doughnut*1]
```

```
df = df
from matplotlib import colors
hmap = colors.ListedColormap(['lightblue', 'lightgrey'])
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=labels).plot(column='cl', categorical=True, \
        k=2, cmap=hmap, linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.show()
```



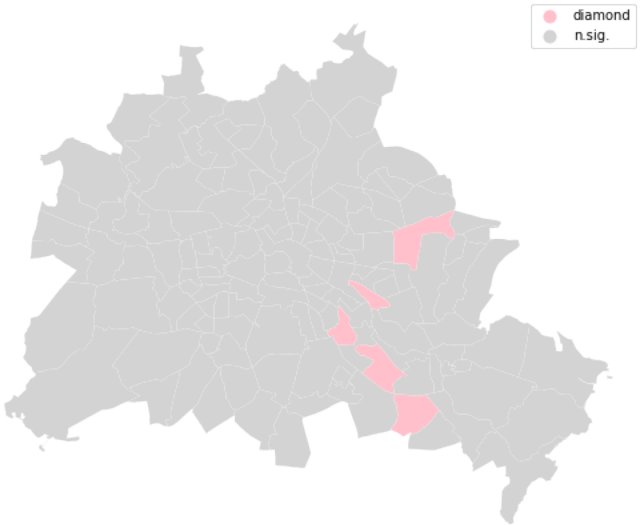
Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
spots = ['n.sig.', 'diamond']
labels = [spots[i] for i in diamond*1]
```

```
df = df
from matplotlib import colors
hmap = colors.ListedColormap(['pink', 'lightgrey'])
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=labels).plot(column='cl', categorical=True, \
        k=2, cmap=hmap, linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.show()
```



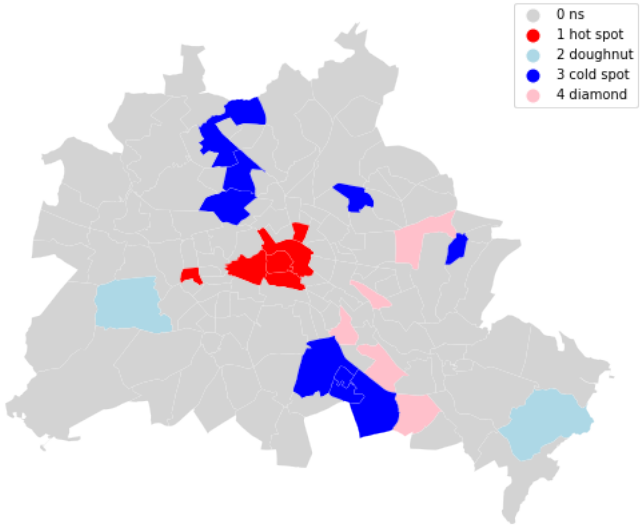
Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
sig = 1 * (li.p_sim < 0.05)
hotspot = 1 * (sig * li.q==1)
coldspot = 3 * (sig * li.q==3)
doughnut = 2 * (sig * li.q==2)
diamond = 4 * (sig * li.q==4)
spots = hotspot + coldspot + doughnut + diamond
spots
```

```
array([0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 3, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0,
       0, 0, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 3, 3, 0, 0, 0, 0, 0,
       3, 0, 0, 0, 3, 0])
```



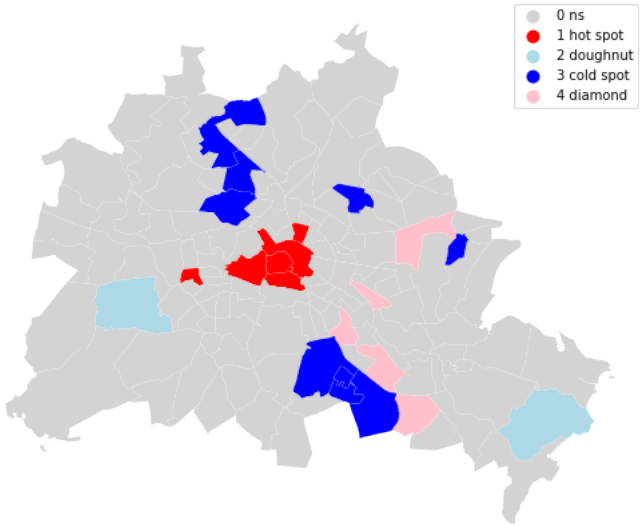
Local Autocorrelation: Hot Spots, Cold Spots, and Spatial Outliers

What do we mean by data streams?

Streaming data is the continuous flow of data generated by various sources. By using stream processing technology, data streams can be processed, stored, analysed, and acted upon as it's generated in real-time.

```
spot_labels = [ '0 ns', '1 hot spot', '2 doughnut', '3 cold spot', '4 diamond']
labels = [spot_labels[i] for i in spots]
```

```
from matplotlib import colors
hmap = colors.ListedColormap([ 'lightgrey', 'red', 'lightblue', 'blue', 'pink'])
f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=labels).plot(column='cl', categorical=True, \
        k=2, cmap=hmap, linewidth=0.1, ax=ax, \
        edgecolor='white', legend=True)
ax.set_axis_off()
plt.show()
```



Introduction to Data Visualisation

Choropleth Maps

Choropleth maps play a prominent role in spatial data science. The word choropleth stems from the root "choro" meaning "region". As such choropleth maps are appropriate for areal unit data where each observation combines a value of an attribute and a polygon.

Choropleth maps derive from an earlier era where cartographers faced technological constraints that precluded the use of unclassed maps where each unique attribute value could be represented by a distinct symbol. Instead, attribute values were grouped into a smaller number of classes with each class being associated with a unique symbol that was in turn applied to all polygons with attribute values falling in the class.

Although today these technological constraints are no longer binding, and unclassed mapping is feasible, there are still good reasons for adopting a classed approach. Chief among these is to reduce the cognitive load involved in parsing the complexity of an unclassed map. A choropleth map reduces this complexity by drawing upon statistical and visualization theory to provide an effective representation of the spatial distribution of the attribute values across the areal units.

The effectiveness of a choropleth map will be a function of the choice of classification scheme together with the colour or symbolization strategy adopted. In broad terms, the classification scheme defines the number of classes as well as the rules for assignment, while the symbolization should convey information about the value differentiation across the classes.

Introduction to Data Visualisation

Spatial Distribution

```
import pandas as pd
import geopandas as gpd
import libpysal.weights as lp
import matplotlib.pyplot as plt
import rasterio as rio
import numpy as np
import contextily as ctx
import shapely.geometry as geom
%matplotlib inline
```

```
df = gpd.read_file('data/neighborhoods.shp')
df.head()
```

	neighbourh	neighbou_1	median_pri	geometry
0	Blankenfelde/Niederschönhausen	Pankow	37.5	POLYGON ((1493006.880 6912074.798, 1492997.641...
1	Helmholtzplatz	Pankow	58.0	POLYGON ((1493245.549 6900059.696, 1493264.362...
2	Wiesbadener Straße	Charlottenburg-Wilm.	50.0	POLYGON ((1481381.452 6885170.698, 1481376.777...
3	Schmöckwitz/Karolinenhof/Rauchfangswerder	Treptow - Köpenick	99.0	POLYGON ((1526159.829 6872101.044, 1526108.176...
4	Müggelheim	Treptow - Köpenick	25.0	POLYGON ((1529265.086 6874326.842, 1529277.554...

Introduction to Data Visualisation

Spatial Distribution

