Spatial Statistics and Geospatial Data Visualization

By: Amr Mohamed

Table of contents:

Importing Packages

In [ ]:

```
#import the required libraries
!pip install srtm.py
!pip install geopandas;
!pip install pyshp;
!pip install rasterio;
!pip install keplergl
!pip install movingpandas
!pip install basemap
```

In [8]:

```
from descartes import PolygonPatch
import shapefile as shp
import matplotlib.pyplot as plt
import geopandas as gpd
import fiona
import numpy as np
import rasterio as rio
from rasterio.plot import show
import pandas as pd
import fiona.transform
import rasterio.sample
from scipy.spatial.distance import directed_hausdorff
from sklearn.cluster import DBSCAN
import movingpandas as mpd
from datetime import datetime, timedelta
import plotly.express as px
from mpl_toolkits.basemap import Basemap
from mpl_toolkits.mplot3d import Axes3D
import pylab
import math
import logging as mod_logging
import srtm
import srtm as mod_srtm
import urllib
import zipfile
import os
import scipy.io
import seaborn as sns
import matplotlib as mpl
import branca.colormap as cm
import folium
from scipy.stats import skewnorm
from folium import plugins
import geemap
%matplotlib inline
from pyvista import set_plot_theme
# Importing auxiliary libraries
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image
set_plot_theme('document')
mpl.style.use('default')
%matplotlib inline
import warnings
warnings.simplefilter("ignore")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```r
# rayshader
library(rayshader)
library(tiff)
library(raster)
library(geoviz)

# chicago
options(warn=-1)
library(ggplot2)
library(tidyverse)
library(rgeos)
library(RColorBrewer)
library(maptools)
library(plotly)
library(scatterplot3d)
library(tidyverse)
library(RNHANES)
library(latticeExtra)

# spatial statistics
options(warn=-1)
library(spatstat)
library(raster)
```

In [2]:
```
%env HV_DOC_HTML=true
```

```
env: HV_DOC_HTML=true
```

In [ ]:
```
%load_ext rpy2.ipython
```

1: Key concepts in Geospatial data

1.1: Raster and vector data

Explain in your notebook the differences between raster and polygon data:

Polygons are a type of vector data that consist of a series of connected vertices that form a closed shape. They are often used to represent areas such as countries, states, and neighborhoods.Vector data consists of points, lines, and polygons. Each feature is represented by one or more vertices, which are connected by lines to form the feature. On the other hand, Raster data is any pixelated data where each pixel is associated with a specific geographical

location. The value of a pixel can be continuous (e.g. elevation) or categorical (e.g. land use). Moreover, raster data is accompanied by spatial information that connects the data to a

Design a non-interactive plot of French school districts based on the dataset provided by the instructor. Each district should be represented with a different color.

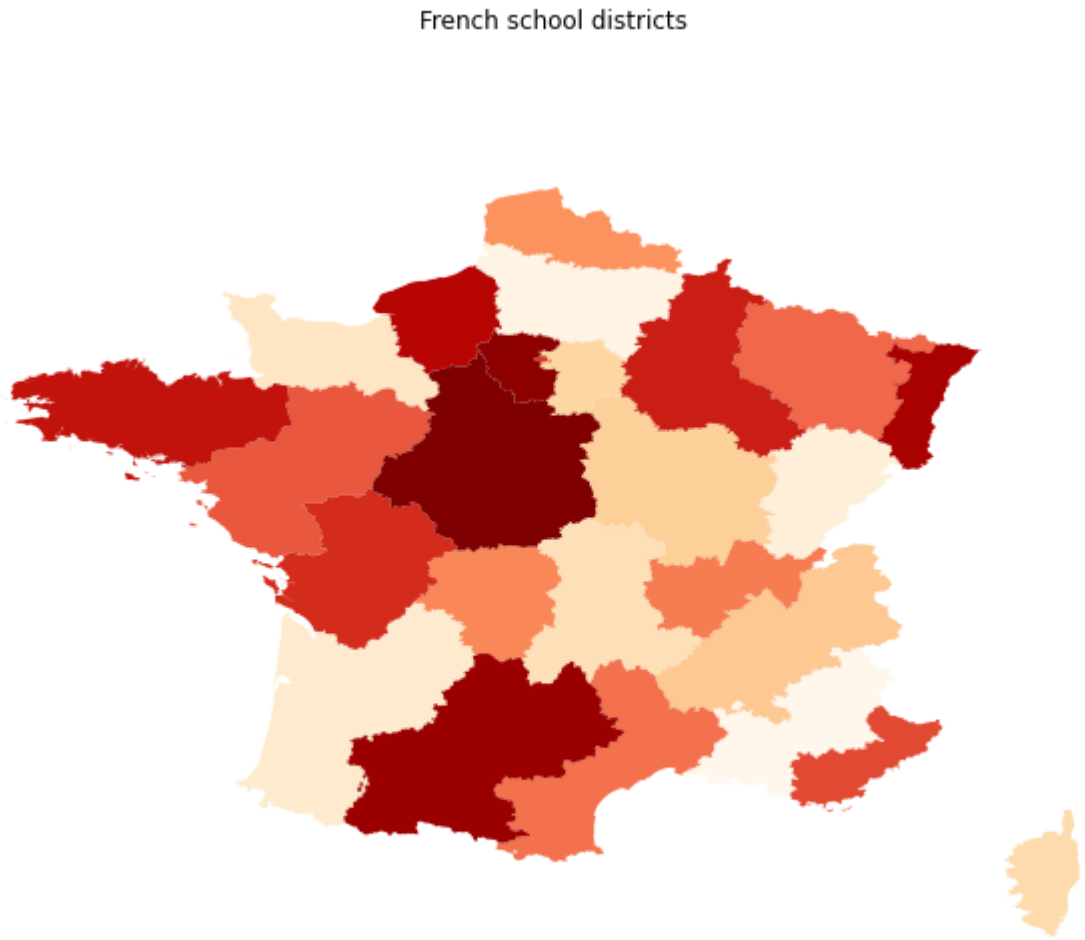
In [44]:
```
# reading the shapefile
fname = "//content//drive//MyDrive//GSP//academies-20160209-shp//academies
schools = gpd.read_file(fname)
schools.head()
```

Out[44]:

	name	vacances	wikipedia	geometry
0	Académie d'Aix-Marseille	Zone B	fr:Académie d'Aix-Marseille	MULTIPOLYGON (((4.23013 43.46039, 4.23025 43.4...
1	Académie d'Amiens	Zone B	fr:Académie d'Amiens (éducation)	MULTIPOLYGON (((1.38014 50.06499, 1.38214 50.0...
2	Académie de Besançon	Zone A	fr:Académie de Besançon	MULTIPOLYGON (((5.25202 46.94451, 5.25208 46.9...
3	Académie de Bordeaux	Zone A	fr:Académie de Bordeaux (éducation)	MULTIPOLYGON (((-1.79102 43.37292, -1.79048 43...
4	Académie de Caen	Zone B	fr:Académie de Caen (éducation)	MULTIPOLYGON (((-1.94877 49.71649, -1.94836 49...

In [45]:

```
fig, ax = plt.subplots(1, 1, figsize=(10,10))
schools.plot(cmap="OrRd",ax=ax)
plt.xlim(left=-5, right = 10)
plt.ylim(top=53, bottom = 40)
plt.title('French school districts')
plt.axis("off");
```

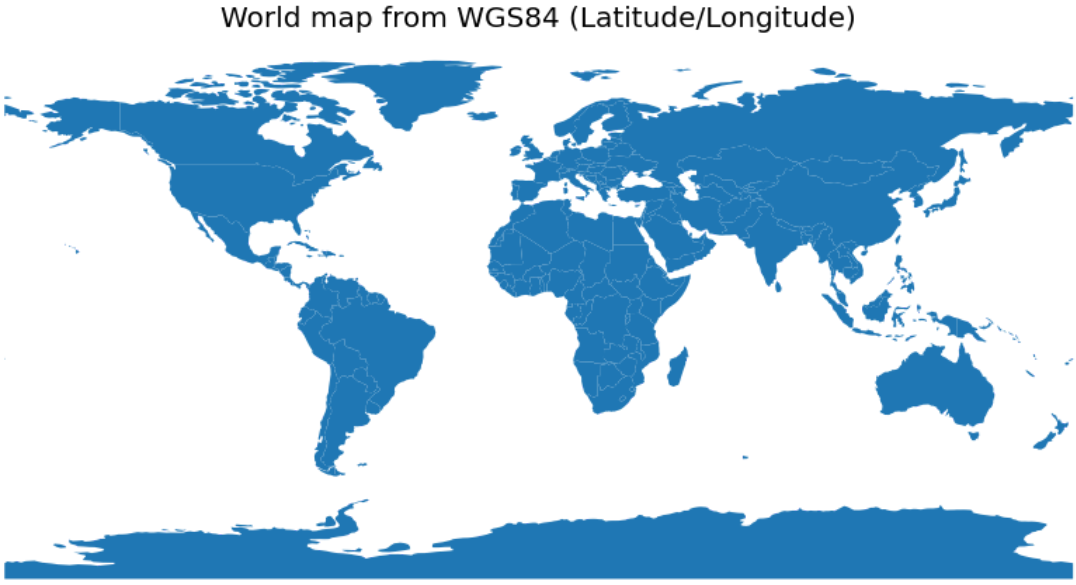


Find a shapefile representing the entire world, and design a set of maps based on different projections / coordinate reference systems (CRS) (ex.: Mercator, etc.).

In [23]:
```
# importing data from geopandas datasets representing the earth
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
cities = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
world.crs # original projection
```

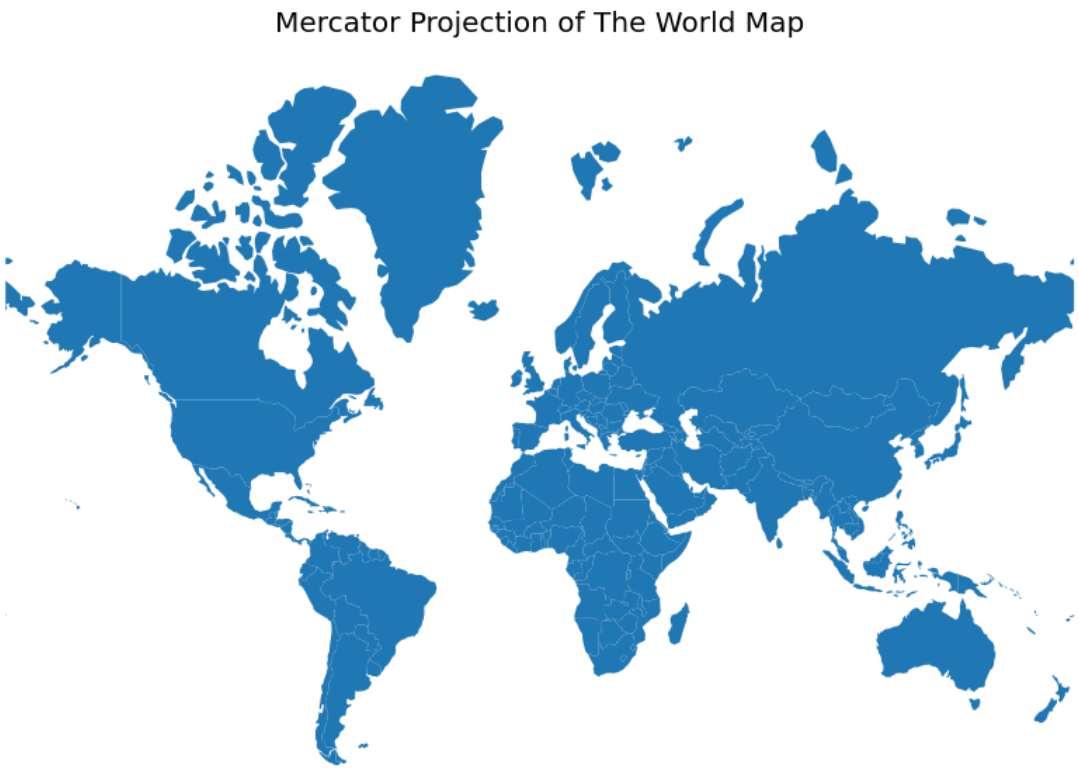
Out[23]:
```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

In [41]:
```
world.plot(figsize=(15, 10))
plt.title('World map from WGS84 (Latitude/Longitude)',fontdict={'fontsize'
plt.axis("off");
```



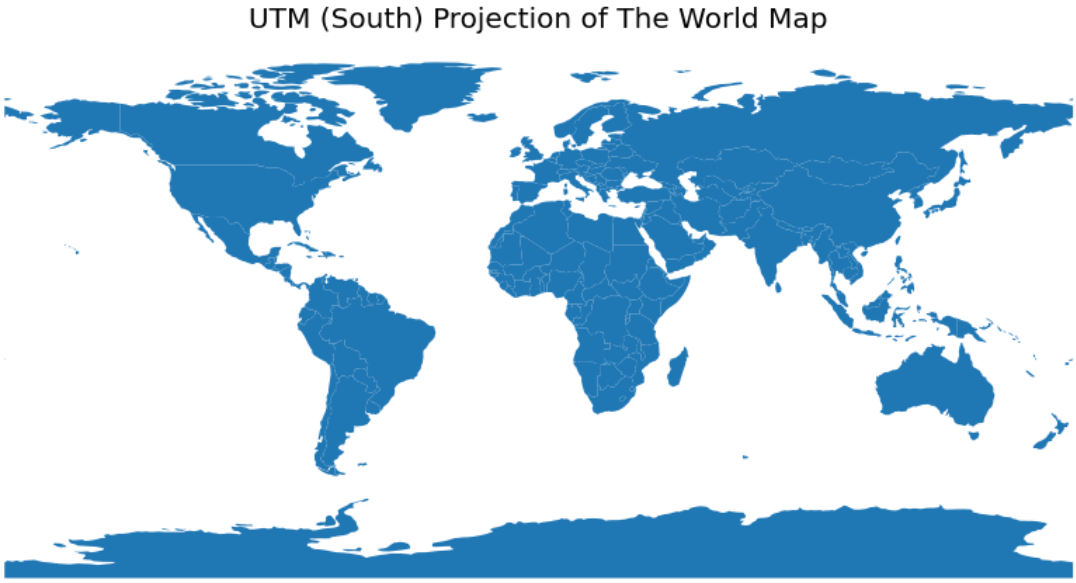
In [40]:

```
world_proj2 = world[(world.name != "Antarctica") & (world.name != "Fr. S.
world_proj2 = world_proj2.to_crs("EPSG:3395") # world.to_crs(epsg=3395) wo
mercator = world_proj2.plot(figsize=(15, 10));
mercator.set_title("Mercator Projection of The World Map",fontsize=20)
plt.axis("off");
```



In [39]:

```
#another testing of crs
#  check thos link for different crs: https://www.nceas.ucsb.edu/sites/def
world_test = world.set_crs(32733, allow_override=True)

world_test = world_test.plot(figsize=(15, 10));
world_test.set_title("UTM (South) Projection of The World Map",fontsize=20
plt.axis("off");
```



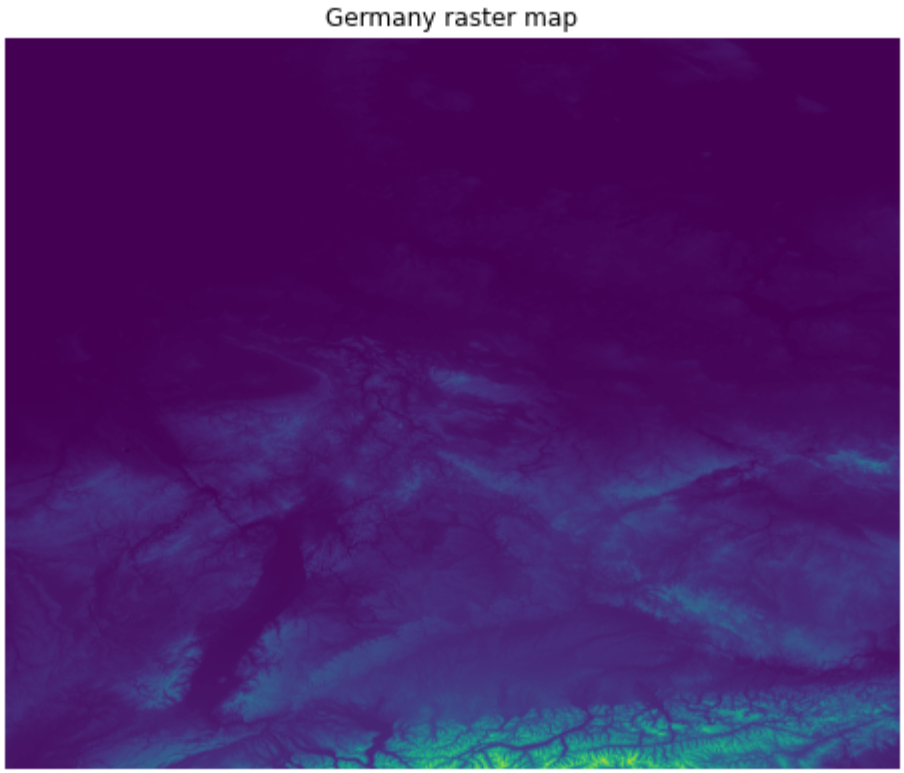
Write down a paragraph (in your notebook only) explaining the concept of CRS

Coordinate reference system (CRS) is a framework used to precisely measure locations on the surface of the Earth as coordinates. It provides a standardized way of describing locations. Many different CRS are used to describe geographic data. The CRS that is chosen depends on when the data was collected, the geographic extent of the data, the purpose of the data

Download a raster dataset from the SRTM website (elevation data from the entire world, +-30 meters in precision). Design a 2D map of the area you selected.

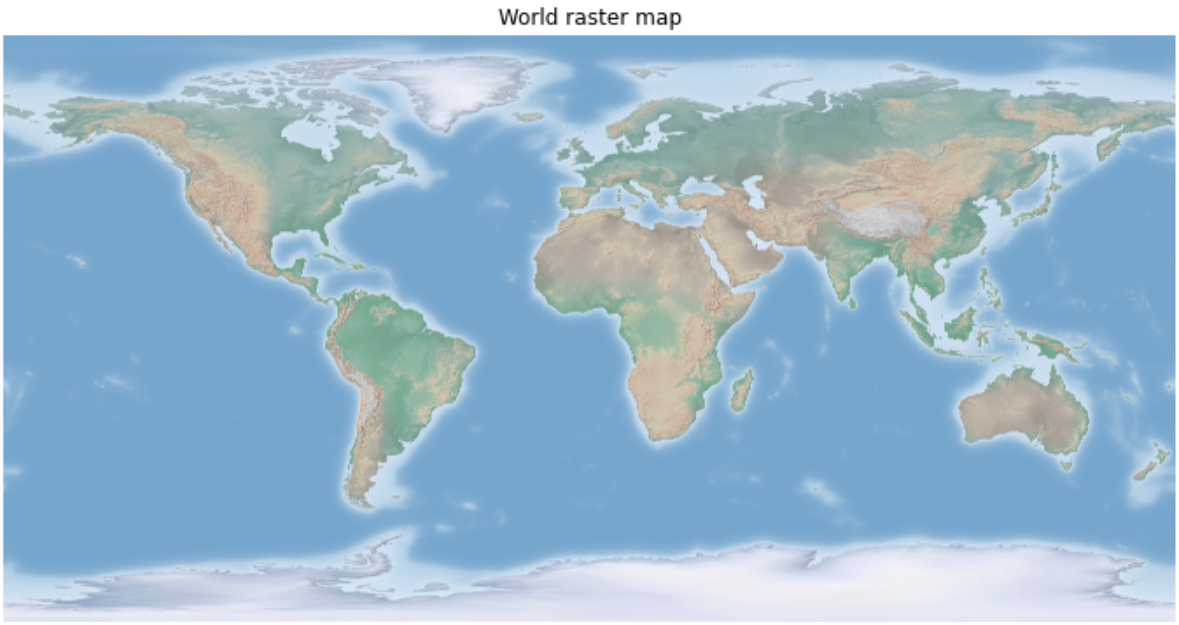
In [38]:

```
fpath = '//content//drive//MyDrive//GSP//srtm_germany_dsm//srtm_germany_ds

def rasterio_open(f):
    return rio.open(f)

src_image = rasterio_open(fpath)
fig, ax = plt.subplots(1, figsize=(8, 8))
show(src_image, ax=ax,cmap='viridis')
plt.title('Germany raster map')
plt.axis("off")
plt.show()
```



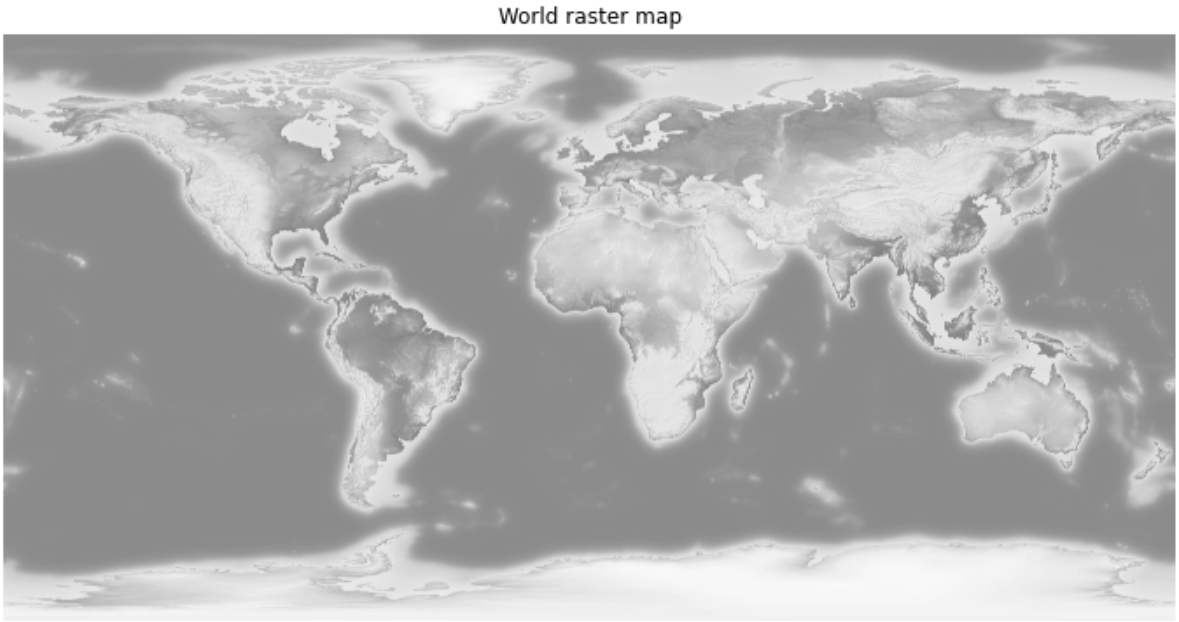
In [37]:

```
file = "//content//drive//MyDrive//GSP//HYP_LR_SR_W//HYP_LR_SR_W.tif"
src_image = rasterio_open(file)
fig, ax = plt.subplots(1, figsize=(12, 12))
show(src_image, ax=ax,cmap='viridis')
plt.title('World raster map')
plt.axis("off")
plt.show()
```



In [36]:

```
dem = rio.open(file)
dem_array = dem.read(1).astype('float64')

dem_array = dem.read(1).astype('float64')
fig, ax = plt.subplots(1, figsize=(12, 12))
show(dem_array, cmap='Greys_r', ax=ax)
plt.title('World raster map')
plt.axis("off")
plt.show()
```



1.2 Digital Elevation Models

In [13]:
```
elevation_data = srtm.get_data(local_cache_dir="//content//drive//MyDrive/
print('CGN Airport elevation (meters):', elevation_data.get_elevation(50.8
```

```
CGN Airport elevation (meters): 74
```

In [16]:
```
image = elevation_data.get_image((500, 500), (45, 46), (13, 14), 300)
```

In [17]:
```
mod_logging.basicConfig(level=mod_logging.DEBUG,
                        format='%(asctime)s %(name)-12s %(levelname)-8s %(

geo_elevation_data = srtm.get_data()

```

```
Creating /root/.cache/srtm
```

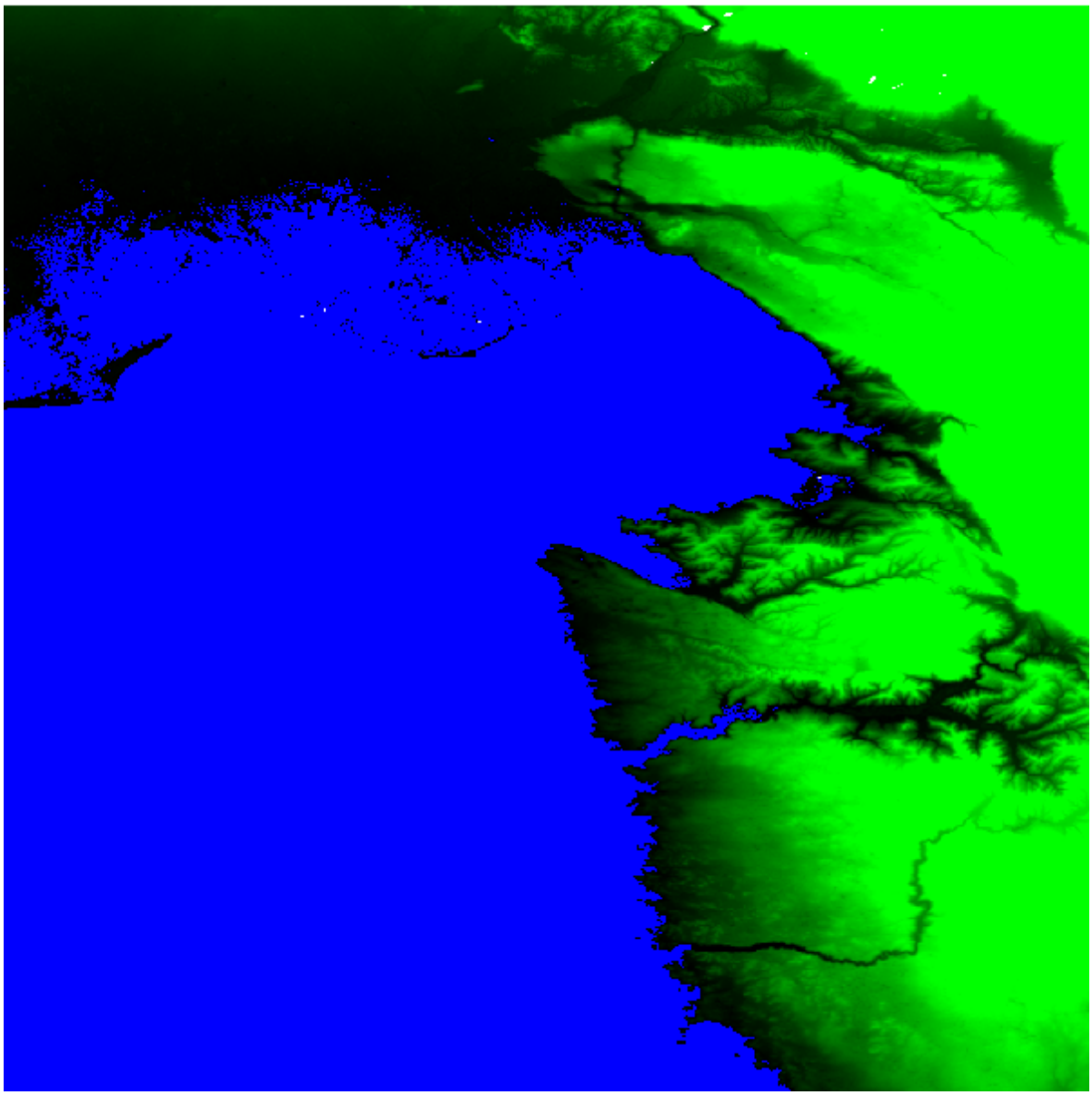
In [18]:
```
image = geo_elevation_data.get_image((400, 400), (45, 46), (13, 14), 300)
image.show()
```

```
4 2884802
```

In [21]:
```
print('                Digital Elevation Model of Istra and Trieste')
geo_elevation_data.get_image((600, 600), (45, 46), (13, 14), 300)
```

