Aerogram, TBIU IIT Delhi
Research Internship Project Report

By Peya Mowar and Mini Jain

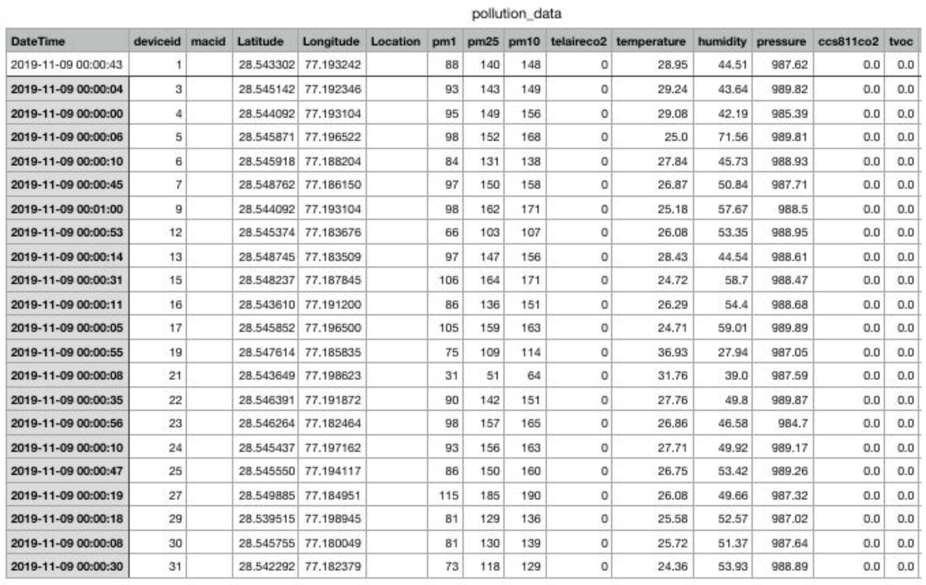
Introduction

About Aerogram:

Aerogram, a startup led by Dr Sarita Ahlawat with alumni of IIT-Delhi, is devising a network to predict real time air quality in a local mapped area. The designed device is equipped to monitor multiple pollutants. The startup is aligned with its vision to be the leading provider of data on air quality and be the most preferred air pollution management organization. The immediate aim is provisioning of real-time local air quality data based on micro-mapping of defined air pollutants to enable masses in taking timely and precise actions to limit exposure. Also, the data is to be leveraged by concerned authorities to help adopt suitable strategies to reduce/remove the sources of pollution. In order to achieve accuracy and efficiency in achieving the mission, Aerogram designs and builds its own low-cost AQM devices which are used to establish a robust network wherein the devices are technically capable of smooth communication amongst each other which allows effective calibration. The accuracy of data achieved via a robust network enables conducting a more in-depth analytics to generate reports and information which is shared with the concerned users.

The data received at the server from the sensors planted in the IIT campus was downloaded in CSV format as follows:

Pollution_data.csv :

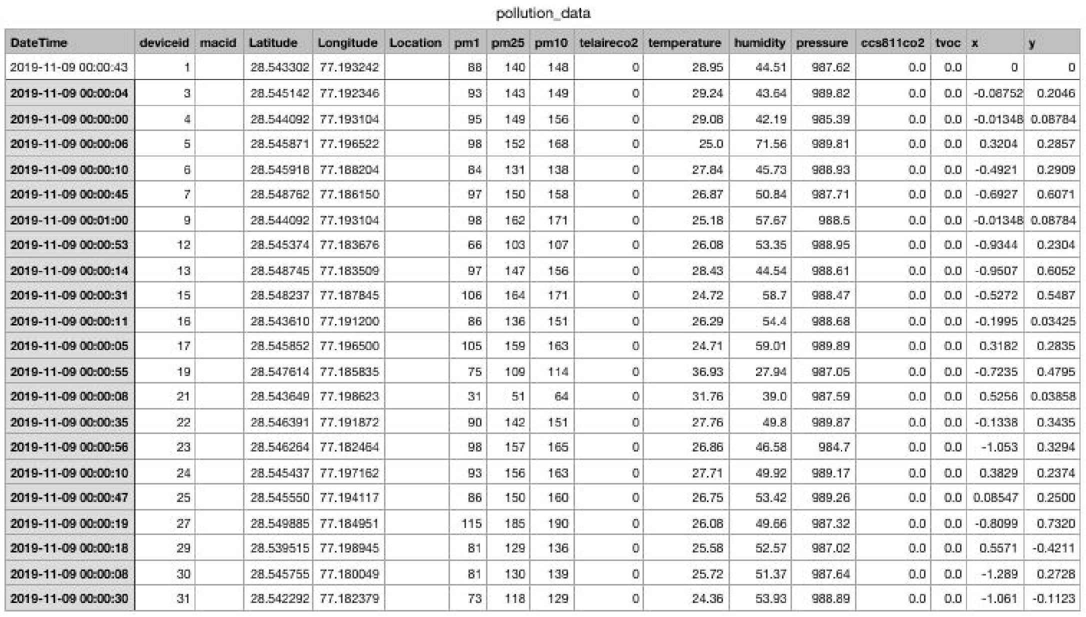


pollution_data

DateTime	deviceid	macid	Latitude	Longitude	Location	pm1	pm25	pm10	telaireco2	temperature	humidity	pressure	ccs811co2	tvoc
2019-11-09 00:00:43	1		28.543302	77.193242		88	140	148	0	28.95	44.51	987.62	0.0	0.0
2019-11-09 00:00:04	3		28.545142	77.192346		93	143	149	0	29.24	43.64	989.82	0.0	0.0
2019-11-09 00:00:00	4		28.544092	77.193104		95	149	156	0	29.08	42.19	985.39	0.0	0.0
2019-11-09 00:00:06	5		28.545871	77.196522		98	152	168	0	25.0	71.56	989.81	0.0	0.0
2019-11-09 00:00:10	6		28.545918	77.188204		84	131	138	0	27.84	45.73	988.93	0.0	0.0
2019-11-09 00:00:45	7		28.548762	77.186150		97	150	158	0	26.87	50.84	987.71	0.0	0.0
2019-11-09 00:01:00	9		28.544092	77.193104		98	162	171	0	25.18	57.67	988.5	0.0	0.0
2019-11-09 00:00:53	12		28.545374	77.183676		66	103	107	0	26.08	53.35	988.95	0.0	0.0
2019-11-09 00:00:14	13		28.548745	77.183509		97	147	156	0	28.43	44.54	988.61	0.0	0.0
2019-11-09 00:00:31	15		28.548237	77.187845		106	164	171	0	24.72	58.7	988.47	0.0	0.0
2019-11-09 00:00:11	16		28.543610	77.191200		86	136	151	0	26.29	54.4	988.68	0.0	0.0
2019-11-09 00:00:05	17		28.545852	77.196500		105	159	163	0	24.71	59.01	989.89	0.0	0.0
2019-11-09 00:00:55	19		28.547614	77.185835		75	109	114	0	36.93	27.94	987.05	0.0	0.0
2019-11-09 00:00:08	21		28.543649	77.198623		31	51	64	0	31.76	39.0	987.59	0.0	0.0
2019-11-09 00:00:35	22		28.546391	77.191872		90	142	151	0	27.76	49.8	989.87	0.0	0.0
2019-11-09 00:00:56	23		28.546264	77.182464		98	157	165	0	26.86	46.58	984.7	0.0	0.0
2019-11-09 00:00:10	24		28.545437	77.197162		93	156	163	0	27.71	49.92	989.17	0.0	0.0
2019-11-09 00:00:47	25		28.545550	77.194117		86	150	160	0	26.75	53.42	989.26	0.0	0.0
2019-11-09 00:00:19	27		28.549885	77.184951		115	185	190	0	26.08	49.66	987.32	0.0	0.0
2019-11-09 00:00:18	29		28.539515	77.198945		81	129	136	0	25.58	52.57	987.02	0.0	0.0
2019-11-09 00:00:08	30		28.545755	77.180049		81	130	139	0	25.72	51.37	987.64	0.0	0.0
2019-11-09 00:00:30	31		28.542292	77.182379		73	118	129	0	24.36	53.93	988.89	0.0	0.0

To plot this data into a 2D plot, x and y or cartesian coordinates for each point were found by taking the geographical coordinates of the first point as reference or origin using haversine formula. Tool used to calculate these is : https://www.movable-type.co.uk/scripts/latlong.html
These cartesian coordinates were found manually using this tool and added to the excel sheet in the form of two columns x and y as follows:

Pollution_data.csv : (Updated)