```
df = gpd.read_file('data/neighborhoods.shp')
df.head()
```

	neighbourh	neighbou_1	median_pri	geometry
0	Blankenfelde/Niederschönhausen	Pankow	37.5	POLYGON ((1493006.880 6912074.798, 1492997.641...
1	Helmholtzplatz	Pankow	58.0	POLYGON ((1493245.549 6900059.696, 1493264.362...
2	Wiesbadener Straße	Charlottenburg-Wilm.	50.0	POLYGON ((1481381.452 6885170.698, 1481376.777...
3	Schmöckwitz/Karolinenhof/Rauchfangswerder	Treptow - Köpenick	99.0	POLYGON ((1526159.829 6872101.044, 1526108.176...
4	Müggelheim	Treptow - Köpenick	25.0	POLYGON ((1529265.086 6874326.842, 1529277.554...

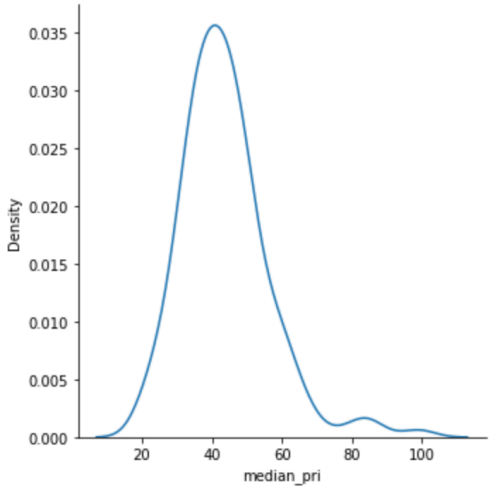
```
df = df
df['median_pri'].fillna((df['median_pri'].mean()), inplace=True)
```

```
import seaborn as sbn
```

We can use `seaborn` to visualize the statistical distribution of the median price of listings across neighborhoods:

```
sbn.displot(df['median_pri'], kind= 'kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x1511fca10>
```



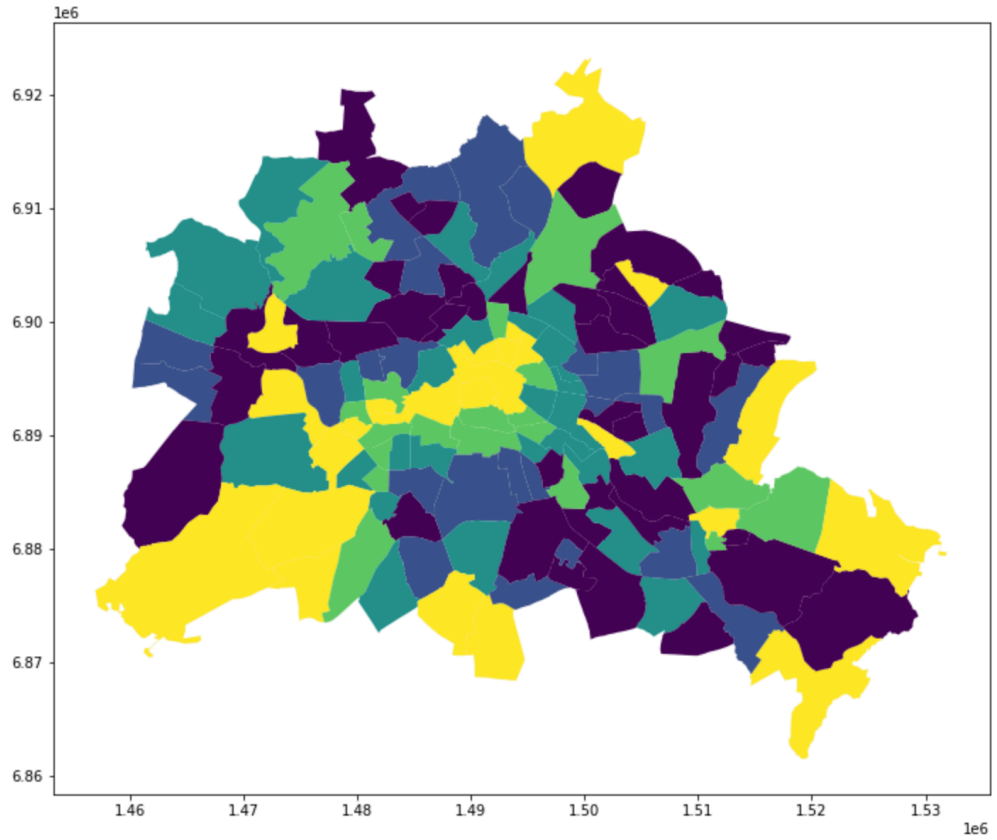
Introduction to Data Visualisation

Spatial Distribution

Plotting a choropleth without prior knowledge of this urban area may make interpretation of the visualization challenging. Therefore, let us specify some option in the form of scheme-

```
fig, ax = plt.subplots(figsize=(12,10), subplot_kw={'aspect':'equal'})
df.plot(column='median_pri', scheme='Quantiles', ax=ax)
```

```
<AxesSubplot:>
```



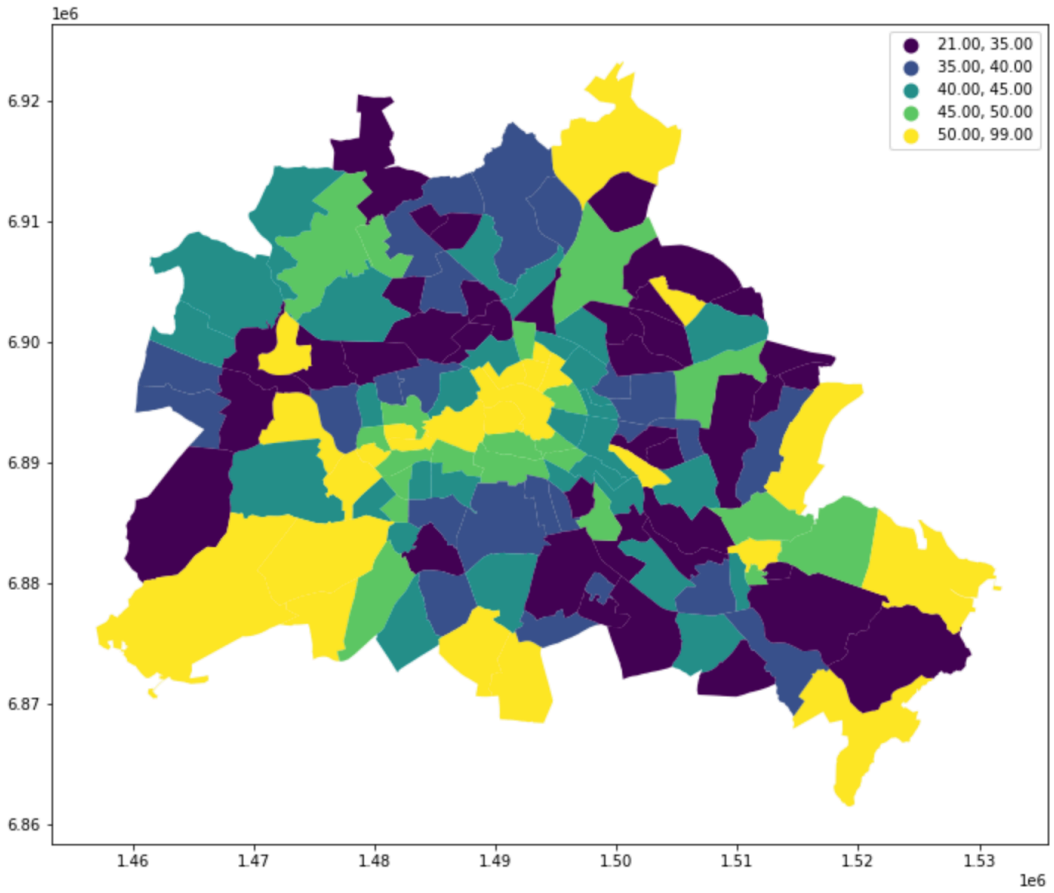
Introduction to Data Visualisation

Spatial Distribution

This has improved the visualization, but there are still questions. For example, how many different colours (hues) do we see? A choropleth map uses different hues to distinguish different value classes, in a similar sense to the bins of a histogram separating observations with different values. Adding a legend argument reveals the nature of the classes:

```
fig, ax = plt.subplots(figsize=(12,10), subplot_kw={'aspect':'equal'})
df.plot(column='median_pri', scheme='Quantiles',  legend=True, ax=ax)
```