```
                Digital Elevation Model of Istra and Trieste
```

Out[21]:

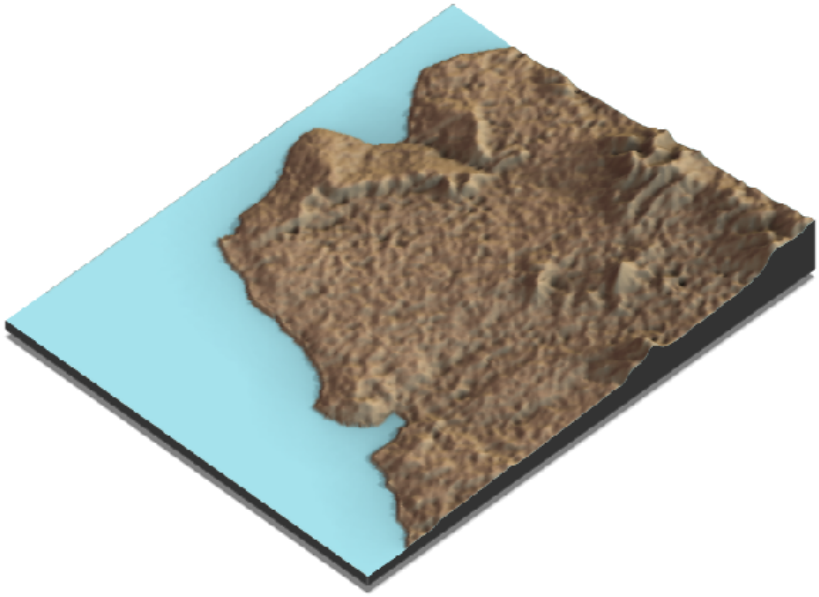


1.2 Digital Elevation Model (Rayshader)

In [2]:
```
# load file
str_name<-'H:/CY Tech Year 4/Semester 1/Geospatial/Datasets/finalmap.tif'
imported_raster=raster(str_name)
elmat = raster_to_matrix(imported_raster)
```

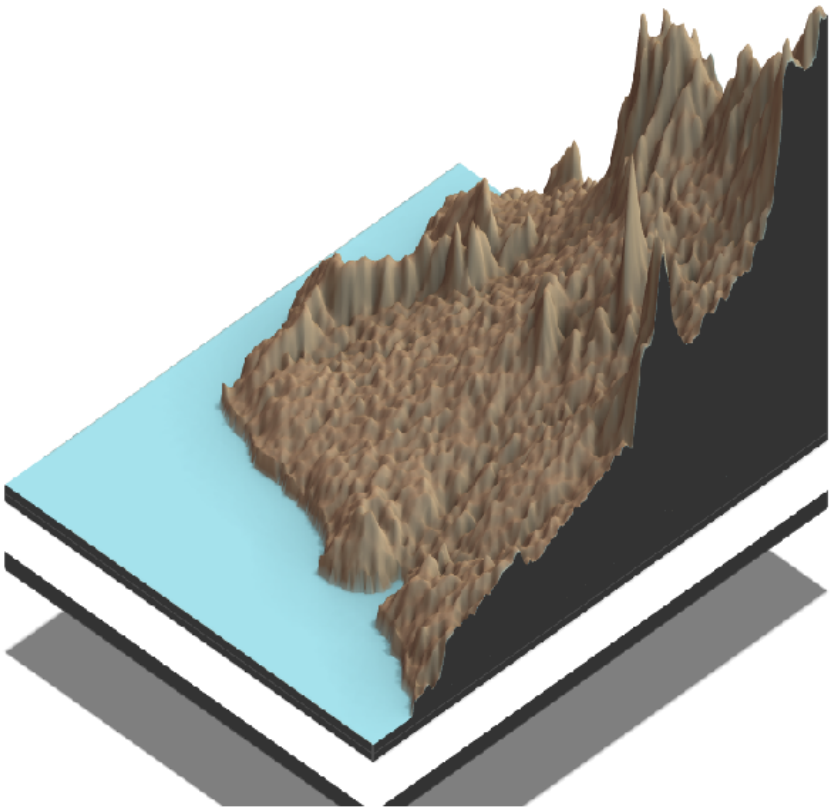
In [5]:

```
# create plot
elmat %>%
  sphere_shade(texture = "desert") %>% #add shade
  add_water(detect_water(elmat), color = "desert") %>% # add water
  add_shadow(ray_shade(elmat, zscale = 3), 0.5) %>% #add shadow
  add_shadow(ambient_shade(elmat), 0) %>% #add shadow
  plot_3d(elmat, zscale = 10, fov = 0, theta = 135, zoom = 1, phi = 45, wi
Sys.sleep(0.2)
render_snapshot()
```



In [6]:

```
elmat %>%
  sphere_shade(texture = "desert") %>%
  add_water(detect_water(elmat), color = "desert") %>%
  add_shadow(ray_shade(elmat, zscale = 3), 0.5) %>%
  add_shadow(ambient_shade(elmat), 0) %>%
  plot_3d(elmat,
            solid = T,
            water = T,
            waterdepth = 0,
            wateralpha = 0.5,
            watercolor = "lightblue",
            waterlinecolor = "white",
            waterlinealpha = 0.5,
            fov=0,theta=135,zoom=0.75,phi=45, windowsize = c(1000,800))
Sys.sleep(0.2)
render_snapshot()
```



2: Interactive data visualization

2.1: Interactive data visualization with Folium

In [23]:
```
fname = './../Datasets/academies-20160209-shp'
french_schools = gpd.read_file(fname)
french_schools.head()
```

Out[23]:

	name	vacances	wikipedia	geometry
0	Académie d'Aix-Marseille	Zone B	fr:Académie d'Aix-Marseille	MULTIPOLYGON (((4.23013 43.46039, 4.23025 43.4...
1	Académie d'Amiens	Zone B	fr:Académie d'Amiens (éducation)	MULTIPOLYGON (((1.38014 50.06499, 1.38214 50.0...
2	Académie de Besançon	Zone A	fr:Académie de Besançon	MULTIPOLYGON (((5.25202 46.94451, 5.25208 46.9...
3	Académie de Bordeaux	Zone A	fr:Académie de Bordeaux (éducation)	MULTIPOLYGON (((-1.79102 43.37292, -1.79048 43...
4	Académie de Caen	Zone B	fr:Académie de Caen (éducation)	MULTIPOLYGON (((-1.94877 49.71649, -1.94836 49...

In [24]:
```
rng = np.random.default_rng()
french_schools["perc. exam success"] = rng.integers(low = 0, high = 100, s
```

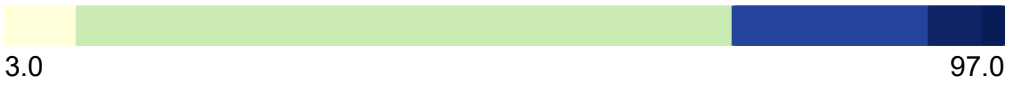
In [25]:
```
x_map=french_schools.centroid.x.mean()
y_map=french_schools.centroid.y.mean()
print(x_map,y_map)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_12768\2399918202.py:1: Us
erWarning: Geometry is in a geographic CRS. Results from 'centroid' are likel
y incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  x_map=french_schools.centroid.x.mean()

-1.3368242488745592 41.07040499085242

C:\Users\Administrator\AppData\Local\Temp\ipykernel_12768\2399918202.py:2: Us
erWarning: Geometry is in a geographic CRS. Results from 'centroid' are likel
y incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  y_map=french_schools.centroid.y.mean()
```

In [26]:

```
#checking color scheme
colormap = cm.linear.YlGnBu_09.to_step(data=french_schools['perc. exam suc
colormap
```

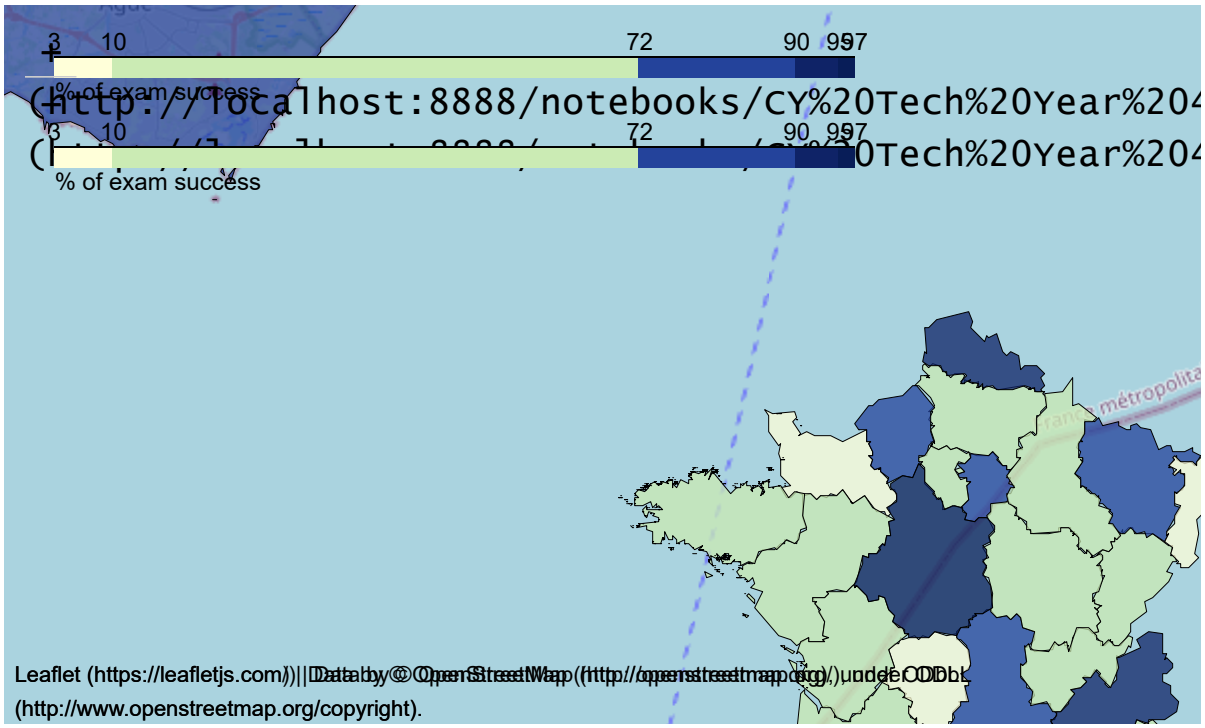
Out[26]:



In [63]:

```
map_ = folium.Map(location=[47, 3], zoom_start=5,tiles=None)
folium.TileLayer('OpenStreetMap',name="Light Map",control=False).add_to(ma
colormap.caption = "% of exam success"
style_function = lambda x: {"weight":0.5,
                            'color':'black',
                            'fillColor':colormap(x['properties']['perc. ex
                            'fillOpacity':0.75}
highlight_function = lambda x: {'fillColor': '#000000',
                                'color':'#000000',
                                'fillOpacity': 0.50,
                                'weight': 0.1}
FRENCH_SCHOOLS=folium.features.GeoJson(
        french_schools,
        style_function=style_function,
        control=False,
        highlight_function=highlight_function,
        tooltip=folium.features.GeoJsonTooltip(fields=['name','perc. exam
            aliases=['Name','% of exam success'],
            style=("background-color: white; color: #333333; font-family:
            sticky=True,zoom_start=0.1
        )
    )
colormap.add_to(map_)
map_.add_child(FRENCH_SCHOOLS)
map_
```

Out[63]:



In [64]:
```
phd = pd.read_csv("./../Datasets/PhD.v3.csv")
phd.head()
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_12768\4257234760.py:1: DtypeWarning: Columns (11) have mixed types. Specify dtype option on import or set low_memory=False.
  phd = pd.read_csv("./../Datasets/PhD.v3.csv")
```

Out[64]:

	Unnamed: 0	Auteur	Identifiant auteur	Titre	Directeur de these	Directeur de these (nom prenom)
0	0	Saeed Al marri	NaN	Le credit documentaire et l'onopposabilite des...	Philippe Delebecque	Delebecque Philippe
1	1	Andrea Ramazzotti	174423705	Application de la PGD a la resolution de probl...	Jean-Claude Grandidier,Marianne Beringhier	Grandidier Jean-Claude,Beringhier Marianne
2	2	OLIVIER BODENREIDER	NaN	Conception d'un outil informatique d'etude des...	Francois Kohler	Kohler Francois
3	3	Emmanuel Porte	NaN	Socio-histoire des politiques publiques en mat...	Gilles Pollet	Pollet Gilles
4	4	Arthur Devriendt	NaN	LES TECHNOLOGIES DE L'INFORMATION ET DE LA COM...	Gabriel Dupuy	Dupuy Gabriel

5 rows × 23 columns

In [65]:
```
phd_count = phd.groupby(["etablissement_rec"])["etablissement_rec"].count(
phd_count.head()
```

Out[65]:

	etablissement_rec	count
0	AgroParisTech	1218
1	Agrocampus Ouest	531
2	Aix-Marseille Université	20873
3	Arts et Métiers Sciences et Technologies	857
4	Avignon Université	1080

In [66]:
```
opendata = pd.read_csv("./../Datasets/opendata.csv",sep=";")[['uai - ident
opendata.head()
```

Out[66]:

	uai - identifiant	Libellé	Géolocalisation
0	0751875F	École nationale supérieure d'architecture de P...	48.89325,2.38192
1	0694123G	École normale supérieure de Lyon	45.7337,4.83376
2	0951214D	École supérieure des sciences économiques et c...	49.03397,2.078427
3	0912408Y	Université Paris-Saclay	48.70997,2.1531
4	0597139P	Centrale Lille Institut	50.60648,3.13757

In [67]:
```
opendata_phd_count = pd.merge(left=opendata, right=phd_count, left_on='Lib
opendata_phd_count.head()
```

Out[67]:

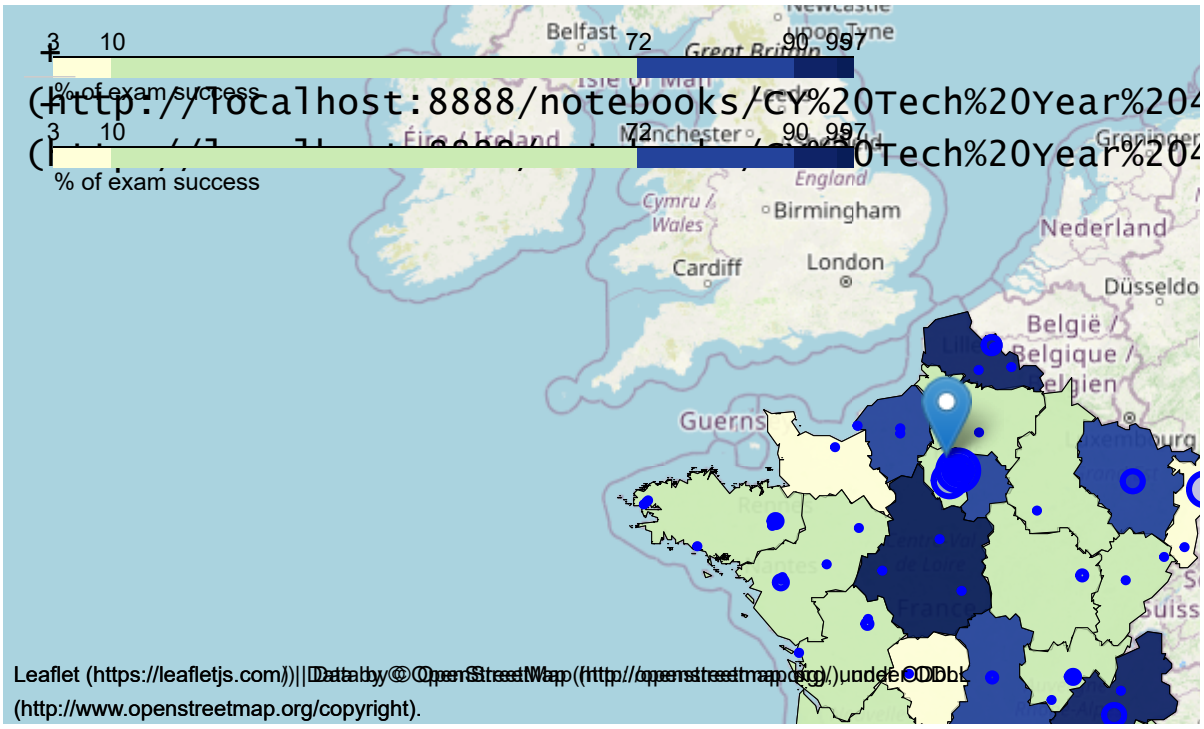
	uai - identifiant	Libellé	Géolocalisation	etablissement_rec	count
0	0694123G	École normale supérieure de Lyon	45.7337,4.83376	École normale supérieure de Lyon	941
1	0951214D	École supérieure des sciences économiques et c...	49.03397,2.078427	École supérieure des sciences économiques et c...	44
2	0912408Y	Université Paris-Saclay	48.70997,2.1531	Université Paris-Saclay	24684
3	0597139P	Centrale Lille Institut	50.60648,3.13757	Centrale Lille Institut	384
4	0331766R	Université Bordeaux Montaigne	44.8015,-0.615058	Université Bordeaux Montaigne	2714

In [68]:
```
CYlocation = list(map(float,str(opendata_phd_count[opendata_phd_count.Libe
```

In [69]:

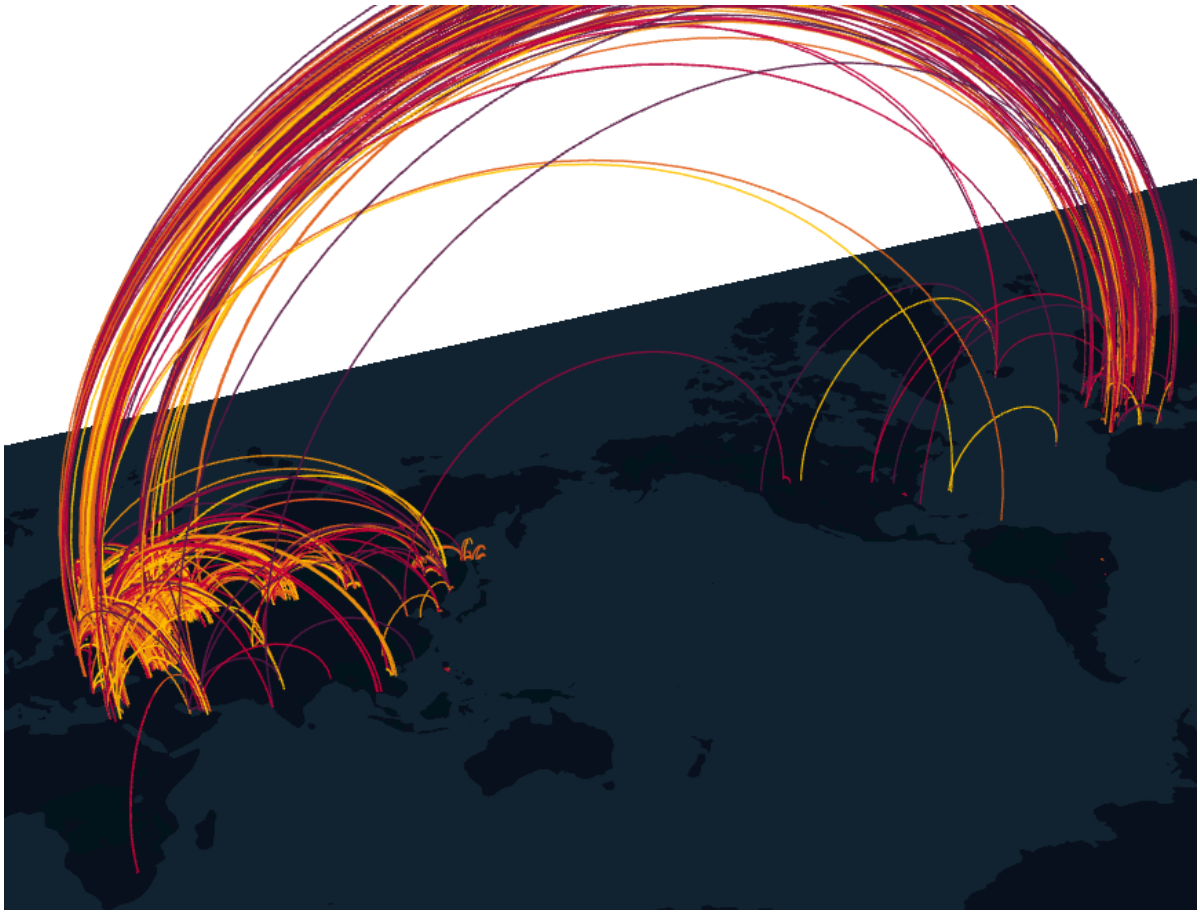
```
#adding a marker on CY Cergy Paris Université location
folium.Marker(CYlocation, popup='CY Tech').add_to(map_)
#adding bubbles to the map based on the count of phds per institution
for i in range(0,len(opendata_phd_count)):
   coordinates = opendata_phd_count["Géolocalisation"][i]
   folium.Circle(
      location=coordinates.split(','),
      popup=opendata_phd_count.iloc[i]['Libellé'],
      radius=float(opendata_phd_count.iloc[i]['count']),
      color='blue',
      fill=True,
      fill_color='blue'
   ).add_to(map_)
map_
```

Out[69]:



2.2: Interactive mapping using Kepler

not: maps are sceeenshots from the html generated kepler map



2.3 DBSCAN and its application to trajectories clustering

In [46]:

```
def kMedoids(D, k, tmax=100):
    # determine dimensions of distance matrix D
    m, n = D.shape

    np.fill_diagonal(D, math.inf)

    if k > n:
        raise Exception('too many medoids')
    # randomly initialize an array of k medoid indices
    M = np.arange(n)
    np.random.shuffle(M)
    M = np.sort(M[:k])

    # create a copy of the array of medoid indices
    Mnew = np.copy(M)

    # initialize a dictionary to represent clusters
    C = {}
    for t in range(tmax):
        # determine clusters, i. e. arrays of data indices
        J = np.argmin(D[:,M], axis=1)

        for kappa in range(k):
            C[kappa] = np.where(J==kappa)[0]
        # update cluster medoids
        for kappa in range(k):
            J = np.mean(D[np.ix_(C[kappa],C[kappa])],axis=1)
            j = np.argmin(J)
            Mnew[kappa] = C[kappa][j]
        np.sort(Mnew)
        # check for convergence
        if np.array_equal(M, Mnew):
            break
        M = np.copy(Mnew)
    else:
        # final update of cluster memberships
        J = np.argmin(D[:,M], axis=1)
        for kappa in range(k):
            C[kappa] = np.where(J==kappa)[0]

    np.fill_diagonal(D, 0)

    # return results
    return M, C
```

In [47]:

```
# from kmedoid import kMedoids # kMedoids code is adapted from https://git

# Some visualization stuff, not so important
sns.set()
plt.rcParams['figure.figsize'] = (12, 12)
```

In [48]:

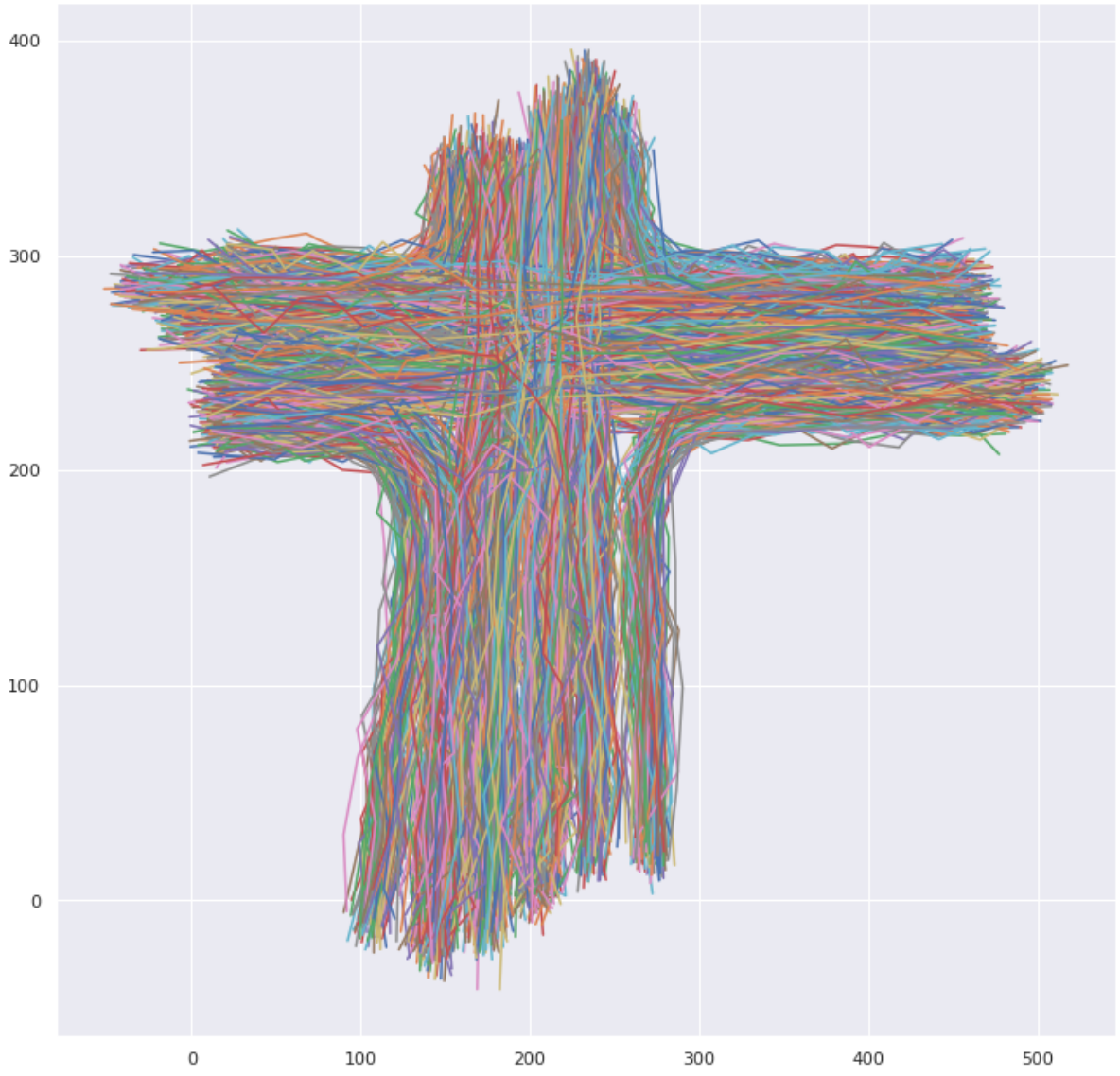
```
# Utility Functions

color_lst = plt.rcParams['axes.prop_cycle'].by_key()['color']
color_lst.extend(['firebrick', 'olive', 'indigo', 'khaki', 'teal', 'saddle
                  'skyblue', 'coral', 'darkorange', 'lime', 'darkorchid', '

def plot_cluster(traj_lst, cluster_lst):
    '''
    Plots given trajectories with a color that is specific for every traje
    Outlier trajectories which are specified with -1 in `cluster_lst` are
    '''
    cluster_count = np.max(cluster_lst) + 1

    for traj, cluster in zip(traj_lst, cluster_lst):

        if cluster == -1:
            # Means it it a noisy trajectory, paint it black
            plt.plot(traj[:, 0], traj[:, 1], c='k', linestyle='dashed')

        else:
            plt.plot(traj[:, 0], traj[:, 1], c=color_lst[cluster % len(col
    plt.show()
```

In [50]:

```
dataset_link = '//content//drive//MyDrive//GSP//CVRR_dataset_trajectory_cl
data_folder = 'data'
filename = '%s/cross.mat' % data_folder

# Import dataset
traj_data = scipy.io.loadmat('//content//drive//MyDrive//GSP//CVRR_dataset

traj_lst = []
for data_instance in traj_data:
    traj_lst.append(np.vstack(data_instance[0]).T)
```

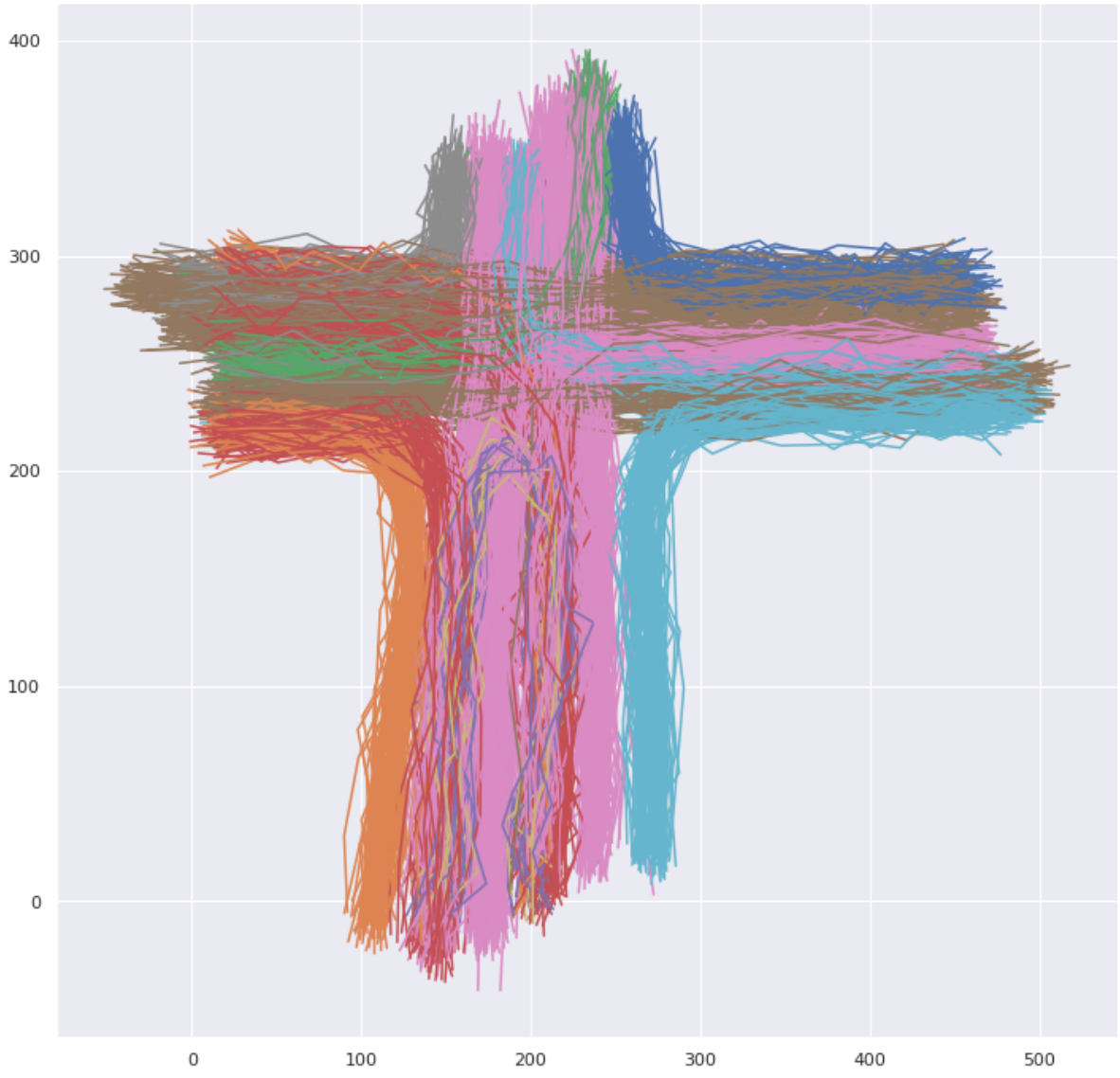
In [51]:

```
# Plotting

for traj in traj_lst:
    plt.plot(traj[:, 0], traj[:, 1])
```



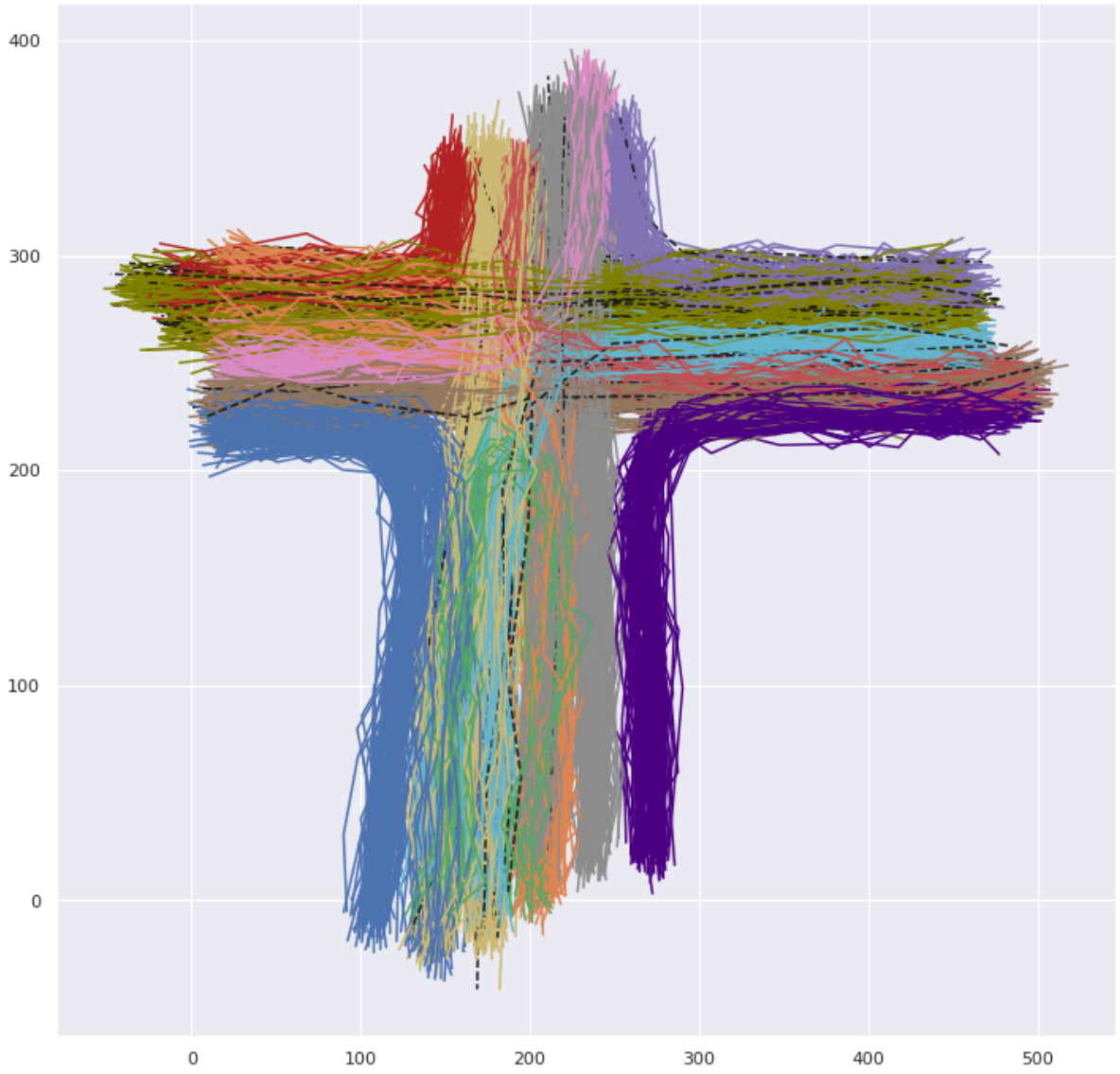
In [52]:

```
# 2 - Trajectory segmentation

degree_threshold = 5

for traj_index, traj in enumerate(traj_lst):

    hold_index_lst = []
    previous_azimuth= 1000

    for point_index, point in enumerate(traj[:-1]):
        next_point = traj[point_index + 1]
        diff_vector = next_point - point
        azimuth = (math.degrees(math.atan2(*diff_vector)) + 360) % 360

        if abs(azimuth - previous_azimuth) > degree_threshold:
            hold_index_lst.append(point_index)
            previous_azimuth = azimuth
    hold_index_lst.append(traj.shape[0] - 1) # Last point of trajectory is

    traj_lst[traj_index] = traj[hold_index_lst, :]
# 3 - Distance matrix

def hausdorff( u, v):
    d = max(directed_hausdorff(u, v)[0], directed_hausdorff(v, u)[0])
    return d

traj_count = len(traj_lst)
D = np.zeros((traj_count, traj_count))

# This may take a while
for i in range(traj_count):
    for j in range(i + 1, traj_count):
        distance = hausdorff(traj_lst[i], traj_lst[j])
        D[i, j] = distance
        D[j, i] = distance

k = 10 # The number of clusters
medoid_center_lst, cluster2index_lst = kMedoids(D, k)

cluster_lst = np.empty((traj_count,), dtype=int)

for cluster in cluster2index_lst:
    cluster_lst[cluster2index_lst[cluster]] = cluster

plot_cluster(traj_lst, cluster_lst)
```


400
300
200
100
0
0
100
200
300
400
500

In [53]:

```
mdl = DBSCAN(eps=400, min_samples=10)
cluster_lst = mdl.fit_predict(D)

plot_cluster(traj_lst, cluster_lst)
```



DBSCAN Based on a set of points, groups together the points that are close to each other based on a distance measurement and a minimum number of points. It also marks as outliers the points that are in low-density regions. It takes into account 2 main parameters: epsilon which specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value, these points are considered neighbors., and minPoints: the minimum number of points to form a dense region.

3: Spatial data visualization challenges

3.1 Space-time trajectory

In [54]:
```
# import data
df_spacetime = pd.read_csv('//content//drive//MyDrive//GSP//MPIAB White St
```

In [55]:
```
df_spacetime.head(2)
```

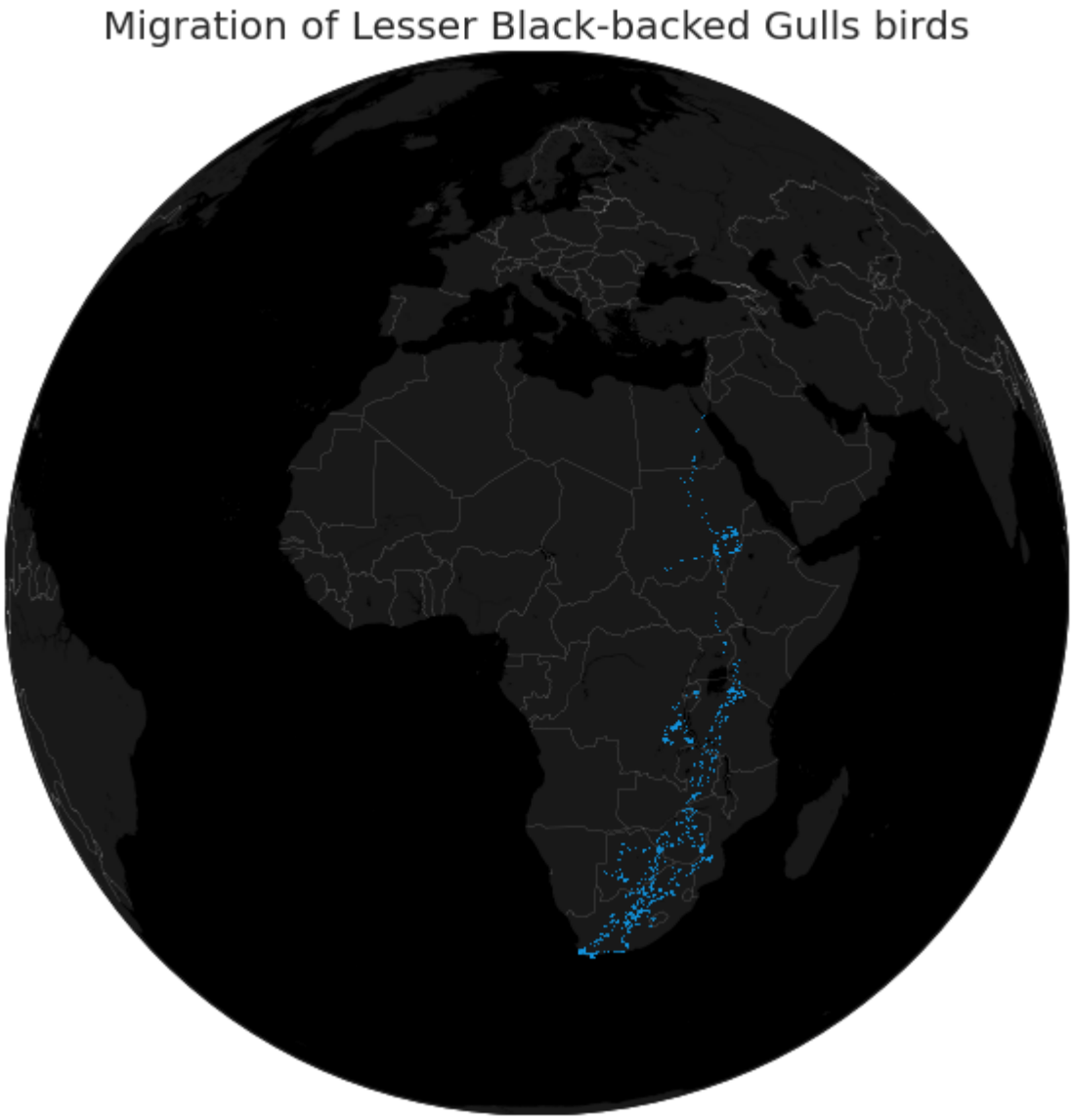
Out[55]:

	event-id	visible	timestamp	location-long	location-lat	algorithm-marked-outlier	argos:altitude	argos:best-level	arg
0	15390742	True	2001-11-24 14:07:30.000	18.507	-33.939	NaN	20.0	-134.0	4
1	15390745	True	2001-11-24 16:44:24.000	18.478	-33.963	NaN	20.0	-129.0	4

2 rows × 30 columns

In [65]:

```
#initialise the figure object
fig = plt.figure(figsize=(14,10))

#initialise the Map using Basemap. You can manually set the area of the wo
#be displayed. You can also get various variations of the maps.

m = Basemap(projection='ortho', lat_0=10, lon_0=13, resolution='i')

m.fillcontinents(color='#191919',lake_color='#000000') # dark grey land, b
m.drawmapboundary(fill_color='#000000')                # black background
m.drawcountries(linewidth=0.1, color="w")              # draw countries

#extract the coordinates into the variables x and y
x, y = m(df_spacetime['location-long'].values, df_spacetime['location-lat'


#plot the positions of the birds as they migrate using the scatter plot.
#Refer the documentation of Basemap for a better understanding of the vari
m.scatter(x, y, c="#1292db", lw=0, alpha=1, zorder=5, s = 1)

#title for the image
plt.title("Migration of Lesser Black-backed Gulls birds",fontsize=20)

#plot the image
plt.show()
```

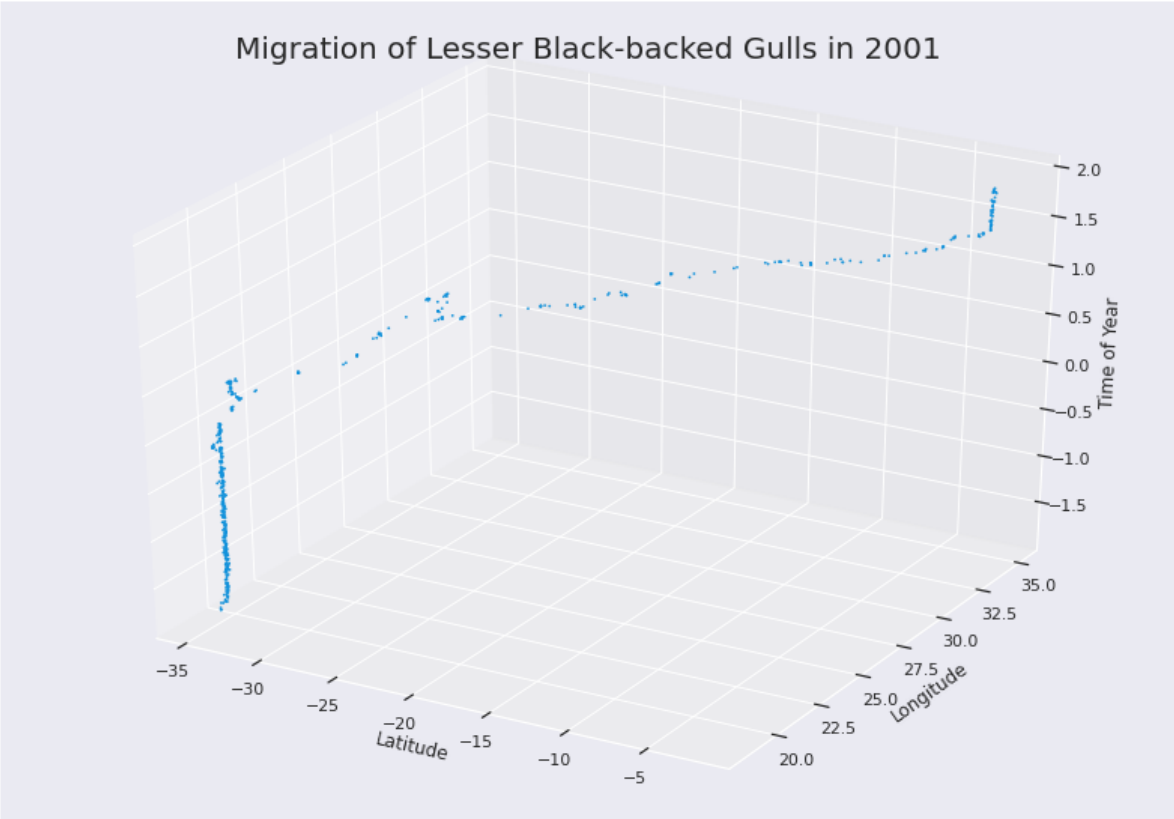


In [ ]:
```
1 set(df_spacetime['tag-local-identifier'])
```

Out[35]: {1979, 1983, 20523, 20525, 20526, 20527}

In [66]:

```
def plot_3d(df):
    #initialise a figure object
    fig = plt.figure(figsize=(14,10))

    #add a 3D object
    ax = fig.add_subplot(111, projection='3d')

    groups = df.groupby('tag-local-identifier')

    # Plot
    # fig, ax = plt.subplots()
    # ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling
    # for name, group in groups:
    #     ax.scatter(df['location-lat'], df['location-long'],df['event-id'
    # ax.legend()

    #scatter plot
    ax.scatter(df['location-lat'], df['location-long'],df['event-id'], c="



    ax.set_xlabel('Latitude')
    ax.set_ylabel('Longitude')
    ax.set_zlabel('Time of Year')
    ax.set_title("Migration of Lesser Black-backed Gulls in 2001",fontsize
    plt.show()

# d = pd.read_csv('migration_original.csv')
#91732 is the id of the bird whose trajectory we are interested in.
#feel free to change the id of the bird
k = df_spacetime.loc[df_spacetime['tag-local-identifier'] == 20527]

#normalise the time axis
#use the event-id to determine the time stamp
#Note: event-id is in increasing order between a range of numbers,
#so we can easily normalise for a given bird
#In the next post, I will plot using time-stamps
#and that will make things more clear

k['event-id'] = (k['event-id'] - k['event-id'].mean())/k['event-id'].std(d

plot_3d(k)

```


Migration of Lesser Black-backed Gulls in 2001
Latitude
−35
−30
−25
−20
−15
−10
−5
Longitude
20.0
22.5
25.0
27.5
30.0
32.5
35.0
Time of Year
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

3.2: Chicago criminality

In [74]:
```
# load file
chicago_Df <- read.csv('H:/CY Tech Year 4/Semester 1/Geospatial/Datasets/G
head(chicago_Df)
```

ID	Case.Number	Date	Block	IUCR	Primary.Type	Description	L
9446834	HX100098	01/01/2014 12:02:00 AM	072XX S MORGAN ST	1477	WEAPONS VIOLATION	RECKLESS FIREARM DISCHARGE	
9446765	HX100013	01/01/2014 12:03:00 AM	064XX S ROCKWELL ST	143A	WEAPONS VIOLATION	UNLAWFUL POSS OF HANDGUN	
9446921	HX100067	01/01/2014 12:04:00 AM	048XX S PRAIRIE AVE	1460	WEAPONS VIOLATION	POSS FIREARM/AMMO:NO FOID CARD	
9446783	HX100093	01/01/2014 12:05:00 AM	011XX W 50TH ST	143A	WEAPONS VIOLATION	UNLAWFUL POSS OF HANDGUN	LOT/GA
9446811	HX100017	01/01/2014 12:05:00 AM	031XX W WALNUT ST	143A	WEAPONS VIOLATION	UNLAWFUL POSS OF HANDGUN	
9446958	HX100026	01/01/2014 12:05:00 AM	005XX E 67TH ST	0550	ASSAULT	AGGRAVATED PO: HANDGUN	

In [75]:
```
chicago_Df$Date <- format(as.POSIXct(chicago_Df$Date,format='%m/%d/%Y %H:%
```

In [76]:
```
read.city_boundary <- function(file_name)
{
  city <- readShapePoly('./../Datasets/cityboundary.shp')
  city <- fortify(gSimplify(city, tol=100), region='OBJECTID')
  rename(city, c(x="long", y="lat"))
}
```

In [77]:
```
city <- read.city_boundary("H:/CY Tech Year 4/Semester 1/Geospatial/Datase
```

In [78]:
```
1 head(city)
```

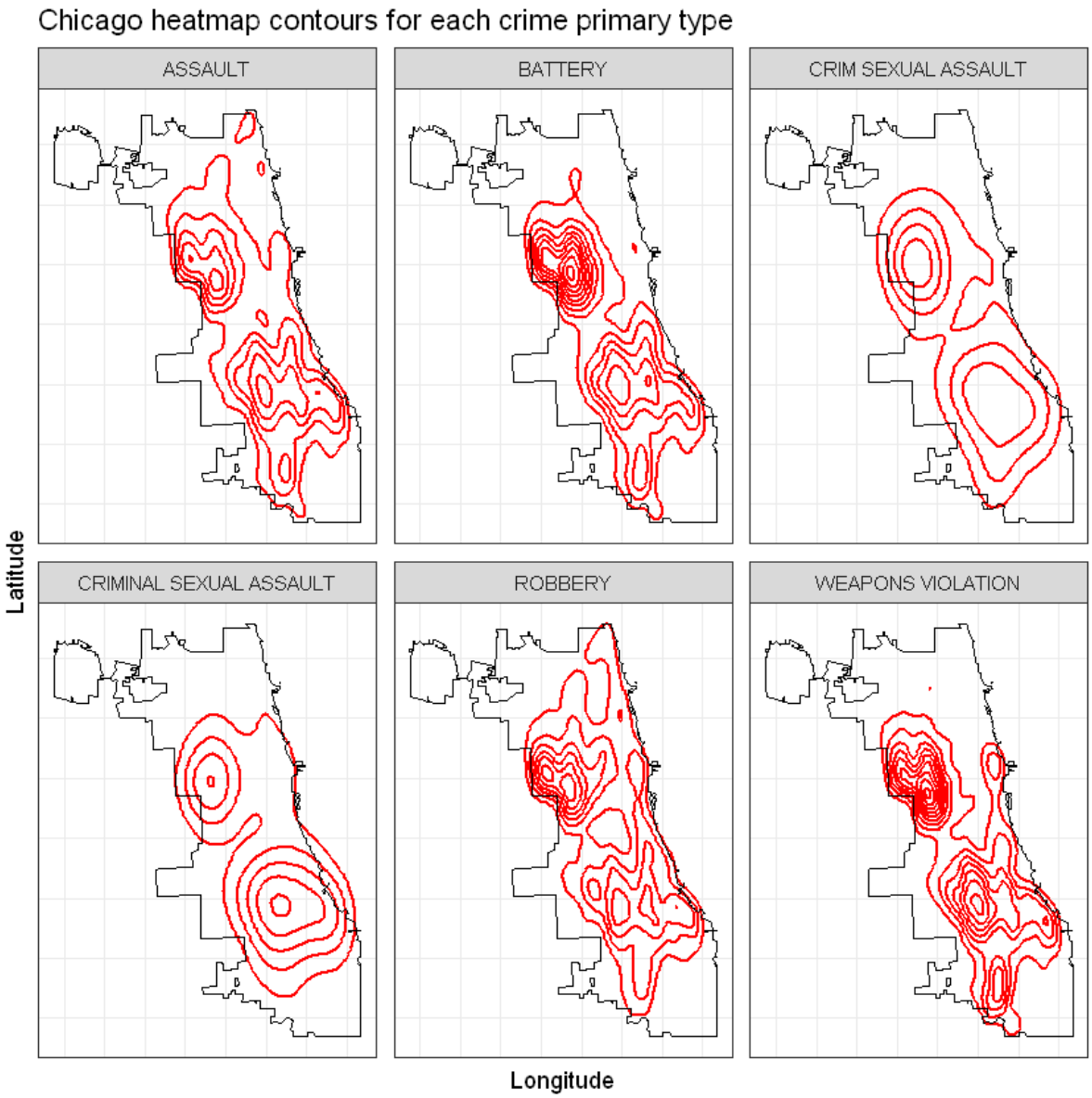
x	y	order	hole	piece	id	group
1092482	1943166	1	FALSE	1	0	0.1
1092430	1945812	2	FALSE	1	0	0.1
1092894	1945822	3	FALSE	1	0	0.1
1092925	1944820	4	FALSE	1	0	0.1
1093784	1944668	5	FALSE	1	0	0.1
1094076	1944670	6	FALSE	1	0	0.1

In [83]:
```
1 options(scipen=999)
```

In [88]:

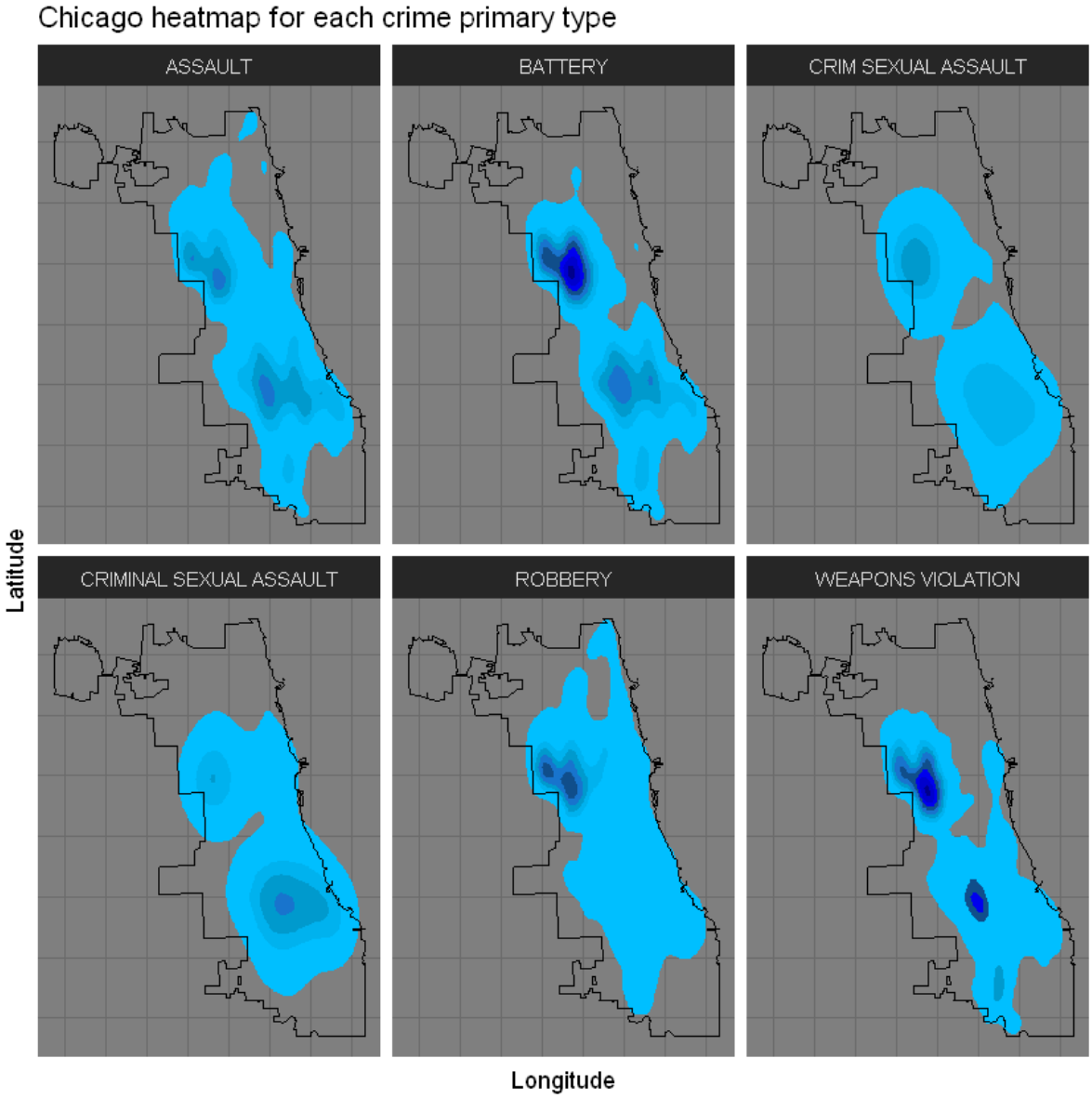
```
ggplot(chicago_Df, aes(x = Longitude, y = Latitude))+ggtitle("Plot of leng
geom_density2d(data = chicago_Df,aes(x = X.Coordinate, y = Y.Coordinate),
  geom_path(data=city, aes(x, y, group=group), size=.2, colour='black') +
  scale_alpha(range = c(0.05,0.5)) +
 scale_alpha_continuous(limits=c(0,0.2),breaks=seq(0,0.2,by=0.025))+
  facet_wrap(~Primary.Type, scales = "free") +
  labs(x = "Longitude", y = "Latitude",title = paste("", "", sep = "\n"),
  guides(colour = FALSE, alpha = FALSE) +
  theme_bw() + theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks = element_blank())+
  labs(title = "Chicago heatmap contours for each crime primary type")
```

```
Scale for 'alpha' is already present. Adding another scale for 'alpha', which
will replace the existing scale.
```



In [89]:

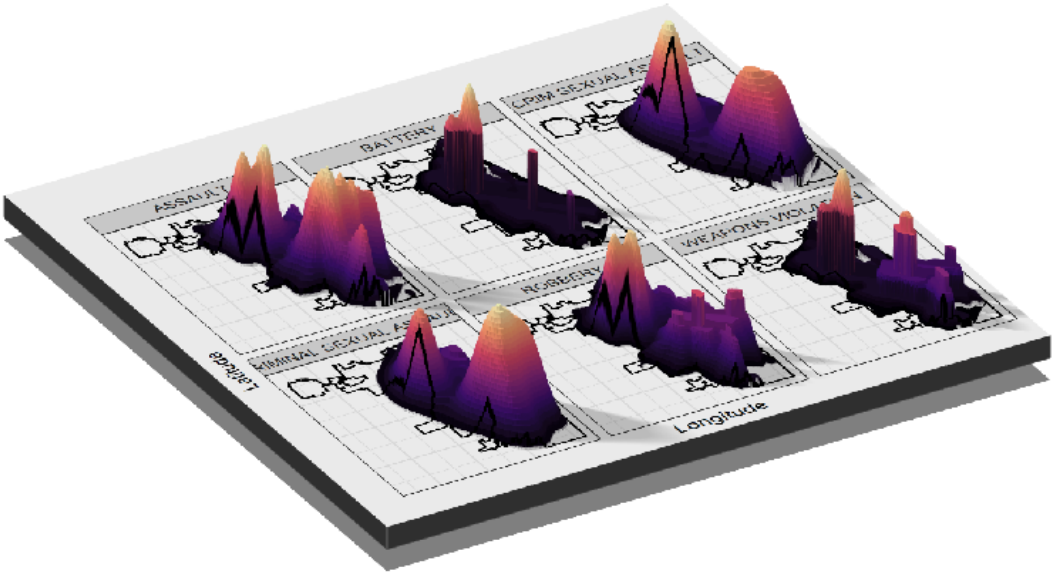
```
ggplot(chicago_Df, aes(x = Longitude, y = Latitude)) +
stat_density_2d(data = chicago_Df,aes(x = X.Coordinate, y = Y.Coordinate,f
# scale_alpha(range = c(0.05,0.5)) +
  scale_fill_manual(values = c("deepskyblue","deepskyblue1","deepskyblue2"
  geom_path(data=city, aes(x, y, group=group), size=.2, colour='black') +
#  scale_fill_gradientn(colours = rev(rainbow(7))+
  facet_wrap(~Primary.Type) +
  labs(x = "Longitude", y = "Latitude",title = paste("", "", sep = "\n"),
  theme_dark() +
  theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks = element_blank(),
  legend.position = "none")+
  labs(title = "Chicago heatmap for each crime primary type")
```



3D Facetgrid of Primary types heatmaps

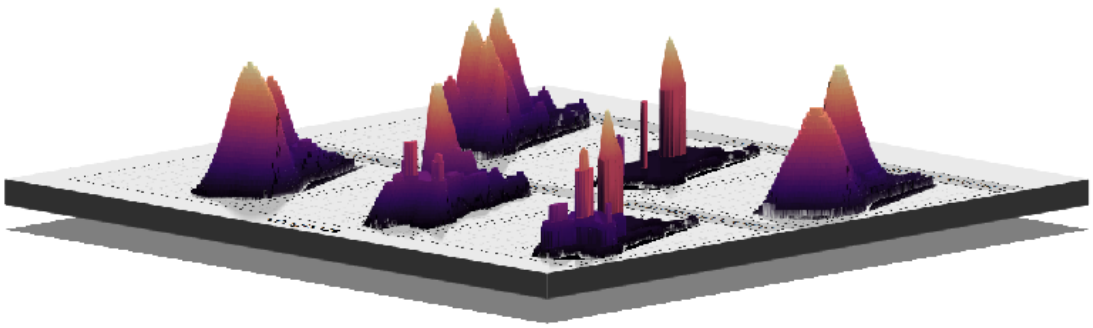
In [37]:

```
ggdiamonds  <- ggplot(chicago_Df, aes(x = Longitude, y = Latitude)) +
stat_density_2d(data = chicago_Df,geom='polygon' ,
                aes(x = X.Coordinate, y = Y.Coordinate,fill = stat(nlevel))) +
  scale_fill_viridis_c(option = "A")+
  geom_path(data=city, aes(x, y, group=group), size=.6,colour='black') +
  facet_wrap(~Primary.Type) +
  labs(x = "Longitude", y = "Latitude",title = paste("", "", sep = "\n"),
  theme_bw() +
  theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks = element_blank(),
  legend.position = "none")
plot_gg(ggdiamonds,multicore = TRUE,width=5,height=5,scale=250,windowsize=
       zoom = 0.55,theta=-30, phi = 30)
Sys.sleep(0.2)
render_snapshot()
```


BATTERY
Latitude
Longitude

In [24]:

```
ggdiamonds  <- ggplot(chicago_Df, aes(x = Longitude, y = Latitude)) +
stat_density_2d(data = chicago_Df,geom='polygon' ,
                aes(x = X.Coordinate, y = Y.Coordinate,fill = stat(nlevel))) +
  scale_fill_viridis_c(option = "A")+
  geom_path(data=city, aes(x, y, group=group), size=.2, colour='black') +
  facet_wrap(~Primary.Type) +
  labs(x = "Longitude", y = "Latitude",title = paste("", "", sep = "\n"),
  theme_bw() +
  theme(
  axis.text.x = element_blank(),
  axis.text.y = element_blank(),
  axis.ticks = element_blank(),
  legend.position = "none")
plot_gg(ggdiamonds,multicore = TRUE,width=5,height=5,scale=250,windowsize=
       zoom = 0.55, phi = 10)
Sys.sleep(0.2)
render_snapshot()
```



lolipop plot

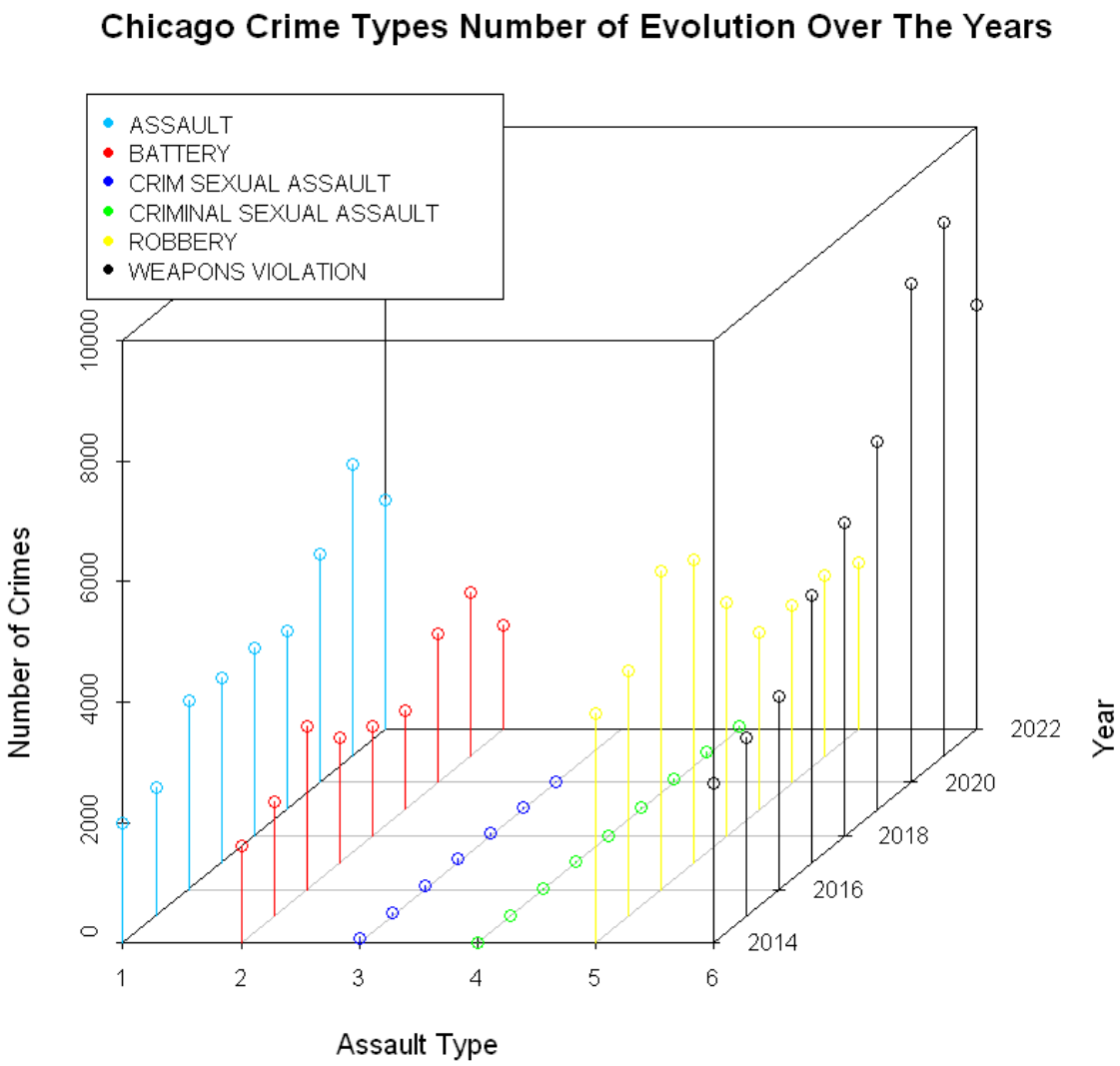
In [97]:
```
PT_date_count <- chicago_Df %>% select(Year,Primary.Type) %>% group_by(Yea
  summarise(n = n())
PT_date_count<-na.omit(PT_date_count)
head(PT_date_count)
```

```
`summarise()` has grouped output by 'Year'. You can override using the `.grou
ps` argument.
```

Year	Primary.Type	n
2014	ASSAULT	1988
2014	BATTERY	1605
2014	CRIM SEXUAL ASSAULT	69
2014	CRIMINAL SEXUAL ASSAULT	4
2014	ROBBERY	3799
2014	WEAPONS VIOLATION	2663

In [104]:

```
cols <- c("deepskyblue","red","blue","green","yellow","black")
with(PT_date_count,
     scatterplot3d(Primary.Type,
                   Year,
                   n,type="h",
                   color=cols[as.numeric(PT_date_count$Primary.Type)],
                   ,main="Chicago Crime Types Number of Evolution Over The
legend("topleft", legend = levels(factor(PT_date_count$Primary.Type)),
      col =  c("deepskyblue","red","blue","green","yellow","black"), pch=1
       cex=0.8)
```



3.3 LIDAR data visualization

In [16]:
```
# pip install geemap[lidar] open3d
```

Download a sample LiDAR dataset (https://drive.google.com/file/d/1H_X1190vL63BoFYa_cVBDxtIa8rG-Usb/view?usp=sharing) from Google Drive. The zip file is 52.1 MB and the uncompressed LAS file is 109 MB.

In [3]:
```
url = (
    'https://drive.google.com/file/d/1H_X1190vL63BoFYa_cVBDxtIa8rG-Usb/vie
)
filename = 'madison.las'
```

In [4]:
```
if not os.path.exists(filename):
    geemap.download_file(url, 'madison.zip', unzip=True)
```

```
Downloading...
From: https://drive.google.com/uc?id=1H_X1190vL63BoFYa_cVBDxtIa8rG-Usb (http
s://drive.google.com/uc?id=1H_X1190vL63BoFYa_cVBDxtIa8rG-Usb)
To: H:\CY Tech Year 4\Semester 1\Geospatial\Noteooks\madison.zip
100%|████████████████████████████████████████████████████████████████████
██████| 54.7M/54.7M [00:03<00:00, 14.0MB/s]

Extracting files...
```

Read the LiDAR data

In [5]:
```
las = geemap.read_lidar(filename)
```

The LAS header.

In [6]:
```
las.header
```

Out[6]:
```
<LasHeader(1.3, <PointFormat(1, 0 bytes of extra dims)>)>
```

The number of points.

In [7]:
```
las.header.point_count
```

Out[7]:
```
4068294
```

The list of features.

In [8]:
```
1 list(las.point_format.dimension_names)
```

Out[8]:
```
['X',
 'Y',
 'Z',
 'intensity',
 'return_number',
 'number_of_returns',
 'scan_direction_flag',
 'edge_of_flight_line',
 'classification',
 'synthetic',
 'key_point',
 'withheld',
 'scan_angle_rank',
 'user_data',
 'point_source_id',
 'gps_time']
```

Inspect data.

In [9]:
```
1 las.X
```

Out[9]:
```
array([5324343, 5324296, 5323993, ..., 5784049, 5784359, 5784667])
```

In [10]:
```
1 las.Y
```

Out[10]:
```
array([8035264, 8035347, 8035296, ..., 7550110, 7550066, 7550026])
```

In [11]:
```
1 las.Z
```

Out[11]:
```
array([36696, 34835, 34826, ..., 36839, 36858, 36842])
```

In [12]:
```
1 las.intensity
```

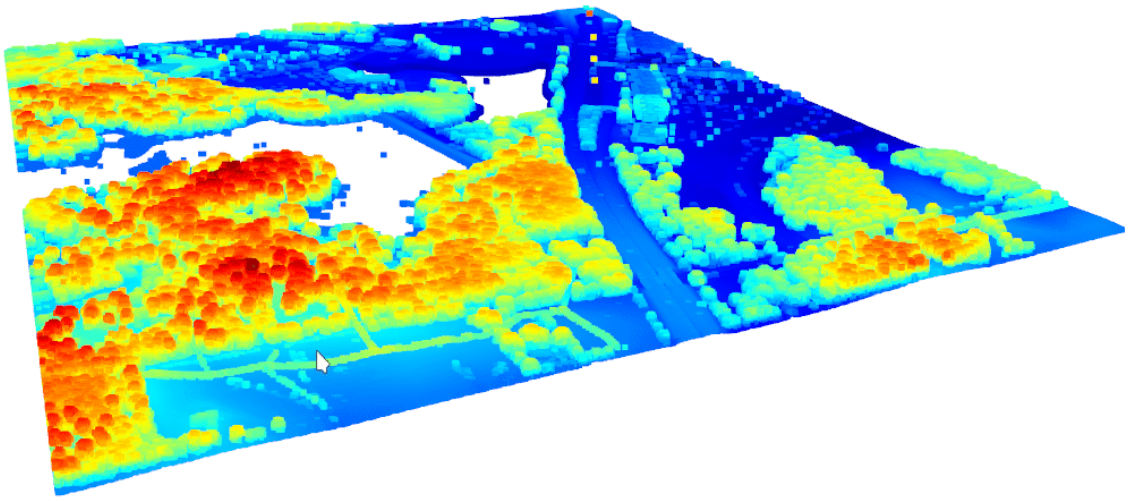
Out[12]:
```
array([ 9, 41, 24, ..., 87, 80, 95], dtype=uint16)
```

Visualize LiDAR data using the open3d (http://www.open3d.org/) backend.

In [15]:
```
1 geemap.view_lidar(filename, backend='open3d')
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Gif of the pop up window output by the code chunk above



4: spatial statistics

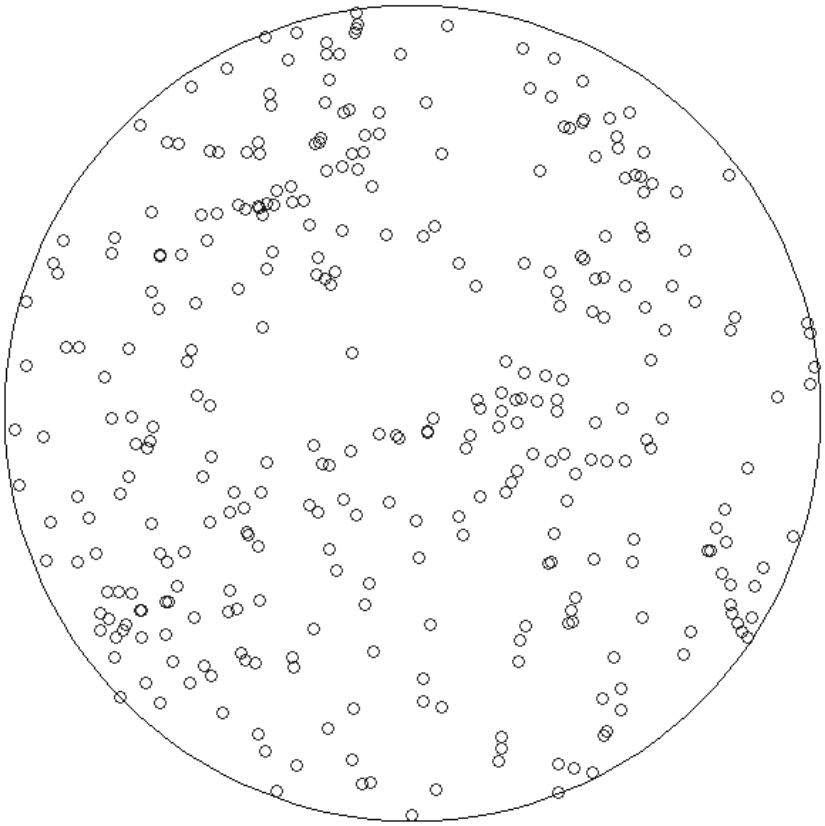
4.1: Introduction to point pattern statistics

In [34]:

```r
#1
disc10 <- disc(10)
# 2
set.seed(123)
p_cluster <- rThomas(kappa = 0.35, scale = 1, mu = 3, win = disc10)
plot(p_cluster)
#3
quadrat.test(p_cluster, alternative = "clustered")
```

```
        Chi-squared test of CSR using quadrat counts

data:  p_cluster
X2 = 53.387, df = 24, p-value = 0.0005142
alternative hypothesis: clustered

Quadrats: 25 tiles (irregular windows)
```

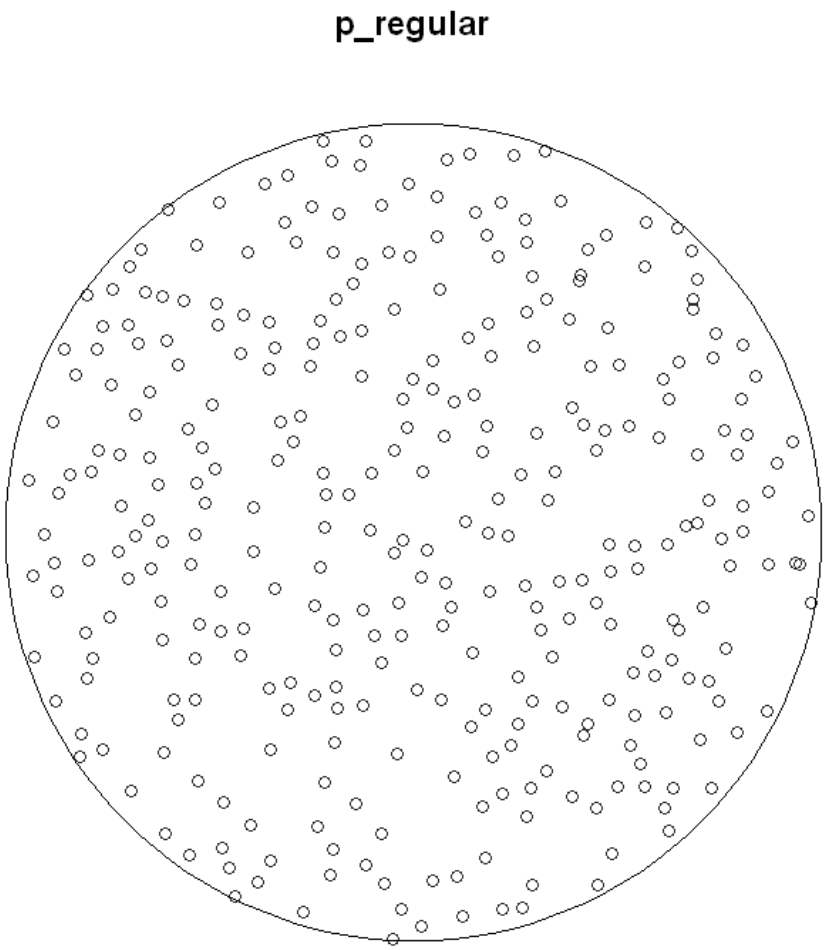
p_cluster



In [35]:

```r
#4
set.seed(123)
p_regular <- rStrauss(beta = 2.9, gamma = 0.025, R = .5, W = disc10)
plot(p_regular)

#5
quadrat.test(p_regular, alternative = "regular")
```

```
        Chi-squared test of CSR using quadrat counts

data:  p_regular
X2 = 8.57, df = 24, p-value = 0.001619
alternative hypothesis: regular

Quadrats: 25 tiles (irregular windows)
```

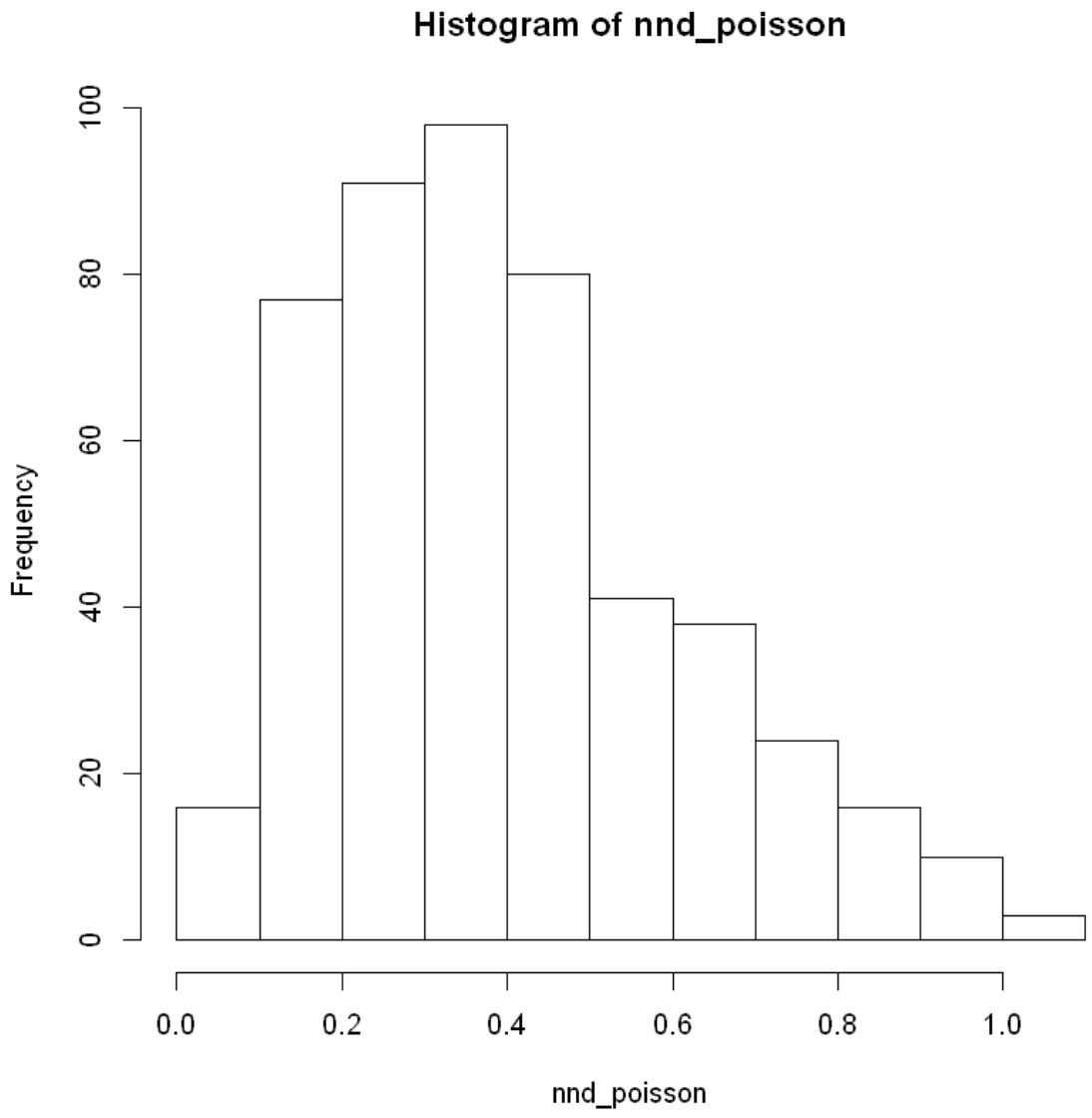


In [36]:

```r
#6

# Create a disc of radius 10
disc10 <- disc(10)

# Compute the rate as count divided by area
lambda <- 500 / area(disc(10))

# Create a point pattern object
p_poisson <- rpoispp(lambda = lambda, win = disc(10))

# Create a point pattern object
p_regular <- rStrauss(beta = 2.9, gamma = 0.025, R = .5, W = disc10)
```

In [37]:

```r
#7

# Calc nearest-neighbor distances for Poisson point data
nnd_poisson <- nndist(p_poisson)

#8
# Draw a histogram of nearest-neighbor distances
hist(nnd_poisson)

#9
# Estimate G(r)
G_poisson <- Gest(p_poisson)
```



In [38]:

```r
# 10
```

$$G(r) = 1 - exp(-lambda * pi * r^2)$$

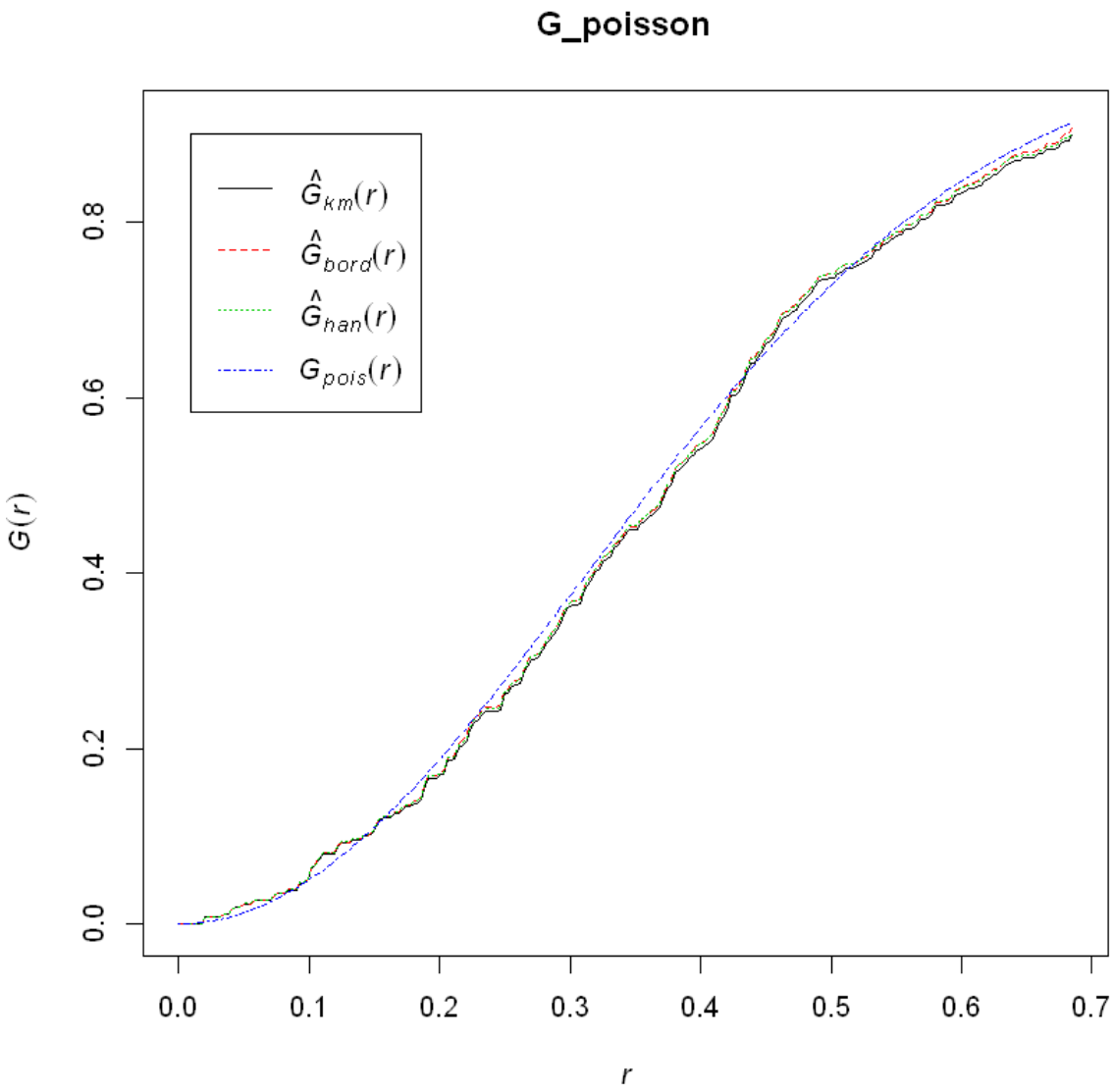
G(r) is the probability of a point having a nearest neighbor within a distance r (the corresponding cdf of the pdf of the NN distances).

To calculate G(r) for a pattern, look at each event in the pattern and find the distance to the nearest event, we do this for every event, and plot a histogram to give an estimate of the probability density function of the nearest neighbor distribution. The corresponding cumulative distribution function is the G(r) we are searching for.

Border correction in spatial statistics involves correcting for the effects of the boundaries of the region on the calculated value of G(r). This is important because the presence of boundaries can influence the distribution of points within the region, leading to inaccurate results. There are several methods that can be used to correct for these effects, including the use of periodic boundary conditions, which involve replicating the region in all dimensions to create an infinite array of identical copies. This can help to mitigate the influence of the boundaries on the calculated value of G(r).