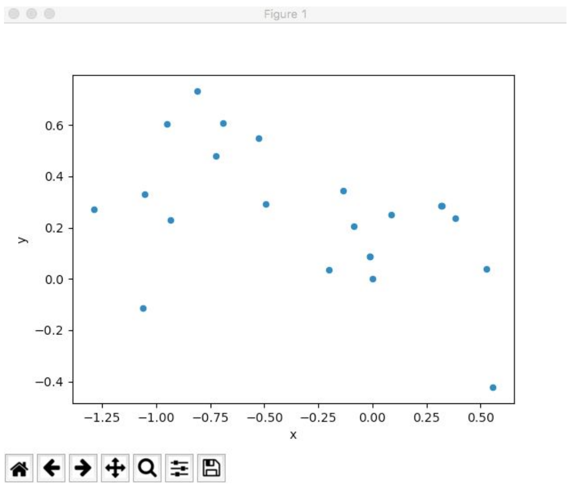
pollution_data

DateTime	deviceid	macid	Latitude	Longitude	Location	pm1	pm25	pm10	telaireco2	temperature	humidity	pressure	ccs811co2	tvoc	x	y
2019-11-09 00:00:43	1		28.543302	77.193242		88	140	148	0	28.95	44.51	987.62	0.0	0.0	0	0
2019-11-09 00:00:04	3		28.545142	77.192346		93	143	149	0	29.24	43.64	989.82	0.0	0.0	-0.08752	0.2046
2019-11-09 00:00:00	4		28.544092	77.193104		95	149	156	0	29.08	42.19	985.39	0.0	0.0	-0.01348	0.08784
2019-11-09 00:00:06	5		28.545871	77.196522		98	152	168	0	25.0	71.56	989.81	0.0	0.0	0.3204	0.2857
2019-11-09 00:00:10	6		28.545918	77.188204		84	131	138	0	27.84	45.73	988.93	0.0	0.0	-0.4921	0.2909
2019-11-09 00:00:45	7		28.548762	77.186150		97	150	158	0	26.87	50.84	987.71	0.0	0.0	-0.6927	0.6071
2019-11-09 00:01:00	9		28.544092	77.193104		98	162	171	0	25.18	57.67	988.5	0.0	0.0	-0.01348	0.08784
2019-11-09 00:00:53	12		28.545374	77.183676		66	103	107	0	26.08	53.35	988.95	0.0	0.0	-0.9344	0.2304
2019-11-09 00:00:14	13		28.548745	77.183509		97	147	156	0	28.43	44.54	988.61	0.0	0.0	-0.9507	0.6052
2019-11-09 00:00:31	15		28.548237	77.187845		106	164	171	0	24.72	58.7	988.47	0.0	0.0	-0.5272	0.5487
2019-11-09 00:00:11	16		28.543610	77.191200		86	136	151	0	26.29	54.4	988.68	0.0	0.0	-0.1995	0.03425
2019-11-09 00:00:05	17		28.545852	77.196500		105	159	163	0	24.71	59.01	989.89	0.0	0.0	0.3182	0.2835
2019-11-09 00:00:55	19		28.547614	77.185835		75	109	114	0	36.93	27.94	987.05	0.0	0.0	-0.7235	0.4795
2019-11-09 00:00:08	21		28.543649	77.198623		31	51	64	0	31.76	39.0	987.59	0.0	0.0	0.5256	0.03858
2019-11-09 00:00:35	22		28.546391	77.191872		90	142	151	0	27.76	49.8	989.87	0.0	0.0	-0.1338	0.3435
2019-11-09 00:00:56	23		28.546264	77.182464		98	157	165	0	26.86	46.58	984.7	0.0	0.0	-1.053	0.3294
2019-11-09 00:00:10	24		28.545437	77.197162		93	156	163	0	27.71	49.92	989.17	0.0	0.0	0.3829	0.2374
2019-11-09 00:00:47	25		28.545550	77.194117		86	150	160	0	26.75	53.42	989.26	0.0	0.0	0.08547	0.2500
2019-11-09 00:00:19	27		28.549885	77.184951		115	185	190	0	26.08	49.66	987.32	0.0	0.0	-0.8099	0.7320
2019-11-09 00:00:18	29		28.539515	77.198945		81	129	136	0	25.58	52.57	987.02	0.0	0.0	0.5571	-0.4211
2019-11-09 00:00:08	30		28.545755	77.180049		81	130	139	0	25.72	51.37	987.64	0.0	0.0	-1.289	0.2728
2019-11-09 00:00:30	31		28.542292	77.182379		73	118	129	0	24.36	53.93	988.89	0.0	0.0	-1.061	-0.1123

Once the cartesian coordinates of the points were found, then a scatter plot was generated for these points to visualize the data using the following python code:

plot_pollution.py

```
import pandas
import os
absolute_path = os.path.abspath(os.path.dirname('pollution_data.csv'))
fh = pandas.read_csv(absolute_path + '/pollution_data.csv')
import matplotlib.pyplot as plt
fh.plot(kind="scatter", x="x", y="y")
plt.show()
```



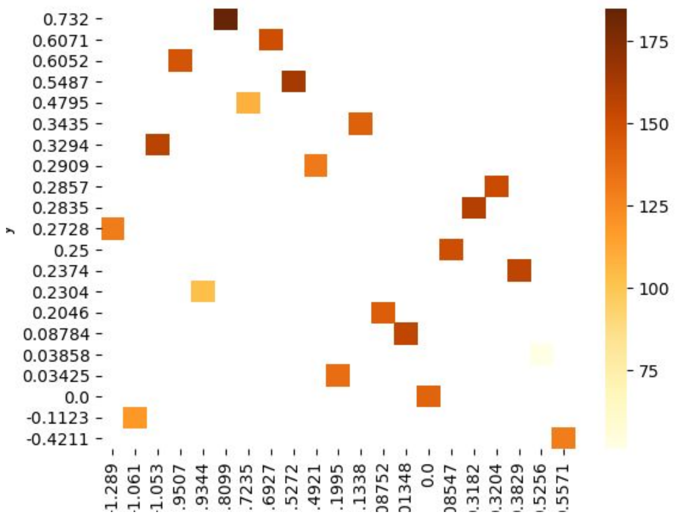
Source: https://www.bigendiandata.com/2017-06-27-Mapping_in_Jupyter/

After the generation of a scatter plot, a heatmap was generated using the Seaborn library in Python. A **heatmap** is a graphical representation of data that uses a system of color-coding to represent different values. **Heatmaps** are used in various forms of analytics but are most commonly used to show user behavior on specific webpages or web page templates.

Sources: https://blog.quantinsti.com/creating-heatmap-using-python-seaborn/
https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
The following code was used to implement the same.

heatmap.py

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
absolute_path = os.path.abspath(os.path.dirname('pollution_data.csv'))
df = pd.read_csv(absolute_path + '/pollution_data.csv')
df1 = df[['deviceid', 'pm25','x', 'y']]
#print(df1.head())
heatmap1_data = pd.pivot_table(df1, values='pm25', index='y', columns = 'x')
fig = sns.heatmap(heatmap1_data, cmap="YlOrBr")
fig.invert_yaxis()
plt.show()
```

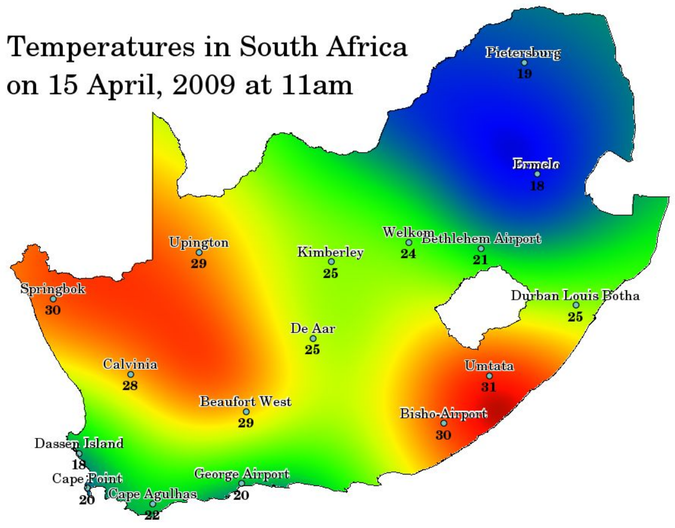


Till now, the data provided to us was modified for our reference and plots were developed for better visualization of the data. Further on, the focus will be on predictive analysis in the form of Spatial Interpolation.

Spatial Interpolation & Data Visualisation

Spatial analysis is the process of manipulating spatial information to extract new information and meaning from the original data. Usually, spatial analysis is carried out with a Geographic Information System (GIS). A GIS usually provides spatial analysis tools for calculating feature statistics and carrying out geoprocessing activities as data interpolation.
It can be used to study geographical patterns with respect to rainfall, pollution, soil composition, etc.

It is the process of using points with known values to estimate values at other unknown points. For example, to make a precipitation (rainfall) map for your country, you will not find enough evenly spread weather stations to cover the entire region. Spatial interpolation can estimate the temperatures at locations without recorded data by using known temperature readings at nearby weather stations. This type of interpolated surface is often called a **statistical surface**. Elevation data, precipitation, snow accumulation, water table, and population density are other types of data that can be computed using interpolation.



Temperature map interpolated from South African Weather Stations.

"Everything is related to everything else, but near things are more related than distant things."
- Waldo Tobler (1970)
This is the basic premise behind interpolation, and near points generally receive higher weights than far away points

The different interpolation methods explored were:
-IDW (Inverse Distance Weighting)
-TIN (Triangulated Irregular Network)
-Spline
-Kriging

References:
https://nptel.ac.in/courses/105107155/
https://mgimond.github.io/Spatial/spatial-interpolation.html
https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/an-overview-of-the-interpolation-tools.htm
http://www.gisresources.com/types-interpolation-methods_3/
https://www.intechopen.com/books/applications-of-spatial-statistics/comparison-of-spatial-interpolation-techniques-using-visualization-and-quantitative-assessment
http://www.bisolutions.us/A-Brief-Introduction-to-Spatial-Interpolation.php

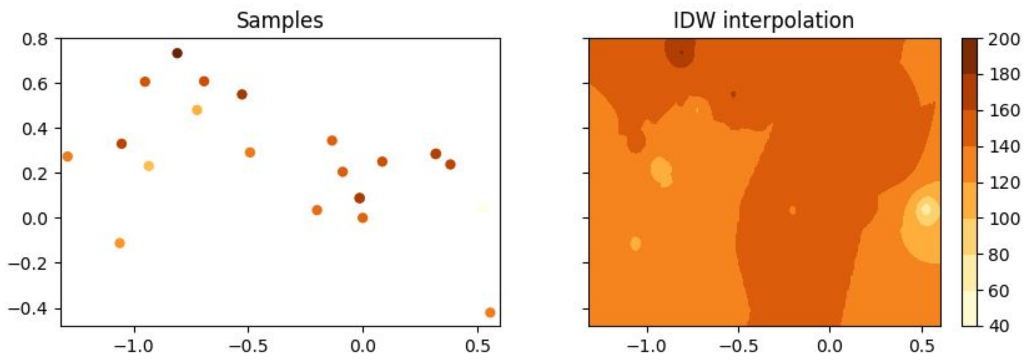
In the **IDW interpolation method**, the sample points are weighted during interpolation such that the influence of one point relative to another declines with distance from the unknown point you want to create. Weighting is assigned to sample points through the use of a weighting coefficient that controls how the weighting influence will drop off as the distance from new point increases. The greater the weighting coefficient, the less the effect points will have if they are far from the unknown point during the interpolation process. As the coefficient increases, the value of the unknown point approaches the value of the nearest observational point.
It is important to notice that the IDW interpolation method also has some disadvantages: the quality of the interpolation result can decrease, if the distribution of sample data points is uneven. Furthermore, maximum and minimum values in the interpolated surface can only occur at sample data points. This often results in small peaks and pits around the sample data points.

A suitable IDW code was found and tested for our pollution data and the results were studied.

The IDW code used was:

IDW.py
```
import numpy as np
from scipy.spatial import cKDTree
import pandas as pd
import matplotlib.pyplot as plt
import os
absolute_path = os.path.abspath(os.path.dirname('pollution_data.csv'))
df = pd.read_csv(absolute_path + '/pollution_data.csv')
df1 = df[['deviceid', 'pm25','x', 'y']]
class tree(object):
    def __init__(self, X=None, z=None, leafsize=10):
        if not X is None:
            self.tree = cKDTree(X, leafsize=leafsize )
        if not z is None:
            self.z = np.array(z)
    def fit(self, X=None, z=None, leafsize=10):
        return self.__init__(X, z, leafsize)
```