```
<AxesSubplot:>
```



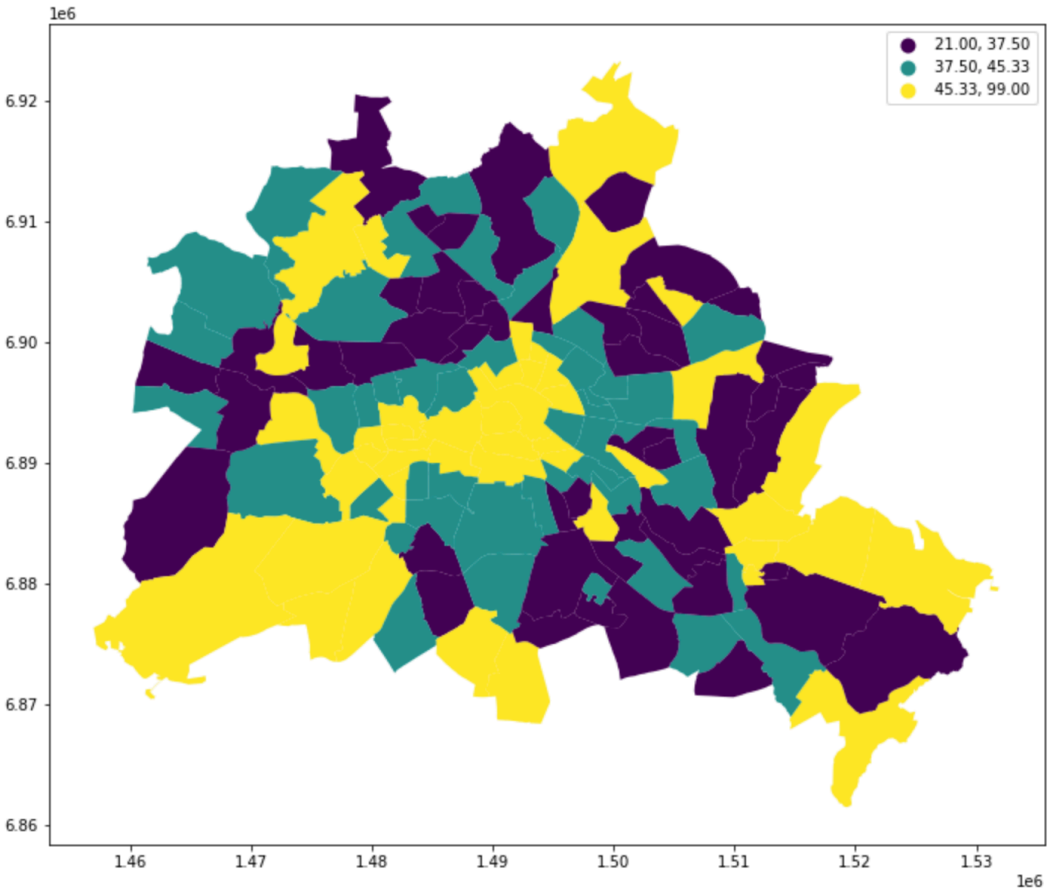
Introduction to Data Visualisation

Spatial Distribution

The five classes in the legend are the quintiles for the attribute distribution. If we instead would like to see the quartile classification, we set a new option K=3

```
fig, ax = plt.subplots(figsize=(12,10), subplot_kw={'aspect':'equal'})
df.plot(column='median_pri', scheme='Quantiles', k=3, legend=True, ax=ax)
```

```
<AxesSubplot:>
```



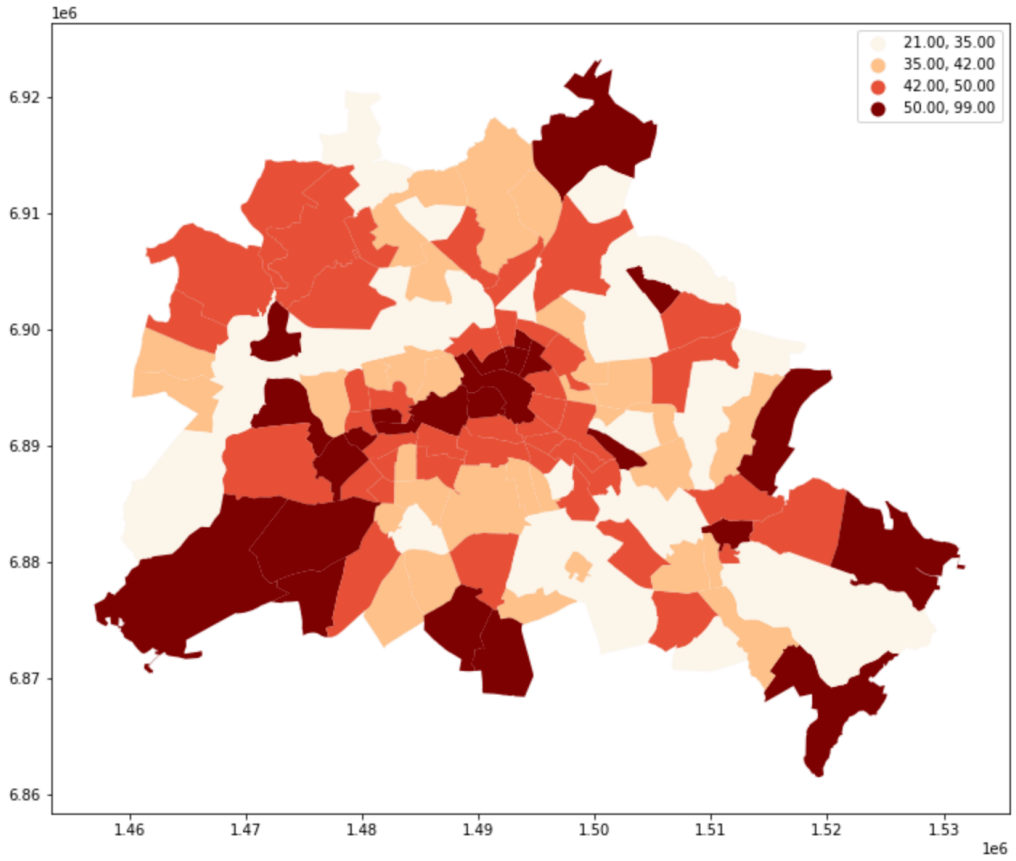
Introduction to Data Visualisation

Spatial Distribution

In addition to changing the number of classes, the color map that defines the hues for the classes can be changed:

```
fig, ax = plt.subplots(figsize=(12,10), subplot_kw={'aspect':'equal'})
df.plot(column='median_pri', scheme='Quantiles', k=4, legend=True, ax=ax,
        cmap='OrRd')
```

```
<AxesSubplot:>
```

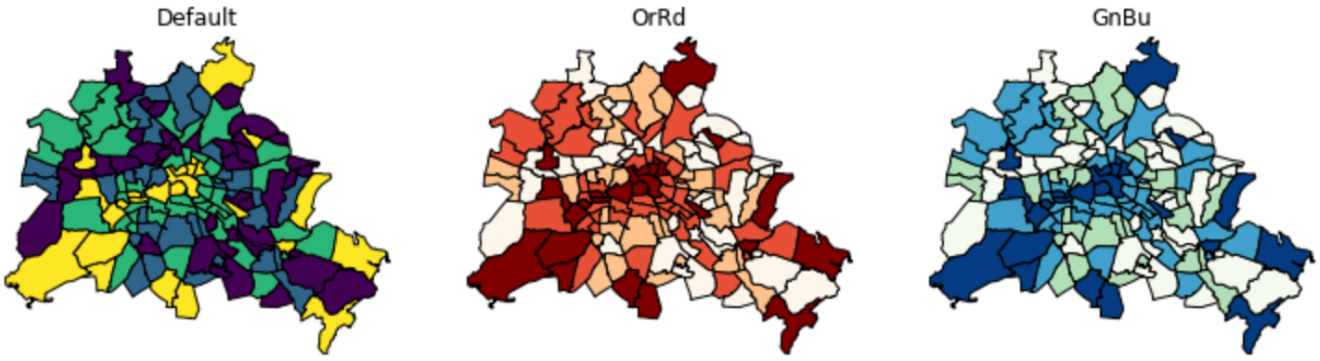


Introduction to Data Visualisation

Spatial Distribution

Both the color ramp and the classification scheme can have substantial impacts on the resulting visualization. GeoPandas makes it straightforward to explore these dimensions. For example, changing the color ramp results in:

```
f,ax = plt.subplots(1,3,figsize=(3.16*4,4))
df.plot(column='median_pri', ax=ax[0], edgecolor='k',
        scheme="quantiles",  k=4)
ax[0].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[0].set_title("Default")
df.plot(column='median_pri', ax=ax[1], edgecolor='k',
        scheme='quantiles', cmap='OrRd', k=4)
ax[1].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[1].set_title("OrRd")
df.plot(column='median_pri', ax=ax[2], edgecolor='k',
        scheme='quantiles', cmap='GnBu', k=4)
ax[2].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[2].set_title("GnBu")
ax[0].axis('off')
ax[1].axis('off')
ax[2].axis('off')
plt.show()
```



Introduction to Data Visualisation

Spatial Distribution

Alternatively we can change the classification scheme:

```
f,ax = plt.subplots(1,3,figsize=(3.16*4,4))
df.plot(column='median_pri', ax=ax[0], edgecolor='k',
        scheme="quantiles", cmap='OrRd', k=4)
ax[0].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[0].set_title("Quartiles")
df.plot(column='median_pri', ax=ax[1], edgecolor='k',
        scheme='equal_interval', cmap='OrRd', k=4)
ax[1].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[1].set_title("Equal Interval")
df.plot(column='median_pri', ax=ax[2], edgecolor='k',
       scheme='fisher_jenks', cmap='OrRd', k=4)
ax[2].axis(df.total_bounds[np.asarray([0,2,1,3])])
ax[2].set_title("Fisher Jenks")
ax[0].axis('off')
ax[1].axis('off')
ax[2].axis('off')
plt.show()
```

Introduction to Data Visualisation

Choosing a Classification Scheme

GeoPandas supports a subset of the map classification schemes that are available in PySAL. The selected ones have a similar function signature that allowed for a simple soft dependency.

Since the initial implementation, PySAL has begun a major refactoring, and as part of this the classification scheme have been moved into their own package: mapclassify. We can explore some of the other classifiers within this package:

```
import mapclassify as mc
mc.CLASSIFIERS
```

```
('BoxPlot',
 'EqualInterval',
 'FisherJenks',
 'FisherJenksSampled',
 'HeadTailBreaks',
```

```
from mapclassify import EqualInterval
y = df['median_pri']
ea5 = mc.EqualInterval(y, k=5)
```

```
ea5
```

```
EqualInterval

   Interval      Count
----------------------
[21.00, 36.60] |    40
(36.60, 52.20] |    74
(52.20, 67.80] |    19
(67.80, 83.40] |     2
(83.40, 99.00] |     3
```

Introduction to Data Visualisation

Comparing Classification Schemes

Choropleth mapping can be used for different purposes. The most common is to select a classification that provides a balance between maximizing the differences between observations in each bin, and minimizing the intra-bin heterogeneity.

The classifiers in PySAL have underlying measures of fit for this purpose, but it is important to keep in mind these should only be used for classifiers with the same number of classes. One such measure is the the absolute deviation around class medians (ADCM). In order to compare the classification methods, we have specified K=5 for various classifiers.

```
import mapclassify as mc

q5 = mc.Quantiles(y, k=5)
ei5 = mc.EqualInterval(y, k=5)
mb5 = mc.MaximumBreaks(y, k=5)
fj5 = mc.FisherJenks(y, k=5)
fits = [c.adcm for c in [q5, ei5, mb5, fj5]]
fits
```

```
[403.77007299270076,
 490.27007299270076,
 1040.7299270072992,
 372.27007299270076]
```

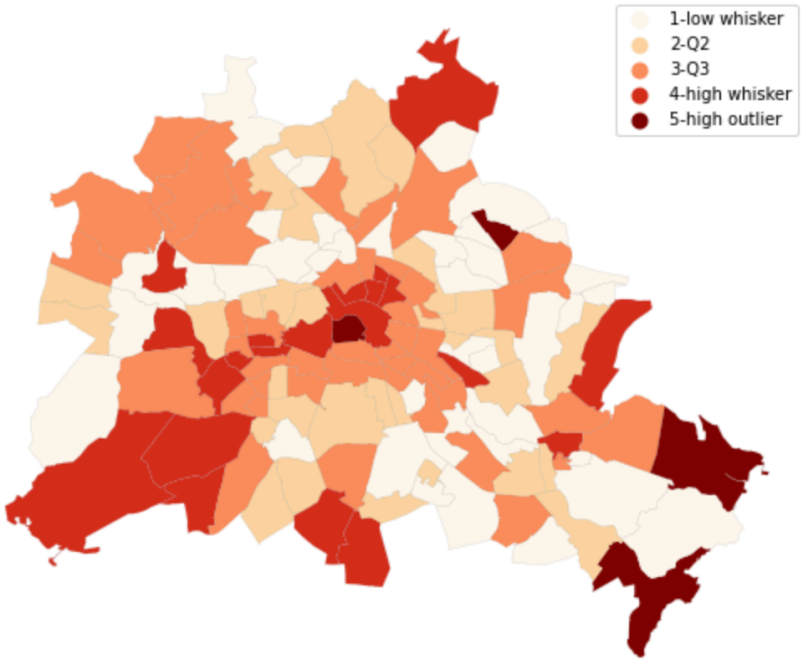
Based on the results above, we can conclude that the Fisher Jenkins turns out to be the best classifier.

Introduction to Data Visualisation

MapClassify and GeoPandas

Although only a subset of the PySAL classifiers are directly accessible from within GeoPandas, it is possible to combine external classifiers with GeoPandas

```
labels = ['0-low outlier', '1-low whisker',
          '2-Q2', '3-Q3', '4-high whisker', '5-high outlier']
bpl = [ labels[b] for b in bp.yb ]

f, ax = plt.subplots(1, figsize=(9, 9))
df.assign(cl=bpl).plot(column='cl', categorical=True, \
                                    k=4, cmap='OrRd', linewidth=0.1, ax=ax,\
                                    edgecolor='grey', legend=True)
ax.set_axis_off()
plt.show()
```



Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

However, when working with spatial data, there are two simultaneous senses of what is near:
- things that similar in attribute (classical kernel regression)
- things that are similar in spatial position (spatial kernel regression)

Below, we'll walk through how to use scikit-learn to fit these two types of kernel regressions, show how it's not super simple to mix the two approaches together, and refer to an approach that does this correctly in another package.

First, though, let's try to predict the log of an Airbnb's nightly price based on a few factors:
- accommodates: the number of people the Airbnb can accommodate
- review_scores_rating: the aggregate rating of the listing
- bedrooms: the number of bedrooms the Airbnb has
- bathrooms: the number of bathrooms the Airbnb has
- beds: the number of beds the Airbnb offers

Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

```
import numpy as np
import libpysal.weights as lp
import geopandas as gpd
import pandas as pd
import shapely.geometry as shp
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
listings = pd.read_csv('./data/berlin-listings.csv.gz')
listings['geometry'] = listings[['longitude','latitude']].apply(shp.Point, axis=1)
listings = gpd.GeoDataFrame(listings)
listings.crs = {'init':'epsg:4269'}
listings = listings.to_crs(epsg=3857)
```

Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

```
listings = pd.read_csv('./data/berlin-listings.csv.gz')
listings['geometry'] = listings[['longitude','latitude']].apply(shp.Point, axis=1)
listings = gpd.GeoDataFrame(listings)
listings.crs = {'init':'epsg:4269'}
listings = listings.to_crs(epsg=3857)
```

