VANGUARD

Geospatial Analysis with QGIS & Python

Version 1.0 (2021 October)

QGIS VANGUARD

What you will learn

- QGIS API (PYQGIS) & Python Library to Query and Visualize Geospatial Data (Day 1)
- QGIS Plugin & Geo-Processing tool for Geovisualization and Analysis (Day 2)
- Spatial Regression (Day 3)

SLA SINGAPORE LAND AUTHORITY @ GeoWorks SINGAPORE

QGIS VANGUARD

Day 1

Morning

- Intro. Python
- QGIS Python console
- JupyterLab setup

Afternoon

- Intro. PYQGIS
- Use PYQGIS in QGIS console
- Use PYQGIS in JupyterLab

Day 2

Morning

- Create a QGIS Plugin
- Load Data
- Spatial Query & Join
- Create Choropleth Maps

Afternoon

- Intro. QGIS Processing framework
- Create QGIS Processing Tools (site selection, TSP)

Day 3

Morning

- Regression Analysis
- Linear Regression
- Spatial Weights
- Spatial Auto-correlation

Afternoon

- Spatial Regression Models
- Final Projects (optional)

SLA SINGAPORE LAND AUTHORITY @ GeoWorks SINGAPORE

Course Organization and Timeline

Morning Session 9:30am to 12:30pm

- Lecture
- Demonstration & Practice (practice together)
- Exercise (do it by yourself)
- Q & A

Break (10:50am - 11:10am)

- Lecture
- Demonstration & Practice (practice together)
- Exercise (do it by yourself)
- Q & A

Afternoon Session (2:00pm to 5:30pm)

Prerequisites

- Basic knowledge in GIS and QGIS
- Basic literacy of programming
- Basic understanding about linear regression

References

- PyQGIS Developer Cookbook
- Python Cheat Sheet
- PySAL open source library of spatial analysis
- Spreg spatial regression models
- Statistical Analysis Handbook
- Geographic Data Science with PySAL and the PyData Stack

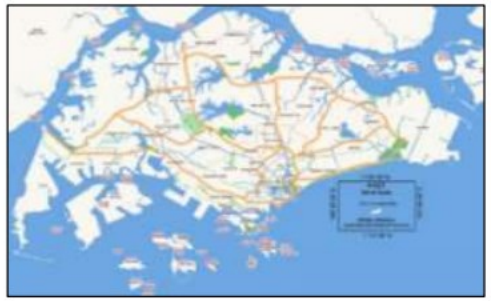
Day 1 Morning

- Use Python Library (in QGIS & Sublime Text) from a GIS Perspective
- Use Python Library (in JupyterLab) from a data science perspective

Day 1 Afternoon

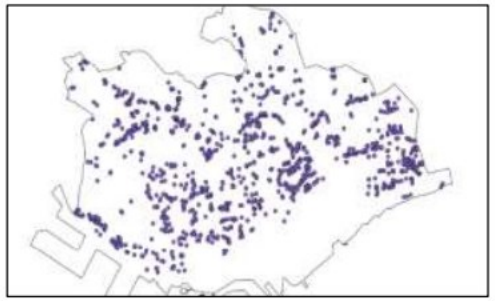
- Use PYQGIS Library (in QGIS)
- Use PYQGIS Library (in JupyterLab) for standalone application (optional)

Recap – Overview

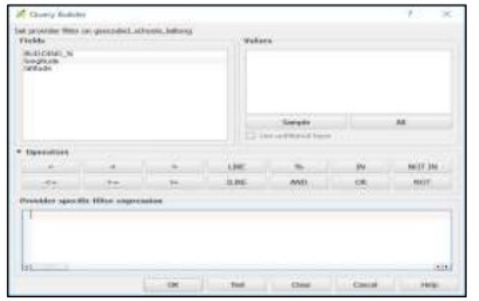


1. Calling Basemap from WMS

2. Styling Map Layers

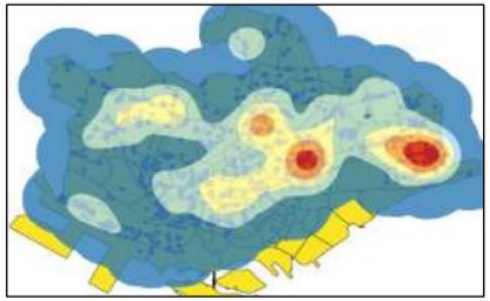


3. Geocoding Textual Data



4. Selecting & Filtering Data

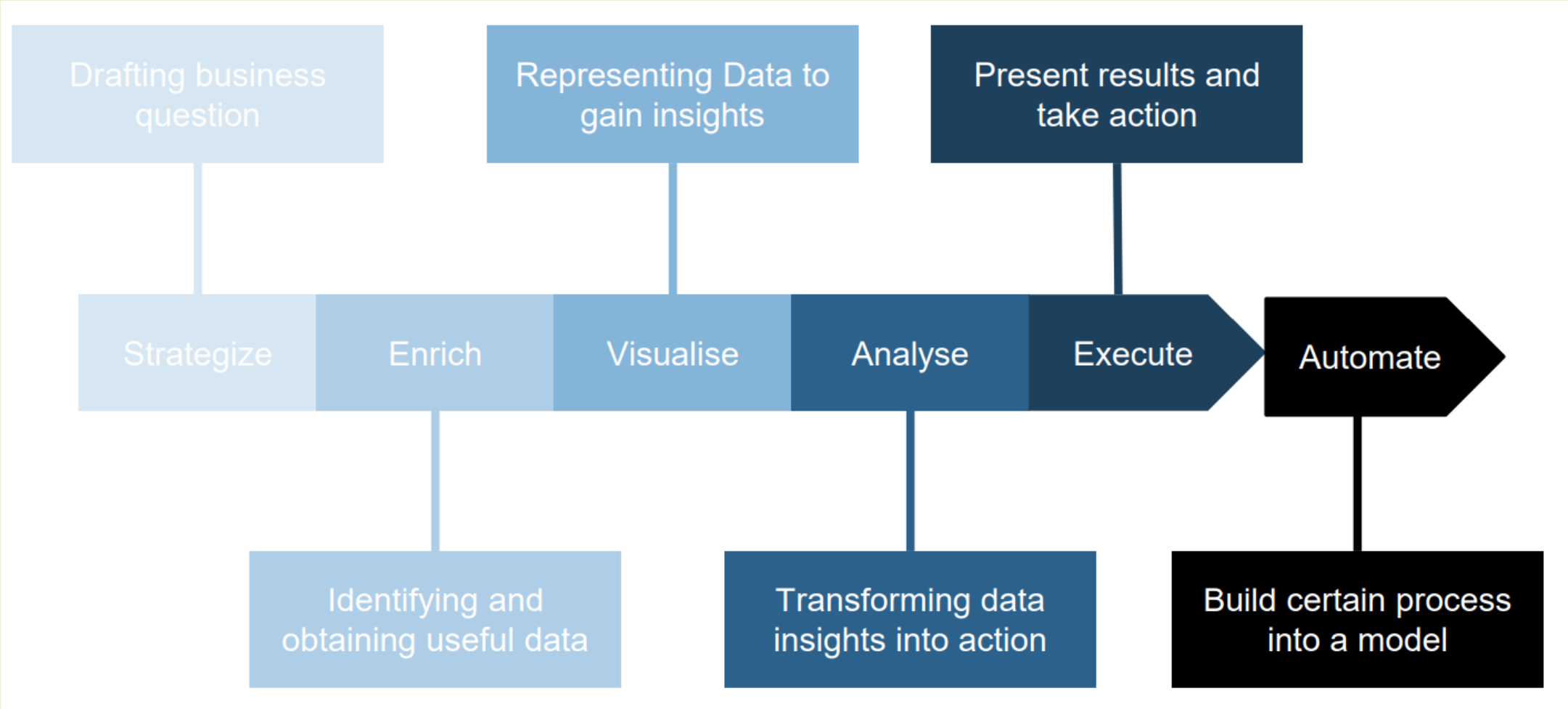
5. Joining & Aggregating Data



6. Generating Heatmap

7. Generating Buffers

8. Using Spatial Query



Qt, PyQt, PyQGIS

Qt	QGIS is built using the Qt platform, which is a free and open-source widget toolkit for creating GUI and cross-platform applications (written in C++).
PyQt	PyQt is the Python interface to Qt. PyQt provides classes and functions to interact with Qt widgets.
PyQGIS	PyQGIS are Python APIs provided by QGIS.