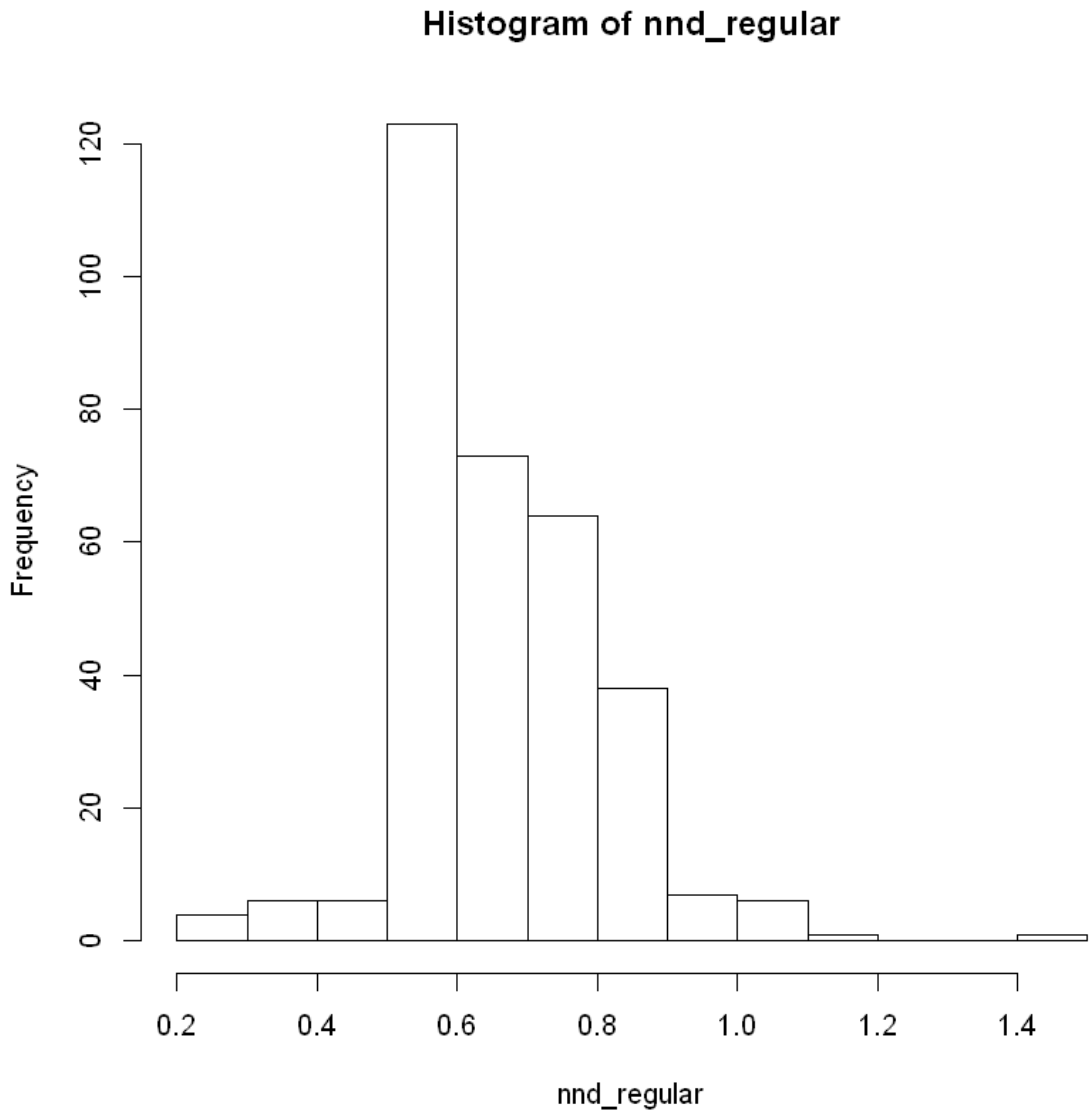
In [16]:

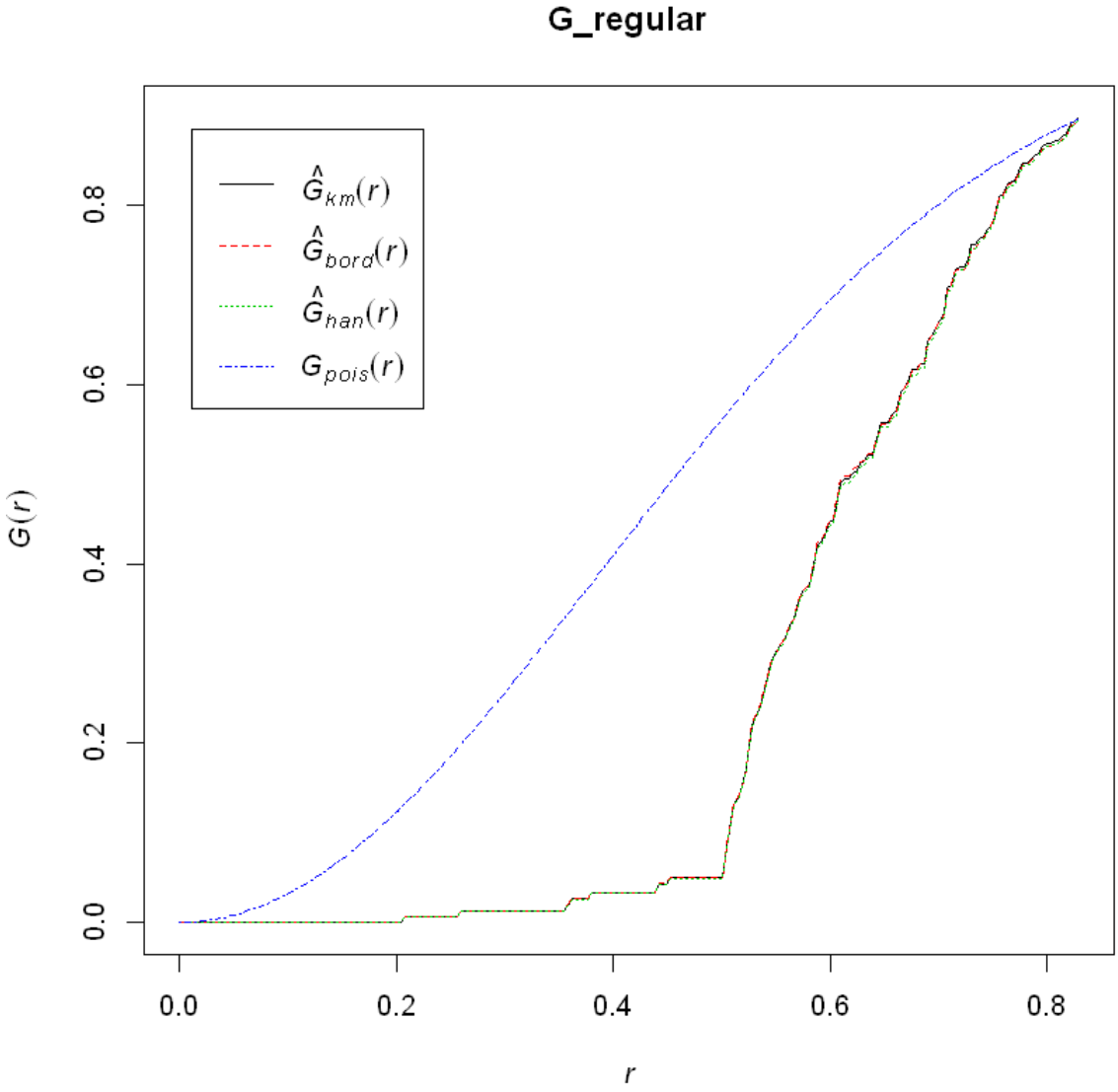
```
#12
# Plot G(r) vs. r
plot(G_poisson)
```



In [17]:

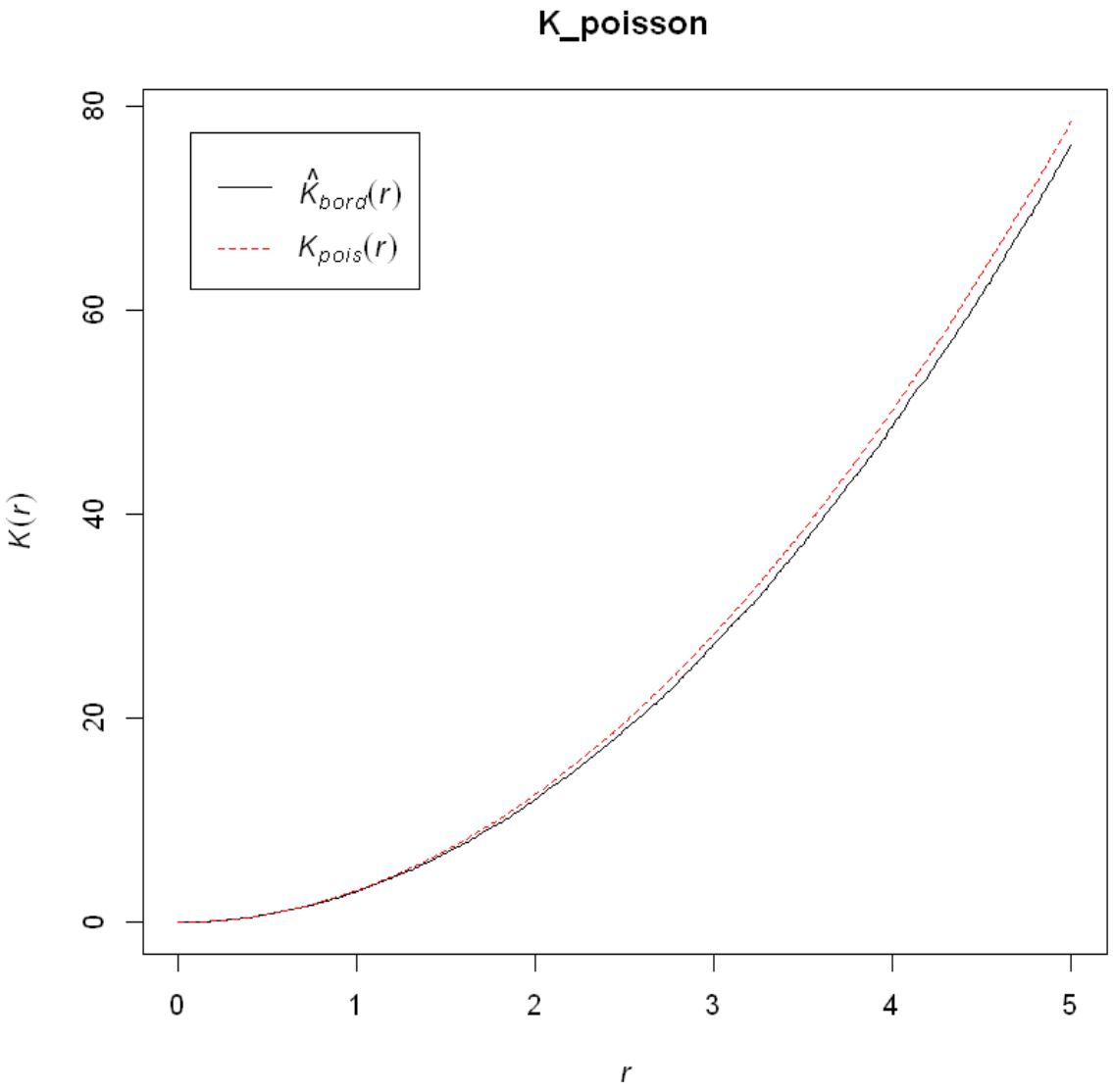
```
#13
# Repeat for regular point data
nnd_regular <- nndist(p_regular)
hist(nnd_regular)
G_regular <- Gest(p_regular)
plot(G_regular)
```




G_regular
$\hat{G}_{km}(r)$
$\hat{G}_{bord}(r)$
$\hat{G}_{han}(r)$
$G_{pois}(r)$
G(r)
r
0.0
0.2
0.4
0.6
0.8

In [18]:

```r
#13
# Estimate the K-function for the Poisson points
K_poisson <- Kest(p_poisson, correction = "border")

# The default plot shows quadratic growth
plot(K_poisson, . ~ r)
```



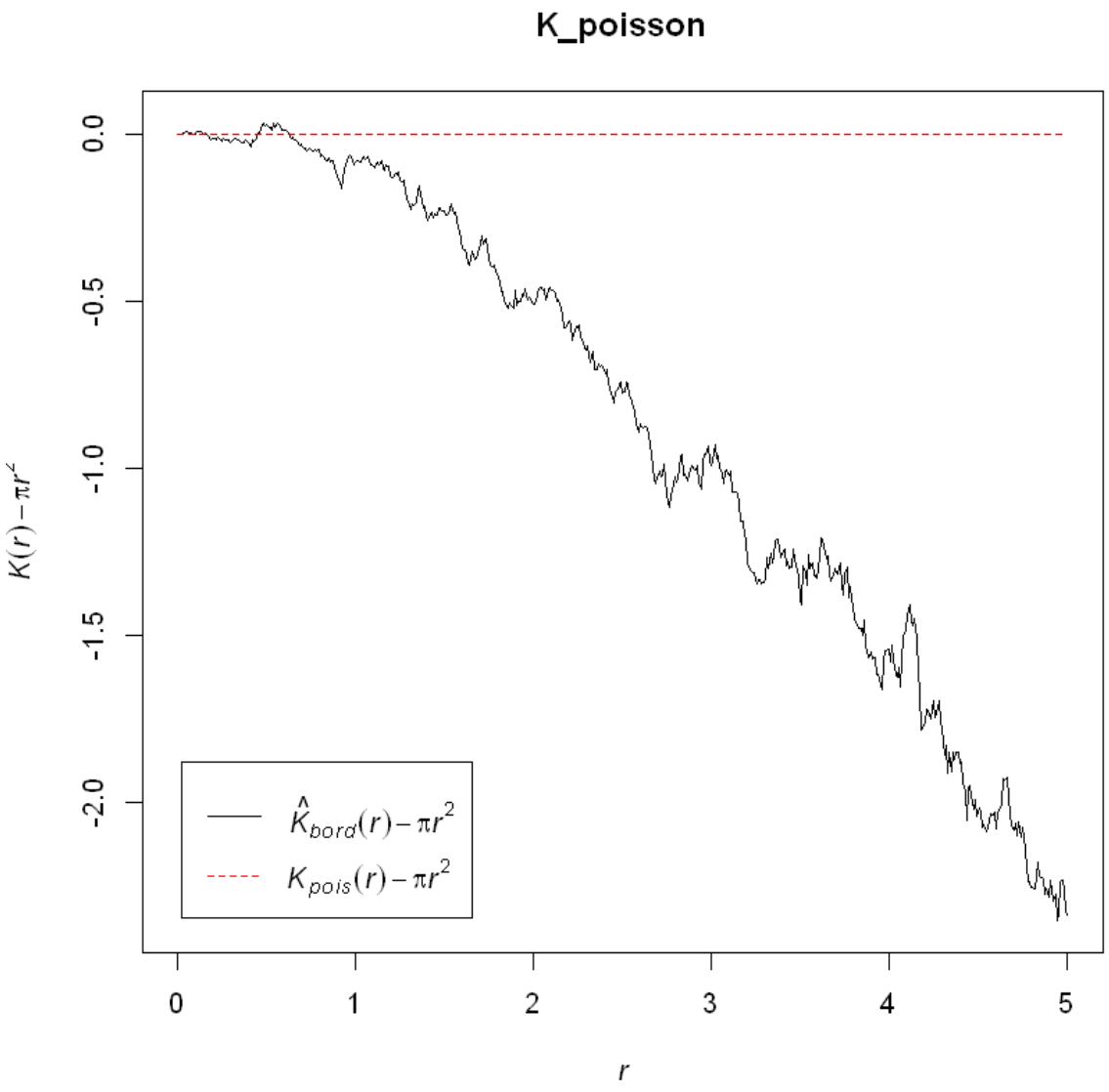
14 Explain the role of the K-function and the envelope, and generally speaking how Ripley's K work

$$K(r) = pi * r^2 - \text{the area of a circle of with radius r}$$

Ripley's K-function is used to compare a given point distribution with a random distribution where the point distribution under investigation is tested against the null hypothesis that the points are distributed randomly and independently. K(r) is defined as the expected number of points within a distance of a point of the process, scaled by the intensity.

An envelope is a tool to assess uncertainties on K-function estimates by randomly sampling points from a uniform Poisson process in the area and computing the K-function of the simulated data. Repeat this process 99 times, and take the minimum and maximum value of K over each of the distance values. If the K-function from the data goes above the top of the envelope then we have evidence for clustering. If the K-function goes below the envelope then there is evidence for an inhibitory process causing points to be spaced out.

In [20]:
```
#15
# Subtract pi * r ^ 2 from the Y-axis and plot
plot(K_poisson, . - pi * r ^ 2 ~ r)
```

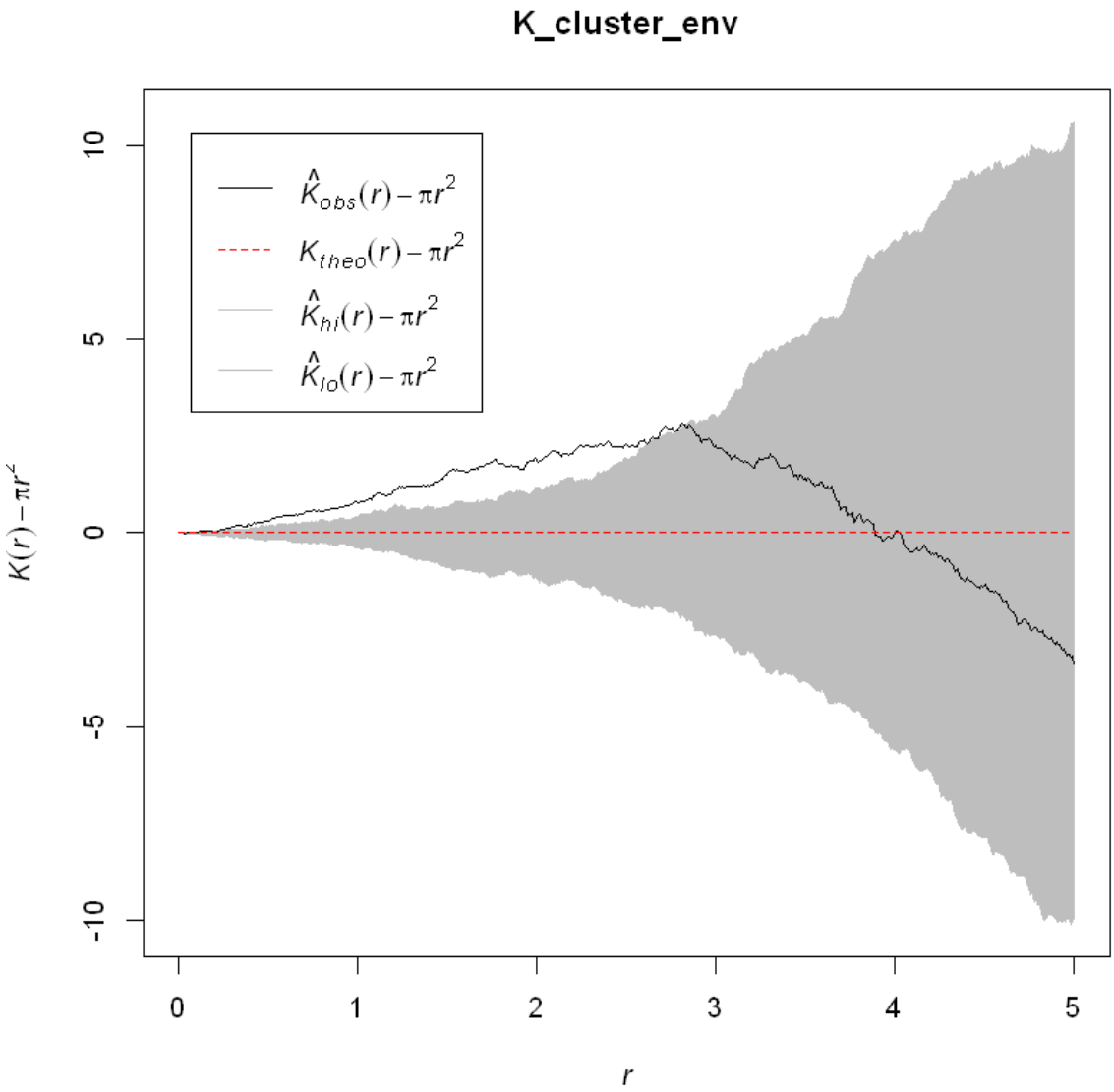


Generally speaking, if the data was generated by a Poisson process, then the line should be close to zero for all values of r, so we subtract pi*r^2 verify this fact. In the case above, we see that the red line stayed at zero for all values of r.

In [23]:
```r
#16 , 17
# Compute envelopes of K under random locations
K_cluster_env <- envelope(p_cluster, Kest, correction = "border")
```

```
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 7
9, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.

Done.
```

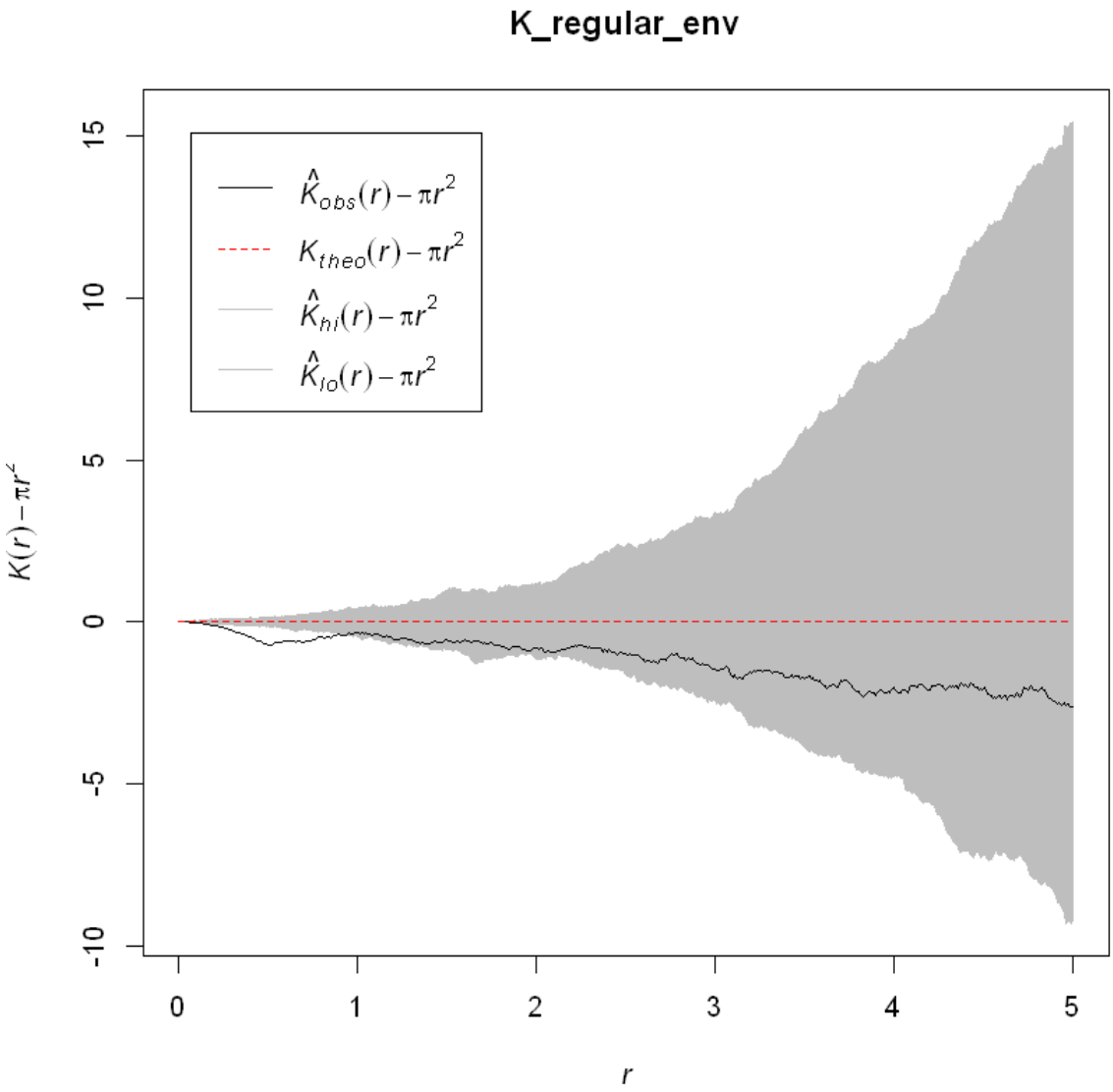
In [24]:
```r
#18 (1)
# Insert the full formula to plot K minus pi * r^2
plot(K_cluster_env, .- pi * r ^ 2 ~ r)
```



In [25]:

```
#18 (2)
# Repeat for regular data
K_regular_env <- envelope(p_regular, Kest, correction = "border")
plot(K_regular_env, .- pi * r ^ 2 ~ r)
```

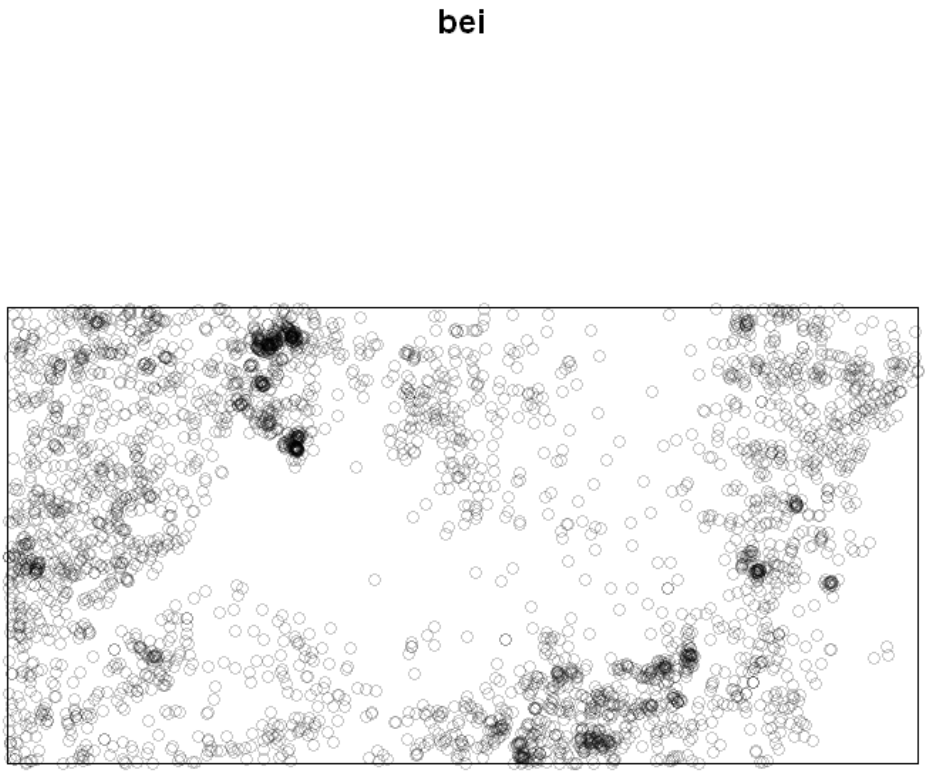
```
Generating 99 simulations of CSR  ...
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 6
0, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 7
9, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,  99.

Done.
```



Real world data

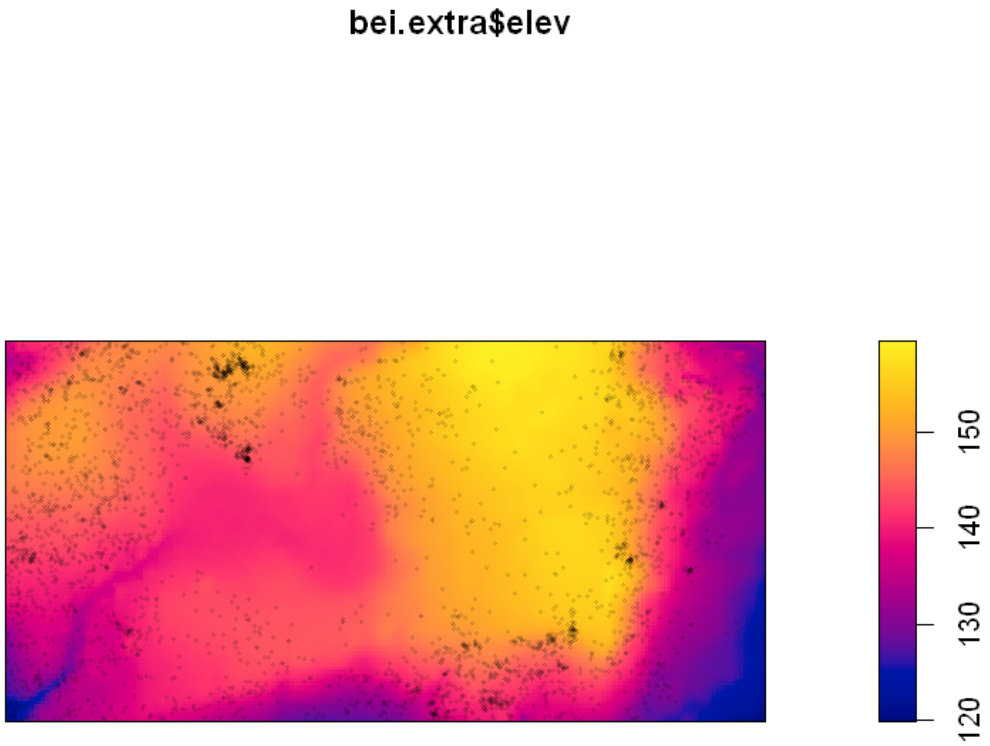
In [40]:
```
1 data(bei)
```

In [43]:
```
1 plot(bei)
```



In [48]:
```
1 data(bei)
```

In [96]:
```
plot(bei.extra$elev)
plot(bei, add = TRUE,cex = 0.3)
```



In [99]:
```
quadrat.test(bei, alternative = "clustered")
```

```
        Chi-squared test of CSR using quadrat counts

data:  bei
X2 = 2009.9, df = 24, p-value < 2.2e-16
alternative hypothesis: clustered

Quadrats: 5 by 5 grid of tiles
```

From the quadrate test above, we can conclude that the tree pattern is clustered as the p-value < 2.2e-16, so we can reject the null hypothesis that the data is coming from a uniform Poisson point process.