```
listings[['scrape_id', 'name',
       'summary', 'space', 'description', 'experiences_offered',
       'neighborhood_overview', 'notes', 'transit', 'access', 'interaction',
       'house_rules', 'thumbnail_url', 'medium_url', 'picture_url',
       'xl_picture_url', 'host_id', 'host_url', 'host_name', 'host_since',
       'host_location', 'host_about']].head()
```

	scrape_id	name	summary	space	description	experiences_offered	neighborhood_overview	notes	transit
0	20170507222235	Kunterbuntes Zimmer mit eigenem Bad für jedermann	Meine Unterkunft ist gut für paare, alleinreis...	NaN	Meine Unterkunft ist gut für paare, alleinreis...	none	NaN	NaN	NaN
1	20170507222235	Modernes Zimmer in Berlin Pankow	Es ist ein schönes gepflegtes und modernes Zim...	Das Haus befindet sich direkt vor eine Tram Ha...	Es ist ein schönes gepflegtes und modernes Zim...	none	NaN	NaN	NaN
2	20170507222235	Gästezimmer Berlin-Pankow	Unser Gästezimmer befindet sich im Dachgeschos...	Wenn Ihr eine anspruchsvolles Ambiente sucht, ...	Unser Gästezimmer befindet sich im Dachgeschos...	none	Unweit der Berliner City, im nördlichen Pankow...	Wir haben zwar keinen offiziellen 24/7 Check-I...	Mit dem ÖPNV oder dem eigenen Fahrzeug seid Ih...
3	20170507222235	Sonniges Doppelzimmer+Nice Price!	Welcome! Hier vermiete ich ein kleines Wohlfüh...	Die Wohnung ist durch Ihre Lage sehr schön hel...	Welcome! Hier vermiete ich ein kleines Wohlfüh...	none	NaN	NaN	- U-Bahn (U2) Richtung Schönhauser Allee (2...
4	20170507222235	Room for womens in Pankow 30 min from the city	Gemütliches Zimmer im ruhigen Teil von Berlin....	Es handelt sich um ein großes, gemütliches Zim...	Gemütliches Zimmer im ruhigen Teil von Berlin....	none	NaN	NaN	Die Tram M1 ist 2 Minuten erreichbar und brauc...

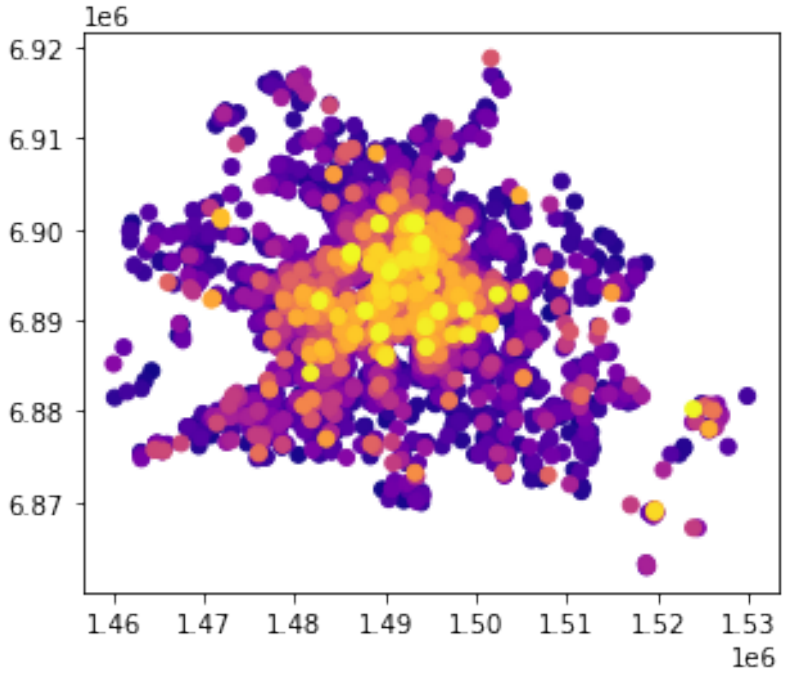
Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

```
listings = pd.read_csv('./data/berlin-listings.csv.gz')
listings['geometry'] = listings[['longitude','latitude']].apply(shp.Point, axis=1)
listings = gpd.GeoDataFrame(listings)
listings.crs = {'init':'epsg:4269'}
listings = listings.to_crs(epsg=3857)
```

```
listings.sort_values('price').plot('price', cmap='plasma')
```



Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

```
model_data = listings[['accommodates', 'review_scores_rating',
                       'bedrooms', 'bathrooms', 'beds',
                       'price', 'geometry']].dropna()
```

```
Xnames = ['accommodates', 'review_scores_rating',
              'bedrooms', 'bathrooms', 'beds' ]
X = model_data[Xnames].values
X = X.astype(float)
y = np.log(model_data[['price']].values)
```

Introduction to Spatial Regression

Introduction to Kernel Regression

Kernel regressions are one exceptionally common way to allow observations to "borrow strength" from nearby observations.

```
model_data = listings[['accommodates', 'review_scores_rating',
                       'bedrooms', 'bathrooms', 'beds',
                       'price', 'geometry']].dropna()
```

```
Xnames = ['accommodates', 'review_scores_rating',
              'bedrooms', 'bathrooms', 'beds' ]
X = model_data[Xnames].values
X = X.astype(float)
y = np.log(model_data[['price']].values)
```

Further, since each listing has a location, I'll extract the set of spatial coordinates coordinates for each listing.

```
coordinates = np.vstack(model_data.geometry.apply(lambda p: np.hstack(p.xy)).values)
```

scikit neighbor regressions are contained in the `sklearn.neighbors` module, and there are two main types:

- `KNeighborsRegressor`, which uses a k-nearest neighborhood of observations around each focal site
- `RadiusNeighborsRegressor`, which considers all observations within a fixed radius around each focal site.

Further, these methods can use inverse distance weighting to rank the relative importance of sites around each focal; in this way, near things are given more weight than far things, even when there's a lot of near things.

```
import sklearn.neighbors as skn
import sklearn.metrics as skm
```

```
shuffle = np.random.permutation(len(y))
train,test = shuffle[:14000],shuffle[14000:]
```

Introduction to Spatial Regression

Introduction to Kernel Regression

So, let's fit three models:

- spatial: using inverse distance weighting on the nearest 500 neighbours geographical space
- attribute: using inverse distance weighting on the nearest 500 neighbours in attribute space
- both: using inverse distance weighting in both geographical and attribute space.

```
KNNR = skn.KNeighborsRegressor(weights='distance', n_neighbors=500)
spatial = KNNR.fit(coordinates[train,:],
                   y[train,:])
KNNR = skn.KNeighborsRegressor(weights='distance', n_neighbors=500)
attribute = KNNR.fit(X[train,:],
                     y[train,])
KNNR = skn.KNeighborsRegressor(weights='distance', n_neighbors=500)
both = KNNR.fit(np.hstack((coordinates,X))[train,:],
                y[train,:])
```

To score them, I'm going to grab their out of sample prediction accuracy and get their % explained variance:

```
sp_ypred =  spatial.predict(coordinates[test,:])
att_ypred = attribute.predict(X[test,:])
both_ypred = both.predict(np.hstack((X,coordinates))[test,:])
```

```
(skm.explained_variance_score(y[test,], sp_ypred),
 skm.explained_variance_score(y[test,], att_ypred),
 skm.explained_variance_score(y[test,], both_ypred))
```

```
(0.1228403051335415, 0.30370650056319404, -3.6399763203576185e-09)
```

Introduction to Spatial Regression

Introduction to Data Borrowing

We may also think towards using weights for the feature engineering. Using the weights matrix, we can construct neighbourhood averages of the data matrix and use these as synthetic features in your model. These often have a strong relationship to the outcome as well, since spatial data is often smooth and attributes of nearby sites often have a spill over impact on each other.

First, we'll construct a Kernel weight from the data that we have, make it an adaptive Kernel bandwidth, and make sure that our kernel weights don't have any self-neighbours. Since we've got the data at each site anyway, we probably shouldn't use that data again when we construct our neighbourhood-smoothed synthetic features.

```
from libpysal.weights.util import fill_diagonal

kW = lp.Kernel.from_dataframe(model_data, fixed=False, function='gaussian', k=100)
kW = fill_diagonal(kW, 0)
WX = lp.lag_spatial(kW, X)
```

```
WX
```

```
array([[  92.07580909, 2813.98214407,   43.08772341,   32.38117883,
          63.8159369 ],
       [  92.09327885, 2803.72220881,   42.13022813,   31.61577382,
          63.17553316],
       [  89.21026817, 2822.25906403,   42.21070891,   32.61703576,
          62.39659919],
       ...,
       [  75.05668356, 2965.92866643,   35.08541713,   32.50997936,
          44.56876865],
       [  80.30300756, 2873.02984261,   36.2196911 ,   32.50706722,
          54.15117167],
       [  81.07968962, 2833.53516619,   36.89406676,   32.7479039 ,
          55.91792687]])
```

Introduction to Spatial Regression

Introduction to Data Borrowing

We may also think towards using weights for the feature engineering. Using the weights matrix, we can construct neighbourhood averages of the data matrix and use these as synthetic features in your model. These often have a strong relationship to the outcome as well, since spatial data is often smooth and attributes of nearby sites often have a spill over impact on each other.

First, we'll construct a Kernel weight from the data that we have, make it an adaptive Kernel bandwidth, and make sure that our kernel weights don't have any self-neighbours. Since we've got the data at each site anyway, we probably shouldn't use that data again when we construct our neighbourhood-smoothed synthetic features.