Integrate QGIS & Python Libraries

PyQGIS

PyQt

Python

QGIS C++ Lib

Qt

Python

NumPy

Pandas

SciPy

Scikit-learn

……

Python Libraries

- **Numpy:** Operations related to linear algebra
- **Pandas & GeoPandas:** data (geospatial) manipulation and analysis
- **Matplotlib & Seaborn:** making plots
- **Scikit-learn:** general-purpose machine learning

Name	Description	Example
a.shape	The shape attribute of NumPy array a keeps a tuple of integers. Each integer describes the number of elements of the axis.	`a = np.array([[1,2],[1,1],[0,0]])` `print(np.shape(a))  # (3, 2)`
a.ndim	The ndim attribute is equal to the length of the shape tuple.	`print(np.ndim(a))  # 2`

*	The asterisk (star) operator performs the Hadamard product, i.e., multiplies two matrices with equal shape element-wise.	`a = np.array([[2, 0], [0, 2]])`
		`b = np.array([[1, 1], [1, 1]])`
		`print(a*b)  # [[2 0] [0 2]]`

np.matmul(a,b), a@b	The standard matrix multiplication operator. Equivalent to the @ operator.	`print(np.matmul(a,b))` `# [[2 2] [2 2]]`
np.arange([start,]stop, [step,])	Creates a new 1D numpy array with evenly spaced values	`print(np.arange(0,10,2))` `# [0 2 4 6 8]`
np.linspace(start, stop, num=50)	Creates a new 1D numpy array with evenly spread elements within the given interval	`print(np.linspace(0,10,3))` `# [ 0.  5. 10.]`
np.average(a)	Averages over all the values in the numpy array	`a = np.array([[2, 0], [0, 2]])` `print(np.average(a))  # 1.0`

<slice> = <val>	Replace the <slice> as selected by the slicing operator with the value <val>.	`a = np.array([0, 1, 0, 0, 0])`
		`a[::2] = 2`
		`print(a)  # [2 1 2 0 2]`

`np.var(a)`	Calculates the variance of a numpy array.	`a = np.array([2, 6])` `print(np.var(a))  # 4.0`
`np.std(a)`	Calculates the standard deviation of a numpy array	`print(np.std(a))  # 2.0`

`np.diff(a)`	Calculates the difference between subsequent values in NumPy array a	`fibs = np.array([0, 1, 1, 2, 3, 5])`
		`print(np.diff(fibs, n=1))`
		`# [1 0 1 1 2]`

`np.cumsum(a)`	Calculates the cumulative sum of the elements in NumPy array a.	`print(np.cumsum(np.arange(5)))` `# [ 0  1  3  6 10]`

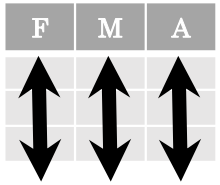
`np.sort(a)`	Creates a new NumPy array with the values from a (ascending).	`a = np.array([10,3,7,1,0])`
		`print(np.sort(a))`
		`# [ 0  1  3  7 10]`

`np.argsort(a)`	Returns the indices of a NumPy array so that the indexed values would be sorted.	`a = np.array([10,3,7,1,0])`
		`print(np.argsort(a))`
		`# [4 3 1 2 0]`

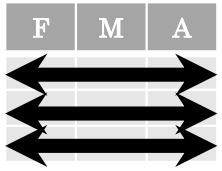
`np.max(a)`	Returns the maximal value of NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.max(a))  # 10`
`np.argmax(a)`	Returns the index of the element with maximal value in the NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.argmax(a))  # 0`
`np.nonzero(a)`	Returns the indices of the nonzero elements in NumPy array a.	`a = np.array([10,3,7,1,0])` `print(np.nonzero(a))  # [0 1 2 3]`

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



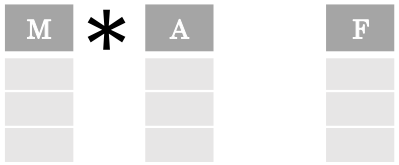
&



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



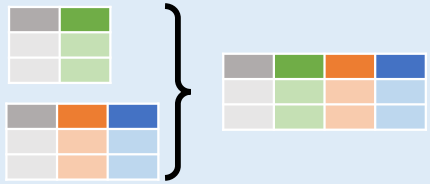
M * A

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames



pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

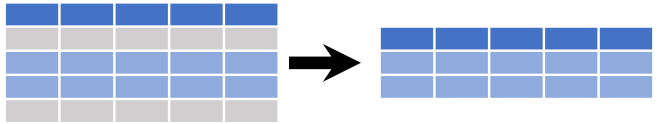
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length','Height'])
Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)			
`<`	Less than	`!=`	Not equal to
`>`	Greater than	`df.column.isin(values)`	Group membership
`==`	Equals	`pd.isnull(obj)`	Is NaN
`<=`	Less than or equals	`pd.notnull(obj)`	Is not NaN
`>=`	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)

`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* **`df.width`**
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples	
`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`'^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

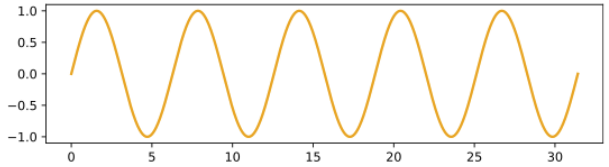
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

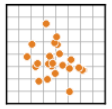
4 Observe



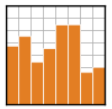
Choose

Matplotlib offers several kind of plots (see Gallery):

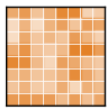
```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

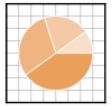


```
Z = np.random.uniform(0, 1, (8,8))

ax.imshow(Z)
```

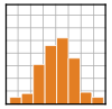


```
Z = np.random.uniform(0, 1, (8,8))

ax.contourf(Z)
```

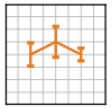
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```



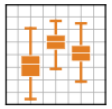
```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0,1,5)
ax.errorbar(X, Y, Y/4)
```



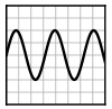
```
Z = np.random.normal(0,1,(100,3))

ax.boxplot(Z)
```



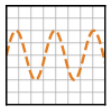
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

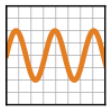
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



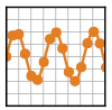
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



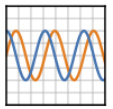
```
X = np.linspace(0,10,100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



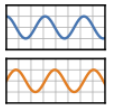
Organize

You can plot several data on the the same figure but you can also split a figure in several subplots (named *Axes*):

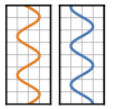
```
X = np.linspace(0,10,100)
Y1, Y1 = np.sin(X), np.cos(X)
ax.plot(X, Y1, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

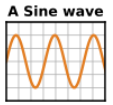


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

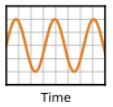


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

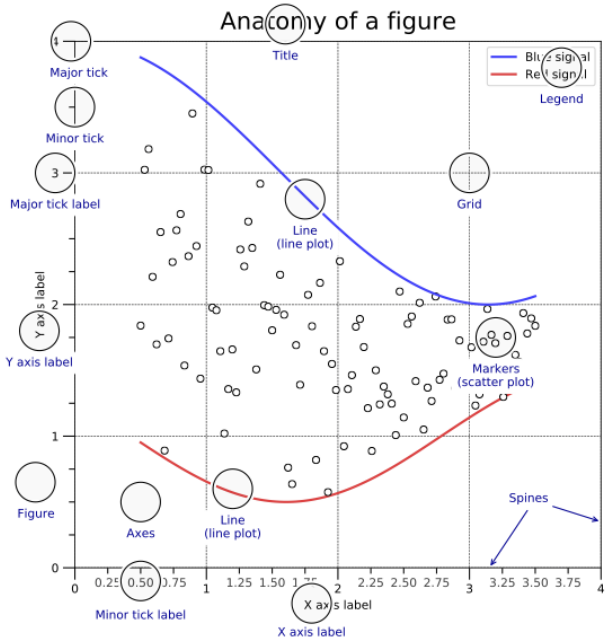
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib for intermediate users

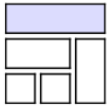
A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.



Figure, axes & spines

```
fig, axs = plt.subplots((3,3))
axs[0,0].set_facecolor("#ddddff")
axs[2,2].set_facecolor("#ffffdd")
```

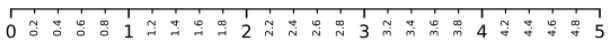
```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```



```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```

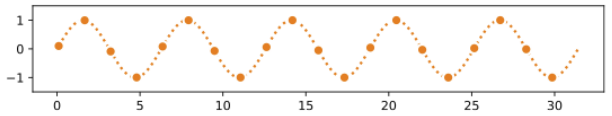
Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```



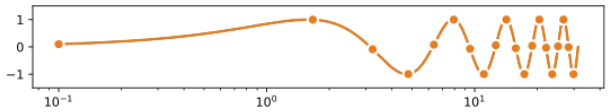
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=25, mec="1.0")
```



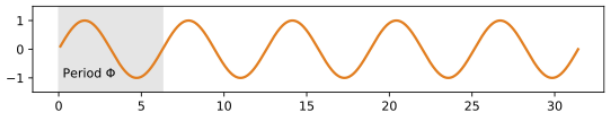
Scales & Projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=25, mec="1.0")
```



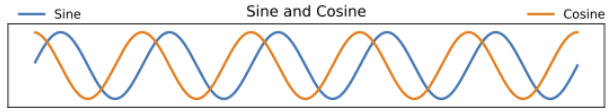
Text & Ornaments

```
ax.fill_betweenx([-1,1],[0],[2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



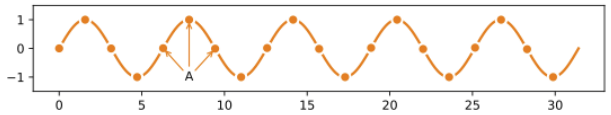
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1),ncol=2,
          mode="expand",  loc="lower left")
```



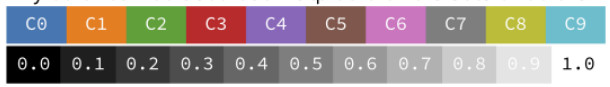
Annotation

```
ax.annotate("A", (X[250],Y[250]),(X[250],-1),
  ha="center", va="center",arrowprops =
  {"arrowstyle" : "->", "color": "C1"})
```



Colors

Any color can be used but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-columns A4 paper with 2cm margins on each side and a column separation of 1cm. The width of a figure is (21 - 2*2 - 1)/2 = 8cm. One inch being 2.54cm, figure size should be 3.15×3.15 in.