4.2: Point pattern statistics with Preston Crime

In [41]:
```
preston_crime_df <- read.table(".//..//Datasets//preston crime-spatstat.cs
colnames(preston_crime_df) <- c('x','y','mark')
summary(preston_crime_df)
```

```
                x                        y                       mark
349847,286386053:   1   425746,70161589 :   1   Non-violent crime:1812
349861,601260038:   1   425778,026335666:   1   Violent crime    : 224
349879,376771875:   1   425778,953254153:   1
349888,695199284:   1   425788,761293382:   1
349902,595749144:   1   425792,560420541:   1
349903,203841941:   1   425793,869626957:   1
(Other)         :2030   (Other)         :2030
```

In [42]:
```
preston_osm <- read.table(".//..//Datasets//osm_preston_gray.csv", sep=';'
colnames(preston_osm) <- c("osm_preston_gray.1","osm_preston_gray.2","osm_
summary(preston_osm)
```

```
osm_preston_gray.1 osm_preston_gray.2 osm_preston_gray.3 osm_preston_gray.4
Min.   :  0.0      Min.   :  0.0      Min.   :  0.0      Min.   :255
1st Qu.:206.0      1st Qu.:206.0      1st Qu.:206.0      1st Qu.:255
Median :224.0      Median :224.0      Median :224.0      Median :255
Mean   :219.8      Mean   :219.8      Mean   :219.8      Mean   :255
3rd Qu.:239.0      3rd Qu.:239.0      3rd Qu.:239.0      3rd Qu.:255
Max.   :254.0      Max.   :254.0      Max.   :254.0      Max.   :255
```

In [45]:
```
filename <- file.choose()
preston_crime <- readRDS(filename)
# preston_osm <- readRDS("osm_preston.rds")
# Get some summary information on the dataset
summary(preston_crime)
```

```
Marked planar point pattern:  2036 points
Average intensity 4.053214e-05 points per square unit

Coordinates are given to 2 decimal places
i.e. rounded to the nearest multiple of 0.01 units

Multitype:
                  frequency proportion    intensity
Non-violent crime      1812  0.8899804 3.607281e-05
Violent crime           224  0.1100196 4.459332e-06

Window: polygonal boundary
single connected closed polygon with 99 vertices
enclosing rectangle: [349773, 357771] x [425706.5, 433705.5] units
                     (7998 x 7999 units)
Window area = 50231700 square units
Fraction of frame area: 0.785
```

In [46]:

```
filename <- file.choose()
preston_osm <- readRDS(filename)
# Get some summary information on the dataset
summary(preston_osm)
```

	osm_preston_gray.1	osm_preston_gray.2	osm_preston_gray.3	osm_preston_gray.4
Min.	0	0	0	255
1st Qu.	206	206	206	255
Median	224	224	224	255
3rd Qu.	239	239	239	255
Max.	254	254	254	255
NA's	0	0	0	0

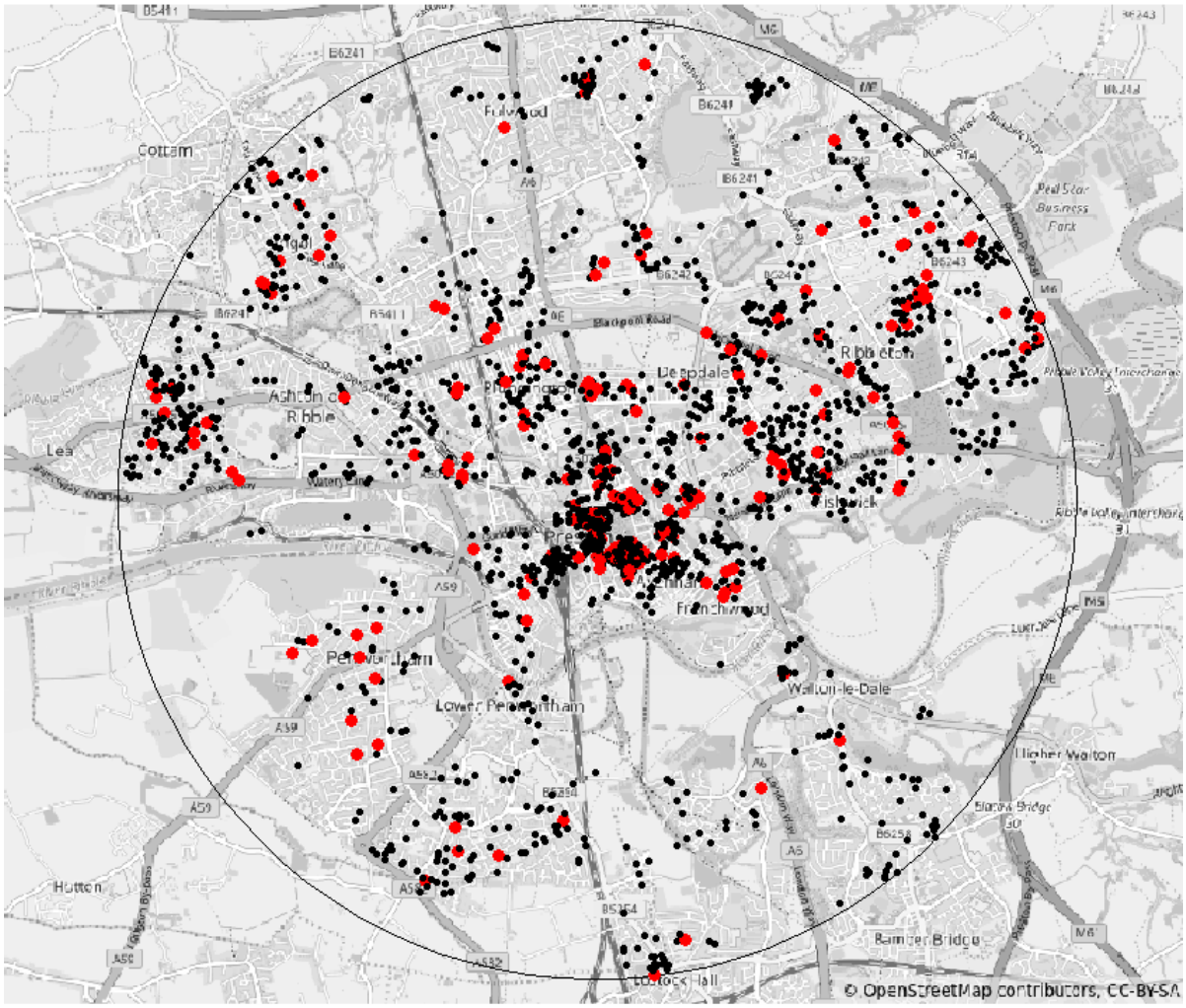
In [47]:

```
# Get a table of marks
table(marks(preston_crime))
```

```
Non-violent crime     Violent crime
             1812               224
```

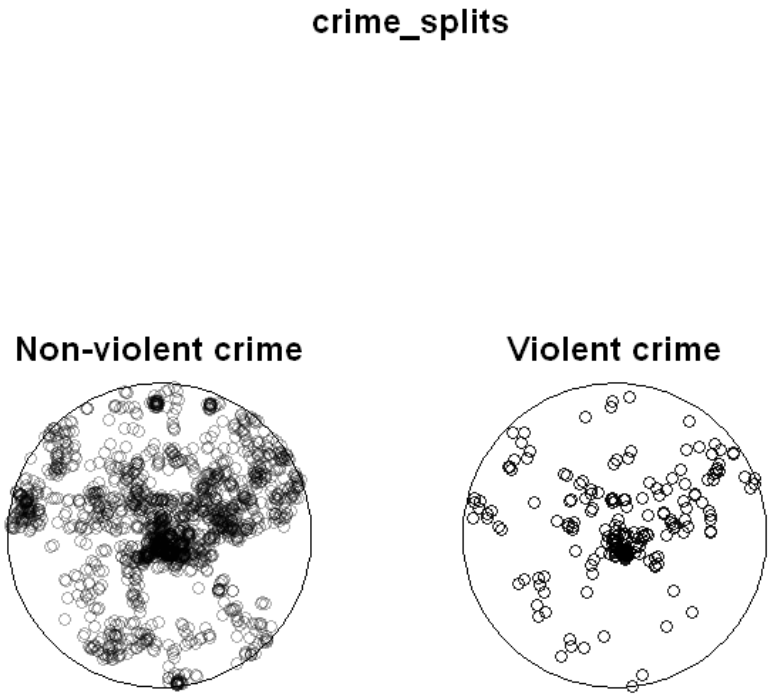
In [48]:

```r
# Define a function to create a map
preston_map <- function(cols = c("green","red"), cex = c(1, 1), pch = c(1,
  plotRGB(preston_osm) # from the raster package
  plot(preston_crime, cols = cols, pch = pch, cex = cex, add = TRUE, show.
}

# Draw the map with colors, sizes and plot character
preston_map(
  cols = c("black", "red"),
  cex = c(0.5, 1),
  pch = c(19,19))
```



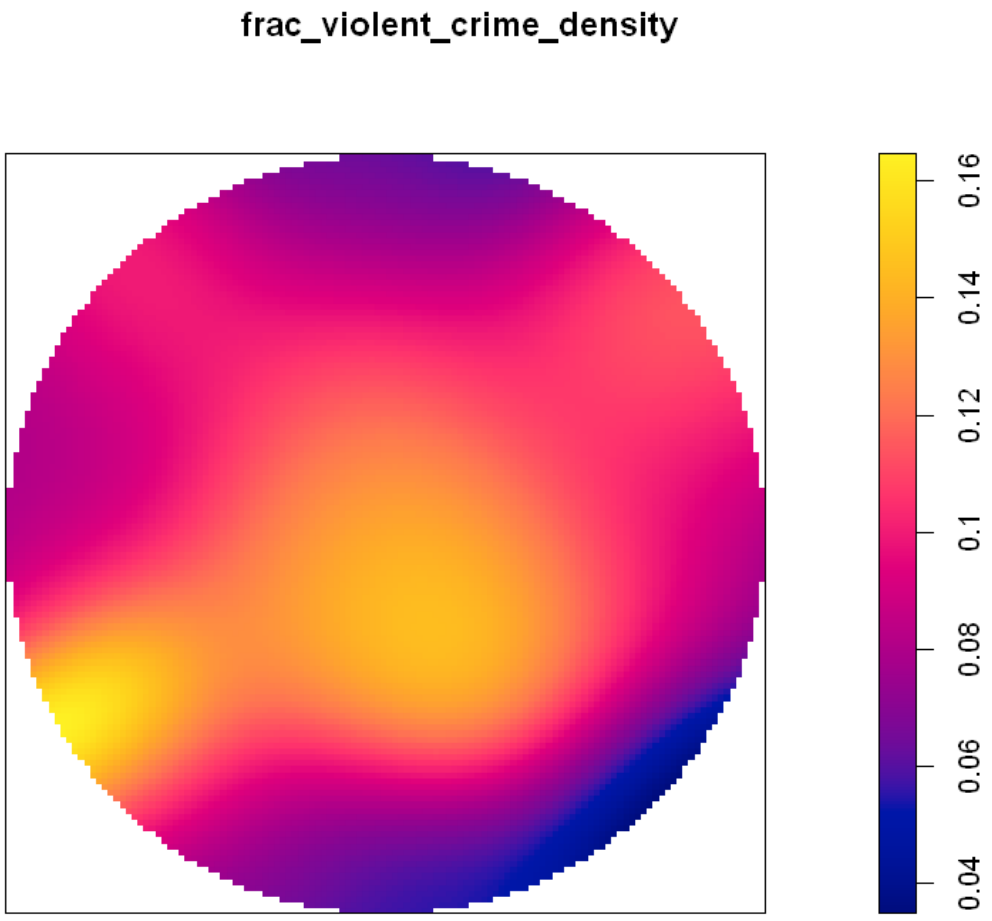
In [49]:

```
# 1

# Use the split function to show the two point patterns
crime_splits <- split(preston_crime)

# 2
# Plot the split crime
plot(crime_splits)
```



In [50]:

```
#3
# Compute the densities of both sets of points
crime_densities <- density(crime_splits)

#4
# Calc the violent density divided by the sum of both
frac_violent_crime_density <- crime_densities[["Violent crime"]] /
  (crime_densities[["Non-violent crime"]] + crime_densities[["Violent crim

#5
# Plot the density of the fraction of violent crime
plot(frac_violent_crime_density)
```



4.3: Areal statistics

In [51]:
```
london_ref <- readRDS(".//..//Datasets//london_eu.rds")
```

In [52]:
```
summary(london_ref)
```

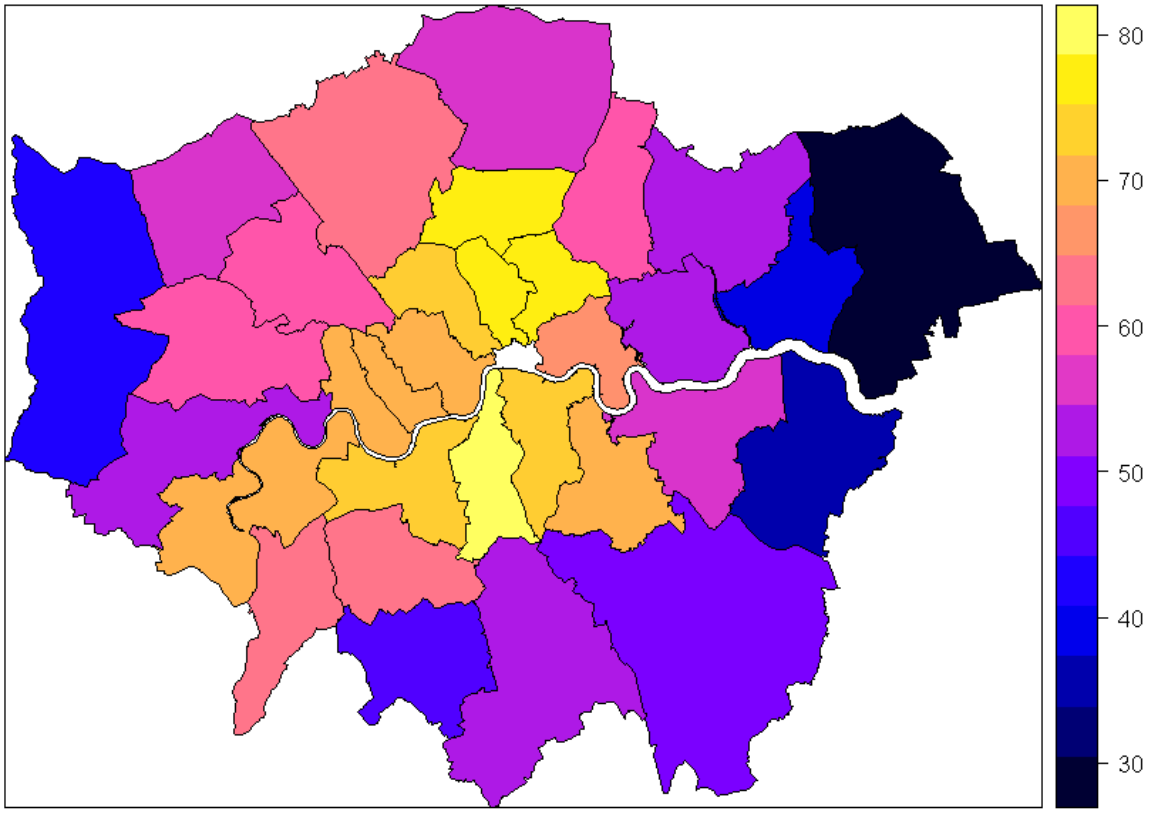
```
Object of class SpatialPolygonsDataFrame
Coordinates:
       min      max
x 503574.2 561956.7
y 155850.8 200933.6
Is projected: TRUE
proj4string :
[+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000
+y_0=-100000 +datum=OSGB36 +units=m +no_defs +ellps=airy
+towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894]
Data attributes:
     NAME             TOTAL_POP        Electorate       Votes_Cast
 Length:32          Min.   :157711   Min.   : 83042   Min.   : 54801
 Class :character   1st Qu.:237717   1st Qu.:143458   1st Qu.:104079
 Mode  :character   Median :272017   Median :168394   Median :116280
                    Mean   :270780   Mean   :169337   Mean   :118025
                    3rd Qu.:316911   3rd Qu.:196285   3rd Qu.:134142
                    Max.   :379691   Max.   :245349   Max.   :182570
     Remain           Leave       Rejected_Ballots   Pct_Remain
 Min.   : 27750   Min.   :17138   Min.   : 60.0    Min.   :30.34
 1st Qu.: 55973   1st Qu.:32138   1st Qu.:105.0    1st Qu.:53.69
 Median : 70254   Median :45263   Median :138.0    Median :61.01
 Mean   : 70631   Mean   :47255   Mean   :139.0    Mean   :60.46
 3rd Qu.: 84287   3rd Qu.:59018   3rd Qu.:164.2    3rd Qu.:69.90
 Max.   :118463   Max.   :96885   Max.   :267.0    Max.   :78.62
   Pct_Leave      Pct_Rejected      Assembly
 Min.   :21.38   Min.   :0.0600   Length:32
 1st Qu.:30.10   1st Qu.:0.0875   Class :character
 Median :38.99   Median :0.1100   Mode  :character
 Mean   :39.54   Mean   :0.1187
 3rd Qu.:46.31   3rd Qu.:0.1500
 Max.   :69.66   Max.   :0.2200
```

In [53]:
```
# Which boroughs voted to "Remain"?
london_ref$NAME[london_ref$Leave < london_ref$Remain]
```

'Lewisham' 'Merton' 'Newham' 'Redbridge' 'Richmond upon Thames' 'Southwark' 'Tower Hamlets' 'Waltham Forest' 'Wandsworth' 'Westminster' 'Barnet' 'Brent' 'Bromley' 'Camden' 'Croydon' 'Ealing' 'Enfield' 'Greenwich' 'Hackney' 'Hammersmith and Fulham' 'Haringey' 'Harrow' 'Hounslow' 'Islington' 'Kensington and Chelsea' 'Kingston upon Thames' 'Lambeth'

In [54]:

```
#1
# Plot a map of the percentage that voted "Remain"
spplot(london_ref, zcol = "Pct_Remain")
```



In [56]:

```
#2
# Use the cartogram and rgeos packages
library(cartogram)
library(rgeos)
library(spdep)

# Make a cartogram, scaling the area to the electorate
carto_ref <- cartogram_cont(london_ref, "Electorate")
plot(carto_ref)
```

```
Loading required package: spData
To access larger datasets in this package, install the spDataLarge
package with: `install.packages('spDataLarge',
repos='https://nowosad.github.io/drat/', type='source')`
Loading required package: sf
Linking to GEOS 3.9.0, GDAL 3.2.1, PROJ 7.2.1
Mean size error for iteration 1: 1.5881743190908
Mean size error for iteration 2: 1.30262460145333
Mean size error for iteration 3: 1.16605703069549
Mean size error for iteration 4: 1.09652671280571
Mean size error for iteration 5: 1.05979301980903
Mean size error for iteration 6: 1.03870642059676
Mean size error for iteration 7: 1.02576724388584
Mean size error for iteration 8: 1.01780225318077
Mean size error for iteration 9: 1.0124614648423
Mean size error for iteration 10: 1.00876130634916
Mean size error for iteration 11: 1.00621346049453
Mean size error for iteration 12: 1.00446957764083
Mean size error for iteration 13: 1.00322569444771
Mean size error for iteration 14: 1.00233570000671
Mean size error for iteration 15: 1.00169692422828
```

In [57]:

```
#3
# Make neighbor list
borough_nb <- poly2nb(london_ref)
print(borough_nb)
```

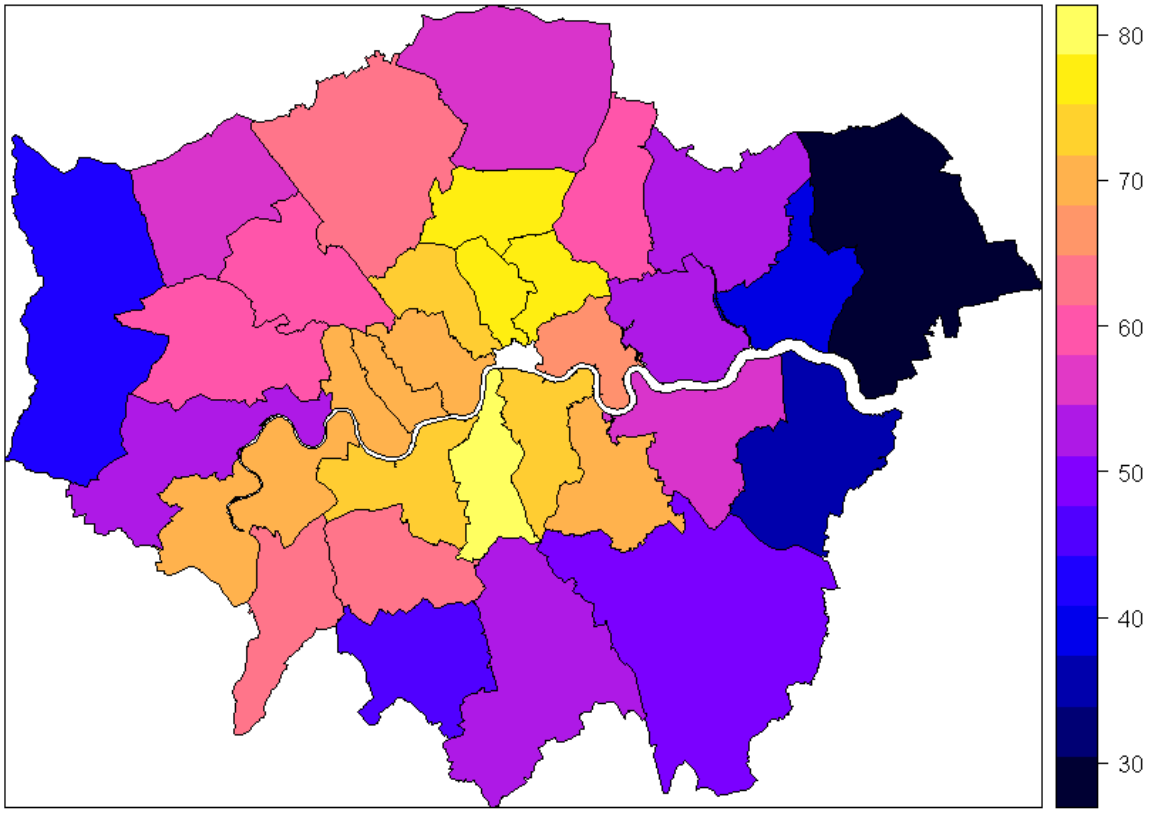
```
Neighbour list object:
Number of regions: 32
Number of nonzero links: 126
Percentage nonzero weights: 12.30469
Average number of links: 3.9375
```

In [58]:

```r
#4
# Get center points of each borough
borough_centers <- coordinates(london_ref)
#5
# Show the connections
plot(london_ref); plot(borough_nb, borough_centers, add = TRUE)
```

In [59]:

```
#6
# Map % Remain
spplot(london_ref, zcol = "Pct_Remain")
```



In [60]:

```
#7
# Run a Moran I MC test on % Remain
moran.mc(
  london_ref$Pct_Remain,
  nb2listw(borough_nb),
  999
)
```

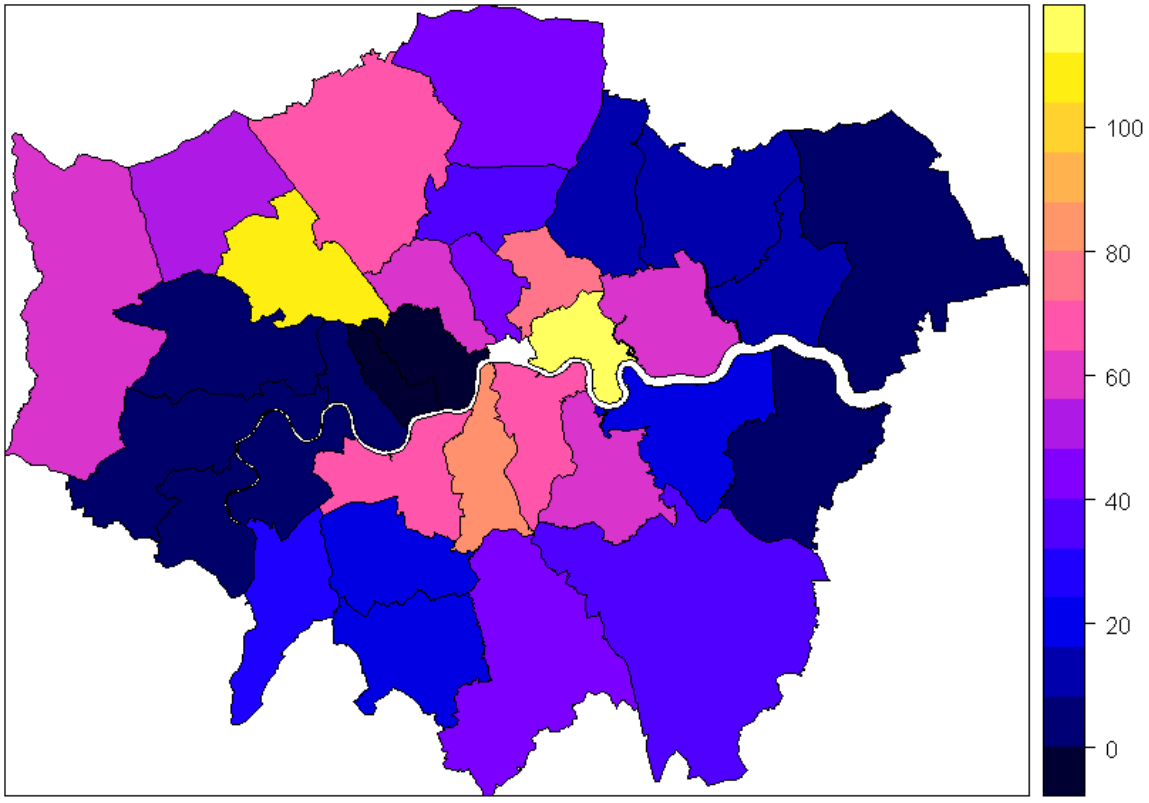
```
        Monte-Carlo simulation of Moran I

data:  london_ref$Pct_Remain
weights: nb2listw(borough_nb)
number of simulations + 1: 1000

statistic = 0.42841, observed rank = 999, p-value = 0.001
alternative hypothesis: greater
```

8) Explain how the Moran’s I distribution is made:
The Moran's I distribution is created by repeating the steps of a permutation test for the Moran's I statistic a large number of times. Specifically, the distribution is created by permuting the values of the dependent variable among the observations, calculating the Moran's I statistic for each permutation, and storing the resulting values. This creates a distribution of possible values of the Moran's I statistic under the null hypothesis of no spatial autocorrelation.

Flu data

In [62]:
```
london <- readRDS(".//..//Datasets//london_2017_2.rds")
```

In [63]:
```
#1
# Map the OBServed number of flu reports
spplot(london, "Flu_OBS")
```



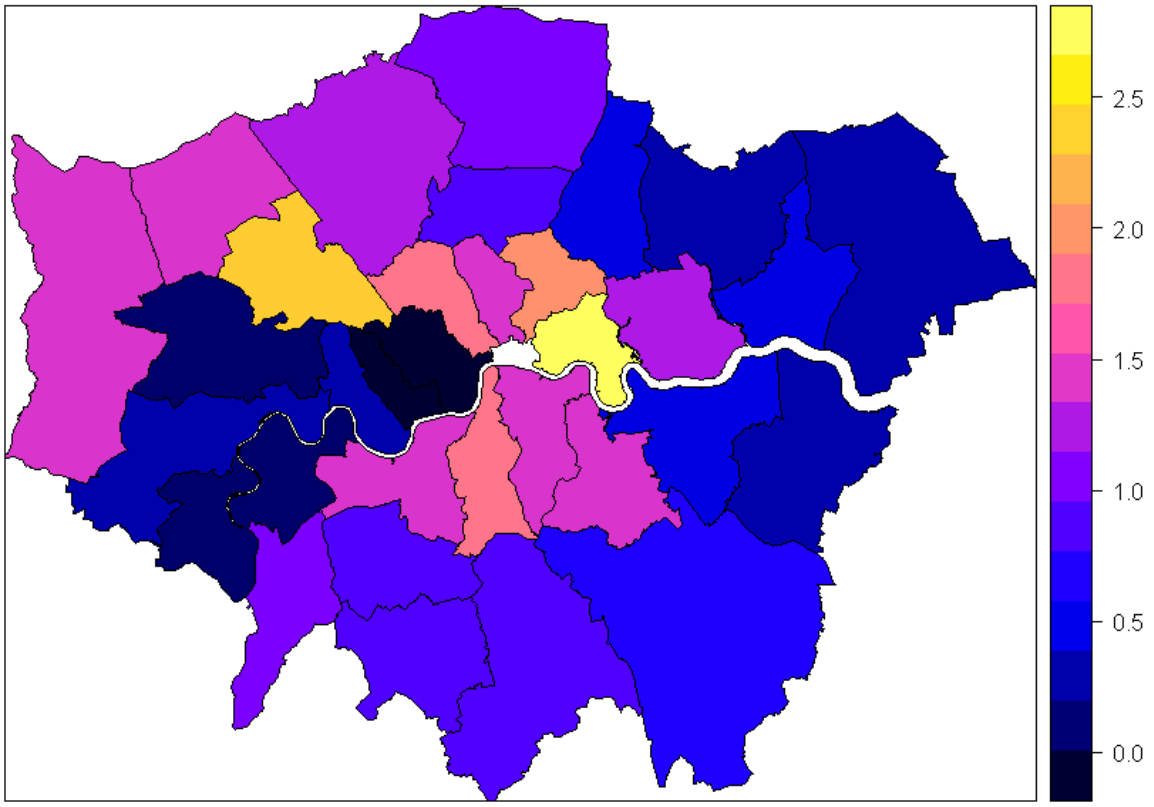
In [64]:
```
#2
# Compute and print the overall incidence of flu
r <- sum(london$Flu_OBS) / sum(london$TOTAL_POP)
r
```

0.000142412774772119

In [65]:
```
#3
# Calculate the expected number for each borough
london$Flu_EXP <- london$TOTAL_POP * r
```

In [66]:
```
#4
# Calculate the ratio of OBServed to EXPected
london$Flu_SMR <- london$Flu_OBS / london$Flu_EXP
```

In [67]:
```
#5
# Map the SMR
spplot(london, "Flu_SMR")
```



SMR corresponds to Standardized Morbidity Ratio, and it is the number of cases per person, but scaled by the overall incidence so that the expected number is 1.

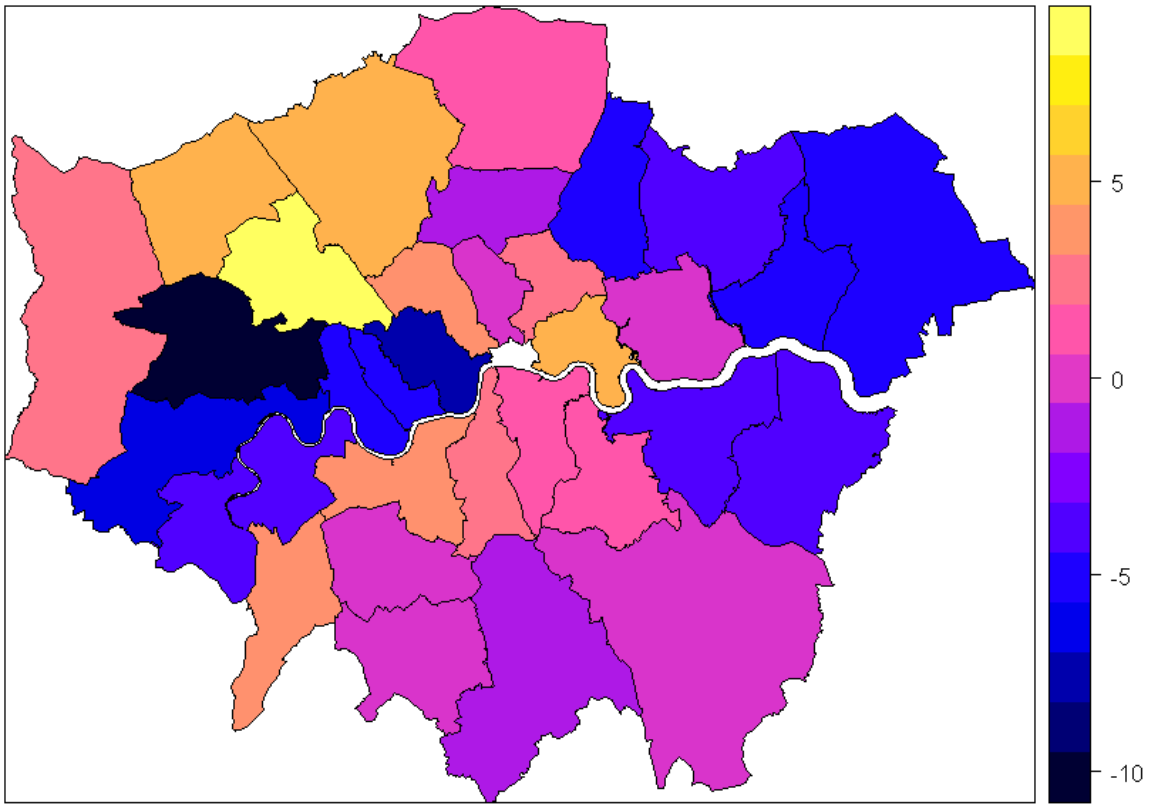
GLMs

In [68]:

```
#1
# Fit a poisson GLM.
model_flu <- glm(
  Flu_OBS ~ HealthDeprivation,
  offset = log(TOTAL_POP),
  data = london,
  family = poisson)
```

In [71]:

```
#2
# Put residuals into the spatial data.
london$Flu_Resid <- residuals(model_flu)
# Map the residuals using spplot
spplot(london, "Flu_Resid")
```



The yellow borough had many more cases of flu than expected; the black borough had many fewer cases.