```
from libpysal.weights.util import fill_diagonal

kW = lp.Kernel.from_dataframe(model_data, fixed=False, function='gaussian', k=100)
kW = fill_diagonal(kW, 0)
WX = lp.lag_spatial(kW, X)
```

```
kW.to_adjlist()[kW.to_adjlist()["focal"]== 1]
```

	focal	neighbor	weight
100	1	0	0.348520
101	1	2	0.255500
102	1	3	0.359750
103	1	4	0.336482
104	1	5	0.387411
...	...	...	...
195	1	17321	0.248255
196	1	17339	0.262199
197	1	19963	0.246456

Introduction to Spatial Regression

Introduction to Data Borrowing

Below are the results for the model with only the covariates used above:

```
import statsmodels.api as sm
Xtable = pd.DataFrame(X, columns=Xnames)
onlyX = sm.OLS(y,sm.add_constant(Xtable)).fit()
onlyX.summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.290
Model:	OLS	**Adj. R-squared:**	0.290
Method:	Least Squares	**F-statistic:**	1269.
Date:	Fri, 25 Jun 2021	**Prob (F-statistic):**	0.00
Time:	01:21:46	**Log-Likelihood:**	-9260.8
No. Observations:	15516	**AIC:**	1.853e+04
Df Residuals:	15510	**BIC:**	1.858e+04
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.9682	0.044	67.590	0.000	2.882	3.054
accommodates	0.1882	0.004	44.362	0.000	0.180	0.197
review_scores_rating	0.0033	0.000	7.378	0.000	0.002	0.004
bedrooms	0.1427	0.008	18.503	0.000	0.128	0.158
bathrooms	0.0062	0.012	0.497	0.619	-0.018	0.031
beds	-0.0482	0.005	-9.221	0.000	-0.058	-0.038

Omnibus:	156.822	**Durbin-Watson:**	1.716
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	265.898
Skew:	-0.005	**Prob(JB):**	1.82e-58
Kurtosis:	3.641	**Cond. No.**	1.17e+03

Introduction to Spatial Regression

Introduction to Data Borrowing

Then, we could fit a model using the neighbourhood average synthetic features as well:

```
withWX = sm.OLS(y,sm.add_constant(XWXtable)).fit()
withWX.summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.311
Model:	OLS	**Adj. R-squared:**	0.311
Method:	Least Squares	**F-statistic:**	701.4
Date:	Fri, 25 Jun 2021	**Prob (F-statistic):**	0.00
Time:	01:21:59	**Log-Likelihood:**	-9026.9
No. Observations:	15516	**AIC:**	1.808e+04
Df Residuals:	15505	**BIC:**	1.816e+04
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.7158	0.114	23.723	0.000	2.491	2.940
accommodates	0.1829	0.004	43.616	0.000	0.175	0.191
review_scores_rating	0.0035	0.000	8.036	0.000	0.003	0.004
bedrooms	0.1465	0.008	19.126	0.000	0.131	0.161
bathrooms	-0.0135	0.012	-1.087	0.277	-0.038	0.011
beds	-0.0448	0.005	-8.616	0.000	-0.055	-0.035
lag_accommodates	0.0162	0.001	16.632	0.000	0.014	0.018
lag_review_scores_rating	-0.0004	4.29e-05	-9.279	0.000	-0.000	-0.000
lag_bedrooms	0.0001	0.002	0.086	0.931	-0.003	0.003
lag_bathrooms	0.0240	0.002	10.595	0.000	0.020	0.028
lag_beds	-0.0150	0.001	-13.860	0.000	-0.017	-0.013

Omnibus:	163.213	**Durbin-Watson:**	1.786
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	278.468
Skew:	-0.023	**Prob(JB):**	3.40e-61
Kurtosis:	3.655	**Cond. No.**	9.71e+04

Introduction to Spatial Clustering

Point data clustering

Clusters are a common concern for spatial analysis. Previously, we observed one way to detect spatial clusters/outliers while performing exploratory data analysis but here we'll learn a few other strategies that can be used on point and lattice data.

```
import numpy as np
import libpysal.weights as lp
import geopandas as gpd
import pandas as pd
import shapely.geometry as shp
import matplotlib.pyplot as plt
import sklearn.metrics as skm
import seaborn as sns
%matplotlib inline
```

```
listings[["geometry"]].head()
```

	geometry
0	POINT (1491246.091 6906289.706)
1	POINT (1491523.302 6905027.621)
2	POINT (1491598.865 6907372.953)
3	POINT (1491189.631 6906107.655)
4	POINT (1490975.657 6906370.805)

Introduction to Spatial Clustering

Point data clustering

Clusters are a common concern for spatial analysis. Previously, we observed one way to detect spatial clusters/outliers while performing exploratory data analysis but here we'll learn a few other strategies that can be used on point and lattice data.

```
listings[["geometry"]].head()
```

	geometry
0	POINT (1491246.091 6906289.706)
1	POINT (1491523.302 6905027.621)
2	POINT (1491598.865 6907372.953)
3	POINT (1491189.631 6906107.655)
4	POINT (1490975.657 6906370.805)

```
import sklearn.cluster as skc
coordinates = listings['geometry'].apply(lambda p: np.hstack(p.xy)).values
coordinates = np.vstack(coordinates)
coordinates
```

```
array([[1491246.09061424, 6906289.70588367],
       [1491523.30240249, 6905027.62143263],
       [1491598.86544802, 6907372.95286508],
       ...,
       [1483626.59167149, 6909789.47565835],
       [1484643.05708275, 6911169.65547772],
       [1524454.54949633, 6867174.51240984]])
```

Introduction to Spatial Clustering

Point data clustering

Clusters are a common concern for spatial analysis. Previously, we observed one way to detect spatial clusters/outliers while performing exploratory data analysis but here we'll learn a few other strategies that can be used on point and lattice data.

```
import sklearn.cluster as skc
coordinates = listings['geometry'].apply(lambda p: np.hstack(p.xy)).values
coordinates = np.vstack(coordinates)
coordinates
```

```
array([[1491246.09061424, 6906289.70588367],
       [1491523.30240249, 6905027.62143263],
       [1491598.86544802, 6907372.95286508],
       ...,
       [1483626.59167149, 6909789.47565835],
       [1484643.05708275, 6911169.65547772],
       [1524454.54949633, 6867174.51240984]])
```

```
clusterer = skc.DBSCAN(eps=1000).fit(coordinates)
```

Now, `clusterer` contains all of the observations found to be components of clusters:

```
clusterer.components_
```

```
array([[1491246.09061424, 6906289.70588367],
       [1491523.30240249, 6905027.62143263],
       [1491598.86544802, 6907372.95286508],
       ...,
       [1484839.95240176, 6903009.17762691],
       [1486389.18508418, 6903758.10772346],
       [1486722.30038881, 6904230.45907474]])
```

Introduction to Spatial Clustering

Point data clustering

Clusters are a common concern for spatial analysis. Previously, we observed one way to detect spatial clusters/outliers while performing exploratory data analysis but here we'll learn a few other strategies that can be used on point and lattice data.