```
fig = plt.figure(figsize=(3.15,3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science Interactively at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
```

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```
(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```
Set context to `"talk"`
Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```
Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```
Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```
Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```
Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```
Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```
Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```
Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
>>> sns.boxplot(data=iris,orient="h")
```
Boxplot

Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```
Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```
Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```
Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```
Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```
Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```
Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
                transparent=True)
```
Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```
Clear an axis
Clear an entire figure
Close a window

DataCamp
Learn Python for Data Science Interactively

Day 1 Afternoon

- Overview of PYQGIS
- Use PYQGIS in QGIS
- Use PYQGIS Library in JupyterLab (optional)

Organization of QGIS Libraries

core library	The CORE library contains all basic GIS functionality
gui library	The GUI library is build on top of the CORE library and adds reusable GUI widgets
analysis library	The ANALYSIS library is built on top of CORE library and provides high level tools for carrying out spatial analysis on vector and raster data
server library	The SERVER library is built on top of the CORE library and adds map server components to QGIS
3D library	The 3D library is build on top of the CORE library and Qt 3D framework
plugin classes	Contains classes related to implementation of QGIS plugins
QgsQuick library	The QgsQuick library is built on top of the CORE library and Qt Quick/QML framework

Organization of PyQGIS API

qgis.core	The CORE library contains all basic GIS functionality
qgis.gui	The GUI library is build on top of the CORE library and adds reusable GUI widgets
qgis.processing	The processing library contains classes for processing toolbox
qgis.analysis	The ANALYSIS library is built on top of CORE library and provides high level tools for carrying out spatial analysis on vector and raster data
qgis.server	The SERVER library is built on top of the CORE library and adds map server components to QGIS
qgis.3d	The 3D library is build on top of the CORE library and Qt 3D framework

qgis.gui.QgsProject

QgisInterface

A reference to QgisInterface is typically available through the **iface** global variable.

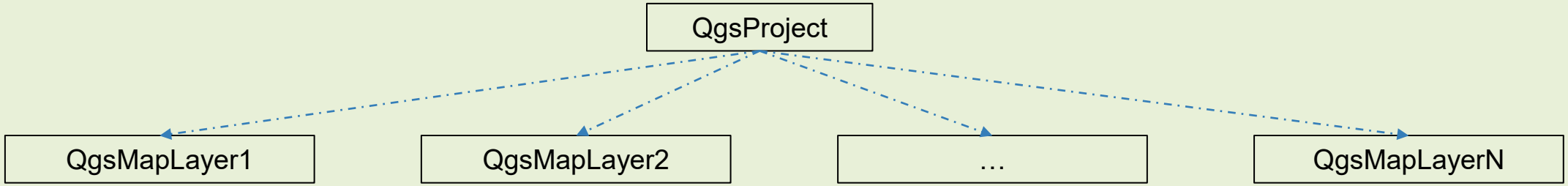
Get a reference to the map canvas displayed within the main application window, using the **mapCanvas**() method.

Retrieve the currently active layer within the project, using the **activeLayer**() method.

Get a reference to the application's main window by calling the **mainWindow**() method.

Get a reference to the QGIS system's message bar by calling the **messageBar**() method.

qgis.core.QgsProject



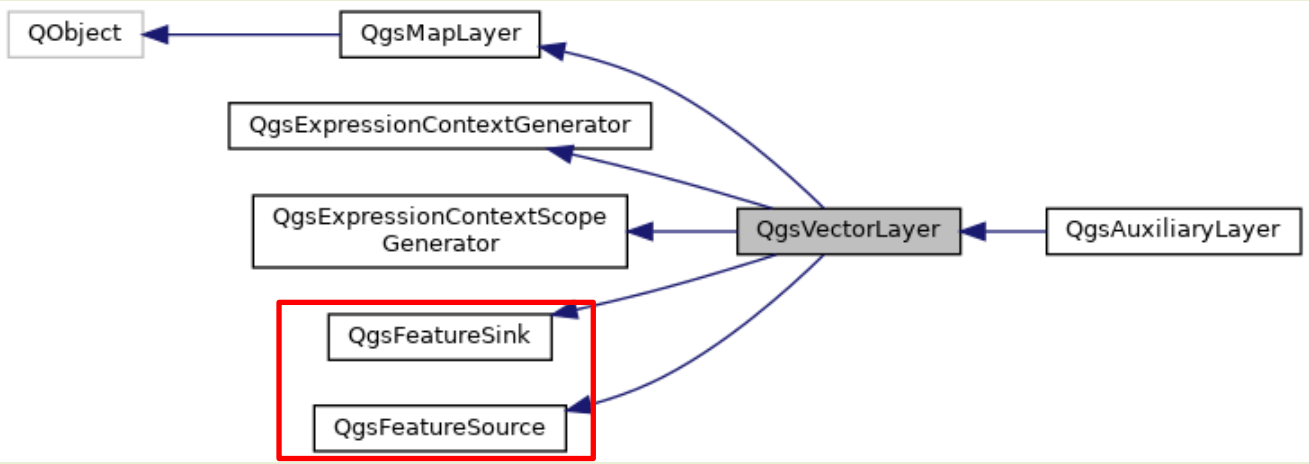
qgis.core.QgsProject example

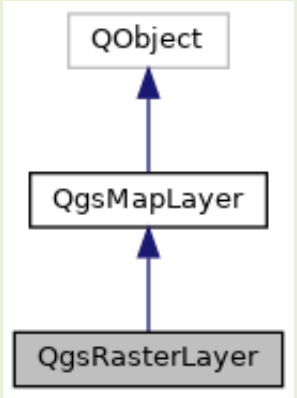
```
import os

# change your current work directory
os.chdir("/Users/hs/Projects/pyqgis")

# as QgsProject is Singleton class, we need to get the instance of it
project = QgsProject.instance()

# open the project
project_name = 'Data/QgsProject.qgs'
project.read(project_name)
```

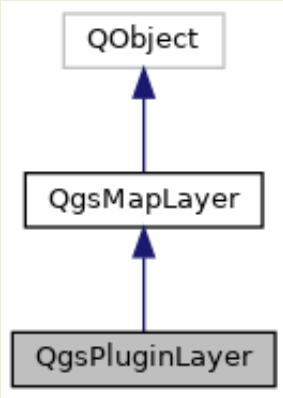
qgis.core.QgsMapLayer example

```
# list of layer names using list comprehension
layers = project.mapLayers()
l=[layer.name() for layer in layers.values()]

# dictionary with key = layer name and value = layer object
layers_list = {}
for l in layers.values():
  layers_list[l.name()] = l