```
  def __call__(self, X, k=6, eps=1e-6, p=2, regularize_by=1e-9):
      self.distances, self.idx = self.tree.query(X, k, eps=eps, p=p)
      self.distances += regularize_by
      weights = self.z[self.idx.ravel()].reshape(self.idx.shape)
          mw = np.sum(weights/self.distances, axis=1) / np.sum(1./self.distances, axis=1)
      return mw
   def transform(self, X, k=23, p=2, eps=1e-6, regularize_by=1e-9):
      return self.__call__(X, k, eps, p, regularize_by)
data = df1.values
X1 = data[:,2:4]
z1 = data[:,1]
idw_tree = tree(X1, z1)
spacing1 = np.linspace(-1.32,0.6,200)
spacing2 = np.linspace(-0.48,0.8,100)
X2 = np.meshgrid(spacing1, spacing2)
grid_shape = X2[0].shape
X2 = np.reshape(X2, (2, -1)).T
z2 = idw_tree(X2)
fig, (ax2, ax3) = plt.subplots(1,2, sharex=True, sharey=True, figsize=(10,3))
ax2.scatter(X1[:,0], X1[:,1], c=z1, linewidths=0, cmap="YlOrBr")
ax2.set_title('Samples')
ax3.contourf(spacing1, spacing2, z2.reshape(grid_shape), cmap="YlOrBr")
im=ax3.contourf(spacing1, spacing2, z2.reshape(grid_shape), cmap="YlOrBr")
ax3.set_title('IDW interpolation')
plt.colorbar(im)
plt.show()
```



Source: https://github.com/paulbrodersen/inverse_distance_weighting

A **3D surface plot** for the pollution data was then generated using the Mayavi library on the Enthought Canopy Platform using the following python code :

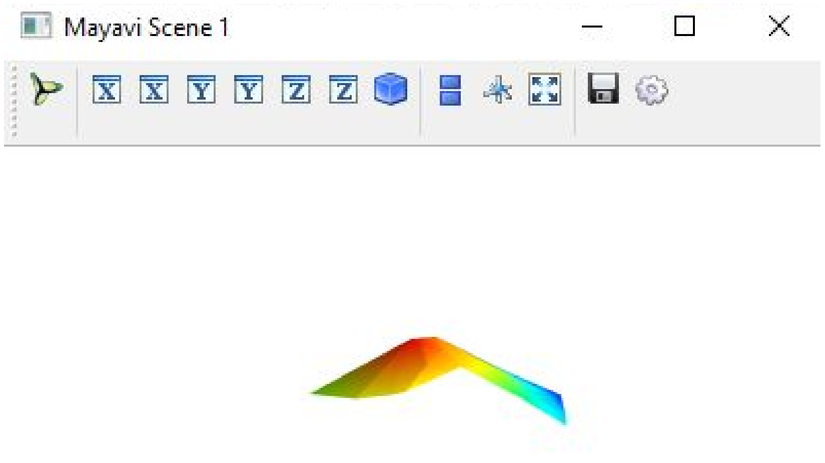
3dplotting.py
```
import numpy as np
import pandas as pd
from mayavi import mlab
df=pd.read_csv('C:\\Users\\minij\\Desktop\\Aerogram\\pollution_data.csv')
data=df.values
X1=data[:,15]
Y1=data[:,16]
X1=np.array(X1, dtype='float64')
Y1=np.array(Y1, dtype='float64')
#X1=np.reshape(X1,-1)
Z1=data[:,7]
Z1=np.array(Z1)
def f(x, y):
        return np.exp(-(x ** 2 + y ** 2))
Z1=f(X1,Y1)
mlab.figure(1, fgcolor=(0, 0, 0), bgcolor=(1, 1, 1))
# Visualize the points
pts = mlab.points3d(X1,Y1,Z1,Z1, scale_mode='none', scale_factor=0.2)
# Create and visualize the mesh
mesh = mlab.pipeline.delaunay2d(pts)
surf = mlab.pipeline.surface(mesh)
pts.remove()
mlab.view(47, 57, 8.2, (0.1, 0.15, 0.14))
mlab.show()
```

Source:https://docs.enthought.com/mayavi/mayavi/auto/example_surface_from_irregular_data.html#example-surface-from-irregular-data

Mayavi2 is a general-purpose, cross-platform tool for 3-D scientific data visualization. Its features include:

- Visualization of scalar, vector and tensor data in 2 and 3 dimensions.
- Easy scriptability using Python.
- Easy extensibility via custom sources, modules, and data filters.
- Reading several file formats: VTK (legacy and XML), PLOT3D, etc.
- Saving of visualizations.
- Saving rendered visualization in a variety of image formats.
- Convenient functionality for rapid scientific plotting via mlab

Enthought Canopy provides Python 2.7 and 3.5, with easy installation and updates via a graphical package manager of over 450 pre-built and tested scientific and analytic Python packages from the Enthought Python Distribution. These include NumPy, Pandas, SciPy, matplotlib, scikit-learn, and Jupyter / IPython. Canopy also provides an integrated analysis environment, with an editor, IPython console, Data Import Tool, debugger, and a documentation browser.

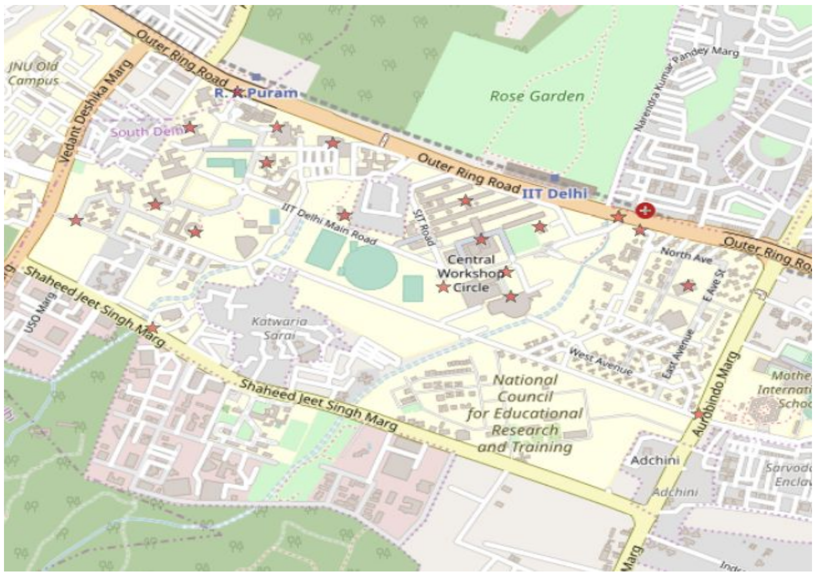


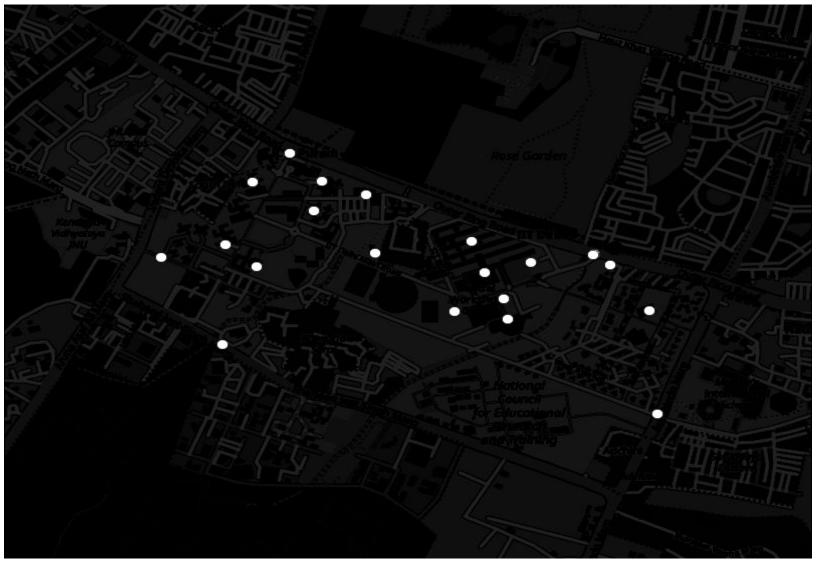
Then, a GIS software called QGIS was chosen to be explored. Further plotting of maps has been done in the QGIS software.