```
clusterer = skc.DBSCAN(eps=1000).fit(coordinates)
```

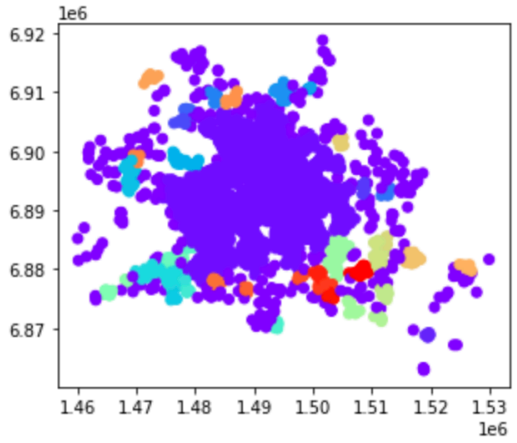
Now, `clusterer` contains all of the observations found to be components of clusters:

```
clusterer.components_
```

```
array([[1491246.09061424, 6906289.70588367],
       [1491523.30240249, 6905027.62143263],
       [1491598.86544802, 6907372.95286508],
       ...,
       [1484839.95240176, 6903009.17762691],
       [1486389.18508418, 6903758.10772346],
       [1486722.30038881, 6904230.45907474]])
```

```
listings.assign(labels=clusterer.labels_).plot('labels', k=nclusters, cmap='rainbow')
```

```
<AxesSubplot:>
```



Introduction to Spatial Clustering

Areal Clusters

Areal clusters are usually slightly harder to identify, since we might mean two things when we say "areal cluster"-

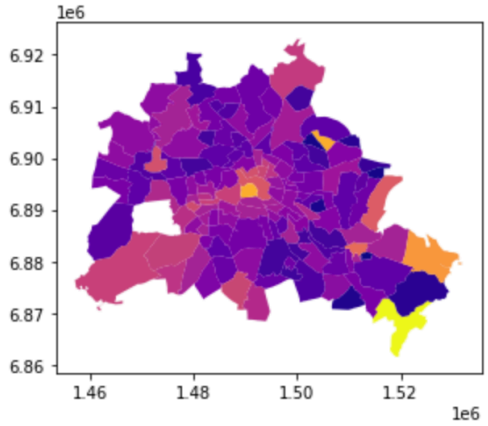
1. The first sense refers to clusters in data that can be visualized, kind of like what we were talking about while performing visual analysis in second notebook.
2. The second sense refers to contiguous areas of similarity in data, which require slightly different approaches to discover.

Let us tackle the first sense first at the neighbourhood level.

```
neighborhood = gpd.read_file('./data/berlin-neighbourhoods.geojson').to_crs(epsg=3857)
median_prices = listings.groupby('neighbourhood_cleansed').price.median().to_frame('median_price')
neighborhood = neighborhood.merge(median_prices, left_on='neighbourhood', right_index=True)
neighborhood = gpd.GeoDataFrame(neighborhood)
```

```
neighborhood.plot('median_price', cmap='plasma')
```

```
<AxesSubplot:>
```



Introduction to Spatial Clustering

Aerial Clusters

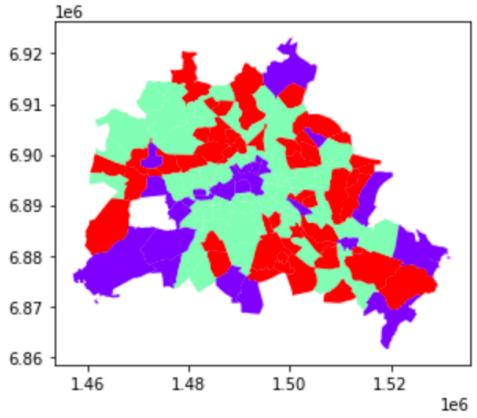
Areal clusters are usually slightly harder to identify, since we might mean two things when we say "areal cluster”-

1. The first sense refers to clusters in data that can be visualized, kind of like what we were talking about while performing visual analysis in second notebook.
2. The second sense refers to contiguous areas of similarity in data, which require slightly different approaches to discover.

Let us tackle the first sense first at the neighbourhood level.

```
neighborhood_priceclusters = skc.AgglomerativeClustering(n_clusters=3).fit(neighborhood[['median_price']])
neighborhood.assign(labels=neighborhood_priceclusters.labels_).plot('labels', cmap='rainbow')
```

```
<AxesSubplot:>
```



Introduction to Spatial Clustering

Aerial Clusters

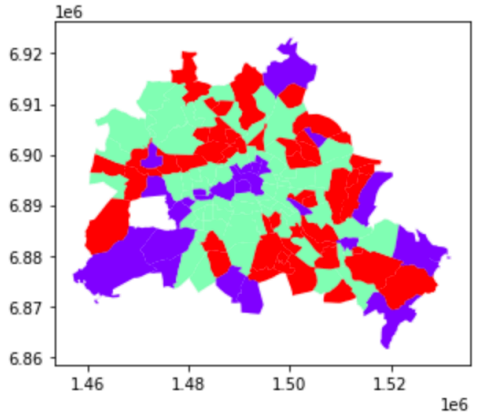
Areal clusters are usually slightly harder to identify, since we might mean two things when we say "areal cluster”-

1. The first sense refers to clusters in data that can be visualized, kind of like what we were talking about while performing visual analysis in second notebook.
2. The second sense refers to contiguous areas of similarity in data, which require slightly different approaches to discover.

Let us tackle the first sense first at the neighbourhood level.

```
neighborhood_priceclusters = skc.AgglomerativeClustering(n_clusters=3).fit(neighborhood[['median_price']])
neighborhood.assign(labels=neighborhood_priceclusters.labels_).plot('labels', cmap='rainbow')
```

```
<AxesSubplot:>
```



Note that these are all broken up around the map; while we've found three clusters with similar prices, there is no guarantee that these solutions will be connected spatially, and indeed we often find that this is the case.

Introduction to Spatial Clustering

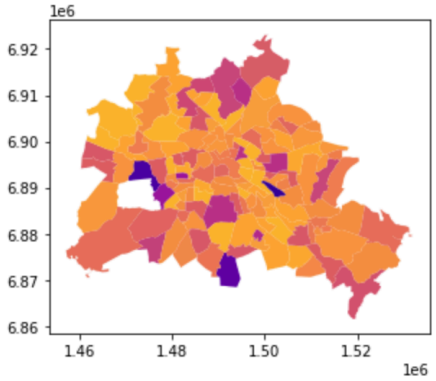
Aerial Clusters

Therefore, in order to measure the quality of clusters in the map, we can use the silhouette score, which is a standardized distance between each observation's data and the "decision boundary," or the nearest cluster that we didn't put the observation into.

```
silhouettes = skm.silhouette_samples(neighborhood[['median_price']], neighborhood_priceclusters.labels_)
```

```
neighborhood.assign(strength = silhouettes).plot('strength', cmap='plasma', vmin=-.5,vmax=1)
```

```
<AxesSubplot:>
```



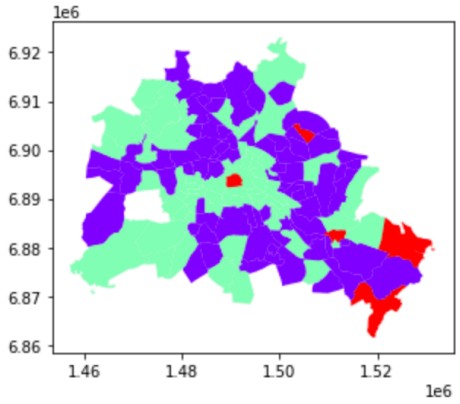
Introduction to Spatial Clustering

FisherJenks Clusters

The mapclassify package has some algorithms to do clustering as well, such as the Fisher Jenks optimal classification method, which finds solutions to k-means-style problems fast & efficiently:

```
from mapclassify.classifiers import FisherJenks
fisher_jenks = FisherJenks(neighborhood.median_price, k=3).yb
neighborhood.assign(labels=fisher_jenks).plot('labels', cmap='rainbow')
```

```
<AxesSubplot:>
```



Introduction to Spatial Clustering

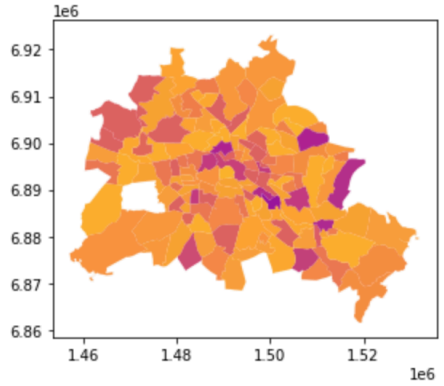
FisherJenks Clusters

The mapclassify package has some algorithms to do clustering as well, such as the Fisher Jenks optimal classification method, which finds solutions to k-means-style problems fast & efficiently:

```
silhouettes_fj = skm.silhouette_samples(neighborhood[['median_price']], fisher_jenks)
```

```
neighborhood.assign(strength = silhouettes_fj).plot('strength', cmap='plasma', vmin=-.5,vmax=1)
```

```
<AxesSubplot:>
```




```
skm.silhouette_score(neighborhood[['median_price']], fisher_jenks)
```

```
0.5605668091719826
```

```
skm.silhouette_score(neighborhood[['median_price']], neighborhood_priceclusters.labels_)
```

```
0.5335887580400264
```

Introduction to Spatial Clustering

Probabilistic Clustering

Sometimes, we'd rather treat clustering assignments as "predictions" in a probabilistic framework. Often, this is done either through latent dirichlet allocation or, more simply, through Gaussian mixture models. scikit-learn provides gaussian mixture methods which are also fast and simple to use.