print(layers_list)
```

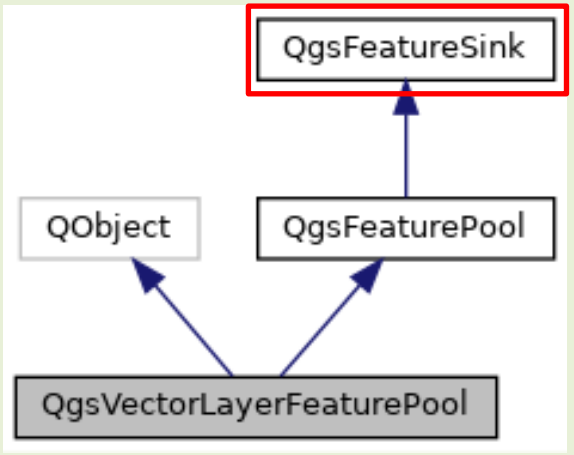
qgis.core.QgsVectorLayer







qgis.core. QgsVectorLayer



Public Member Functions

QgsVectorLayerFeaturePool (QgsVectorLayer *layer)

Creates a new feature pool for *layer*. More...

bool **addFeature** (QgsFeature &feature, QgsFeatureSink::Flags flags=QgsFeatureSink::Flags()) override

Adds a single *feature* to the sink. More...

bool **addFeatures** (QgsFeatureList &features, QgsFeatureSink::Flags flags=QgsFeatureSink::Flags()) override

Adds a list of *features* to the sink. More...

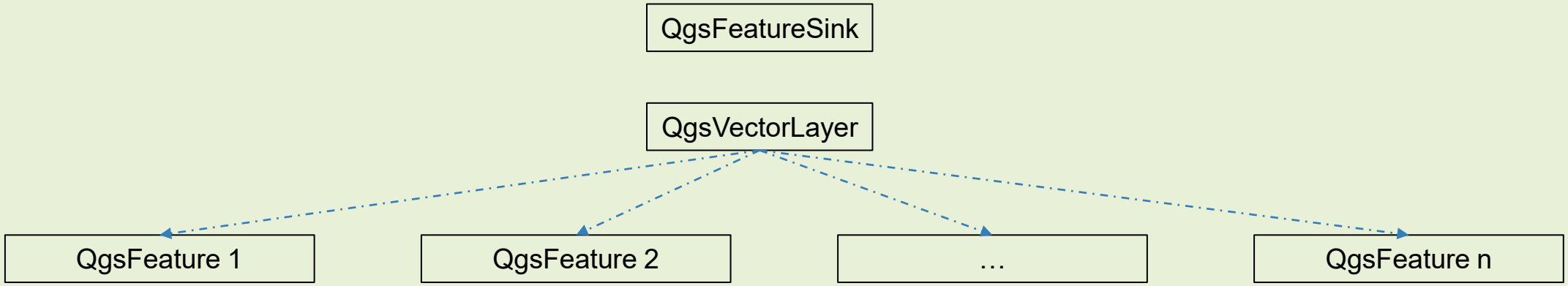
void **deleteFeature** (QgsFeatureId fid) override

Removes a feature from this pool. More...

void **updateFeature** (QgsFeature &feature) override

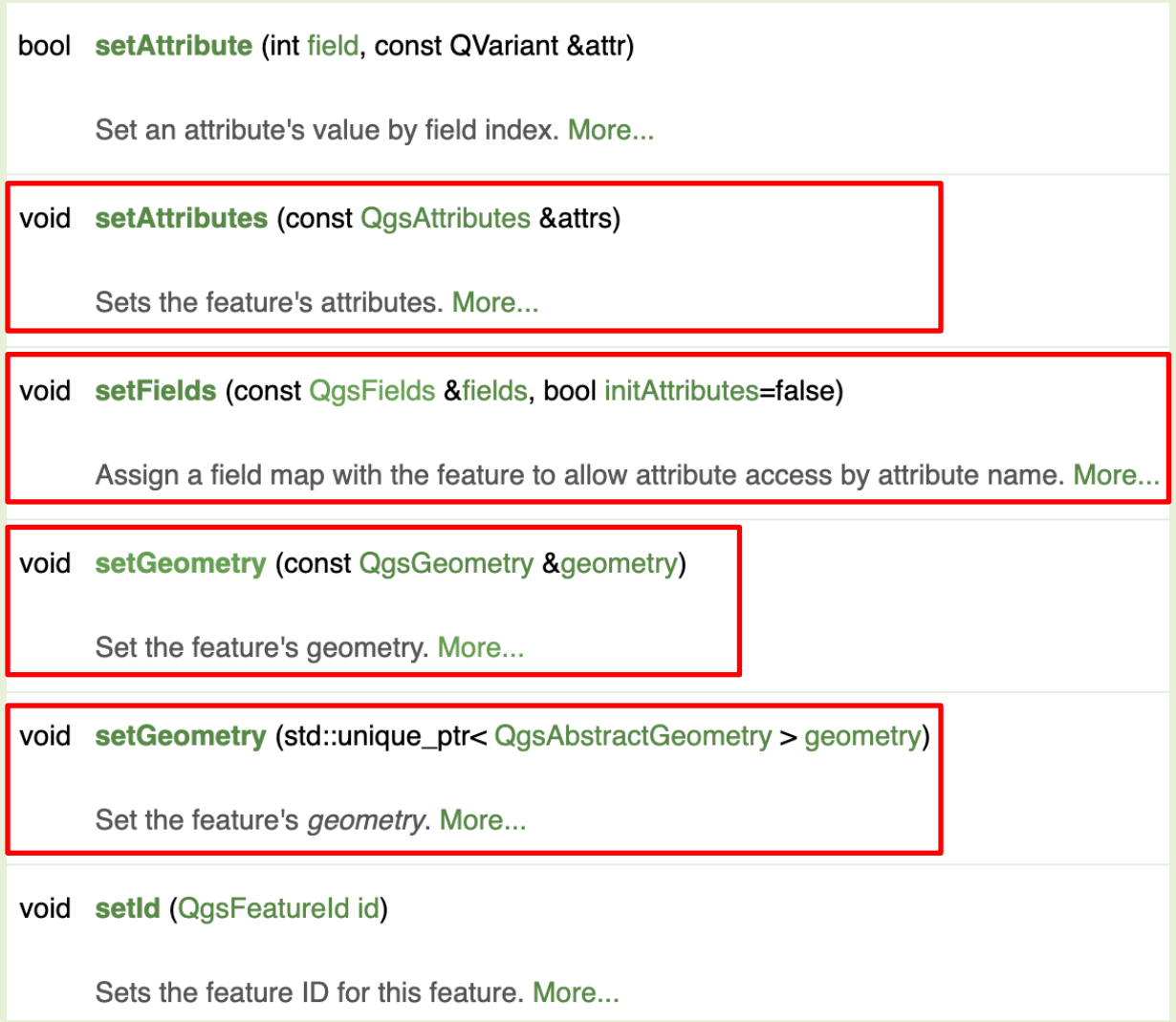
Updates a feature in this pool. More...

qgis.core. QgsVectorLayer



qgis.core.QgsFeature

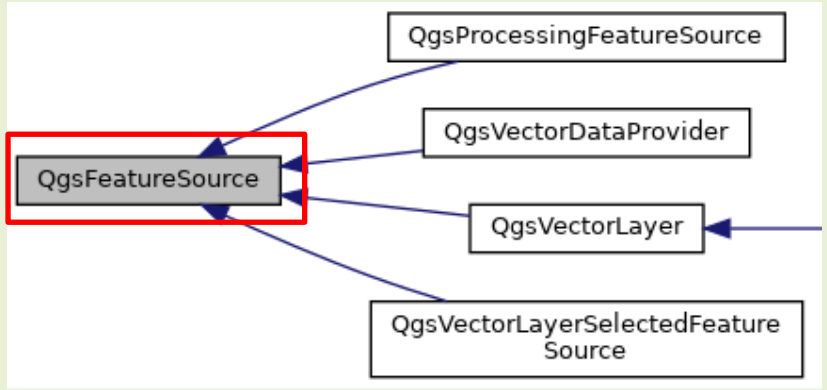
The QgsFeature encapsulates a single feature including its id, geometry and a list of field/values attributes.



qgis.core.QgsFeature example

```
# use QgsInterface global variable iface to get active layer
layer = iface.activeLayer()

features = layer.getFeatures()

for feature in features:
    # retrieve every feature with its geometry and attributes
    print("Feature ID: ", feature.id())