A Geographic Information System is a system designed to capture, store, manipulate, analyze, manage, and present spatial or geographic data. GIS applications are tools that allow users to create interactive queries, analyze spatial information, edit data in maps, and present the results of all these operations.
Reference: https://nptel.ac.in/courses/105107155/

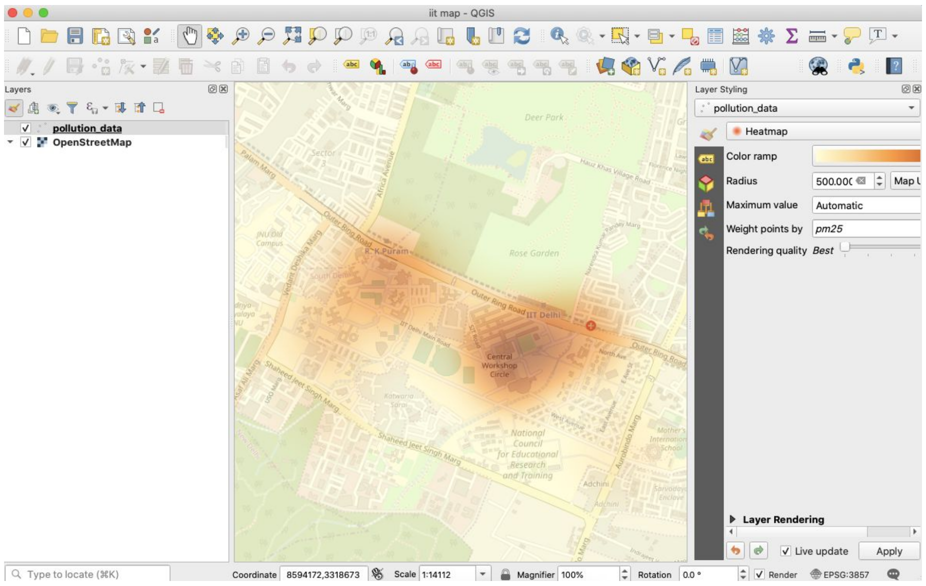
QGIS is a free and open-source cross-platform desktop geographic information system application that supports viewing, editing, and analysis of geospatial data.
Reference: https://qgis.org/en/site/

Plotting our points on a map using QGIS:





Heatmap plotting in QGIS:



Layer Styling
pollution_data
Heatmap

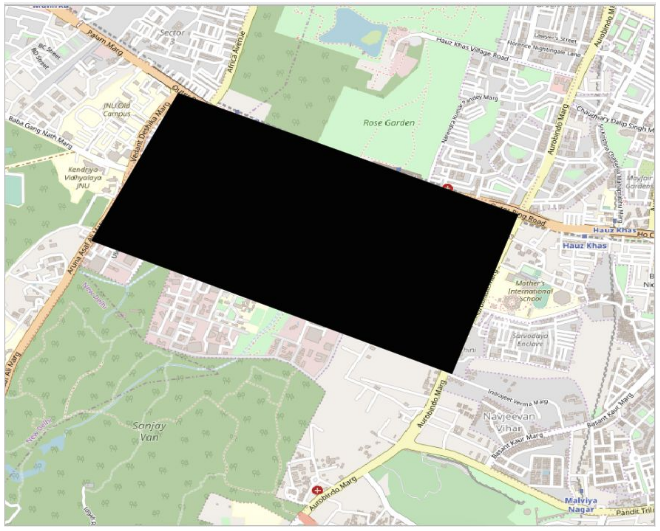
Color ramp	
Radius	500.000
Maximum value	Automatic
Weight points by	pm25
Rendering quality	Best

Layer Rendering

Coordinate 8594172,3318673 Scale 1:14112 Magnifier 100% Rotation 0.0 ° Render EPSG:3857

Spatial interpolation was performed in QGIS:

The points and the area of the IIT Campus were extracted as a shape file (.shp). Using these as the bottom most layers, different interpolation techniques were implemented in QGIS which presented themselves as layers over this shape file layer.
(Test.shp and Points.shp)



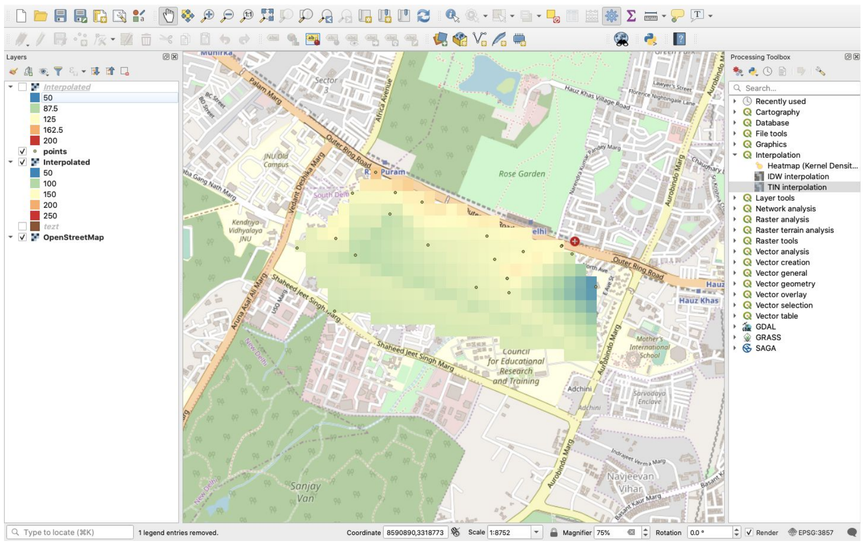
TIN interpolation:

TIN interpolation is another popular tool in GIS. A common TIN algorithm is called **Delaunay triangulation**. It tries to create a surface formed by triangles of nearest neighbour points. To do this, circumcircles around selected sample points are created and their intersections are connected to a network of non overlapping and as compact as possible triangles.
The main disadvantage of the TIN interpolation is that the surfaces are not smooth and may give a jagged appearance. This is caused by discontinuous slopes at the triangle edges and sample data points. In addition, triangulation is generally not suitable for extrapolation beyond the area with collected sample data points.

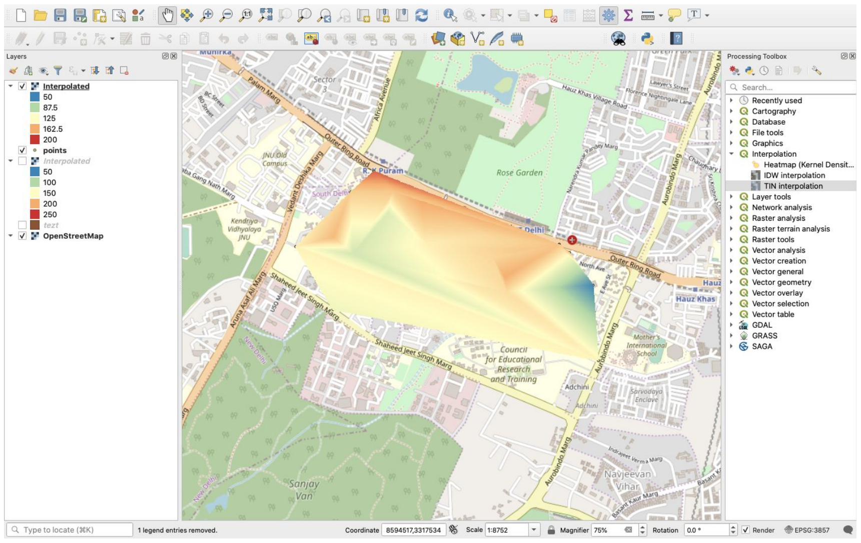
Triangular irregular networks (TIN) have been used by the GIS community for many years and are a digital means to represent surface morphology. TINs are a form of vector-based digital geographic data and are constructed by triangulating a set of vertices (points). The vertices are connected with a series of edges to form a network of triangles. There are different methods of interpolation to form these triangles, such as Delaunay triangulation or distance ordering. The resulting triangulation satisfies the Delaunay triangle criterion, which ensures that no vertex lies within the interior of any of the circumcircles of the triangles in the network. If the Delaunay criterion is satisfied everywhere on the TIN, the minimum interior angle of all triangles is maximized. The result is that long, thin triangles are avoided as much as possible.

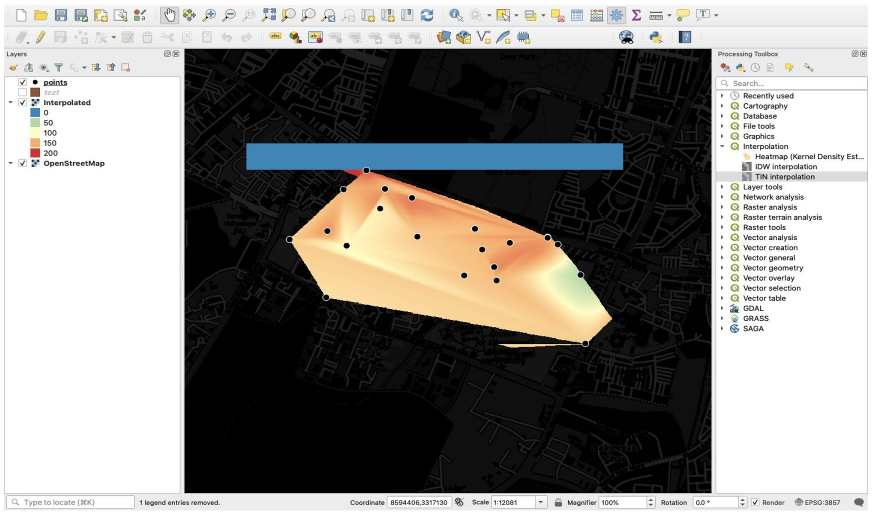
References:
https://desktop.arcgis.com/en/arcmap/10.3/manage-data/tin/fundamentals-of-tin-surfaces.htm

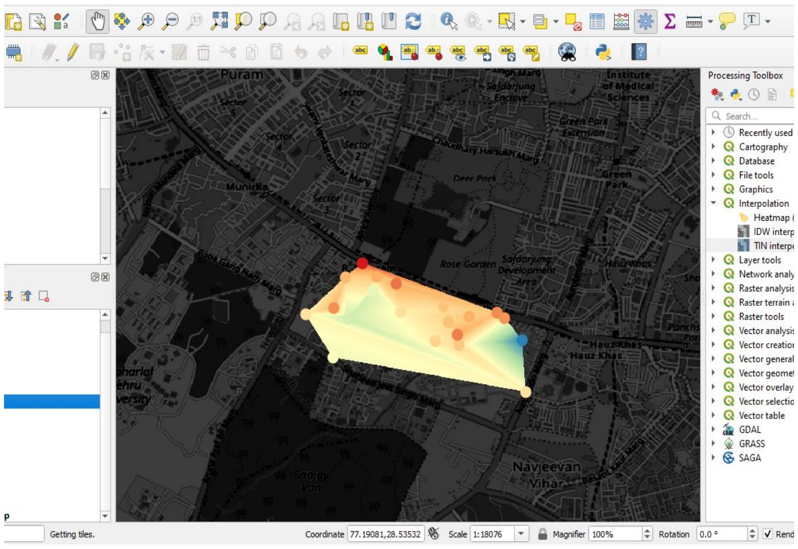
1. (Number of rows set to 23)



2. (Number of rows set to 1000)

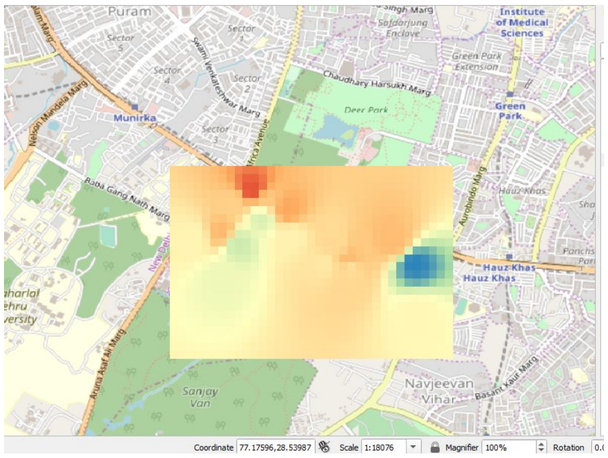




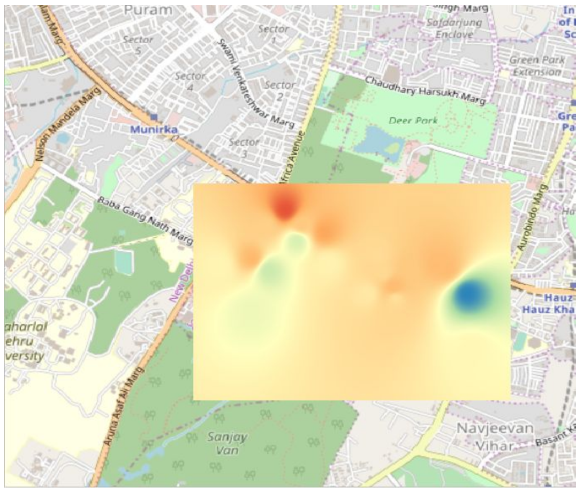


IDW interpolation in QGIS:
interpolation.qgis

1. (Number of rows=23, distance=3)



2. (Number of rows=1000, distance=3)



Kriging was then explored as an interpolation technique.

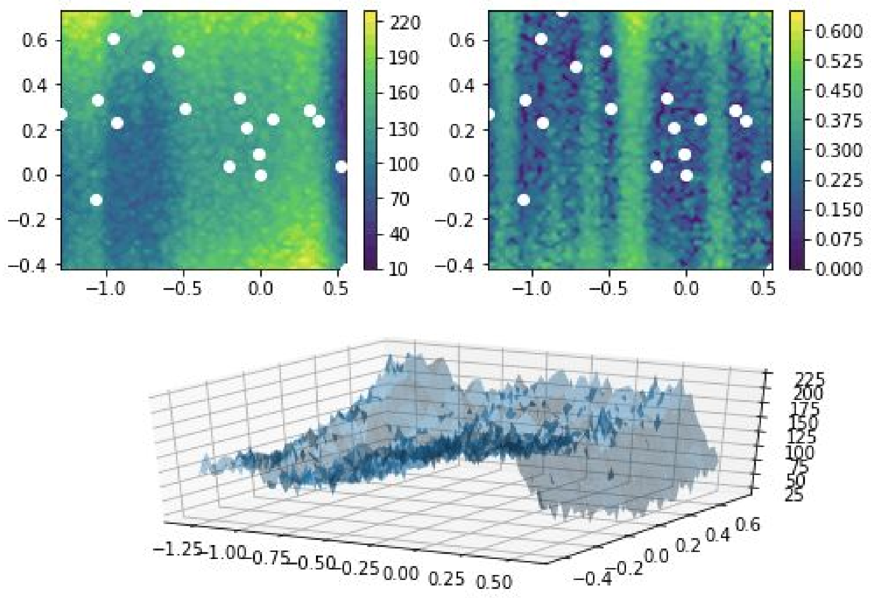