In [39]:

```r
#3
# Compute the neighborhood structure and test spatial correlation of the r
library(spdep)
borough_nb <- poly2nb(london)

moran.mc(london$Flu_Resid, listw = nb2listw(borough_nb), nsim = 999)
```

```
        Monte-Carlo simulation of Moran I

data:  london$Flu_Resid
weights: nb2listw(borough_nb)
number of simulations + 1: 1000

statistic = 0.15059, observed rank = 926, p-value = 0.074
alternative hypothesis: greater
```

In [49]:

```r
#4
# Use R2BayesX
library(R2BayesX)

# Fit a classic GLM
model_flu <- glm(Flu_OBS ~ HealthDeprivation, offset = log(TOTAL_POP),
                 data = london, family = poisson)
summary(model_flu)
```

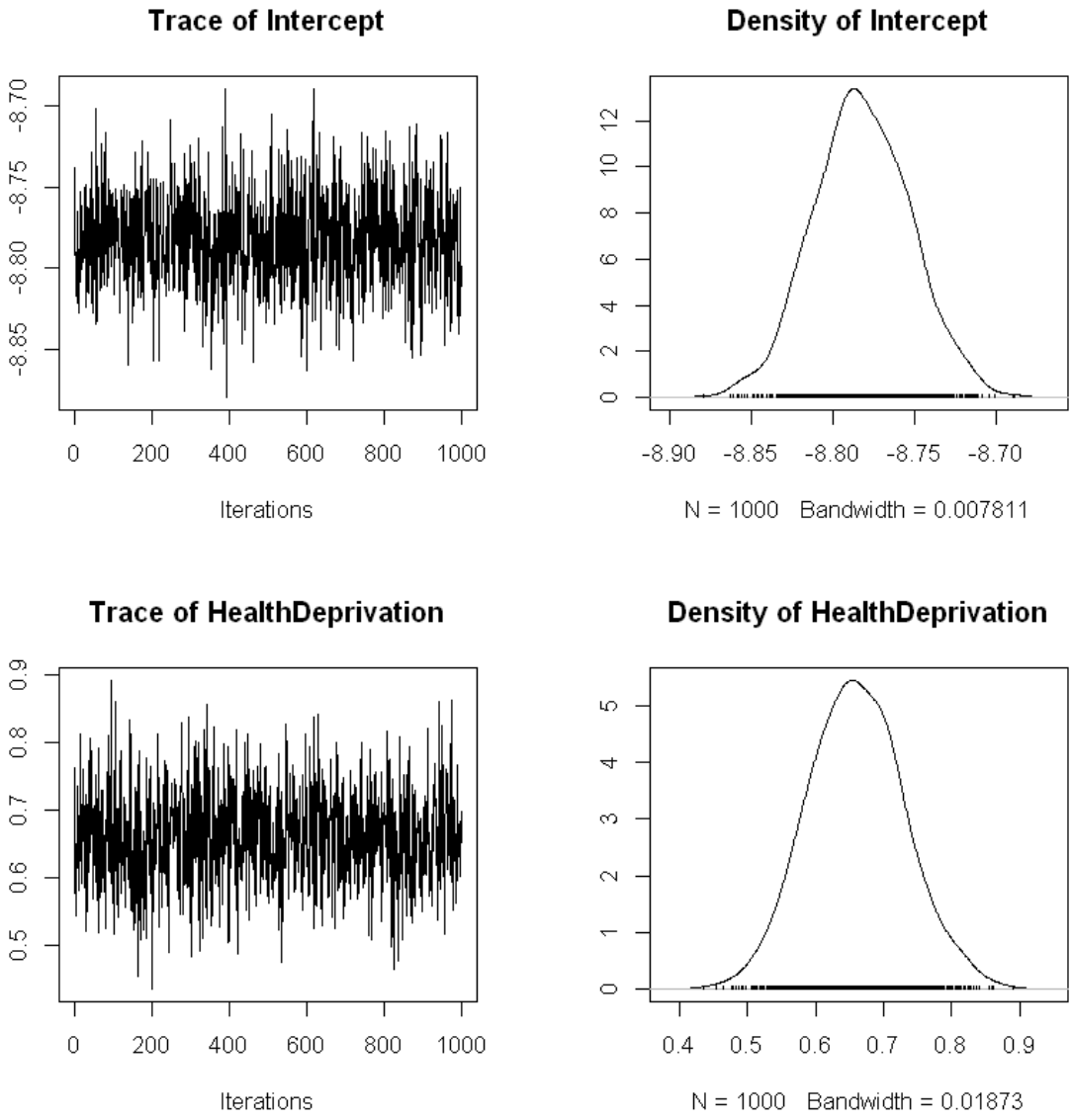
```
Call:
glm(formula = Flu_OBS ~ HealthDeprivation, family = poisson,
    data = london, offset = log(TOTAL_POP))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-9.5361  -4.5285  -0.0499   2.9043   8.2194

Coefficients:
                  Estimate Std. Error  z value Pr(>|z|)
(Intercept)       -8.78190    0.02869 -306.043   <2e-16 ***
HealthDeprivation  0.65689    0.06797    9.665   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 703.75  on 31  degrees of freedom
Residual deviance: 605.03  on 30  degrees of freedom
AIC: 762.37

Number of Fisher Scoring iterations: 5
```

In [50]:

```r
#5
# Fit a Bayesian GLM
bayes_flu <- bayesx(Flu_OBS ~ HealthDeprivation, offset = log(london$TOTAL
                    family = "poisson", data = data.frame(london),
                    control = bayesx.control(seed = 17610407))

# Summarize it
summary(bayes_flu)
```

```
Call:
bayesx(formula = Flu_OBS ~ HealthDeprivation, data = data.frame(london),
    offset = log(london$TOTAL_POP), control = bayesx.control(seed = 1761040
7),
    family = "poisson")

Fixed effects estimation results:

Parametric coefficients:
                     Mean      Sd    2.5%     50%   97.5%
(Intercept)       -8.7825  0.0293 -8.8386 -8.7839 -8.7238
HealthDeprivation  0.6598  0.0703  0.5244  0.6581  0.8062

N = 32  burnin = 2000  method = MCMC  family = poisson
iterations = 12000  step = 10
```

In [51]:
```
#6
# Look at the samples from the Bayesian model
plot(samples(bayes_flu))
```



7) Explain how the density of the healthDeprivation parameter is built and what it means:
The density of the healthDeprivation parameter is called the posterior distribution which is the probability distribution of the healthDeprivation parameter given the data.

In [52]:
```
#8
# Compute adjacency objects
borough_nb <- poly2nb(london)
borough_gra <- nb2gra(borough_nb)
```

In [53]:

```
#9
# Fit spatial model
flu_spatial <- bayesx(
  Flu_OBS ~ HealthDeprivation + sx(i, bs = "spatial", map = borough_gra),
  offset = log(london$TOTAL_POP),
  family = "poisson", data = data.frame(london),
  control = bayesx.control(seed = 17610407)
)
```

```
Note: created new output directory 'C:/Users/ADMINI~1/AppData/Local/Temp/Rtmp
2vdXMR/bayesx1'!
```

In [54]:

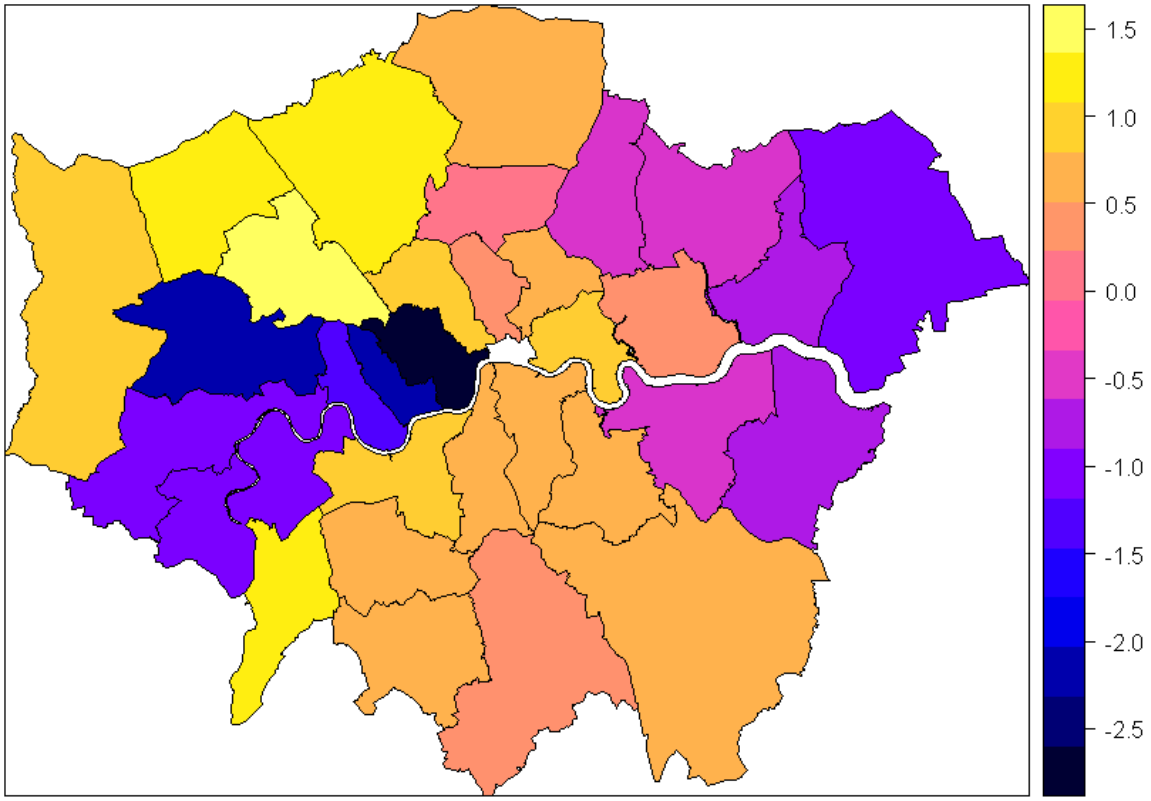
```
# Summarize the model
summary(flu_spatial)
```

```
Call:
bayesx(formula = Flu_OBS ~ HealthDeprivation + sx(i, bs = "spatial",
    map = borough_gra), data = data.frame(london), offset = log(london$TOTAL_
POP),
    control = bayesx.control(seed = 17610407), family = "poisson")

Fixed effects estimation results:

Parametric coefficients:
                     Mean      Sd    2.5%     50%   97.5%
(Intercept)       -9.2118  0.1011 -9.4162 -9.2146 -9.0000
HealthDeprivation  0.8790  0.4366  0.1308  0.8385  1.8318

Smooth terms variances:
            Mean     Sd   2.5%    50%  97.5%    Min    Max
sx(i):mrf 4.6137 1.6530 2.2749 4.3161 8.5822 1.5469 13.634

N = 32  burnin = 2000  method = MCMC  family = poisson
iterations = 12000  step = 10
```

10) by contrast with the spatial model, now we can see the Smooth terms variances.

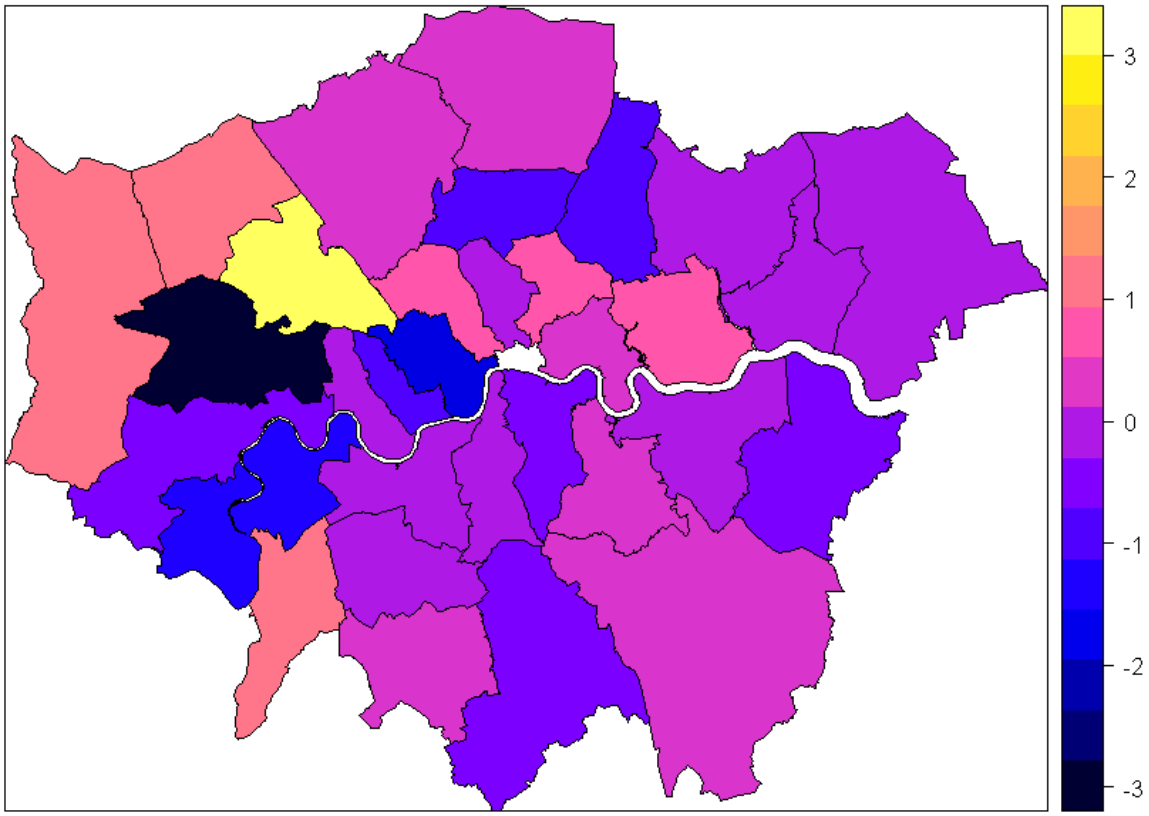
In [55]:

```
#11
# Map the fitted spatial term only
london$spatial <- fitted(flu_spatial, term = "sx(i):mrf")[, "Mean"]
spplot(london, zcol = "spatial")
```



In [56]:

```r
#12
# Map the residuals
london$spatial_resid <- residuals(flu_spatial)[, "mu"]
spplot(london, zcol = "spatial_resid")
```



In [57]:

```r
#13
# Test residuals for spatial correlation
moran.mc(london$spatial_resid, nb2listw(borough_nb), 999)
```

```
        Monte-Carlo simulation of Moran I

data:  london$spatial_resid
weights: nb2listw(borough_nb)
number of simulations + 1: 1000

statistic = -0.21692, observed rank = 46, p-value = 0.954
alternative hypothesis: greater
```

From the test above we fail to reject the null hypothesis that there is no spatial autocorrelation in the residuals, and that the model is correct, since the residuals have no spatial correlation.

5: Geostatistics

note: 3d visualizations in this section are screenshots of the pop up window output by the code

Importing GemPy

In [2]:
```
# Setting options
np.random.seed(1515)
pd.set_option('precision', 2)
```

```
Not subsurface compatibility available

WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS
functions.
```

Importing and creating a set of input data

The data used for the construction of a model in GemPy is stored in Python objects. The main data classes are:

::

- Surface_points
- Orientations
- Grid
- Surfaces
- Series
- Additional data
- Faults

We will see each of this class in further detail in the future.

Most of data can also be generated from raw data that comes in the form of CSV-files (CSV = comma-separated values). Such files might be attained by exporting model data from a different program such as GeoModeller or by simply creating it in spreadsheet software such as Microsoft Excel or LibreOffice Calc.

In this tutorial, all input data is created by importing such CSV-files. These exemplary files can be found in the `input_data` folder in the root folder of GemPy. The data comprises x-, y- and z-positional values for all surface points and orientation measurements. For the latter, poles,

azimuth and polarity are additionally included. Surface points are furthermore assigned a formation. This might be a lithological unit such as "Sandstone" or a structural feature such as "Main Fault". It is decisive to remember that, in GemPy, interface position points mark the **bottom** of a layer. If such points are needed to resemble a top of a formation (e.g. when modeling an intrusion), this can be achieved by defining a respectively inverted orientation measurement.

As we generate our `Data` from CSV-files, we also have to define our model's real extent in x, y and z, as well as declare a desired resolution for each axis. This resolution will in turn determine the number of voxels used during modeling. Here, we rely on a medium resolution of 50x50x50, amounting to 125,000 voxels. The model extent should be chosen in a way that it contains all relevant data in a representative space. As our model voxels are not cubes, but prisms, the resolution can take a different shape than the extent. We don't recommend going much higher than 100 cells in every direction (1,000,000 voxels), as higher resolutions will become increasingly expensive to compute.

In [3]:
```
geo_model = gp.create_model('Tutorial_ch1_1_Basics')
```

In [4]:
```
data_path = 'https://raw.githubusercontent.com/cgre-aachen/gempy_data/mast
# Importing the data from CSV-files and setting extent and resolution
gp.init_data(geo_model, [0, 2000., 0, 2000., 0, 750.], [50, 50, 50],
             path_o=data_path + "/data/input_data/getting_started/"
                                "simple_fault_model_orientations.csv",
             path_i=data_path + "/data/input_data/getting_started/"
                                "simple_fault_model_points.csv",
             default_values=True)
```

```
Active grids: ['regular']
```

Out[4]:
```
Tutorial_ch1_1_Basics  2023-01-21 14:01
```

In [5]:
```
geo_model.surfaces
```

Out[5]:

	surface	series	order_surfaces	color	id
0	Shale	Default series	1	#015482	1
1	Sandstone_1	Default series	2	#9f0052	2
2	Siltstone	Default series	3	#ffbe00	3
3	Sandstone_2	Default series	4	#728f02	4
4	Main_Fault	Default series	5	#443988	5
5	basement	Basement	1	#ff3f20	6

The input data can then be listed using the command `get_data`. Note that the order of formations and respective allocation to series is still completely arbitrary. We will fix this in the following.

In [6]:
```
1 gp.get_data(geo_model, 'surface_points').head()
```

Out[6]:

	X	Y	Z	smooth	surface
0	1000	50	450.00	2.00e-06	Shale
1	1000	150	433.33	2.00e-06	Shale
2	1000	300	433.33	2.00e-06	Shale
3	1000	500	466.67	2.00e-06	Shale
4	1000	1000	533.33	2.00e-06	Shale

In [7]:
```
1 gp.get_data(geo_model, 'orientations').head()
```

Out[7]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	1000	1000	300	0.32	1.00e-12	0.95	0.01	Shale
1	400	1000	420	0.32	1.00e-12	0.95	0.01	Sandstone_2
2	500	1000	300	-0.95	1.00e-12	0.32	0.01	Main_Fault

Declaring the sequential order of geological formations

In [8]:
```
1 geo_model.surfaces
```

Out[8]:

	surface	series	order_surfaces	color	id
0	Shale	Default series	1	#015482	1
1	Sandstone_1	Default series	2	#9f0052	2
2	Siltstone	Default series	3	#ffbe00	3
3	Sandstone_2	Default series	4	#728f02	4
4	Main_Fault	Default series	5	#443988	5
5	basement	Basement	1	#ff3f20	6

In [9]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Fault_Series": 'Main_Fault',
                          "Strat_Series": ('Sandstone_2', 'Siltstone',
                                           'Shale', 'Sandstone_1', 'baseme
                         remove_unused_series=True)
```

Out[9]:

	surface	series	order_surfaces	color	id
4	Main_Fault	Fault_Series	1	#443988	1
0	Shale	Strat_Series	1	#015482	2
1	Sandstone_1	Strat_Series	2	#9f0052	3
2	Siltstone	Strat_Series	3	#ffbe00	4
3	Sandstone_2	Strat_Series	4	#728f02	5
5	basement	Strat_Series	5	#ff3f20	6

In [10]:
```
geo_model.surfaces
```

Out[10]:

	surface	series	order_surfaces	color	id
4	Main_Fault	Fault_Series	1	#443988	1
0	Shale	Strat_Series	1	#015482	2
1	Sandstone_1	Strat_Series	2	#9f0052	3
2	Siltstone	Strat_Series	3	#ffbe00	4
3	Sandstone_2	Strat_Series	4	#728f02	5
5	basement	Strat_Series	5	#ff3f20	6

In [11]:
```
geo_model.stack
```

Out[11]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Erosion	True	False	False
Strat_Series	2	Erosion	True	False	False

In [12]:
```
geo_model.set_is_fault(['Fault_Series'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to F
alse.
```

Out[12]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Fault	True	True	False
Strat_Series	2	Erosion	True	False	False

In [13]:
```
1 geo_model.faults.faults_relations_df
```

Out[13]:

	Fault_Series	Strat_Series
Fault_Series	False	True
Strat_Series	False	False

In [14]:
```
1 geo_model.faults
```

Out[14]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Fault	True	True	False
Strat_Series	2	Erosion	True	False	False

In [15]:
```
1 geo_model.faults.faults_relations_df
```

Out[15]:

	Fault_Series	Strat_Series
Fault_Series	False	True
Strat_Series	False	False

Returning information from our input data

Our model input data, here named "*geo_model*", contains all the information that is essential for the construction of our model. You can access different types of information by using `gp.get_data` or simply by accessiong the atrribues.

We can, for example, return the coordinates of our modeling grid via:

In [16]:
```
1 geo_model.grid
```

Out[16]:
```
Grid Object. Values:
array([[  20. ,   20. ,    7.5],
       [  20. ,   20. ,   22.5],
       [  20. ,   20. ,   37.5],
       ...,
       [1980. , 1980. ,  712.5],
       [1980. , 1980. ,  727.5],
       [1980. , 1980. ,  742.5]])
```

As mentioned before, GemPy's core algorithm is based on interpolation of two types of data: - surface_points and - orientation measurements

(if you want to know more on how this this interpolation algorithm works, checkout our paper: https://www.geosci-model-dev.net/12/1/2019/gmd-12-1-2019.pdf (https://www.geosci-model-dev.net/12/1/2019/gmd-12-1-2019.pdf)).

We introduced the function `get\_data` above. You can also specify which kind of data you want to call, by declaring the string attribute "*dtype*" to be either `'surface_points'` (interfaces) or `'orientations'` \.

Interfaces Dataframe:

In [17]:
```
1 gp.get_data(geo_model, 'surface_points').head()
```

Out[17]:

	X	Y	Z	smooth	surface
52	700	1000	300.0	2.00e-06	Main_Fault
53	600	1000	200.0	2.00e-06	Main_Fault
54	500	1000	100.0	2.00e-06	Main_Fault
55	1000	1000	600.0	2.00e-06	Main_Fault
56	1100	1000	700.0	2.00e-06	Main_Fault

Orientations Dataframe:

In [18]:
```
1 gp.get_data(geo_model, 'orientations')
```

Out[18]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
2	500	1000	300	-0.95	1.00e-12	0.32	0.01	Main_Fault
0	1000	1000	300	0.32	1.00e-12	0.95	0.01	Shale
1	400	1000	420	0.32	1.00e-12	0.95	0.01	Sandstone_2

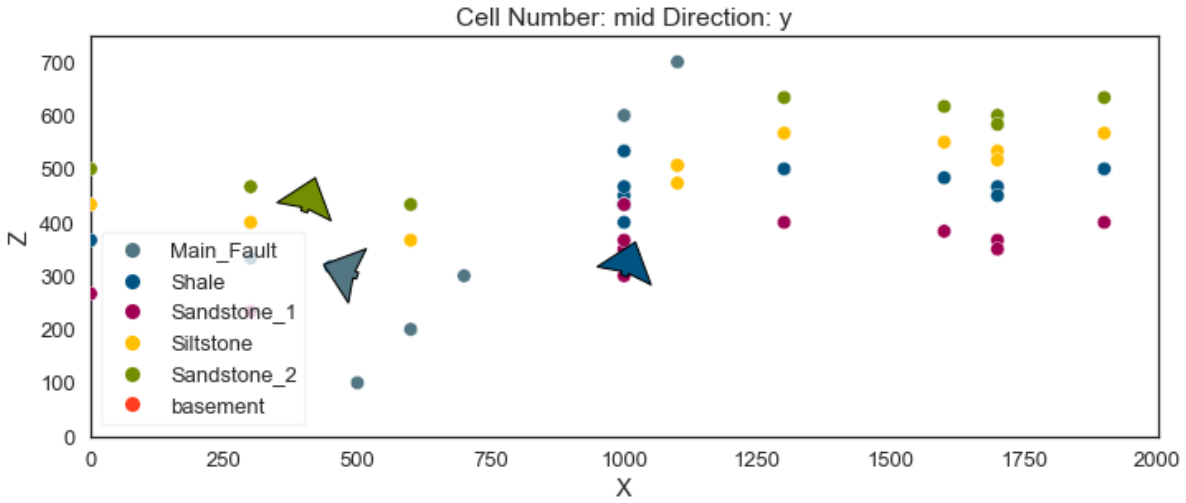
Notice that now all **surfaces** have been assigned to a **series** and are displayed in the correct order (from young to old).

Visualizing input data

We can also visualize our input data. This might for example be useful to check if all points and measurements are defined the way we want them to. Using the function `plot_data` \, we attain a 2D projection of our data points onto a plane of chosen *direction* (we can choose this attribute to be either x, y or z).

In [19]:
```
plot = gp.plot_2d(geo_model, show_lith=False, show_boundaries=False)
plt.show()
```

```
H:\Users\Administrator\lib\site-packages\gempy\plot\plot_api.py:261: UserWarn
ing: Matplotlib is currently using module://matplotlib_inline.backend_inline,
which is a non-GUI backend, so cannot show the figure.
  p.fig.show()
```



Using `plot_data_3D \`, we can also visualize this data in 3D. Note that direct 3D visualization in GemPy requires `the Visualization Toolkit <https://www.vtk.org/> __` (VTK) to be installed.

All 3D plots in GemPy are interactive. This means that we can drag and drop any data point and measurement. The perpendicular axis views in VTK are particularly useful to move points solely on a desired 2D plane. Any changes will then be stored permanently in the "InputData" dataframe. If we want to reset our data points, we will then need to reload our original input data.

Executing the cell below will open a new window with a 3D interactive plot of our data.

In [20]:
```
gpv = gp.plot_3d(geo_model, image=False, plotter_type='basic')
```

```
H:\Users\Administrator\lib\site-packages\pyvista\utilities\helpers.py:507: Us
erWarning: Points is not a float type. This can cause issues when transformin
g or applying filters. Casting to ``np.float32``. Disable this by passing ``f
orce_float=False``.
  warnings.warn(
```

Model generation

Once we have made sure that we have defined all our primary information as desired in our object `DataManagement.InputData` (named `geo_data` in these tutorials), we can continue with the next step towards creating our geological model: preparing the input data for interpolation.

This is done by generating an `InterpolatorData` object (named `interp_data` in these tutorials) from our `InputData` object via the following function:

In [21]:

```python
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    )
```

```
Setting kriging parameters to their default values.
Compiling theano function...