    # fetch attributes
    attrs = feature.attributes()

    # attrs is a list. It contains all the attribute values of this feature
    print(attrs)
```

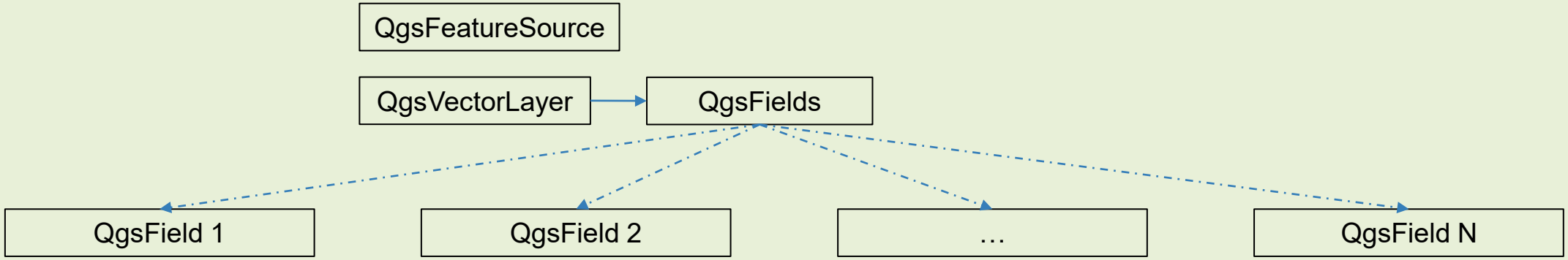
qgis.core.QgsFeatureSource



Public Member Functions

virtual	**~QgsFeatureSource** ()=default
virtual QgsFeatureIds	**allFeatureIds** () const Returns a list of all feature IDs for features present in the source. More...
virtual long	**featureCount** () const =0 Returns the number of features contained in the source, or -1 if the feature count is unknown.
virtual QgsFields	**fields** () const =0 Returns the fields associated with features in the source. More...
virtual QgsFeatureIterator	**getFeatures** (const QgsFeatureRequest &request=QgsFeatureRequest()) const =0 Returns an iterator for the features in the source. More...
virtual FeatureAvailability	**hasFeatures** () const Determines if there are any features available in the source. More...
virtual SpatialIndexPresence	**hasSpatialIndex** () const Returns an enum value representing the presence of a valid spatial index on the source, if it c

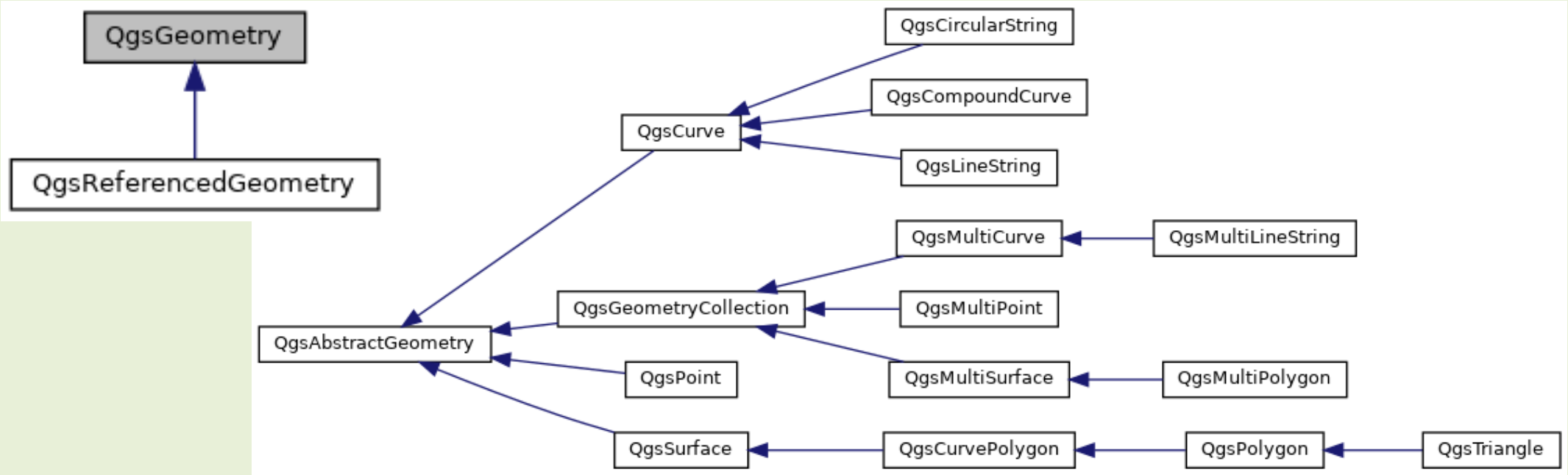
qgis.core.QgsFields



qgis.core.QgsFields example

```
layer = iface.activeLayer()

output_name = 'Data_Day1/output.csv'

# Using*with* statement which takes care of closing the files and handling errors
with open(output_name, 'w') as output_file:
    fieldnames = [field.name() for field in layer.fields()]

    ## write header
    line = ','.join(name for name in fieldnames) + '\n'
    output_file.write(line)

    # write feature attributes
    for f in layer.getFeatures():
        line = ','.join(str(f[name]) for name in fieldnames) + '\n'
        output_file.write(line)

iface.messageBar().pushMessage(
    'Success:', 'Output file written at ' + output_name, level=Qgis.Success)
```

qgis.core.QgsGeometry



qgis.core.QgsGeometry example

```
geom = feature.geometry()
    geomSingleType = QgsWkbTypes.isSingleType(geom.wkbType())
    if geom.type() == QgsWkbTypes.PointGeometry:
        # the geometry type can be of single or multi type
        if geomSingleType:
            x = geom.asPoint()
            print("Point: ", x)
        else:
            x = geom.asMultiPoint()
            print("MultiPoint: ", x)
    elif geom.type() == QgsWkbTypes.LineGeometry:
         # the geometry type can be of single or multi type
        if geomSingleType:
            x = geom.asPolyline()
            print("Line: ", x, "length: ", geom.length())
        else:
            x = geom.asMultiPolyline()
            print("MultiLine: ", x, "length: ", geom.length())
    elif geom.type() == QgsWkbTypes.PolygonGeometry:
         # the geometry type can be of single or multi type
        if geomSingleType:
            x = geom.asPolygon()
            print("Polygon: ", x, "Area: ", geom.area())
        else:
            x = geom.asMultiPolygon()
            print("MultiPolygon: ", x, "Area: ", geom.area())
    else:
        print("Unknown or invalid geometry")
```

Modify a Vector Layer Example

```
feat1 = feat2 = QgsFeature(layer.fields())
fid = 99
feat1.setId(fid)

# add two features (QgsFeature instances)
layer.addFeatures([feat1,feat2])

# delete a feature with specified ID
layer.deleteFeature(fid)

# set new geometry (QgsGeometry instance) for a feature
geometry = QgsGeometry.fromWkt("POINT(7 45)")
layer.changeGeometry(fid, geometry)

# update an attribute with given field index (int) to a given value
fieldIndex =1
value ='My new name'
layer.changeAttributeValue(fid, fieldIndex, value)
```

```
# add new field
layer.dataProvider().addAttributes([QgsField("STATUT", QVariant.Int)])
layer.updateFields()

# remove a field
fieldIndex = layer.fields().indexFromName("STATUT")
layer.dataProvider().deleteAttributes([fieldIndex])
layer.updateFields()
```

Geometry Construction

```
# Construct Geometry from coordinates
gPnt = QgsGeometry.fromPointXY(QgsPointXY(1,1))
print(gPnt)

gLine = QgsGeometry.fromPolyline([QgsPoint(1, 1), QgsPoint(2, 2)])
print(gLine)

gPolygon = QgsGeometry.fromPolygonXY([[QgsPointXY(1, 1),
QgsPointXY(2, 2), QgsPointXY(2, 1)]])
print(gPolygon)