The IDW (inverse distance weighted) and Spline interpolation tools are referred to as deterministic interpolation methods because they are directly based on the surrounding measured values or on specified mathematical formulas that determine the smoothness of the resulting surface. A second family of interpolation methods consists of geostatistical methods, such as kriging, which are based on statistical models that include autocorrelation—that is, the statistical relationships among the measured points. Because of this, geostatistical techniques not only have the capability of producing a prediction surface but also provide some measure of the certainty or accuracy of the predictions.
Kriging assumes that the distance or direction between sample points reflects a spatial correlation that can be used to explain variation in the surface. The Kriging tool fits a mathematical function to a specified number of points, or all points within a specified radius, to determine the output value for each location. Kriging is a multistep process; it includes exploratory statistical analysis of the data, variogram modeling, creating the surface, and (optionally) exploring a variance surface. Kriging is most appropriate when you know there is a spatially correlated distance or directional bias in the data. It is often used in soil science and geology.

Resources for kriging:
https://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-kriging-works.htm
https://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kriging-works.htm
http://www.gitta.info/ContiSpatVar/en/html/Interpolatio_learningObject3.xhtml
https://mgimond.github.io/Spatial/spatial-interpolation.html

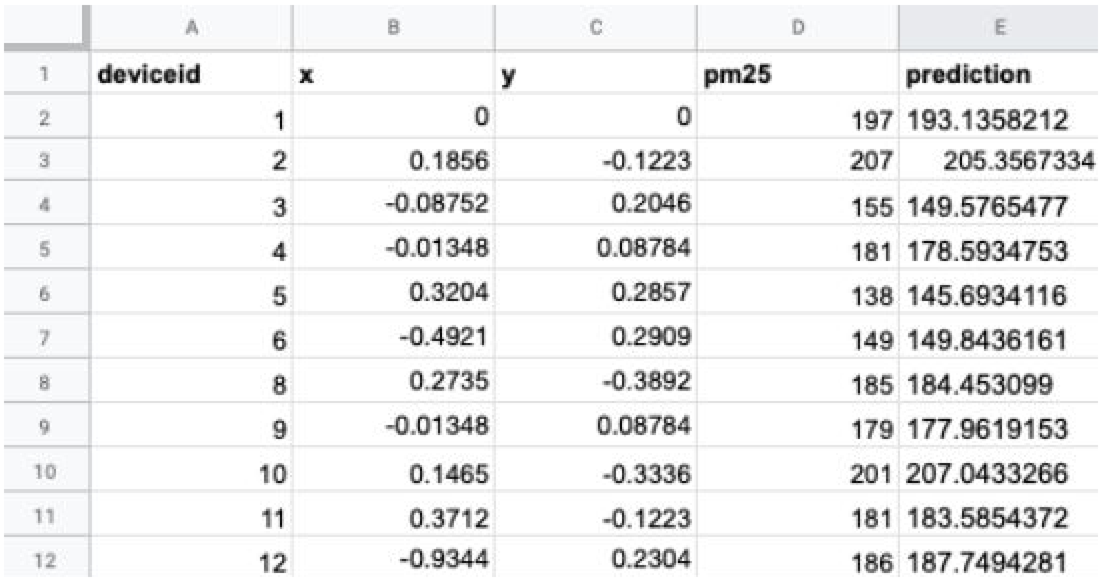
A python library called pyKriging was used to predict pm25 values for coordinates with unknown pm25 values. Along with the prediction, a 3D surface and graphs were also plotted for visualization.

pykriging.ipynb
```
import pyKriging
import pandas as pd
import numpy
import os
from pyKriging.krige import kriging
from pyKriging.samplingplan import samplingplan
df = pd.read_csv('26th - 26th.csv')
df1 = df[['deviceid', 'pm25','x', 'y']]
data = df1.values
X1 = data[:,2:4]
z1 = data[:,1]
k = kriging(X1, z1, name='universal')
k.train()
print(k.predict([0.08547,0.25]))
k.plot()
```

Source: http://pykriging.com/



The predictions were stored in an excel sheet:
https://docs.google.com/spreadsheets/d/1RJ1Ng3rBWIBCM80UOYSmaXI9ytEqWCRvbJ6HRYGuCEo/edit#gid=1158805120



	A	B	C	D	E
1	deviceid	x	y	pm25	prediction
2	1	0	0	197	193.1358212
3	2	0.1856	-0.1223	207	205.3567334
4	3	-0.08752	0.2046	155	149.5765477
5	4	-0.01348	0.08784	181	178.5934753
6	5	0.3204	0.2857	138	145.6934116
7	6	-0.4921	0.2909	149	149.8436161
8	8	0.2735	-0.3892	185	184.453099
9	9	-0.01348	0.08784	179	177.9619153
10	10	0.1465	-0.3336	201	207.0433266
11	11	0.3712	-0.1223	181	183.5854372
12	12	-0.9344	0.2304	186	187.7494281

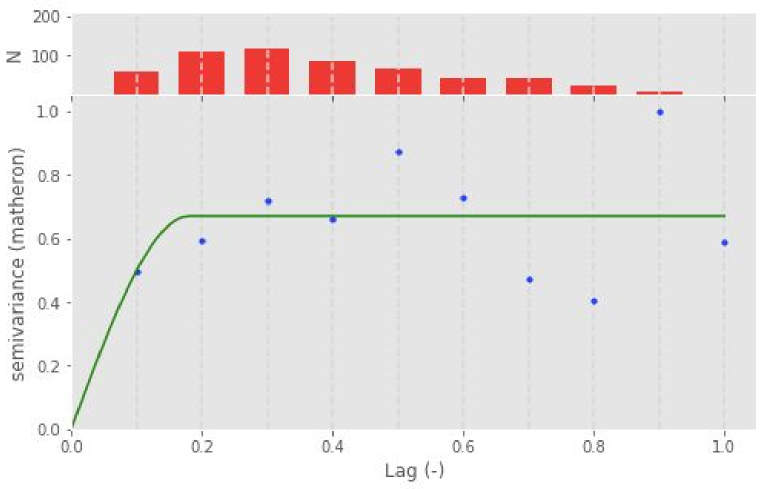
The problem with the predictions was that they were inconsistent, i.e. over multiple iterations of the same code and same set of data points, the code predicted different pm25 values for a set of coordinates.

Kriging was studied in further detail and a variogram model was created. A python library called scikit-gstat was used to get the value of the range, nugget and sill as well as the model curve and the equation for the curve.

SciKit kridging.ipynb