H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\theano\scan_module\scan_perform_ext.
py:75: UserWarning: The file scan_perform.c is not available. This donot happ
en normally. You are probably in a strangesetup. This mean Theano can not use
the cython code for scan. If youwant to remove this warning, use the Theano f
lag'cxx=' (set to an empty string) to disable all ccode generation.
  warnings.warn(
```

```
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                     values
range              2926.17
$C_o$            203869.05
drift equations     [3, 3]
```

Out[21]: `<gempy.core.interpolator.InterpolatorModel at 0x15aab923eb0>`

This function rescales the extent and coordinates of the original data (and store it in the attribute `geo_data_res` which behaves as a usual `InputData` object) and adds mathematical parameters that are needed for conducting the interpolation. The computation of this step may take a while, as it also compiles a theano function which is required for the model computation. However, should this not be needed, we can skip it by declaring `compile_theano = False` in the function.

Furthermore, this preparation process includes an assignment of numbers to each formation. Note that GemPy's always creates a default basement formation as the last formation number. Afterwards, numbers are allocated from youngest to oldest as defined by the sequence of series and formations. On the property `formations` on our interpolation data, we can find out which number has been assigned to which formation:

The parameters used for the interpolation can be returned using the function `get_kriging_parameters`. These are generated automatically from the original data, but can be changed if needed. However, users should be careful doing so, if they do not fully understand their significance.

In [54]:
```
1 gp.get_data(geo_model, 'kriging')
```

Out[54]:

	values
range	2926.17
C_o	203869.05
drift equations	[3, 3]

At this point, we have all we need to compute our full model via `compute_model`. By default, this will return two separate solutions in the form of arrays. The first gives information on the lithological formations, the second on the fault network in the model. These arrays consist of two subarrays as entries each:

1. Lithology block model solution:
    - Entry [0]: This array shows what kind of lithological formation is found in each voxel, as indicated by a respective formation_number.
    - Entry [1]: Potential field array that represents the orientation of lithological units and layers in the block model.

2. Fault network block model solution:

   - Entry [0]: Array in which all fault-separated areas of the model are represented by a distinct number contained in each voxel.
   - Entry [1]: Potential field array related to the fault network in the block model.

Below. we illustrate these different model solutions and how thev can be used.

In [55]:
```
sol = gp.compute_model(geo_model)
```

In [56]:
```
sol
```

Out[56]:
```
Lithology ids
  [   6.    6.    6. ... -100. -100. -100.]
```

In [57]:
```
geo_model.solutions
```

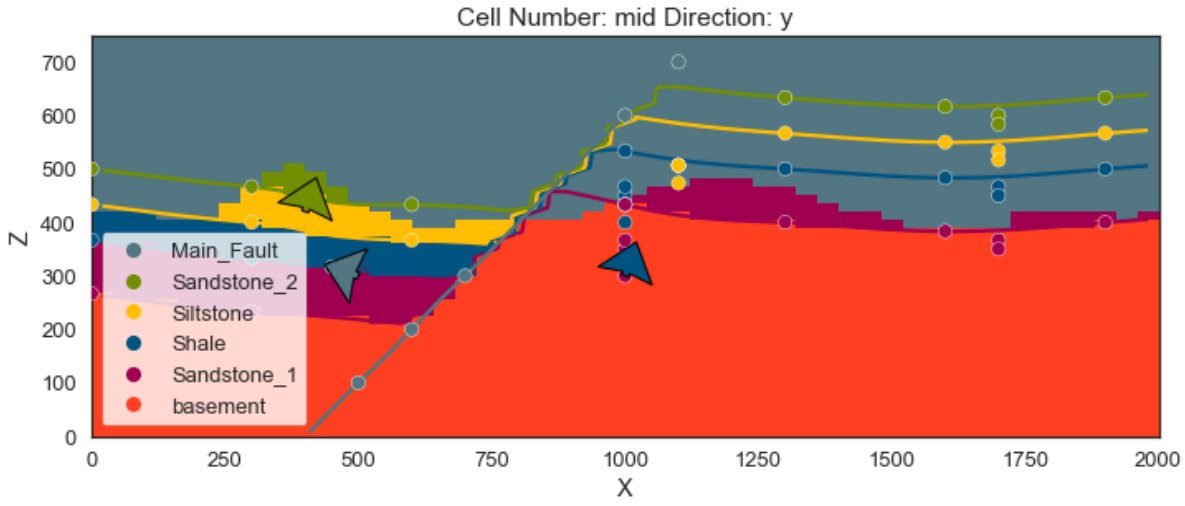
Out[57]:
```
Lithology ids
  [   6.    6.    6. ... -100. -100. -100.]
```

Direct model visualization in GemPy

Model solutions can be easily visualized in 2D sections in GemPy directly. Let's take a look at our lithology block:

In [58]:
```
gp.plot_2d(geo_model, show_data=True)
plt.show()
```

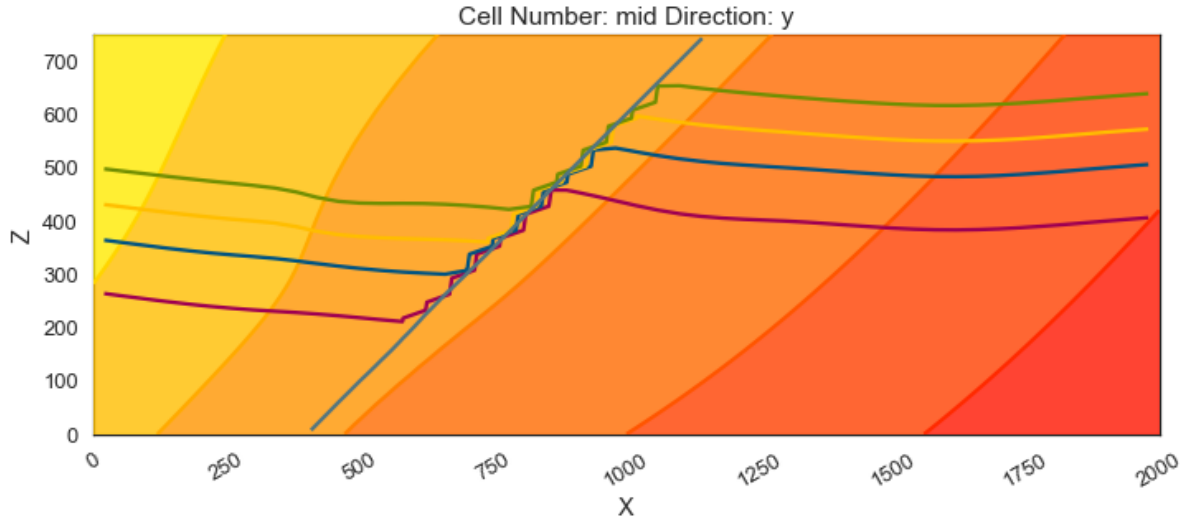
```
H:\Users\Administrator\lib\site-packages\gempy\plot\plot_api.py:261: UserWarn
ing: Matplotlib is currently using module://matplotlib_inline.backend_inline,
which is a non-GUI backend, so cannot show the figure.
  p.fig.show()
```



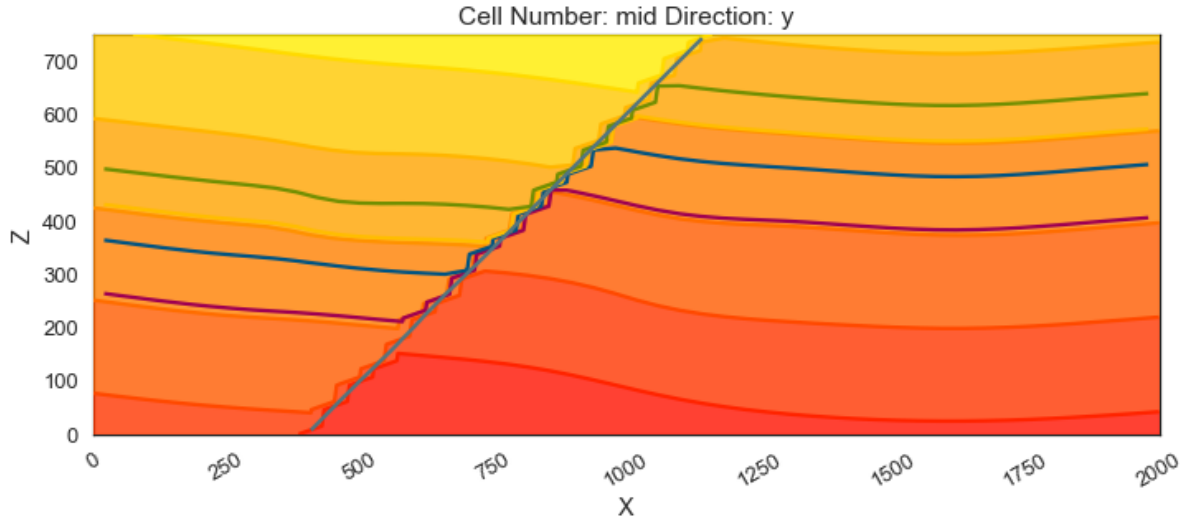
With `cell_number=25` and remembering that we defined our resolution to be 50 cells in each direction, we have chosen a section going through the middle of our block. We have moved 25 cells in `direction='y'`, the plot thus depicts a plane parallel to the x- and y-axes. Setting `plot_data=True`, we could plot original data together with the results. Changing the values for `cell_number` and `direction`, we can move through our 3D block model and explore it by looking at different 2D planes.

We can do the same with out lithological scalar-field solution:

In [59]:
```
gp.plot_2d(geo_model, show_data=False, show_scalar=True, show_lith=False)
plt.show()
```



In [60]:
```
gp.plot_2d(geo_model, series_n=1, show_data=False, show_scalar=True, show_
plt.show()
```



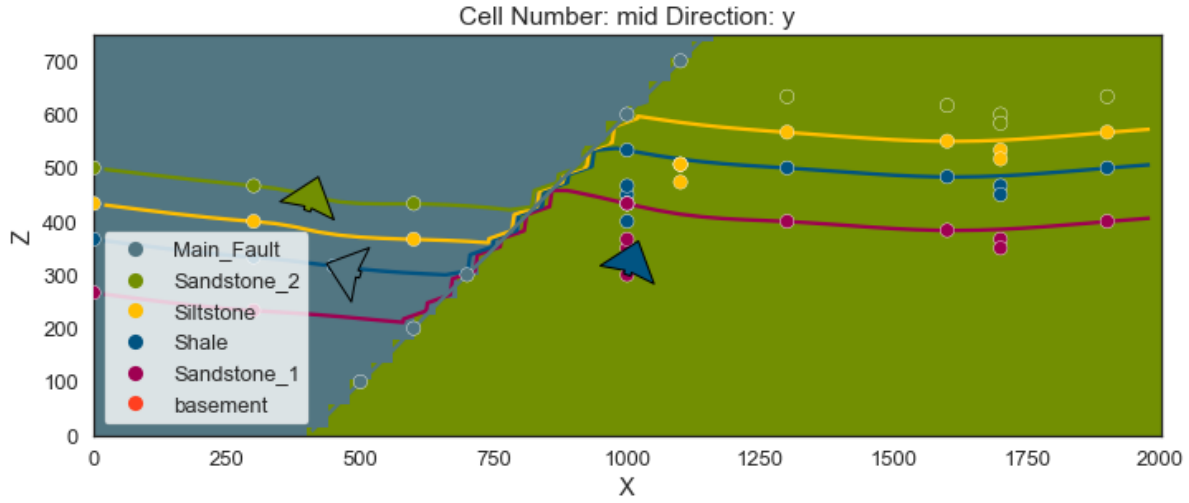
This illustrates well the fold-related deformation of the stratigraphy, as well as the way the layers are influenced by the fault.

The fault network modeling solutions can be visualized in the same way:

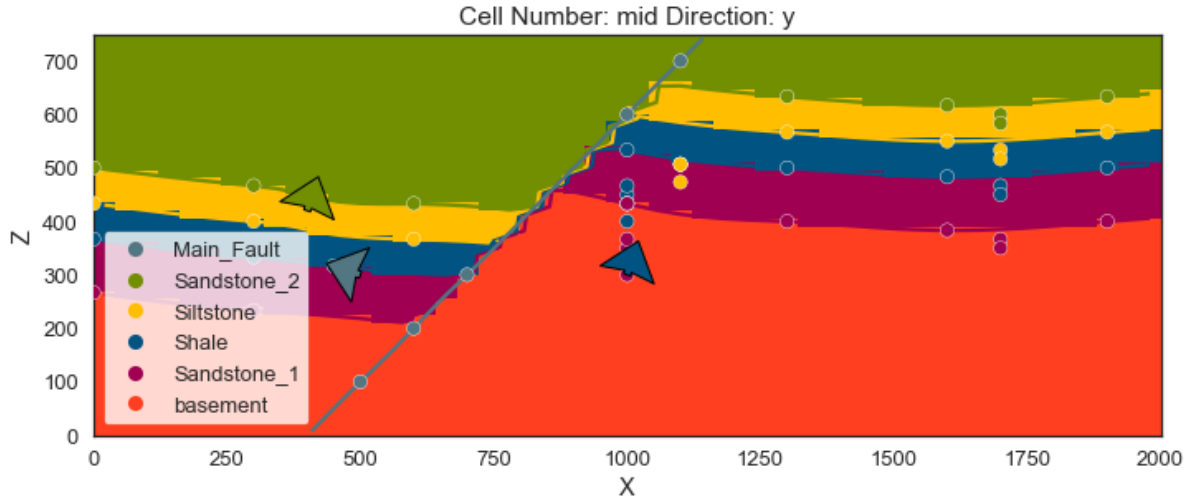
In [61]:
```
1 geo_model.solutions.scalar_field_at_surface_points
```

Out[61]:
```
array([[0.03075848, 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.83598092, 0.80357372, 0.77174354, 0.72471042]])
```

In [62]:
```
1 gp.plot_2d(geo_model, show_block=True, show_lith=False)
2 plt.show()
```



In [63]:
```
1 gp.plot_2d(geo_model, series_n=1, show_block=True, show_lith=False)
2 plt.show()
```

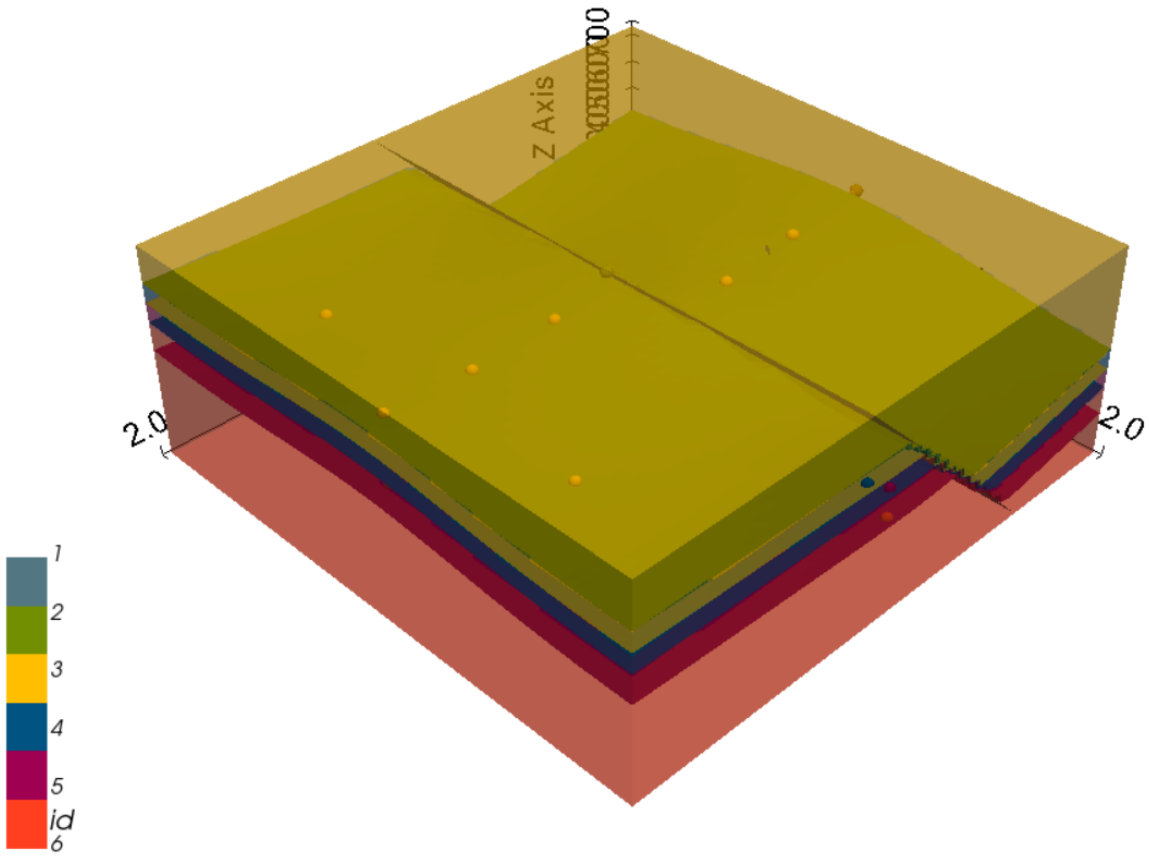


Marching cubes and vtk visualization

In addition to 2D sections we can extract surfaces to visualize in 3D renderers. Surfaces can be visualized as 3D triangle complexes in VTK (see function plot_surfaces_3D below). To create these triangles, we need to extract respective vertices and simplices from the potential fields of lithologies and faults. This process is automatized in GemPy with the function `get_surface \` .

In [64]:
```
ver, sim = gp.get_surfaces(geo_model)
gpv = gp.plot_3d(geo_model, image=False, plotter_type='basic')
```

```
H:\Users\Administrator\lib\site-packages\pyvista\plotting\tools.py:622: PyVis
taDeprecationWarning: The usage of `parse_color` is deprecated in favor of th
e new `Color` class.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\pyvista\utilities\helpers.py:507: Us
erWarning: Points is not a float type. This can cause issues when transformin
g or applying filters. Casting to ``np.float32``. Disable this by passing ``f
orce_float=False``.
  warnings.warn(
```



Using the rescaled interpolation data, we can also run our 3D VTK visualization in an interactive mode which allows us to alter and update our model in real time. Similarly to the interactive 3D visualization of our input data, the changes are permanently saved (in the `InterpolationInput.dataframe` object). Additionally, the resulting changes in the geological models are re-computed in real time.

Adding topography

In [65]:
```
1 geo_model.set_topography(d_z=(350, 750))
```

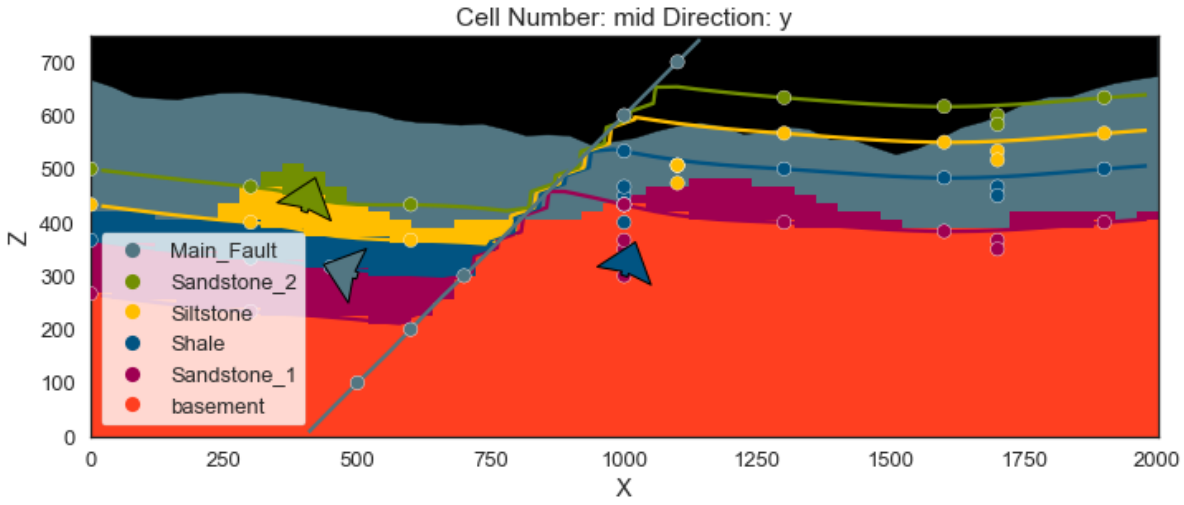
```
Active grids: ['topography']
```

Out[65]:
```
Grid Object. Values:
array([[   0.        ,    0.        ,  674.04326366],
       [   0.        ,   40.81632653,  651.65388764],
       [   0.        ,   81.63265306,  633.17839754],
       ...,
       [2000.        , 1918.36734694,  727.36375607],
       [2000.        , 1959.18367347,  711.62311727],
       [2000.        , 2000.        ,  699.1870782 ]])
```

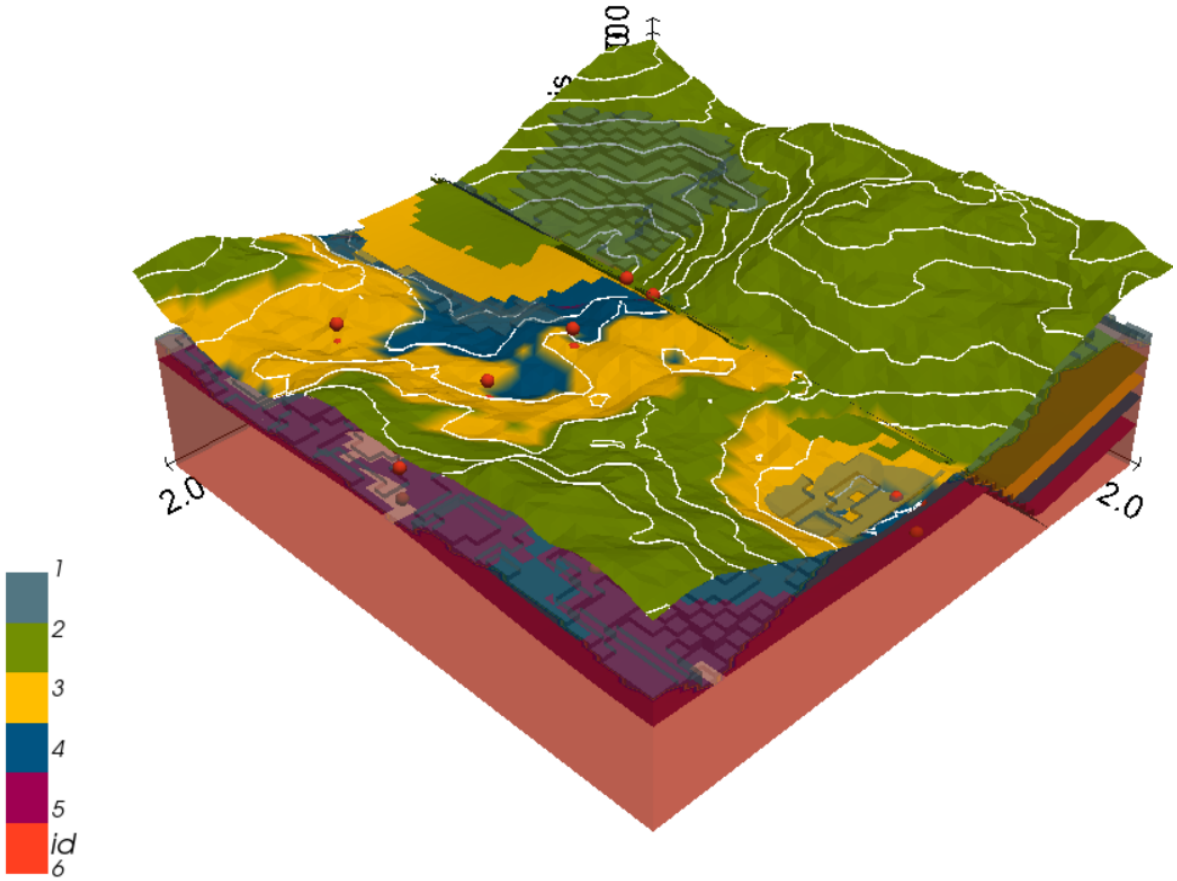
In [66]:

```
gp.compute_model(geo_model)
gp.plot_2d(geo_model, show_topography=True)
plt.show()


# sphinx_gallery_thumbnail_number = 9
gpv = gp.plot_3d(geo_model, plotter_type='basic', show_topography=True, sh
                 show_lith=True,
                 image=False)
```

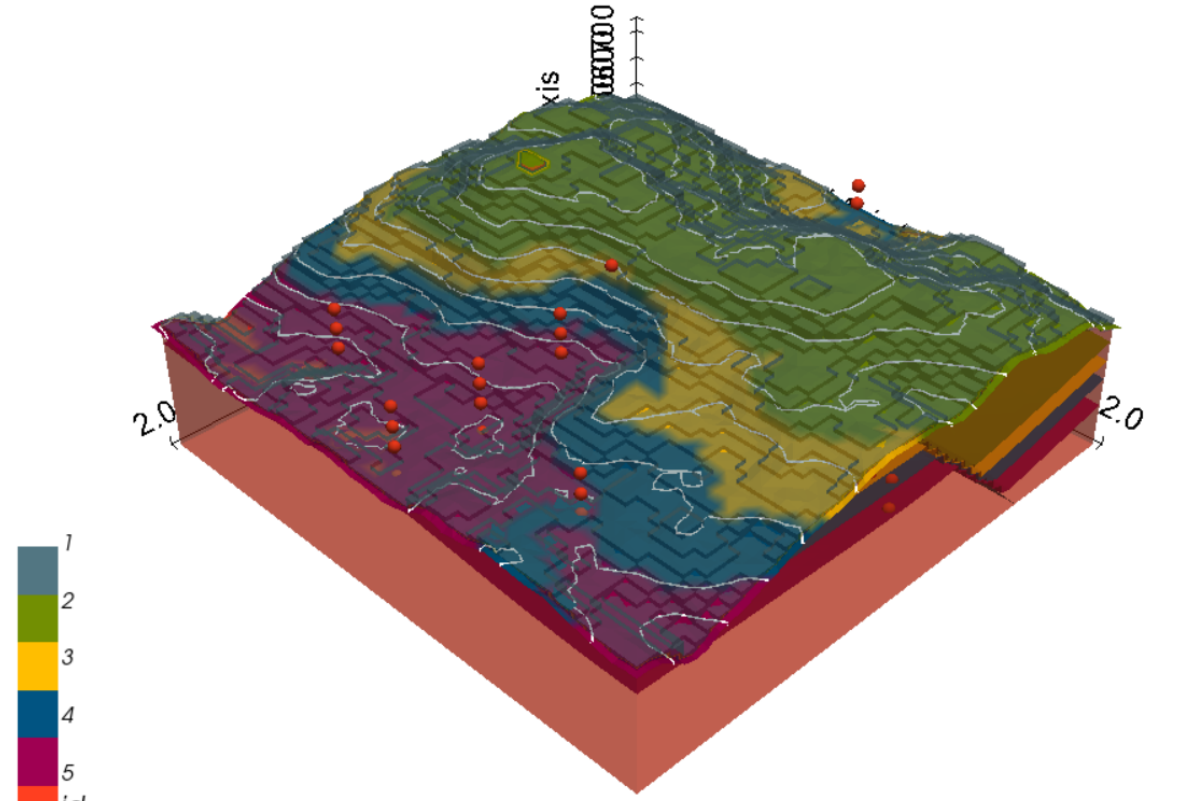
```
H:\Users\Administrator\lib\site-packages\gempy\core\solution.py:180: VisibleD
eprecationWarning: Creating an ndarray from ragged nested sequences (which is
a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shap
es) is deprecated. If you meant to do this, you must specify 'dtype=object' w
hen creating the ndarray.
  self.geological_map = np.array(
H:\Users\Administrator\lib\site-packages\gempy\plot\plot_api.py:261: UserWarn
ing: Matplotlib is currently using module://matplotlib_inline.backend_inline,
which is a non-GUI backend, so cannot show the figure.
  p.fig.show()
```



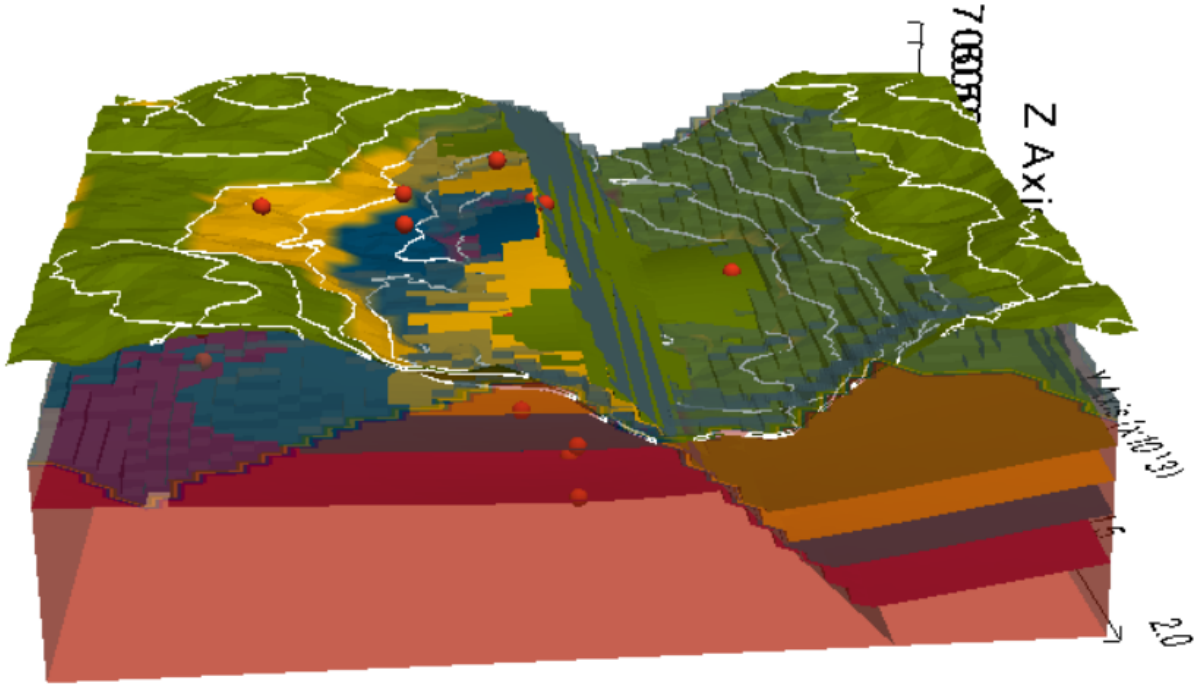

```
H:\Users\Administrator\lib\site-packages\pyvista\plotting\tools.py:622: PyVis
taDeprecationWarning: The usage of `parse_color` is deprecated in favor of th
e new `Color` class.
  warnings.warn(
H:\Users\Administrator\lib\site-packages\pyvista\utilities\helpers.py:507: Us
erWarning: Points is not a float type. This can cause issues when transformin
g or applying filters. Casting to ``np.float32``. Disable this by passing ``f
orce_float=False``.
  warnings.warn(
```


2.0
2.0

1
2
3
4
5
id
6


2.0
2.0

1
2
3
4
5
id
6



Compute at a given location

This is done by modifying the grid to a custom grid and recomputing. Notice that the results are given as *grid + surfaces_points_ref + surface_points_rest locations*

In [52]:
```
x_i = np.array([[3, 5, 6]])
sol = gp.compute_model(geo_model, at=x_i)
```

```
Active grids: []
```

Therefore if we just want the value at **x_i**:

In [53]:
```
sol.custom
```

Out[53]:
```
array([array([[6.]]), array([[0.18630133],
                             [0.63163565]])], dtype=object)
```

This return the id, and the scalar field values for each series