# Construct Geometry from well known text
geom = QgsGeometry.fromWkt("POINT(3 4)")
print(geom)
```

Exploring Data using PyQGIS

Try Out Time

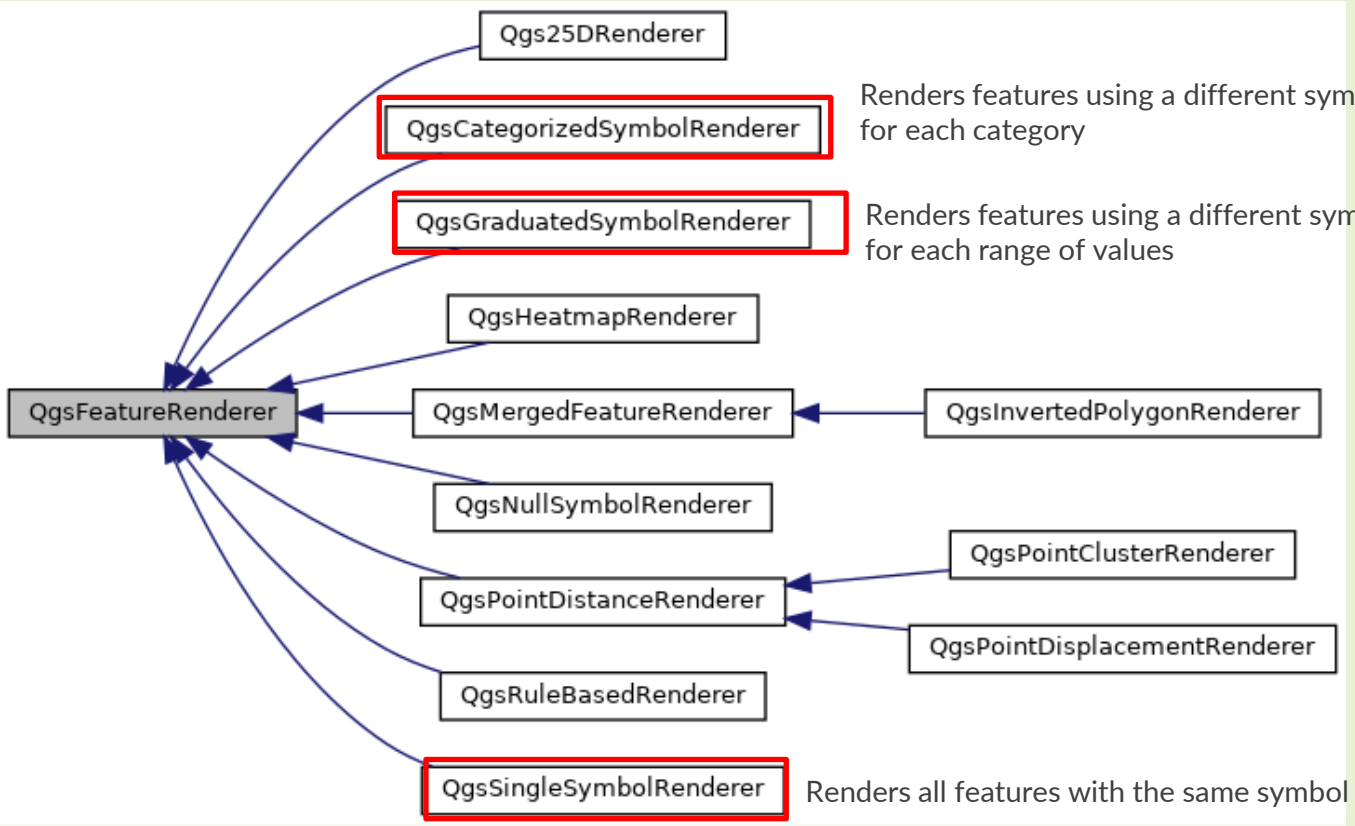
1) Open QGIS Project Data/QgsProject.qgs

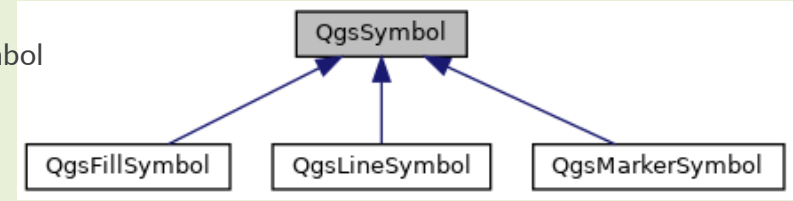
2) Find the layer MP14 Subzone

1) Iterate all features of this layer and export all attributes into a csv file.

2) Iterate all geometry of this layer and export centroid coordinates (X, Y) into a csv file.

qgis.core - Display Vector Data





The renderer chooses the symbol to use for a given feature.

The symbol that does the actual drawing.

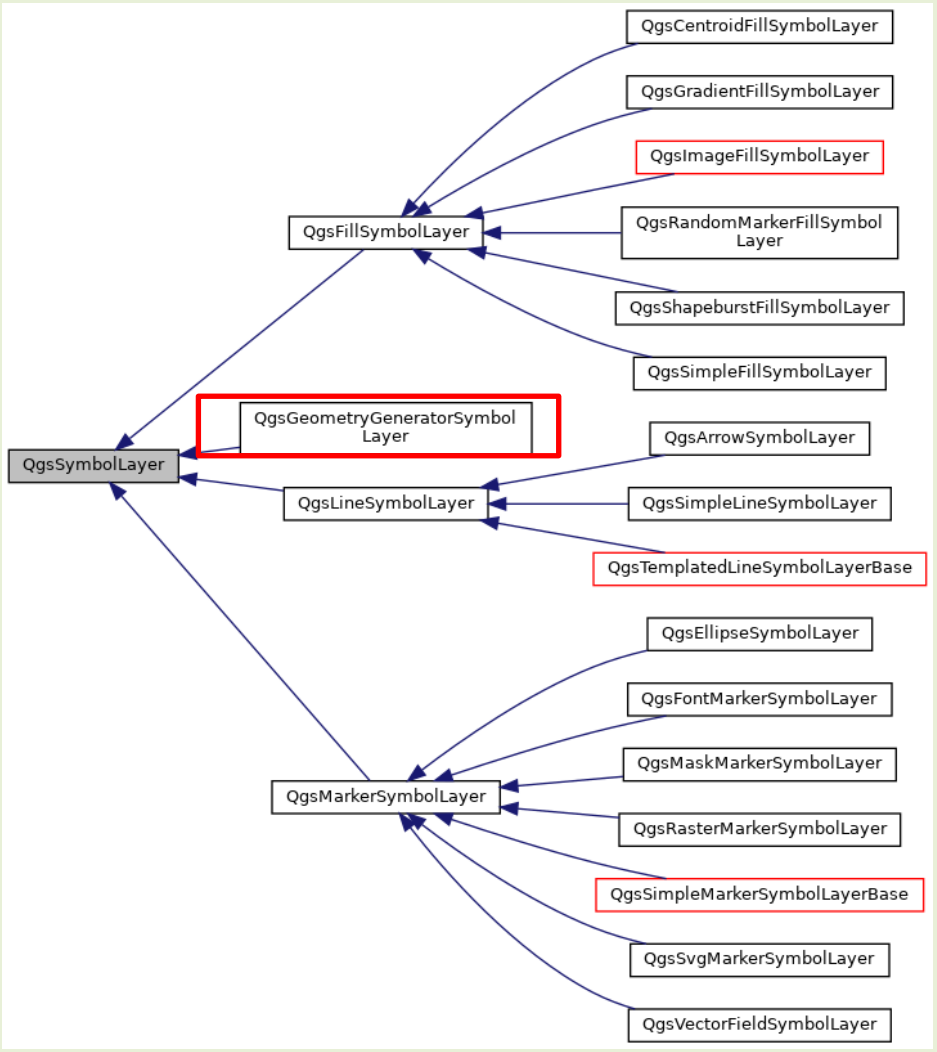
Display Vector Data Example

```
# create a simple symbol with red square
symbol = QgsMarkerSymbol.createSimple({'name': 'x', 'color': 'red'})
layer.renderer().setSymbol(symbol)