```
import skgstat as skg
from skgstat import Variogram, OrdinaryKriging
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
plt.style.use('ggplot')
%env SKG_SUPPRESS = true
df = pd.read_csv('/Users/peyamowar/26th - 26th.csv')
df1 = df[['deviceid', 'pm25','x', 'y']]
data = df1.values
X1 = data[:,2:4]
z1 = data[:,1]
V = skg.Variogram(coordinates=X1, values=z1)
print(V)
V.plot()
```

```
spherical Variogram
-------------------
Estimator:         matheron
Effective Range:   0.85
Sill:              1959702.57
Nugget:            0.00
```



Source: https://mmaelicke.github.io/scikit-gstat/index.html
https://github.com/mmaelicke/scikit-gstat

This code could accurately predict the best suited variogram model for the given data along with the values of the variogram parameters: sill, range and nugget.

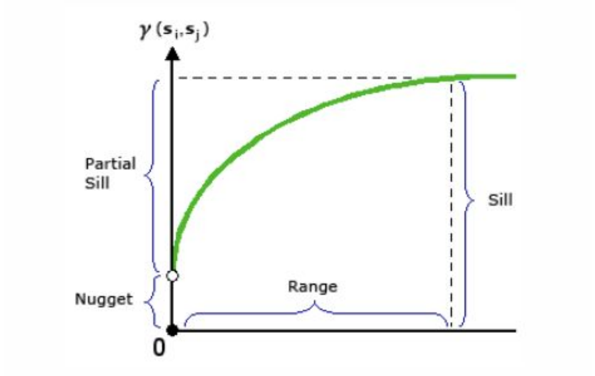
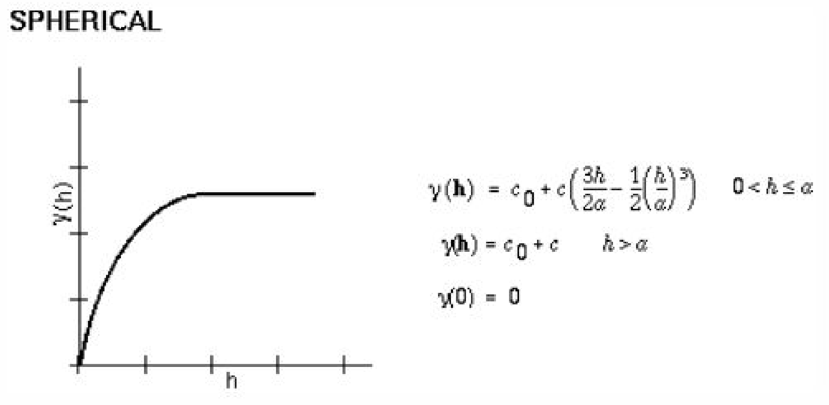


Illustration of Range, Sill, and Nugget components

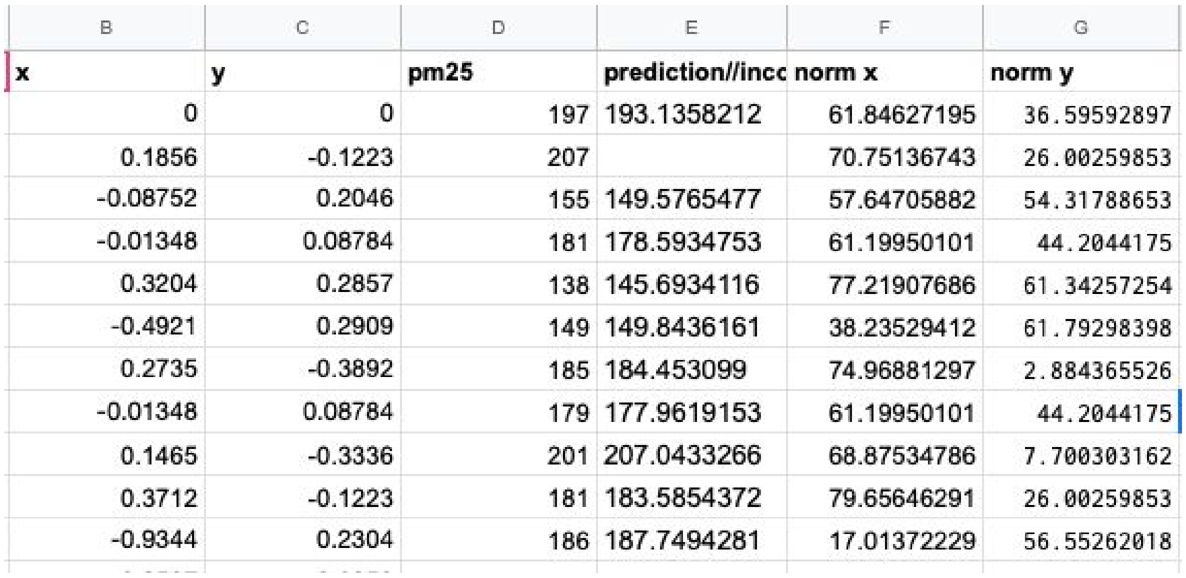


SPHERICAL
$\gamma(h)$
h
$\gamma(\mathbf{h}) = c_0 + c\left(\frac{3h}{2a} - \frac{1}{2}\left(\frac{h}{a}\right)^3\right) \quad 0 < h \le a$
$\gamma(\mathbf{h}) = c_0 + c \quad h > a$
$\gamma(0) = 0$

Spherical semivariance model illustration

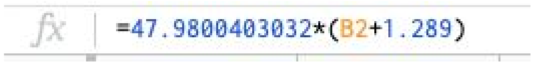
By observation and understanding, the following conclusions were made:
C_0 = Nugget
α = Range
C = Partial Sill = Sill - Nugget

Running the further code for prediction after creating the variogram showed an error: not enough neighbours to predict. After taking a look into the sample dataset on the site, it was observed that the sample x and y values ranged from 0 to 100. Hence, our data was then normalised using the formula provided in:
https://stats.stackexchange.com/questions/25894/changing-the-scale-of-a-variable-to-0-100
And the normalised values were added to the sample dataset as following:



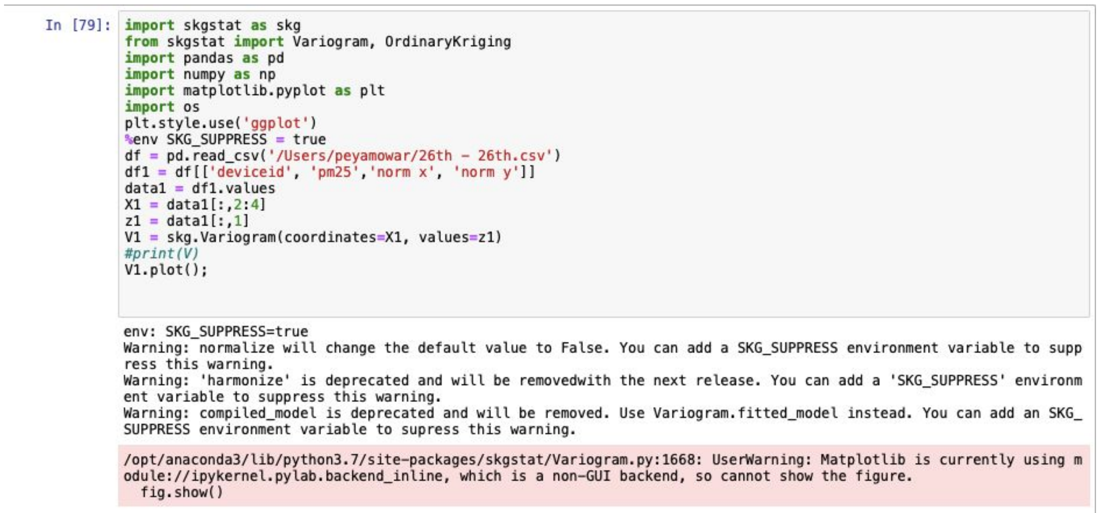
B	C	D	E	F	G
x	y	pm25	prediction//incc	norm x	norm y
0	0	197	193.1358212	61.84627195	36.59592897
0.1856	-0.1223	207		70.75136743	26.00259853
-0.08752	0.2046	155	149.5765477	57.64705882	54.31788653
-0.01348	0.08784	181	178.5934753	61.19950101	44.2044175
0.3204	0.2857	138	145.6934116	77.21907686	61.34257254
-0.4921	0.2909	149	149.8436161	38.23529412	61.79298398
0.2735	-0.3892	185	184.453099	74.96881297	2.884365526
-0.01348	0.08784	179	177.9619153	61.19950101	44.2044175
0.1465	-0.3336	201	207.0433266	68.87534786	7.700303162
0.3712	-0.1223	181	183.5854372	79.65646291	26.00259853
-0.9344	0.2304	186	187.7494281	17.01372229	56.55262018

Formula used:



```
fx  =47.9800403032*(B2+1.289)
```

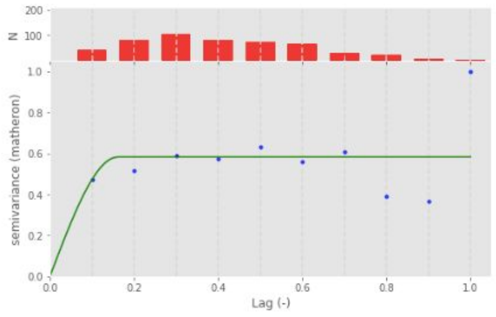
Now using these normalized values of x and y as dataset, again the variograms were generated:



```
In [79]: import skgstat as skg
         from skgstat import Variogram, OrdinaryKriging
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import os
         plt.style.use('ggplot')
         %env SKG_SUPPRESS = true
         df = pd.read_csv('/Users/peyamowar/26th - 26th.csv')
         df1 = df[['deviceid', 'pm25','norm x', 'norm y']]
         data1 = df1.values
         X1 = data1[:,2:4]
         z1 = data1[:,1]
         V1 = skg.Variogram(coordinates=X1, values=z1)
         #print(V)
         V1.plot();
```

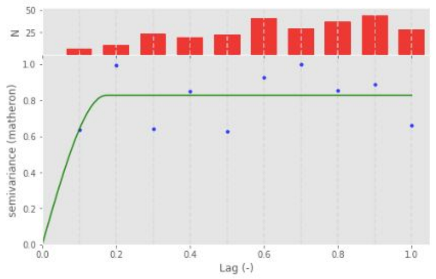
```
env: SKG_SUPPRESS=true
Warning: normalize will change the default value to False. You can add a SKG_SUPPRESS environment variable to supp
ress this warning.
Warning: 'harmonize' is deprecated and will be removedwith the next release. You can add a 'SKG_SUPPRESS' environm
ent variable to suppress this warning.
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an SKG_
SUPPRESS environment variable to supress this warning.
```

```
/opt/anaconda3/lib/python3.7/site-packages/skgstat/Variogram.py:1668: UserWarning: Matplotlib is currently using m
odule://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```



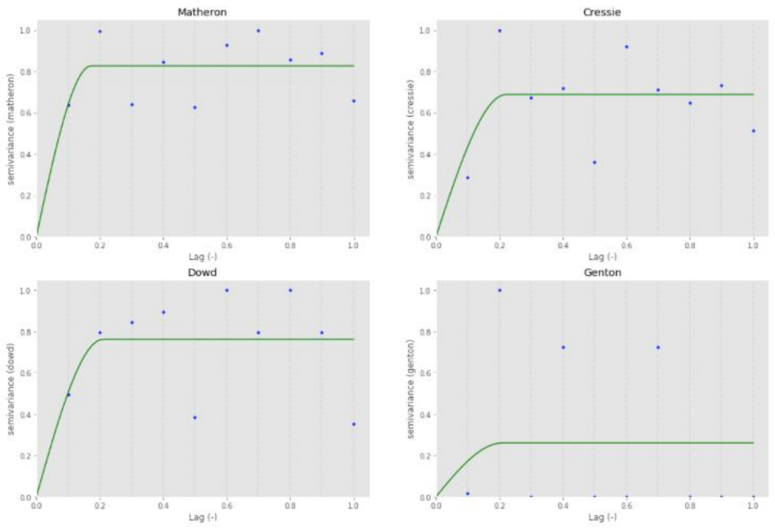
In [80]:
```
V1.maxlag = 'median'
V1.plot(show=False);
```

```
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an SKG_
SUPPRESS environment variable to supress this warning.
```