```
import sklearn.mixture as skmix
gmm = skmix.GaussianMixture(n_components=3, random_state=7052018).fit(neighborhood[['median_price']])
predicted = gmm.predict(neighborhood[['median_price']])
```

```
strengths = gmm.predict_proba(neighborhood[['median_price']])
```

This gives the full matrix of probabilities (over all clusters).

```
strengths[0:5]
```

```
array([[8.12234237e-01, 3.36664575e-04, 1.87429098e-01],
       [2.27849052e-02, 3.43470611e-02, 9.42868034e-01],
       [1.87906330e-01, 5.42612080e-03, 8.06667549e-01],
       [2.66802059e-16, 9.99999846e-01, 1.53895689e-07],
       [9.74290541e-01, 6.14636516e-05, 2.56479954e-02]])
```

Introduction to Spatial Clustering

Probabilistic Clustering

Sometimes, we'd rather treat clustering assignments as "predictions" in a probabilistic framework. Often, this is done either through latent dirichlet allocation or, more simply, through Gaussian mixture models. scikit-learn provides gaussian mixture methods which are also fast and simple to use.

```
import sklearn.mixture as skmix
gmm = skmix.GaussianMixture(n_components=3, random_state=7052018).fit(neighborhood[['median_price']])
predicted = gmm.predict(neighborhood[['median_price']])
```

```
strengths = gmm.predict_proba(neighborhood[['median_price']])
```

This gives the full matrix of probabilities (over all clusters).

```
strengths[0:5]
```

```
array([[8.12234237e-01, 3.36664575e-04, 1.87429098e-01],
       [2.27849052e-02, 3.43470611e-02, 9.42868034e-01],
       [1.87906330e-01, 5.42612080e-03, 8.06667549e-01],
       [2.66802059e-16, 9.99999846e-01, 1.53895689e-07],
       [9.74290541e-01, 6.14636516e-05, 2.56479954e-02]])
```

So, just to focus on the probability of assignment to the "most likely" cluster, we can take the maximum probability over the rows of the strengths matrix:

```
strength = strengths.max(axis=1)
```

```
strength[0:5]
```

```
array([0.81223424, 0.94286803, 0.80666755, 0.99999985, 0.97429054])
```

Introduction to Spatial Clustering

Probabilistic Clustering

Sometimes, we'd rather treat clustering assignments as "predictions" in a probabilistic framework. Often, this is done either through latent dirichlet allocation or, more simply, through Gaussian mixture models. scikit-learn provides gaussian mixture methods which are also fast and simple to use.

```
import sklearn.mixture as skmix
gmm = skmix.GaussianMixture(n_components=3, random_state=7052018).fit(neighborhood[['median_price']])
predicted = gmm.predict(neighborhood[['median_price']])
```

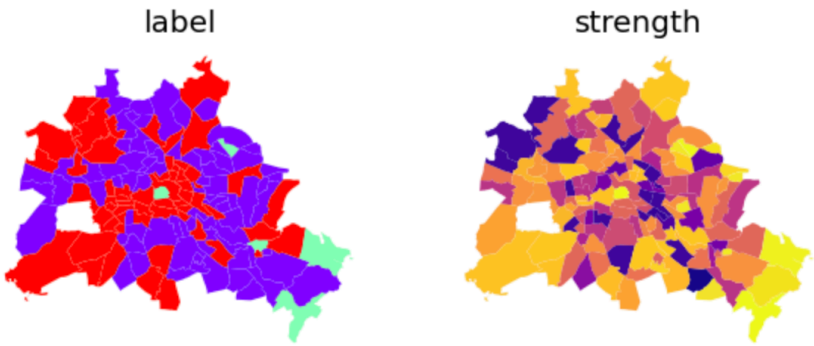
```
strengths = gmm.predict_proba(neighborhood[['median_price']])
```

This gives the full matrix of probabilities (over all clusters).

```
strengths[0:5]
```

```
array([[8.12234237e-01, 3.36664575e-04, 1.87429098e-01],
       [2.27849052e-02, 3.43470611e-02, 9.42868034e-01],
       [1.87906330e-01, 5.42612080e-03, 8.06667549e-01],
       [2.66802059e-16, 9.99999846e-01, 1.53895689e-07],
       [9.74290541e-01, 6.14636516e-05, 2.56479954e-02]])
```

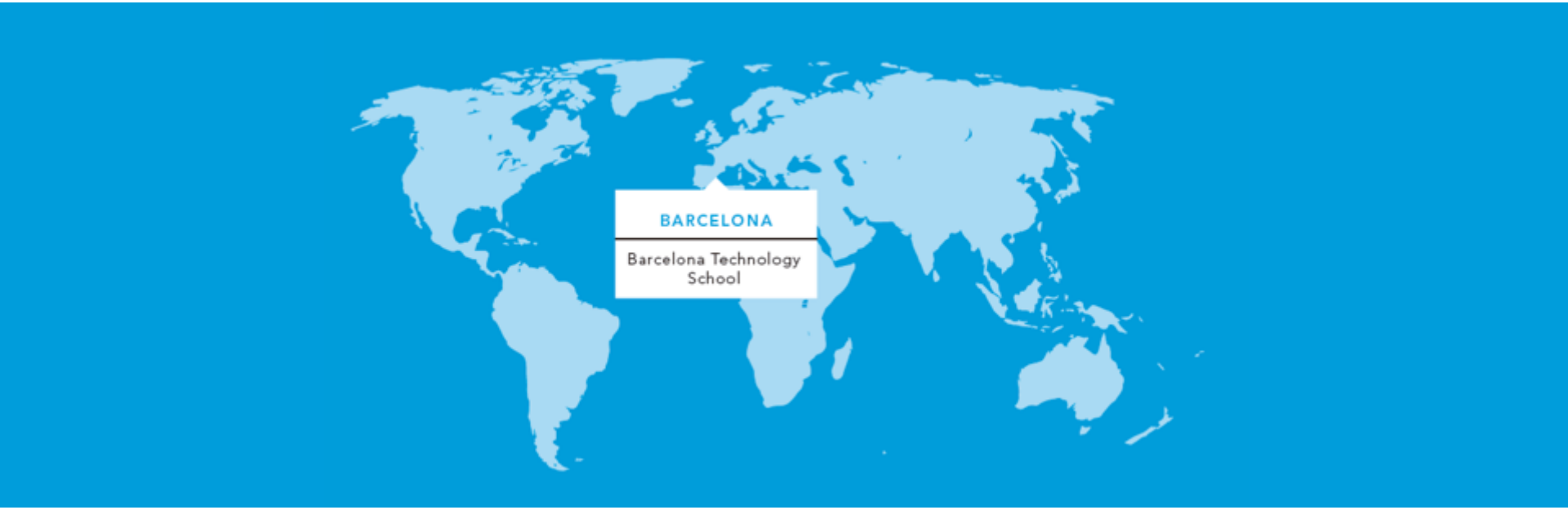
```
f,ax = plt.subplots(1,2,figsize=(10,4))
neighborhood.assign(labels=predicted).plot('labels',ax=ax[0],cmap='rainbow')
neighborhood.assign(strength=strength).plot('strength', cmap='plasma', ax=ax[1], vmin=.5, vmax=1)
for i in range(2):
    ax[i].axis('off')
    ax[i].set_title(["label","strength"][i], fontsize=20)
```



Thank you!!

It was a pleasure to have you in the class

Questions??


BARCELONA
Barcelona Technology School