# show the change
layer.triggerRepaint()
```

qgis.core - Display Vector Data

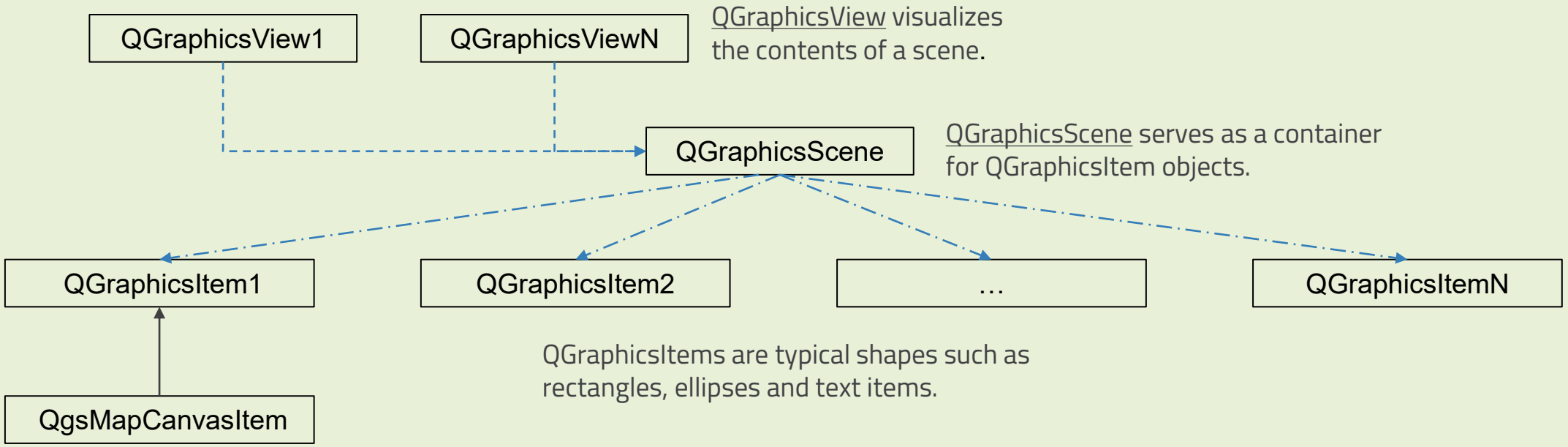
Symbol layers are the building blocks that make up a representation rule.

Symbol layers are categorized by symbol type—fill, line, or marker—to symbolize polygon, line, and point features, respectively.



Map Canvas

The **Map canvas** widget shows the map and allows interaction with the map and layers.



Qt Graphics View Framework

Map Canvas

The **Map canvas** widget shows the map and allows interaction with the map and layers.

map canvas – for viewing of the map
map canvas items – additional items that can be displayed on the map canvas
map tools – for interaction with the map canvas

Expressions, Filtering and Calculating Values

- arithmetic operators: +, -, *, /, ^
- mathematical functions: sqrt, sin, cos, tan, asin, acos, atan
- **conversion functions**: to_int, to_real, to_string, to_date
- geometry functions: $area, $length
- **geometry handling functions**: $x, $y, $geometry, num_geometries, centroid

- comparison: =, !=, >, >=, <, <=
- **pattern matching**: LIKE (using % and _), ~ (regular expressions)
- logical predicates: AND, OR, NOT
- **NULL value checking**: IS NULL, IS NOT NULL