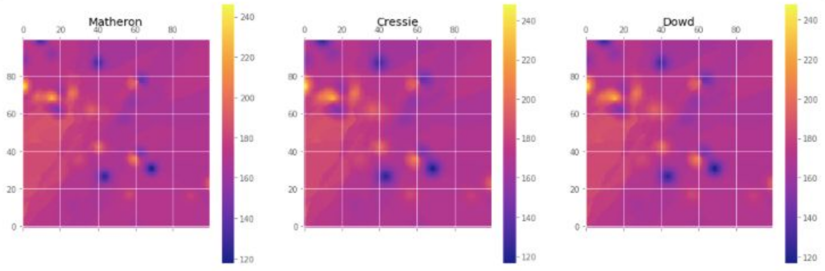


In [81]:
```
fig, _a = plt.subplots(2, 2, figsize=(18, 12))
axes = _a.flatten()
for ax, est in zip(axes, ('matheron', 'cressie', 'dowd', 'genton')):
    V1.estimator = est
    V1.plot(axes=ax, hist=False, show=False)
    ax.set_title(est.capitalize())
```

```
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an
SKG_SUPPRESS environment variable to supress this warning.
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an
SKG_SUPPRESS environment variable to supress this warning.
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an
SKG_SUPPRESS environment variable to supress this warning.
Warning: compiled_model is deprecated and will be removed. Use Variogram.fitted_model instead. You can add an
SKG_SUPPRESS environment variable to supress this warning.
```

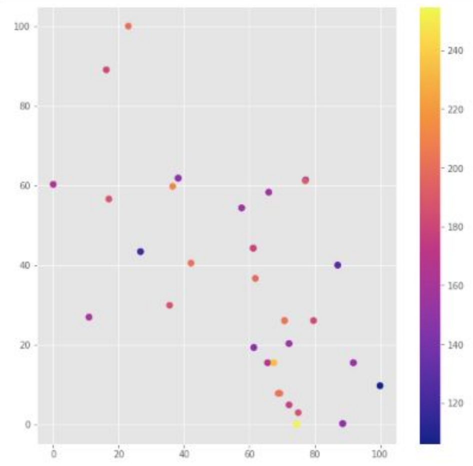


In [82]:
```
xx, yy = np.mgrid[0:99:100j, 0:99:100j]
fig, _a = plt.subplots(1, 3, figsize=(18, 6))
axes = _a.flatten()

for ax, est in zip(axes, ('matheron', 'cressie', 'dowd')):
    V1.estimator = est
    ok = OrdinaryKriging(V1, min_points=5, max_points=15, mode='exact')
    field = ok.transform(xx.flatten(), yy.flatten()).reshape(xx.shape)
    art = ax.matshow(field, origin='lower', cmap='plasma')
    plt.colorbar(art, ax=ax)
    ax.set_title(est.capitalize())
```



Points plotted:

```
In [48]: fig, ax = plt.subplots(1, 1, figsize=(9, 9))
         art = ax.scatter(X1[:,0],X1[:,1], s=50, c=z1, cmap='plasma')
         plt.colorbar(art);
```

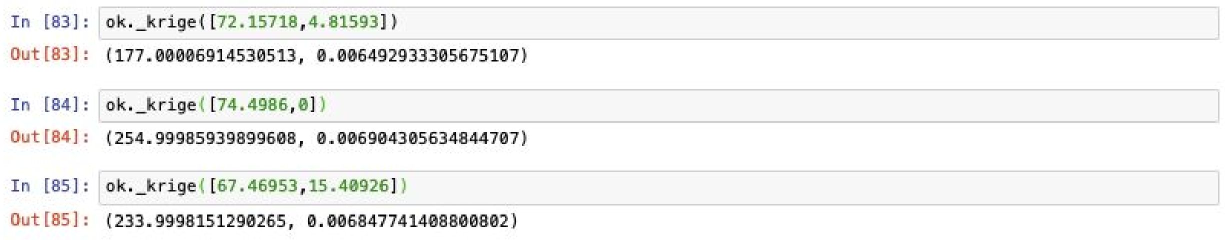


After these plots were created, after going through the code provided in:
https://github.com/mmaelicke/scikit-gstat
It was discovered that there exists a prediction function **_krige()** defined as follows:

```
def _krige(self, p):

        """Algorithm

        Kriging algorithm for one point. This is the place, where the
        algorithm shall be changed and optimized.

        Parameters
        ----------
        p : numpy.array
            point location coordinates of the unobserved location

        Raises
        ------
        SingularMatrixError:
            Raised if the kriging matrix is singular and therefore the
            equation system cannot be solved.
        LessPointsError:
            Raised if there are not the required minimum of points within the
            variogram's radius.

        Notes:
        ------
```

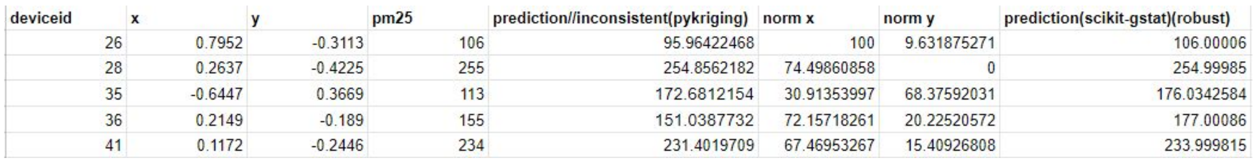
```
Z is calculated as follows:

  .. math::
      \hat{Z} = \sum_i(w_i * z_i)


  where :math:`w_i` is the calulated kriging weight for the i-th point
  and :math:`z_i` is the observed value at that point.

  The kriging variance :math:`\sigma^2` (sigma) is calculate as follows:

  .. math::
      \sigma^2 = \sum_i(w_i * \gamma(p_0 - p_i)) + \lambda

  where :math:`w_i` is again the weight, :math:`\gamma(p_0 - p_i)` is
  the semivairance of the distance between the unobserved location and
  the i-th observation. :math:`\lamda` is the Lagrange multiplier needed
  to minimize the estimation error.

  Returns
  -------
  Z : float
      estimated value at p

  sigma : float
      kriging variance :math:`\sigma^2` for p.

  """
```

This function was used to predict values for analysis:



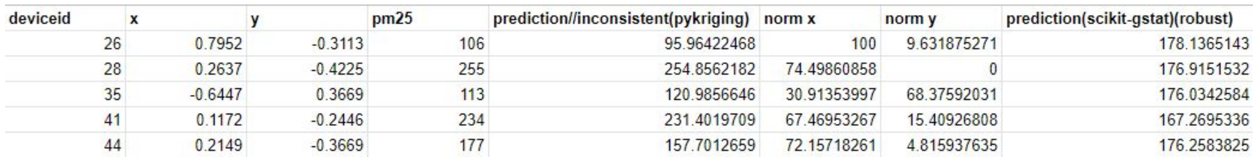
```
In [83]: ok._krige([72.15718,4.81593])
Out[83]: (177.00006914530513, 0.006492933305675107)

In [84]: ok._krige([74.4986,0])
Out[84]: (254.99985939899608, 0.006904305634844707)

In [85]: ok._krige([67.46953,15.40926])
Out[85]: (233.9998151290265, 0.006847741408800802)
```

Firstly, predictions were made for points included in the dataset and saved in: https://docs.google.com/spreadsheets/d/1RJ1Ng3rBWIBCM80UOYSmaXI9ytEqWCRvbJ6HRYGuCEo/edit#gid=1158805120



deviceid	x	y	pm25	prediction//inconsistent(pykriging)	norm x	norm y	prediction(scikit-gstat)(robust)
26	0.7952	-0.3113	106	95.96422468	100	9.631875271	106.00006
28	0.2637	-0.4225	255	254.8562182	74.49860858	0	254.99985
35	-0.6447	0.3669	113	172.6812154	30.91353997	68.37592031	176.0342584
36	0.2149	-0.189	155	151.0387732	72.15718261	20.22520572	177.00086
41	0.1172	-0.2446	234	231.4019709	67.46953267	15.40926808	233.999815

Then, for analysis, predictions were made for a point by removing that point from the dataset. However these predictions weren't very accurate, which can be owing to many reasons like not a good enough or robust mathematical model for prediction, lack of number of data points ie sensors, or not appropriate scalability etc.



deviceid	x	y	pm25	prediction//inconsistent(pykriging)	norm x	norm y	prediction(scikit-gstat)(robust)
26	0.7952	-0.3113	106	95.96422468	100	9.631875271	178.1365143
28	0.2637	-0.4225	255	254.8562182	74.49860858	0	176.9151532
35	-0.6447	0.3669	113	120.9856646	30.91353997	68.37592031	176.0342584
41	0.1172	-0.2446	234	231.4019709	67.46953267	15.40926808	167.2695336
44	0.2149	-0.3669	177	157.7012659	72.15718261	4.815937635	176.2583825

Conclusion

In conclusion, many Spatial Interpolation techniques were explored, out of which Kriging was found to be the most robust and preferred. The last code used for implementing Kriging using scikit-gstat was found to be the most suitable for data analysis and prediction. However, there is high probability of error for isolated locations that aren't already present in the dataset. This error term is also shown in the output in the form of kriging variance.

Scope for future

The scikit-gstat Kriging code needs to be studied in detail to understand the mathematics involved in the algorithm to reduce the error percentage for better prediction results.
The results also need to be analysed to determine better possible locations for existing sensors and the amount and location of new sensors required to make a better and more accurate as well as optimized network.