Qt, PyQt, PyQGIS

Qt	QGIS is built using the Qt platform, which is a free and open-source widget toolkit for creating GUI and cross-platform applications (written in C++).
PyQt	PyQt is the Python interface to Qt. PyQt provides classes and functions to interact with Qt widgets.
PyQGIS	PyQGIS are Python APIs provided by QGIS.

Integrate QGIS & Python Libraries

PyQGIS

PyQt

Python

QGIS C++ Lib

Qt

Python

NumPy

Pandas

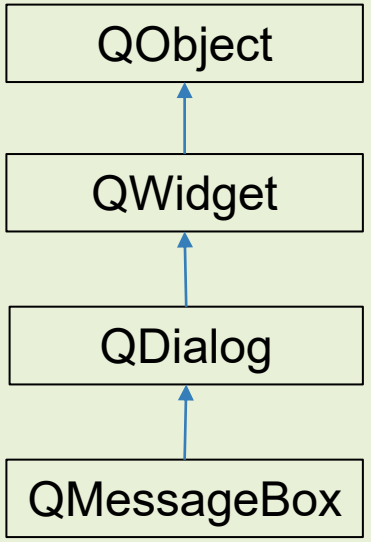
SciPy

Scikit-learn

……

PyQt Example

- Example in QGIS
  - mb = QMessageBox()
- Tell class of an Object:
  - type(mb)
- The attributes and methods of any object
  - dir(mb)

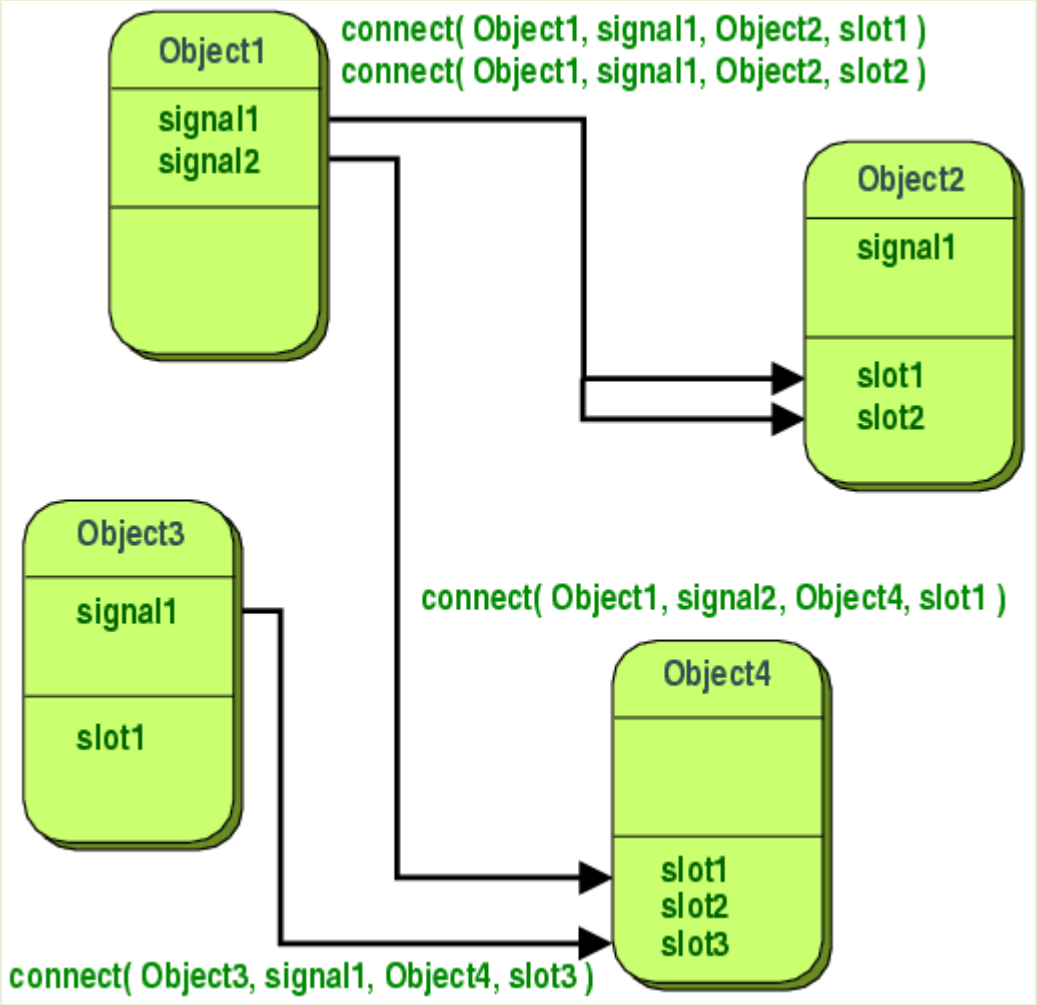


Qt Object Model

The most crucial concept in GUI programming

signals and slots

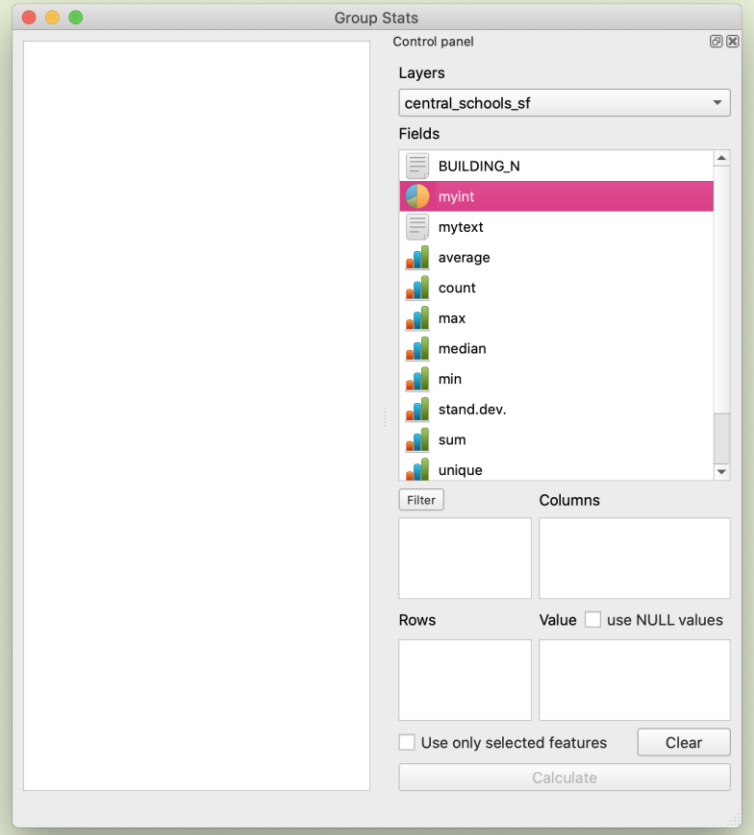
A signal is a trigger which can be emitted. The signal can be connected to a slot, which needs to be Python callable (a method or a class, anything implementing the __call__ magic).

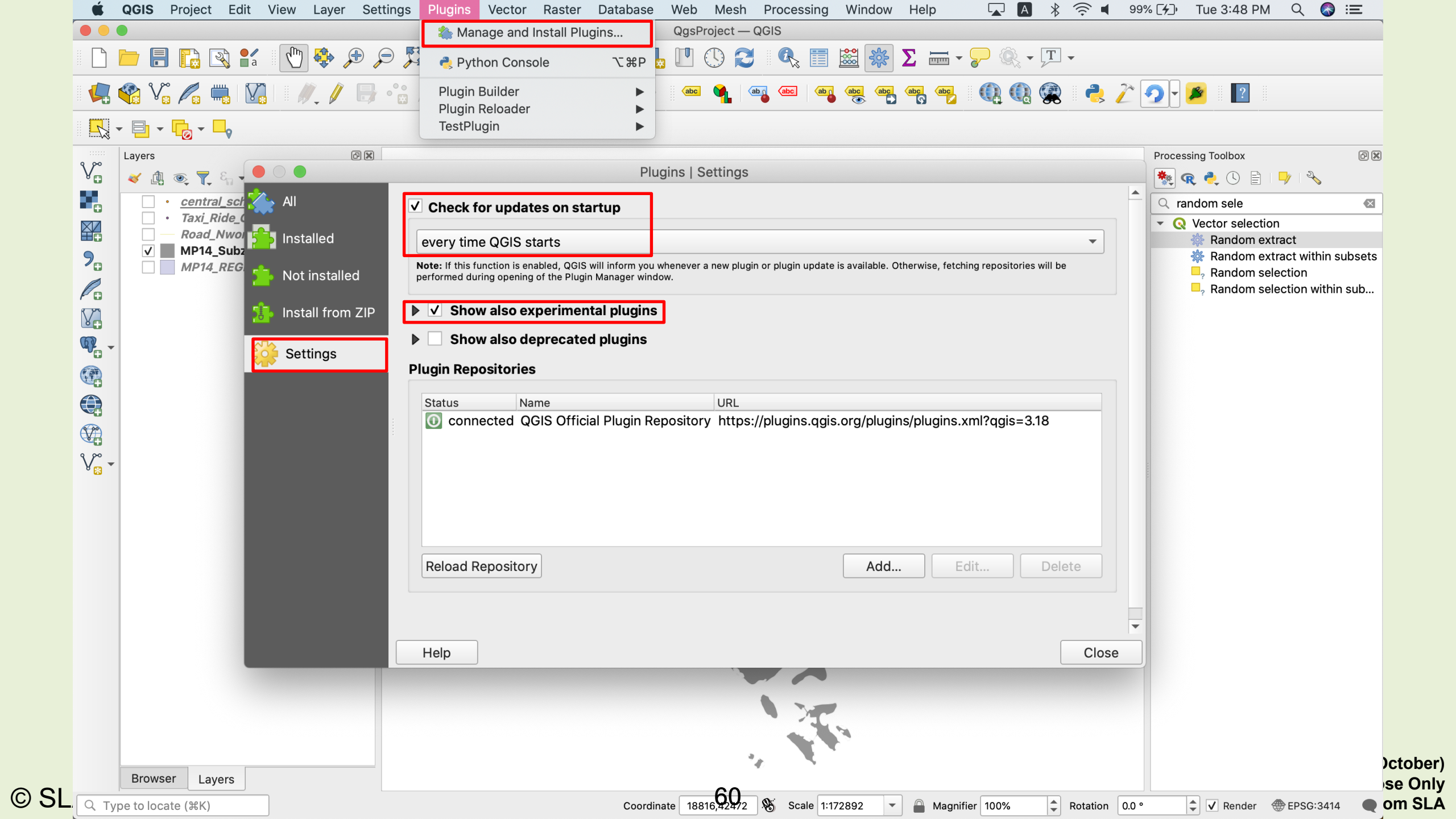


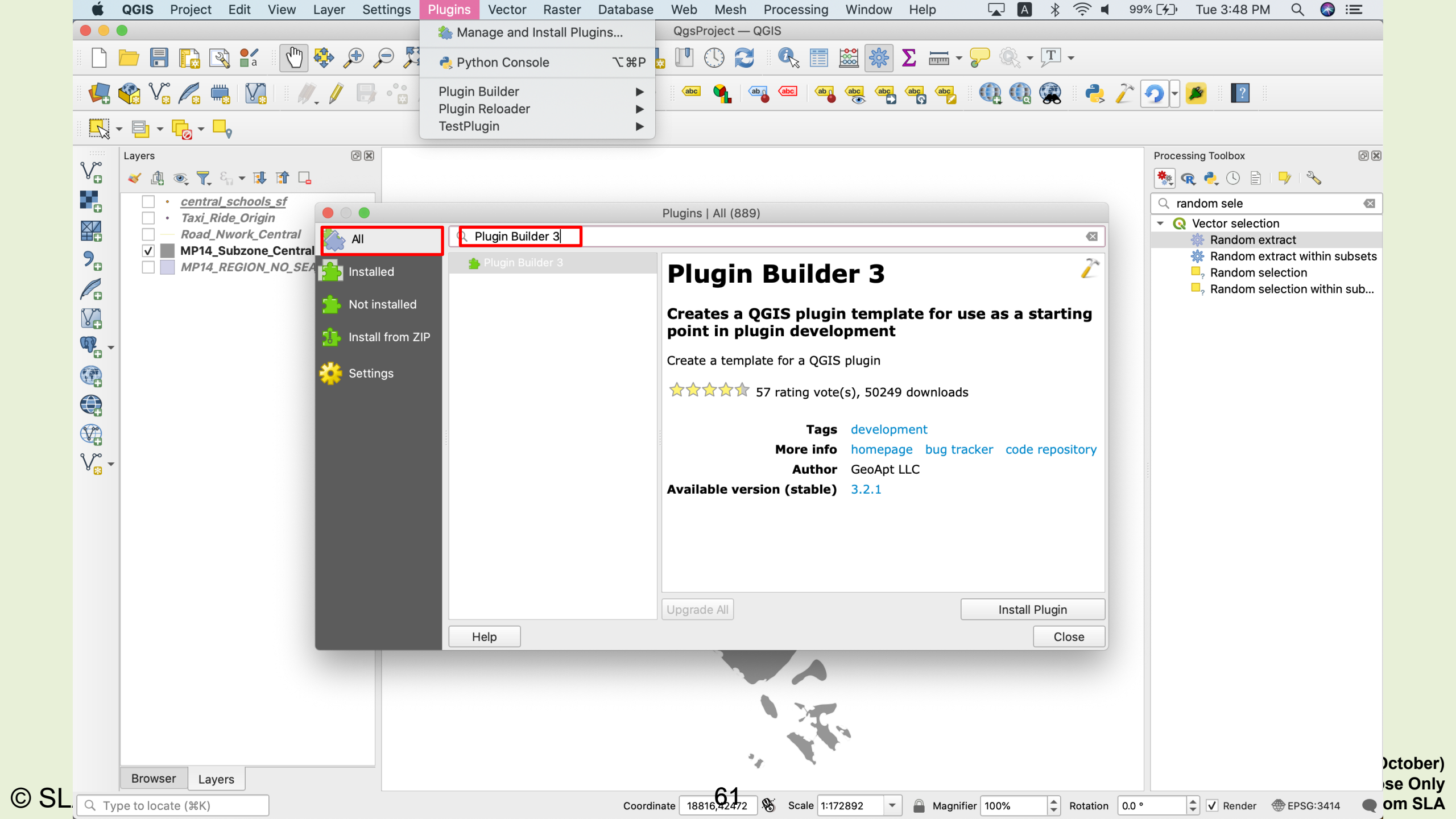
Building a Python Plugin

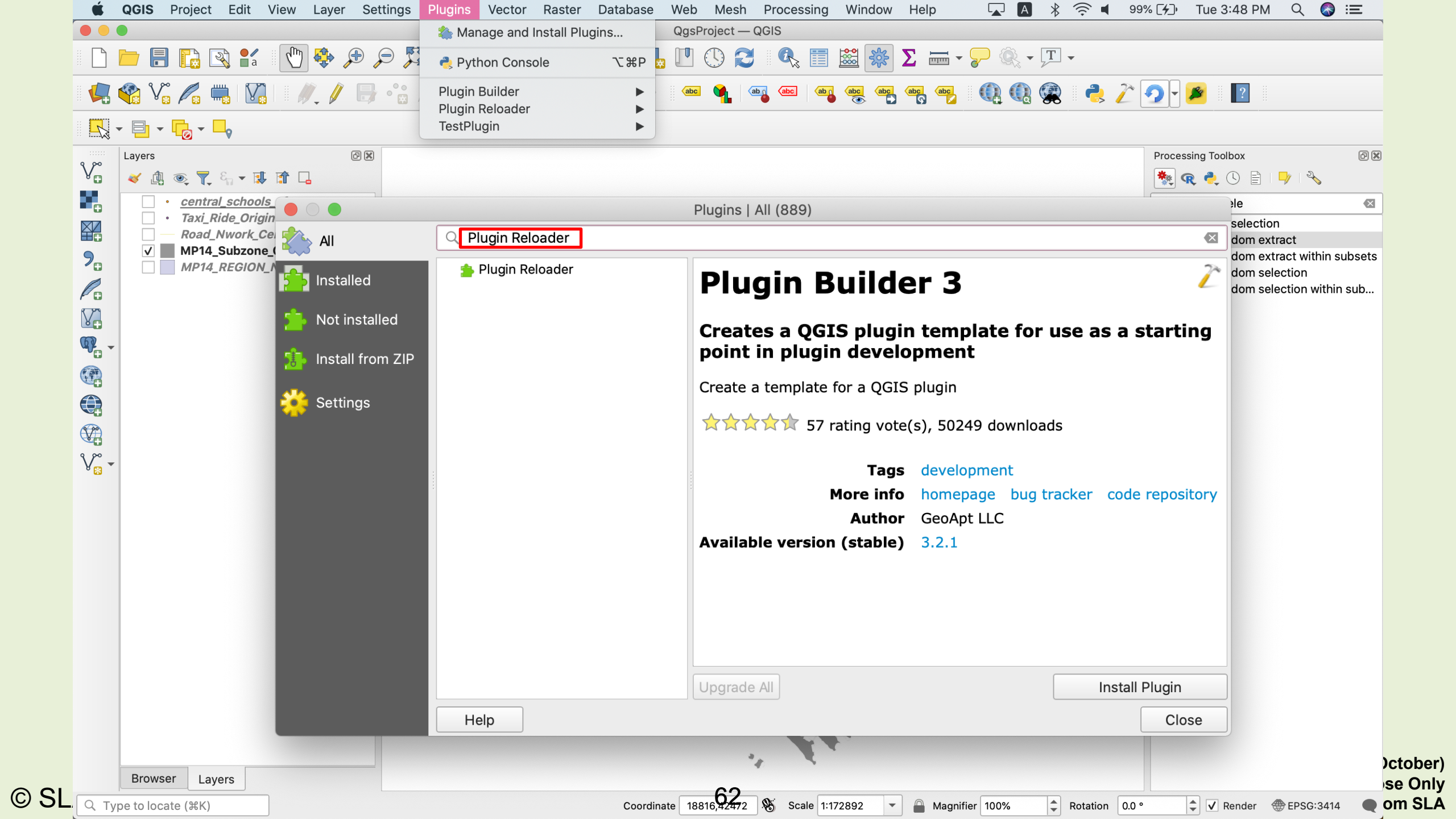
Plugins are a great way to extend the functionality of QGIS.

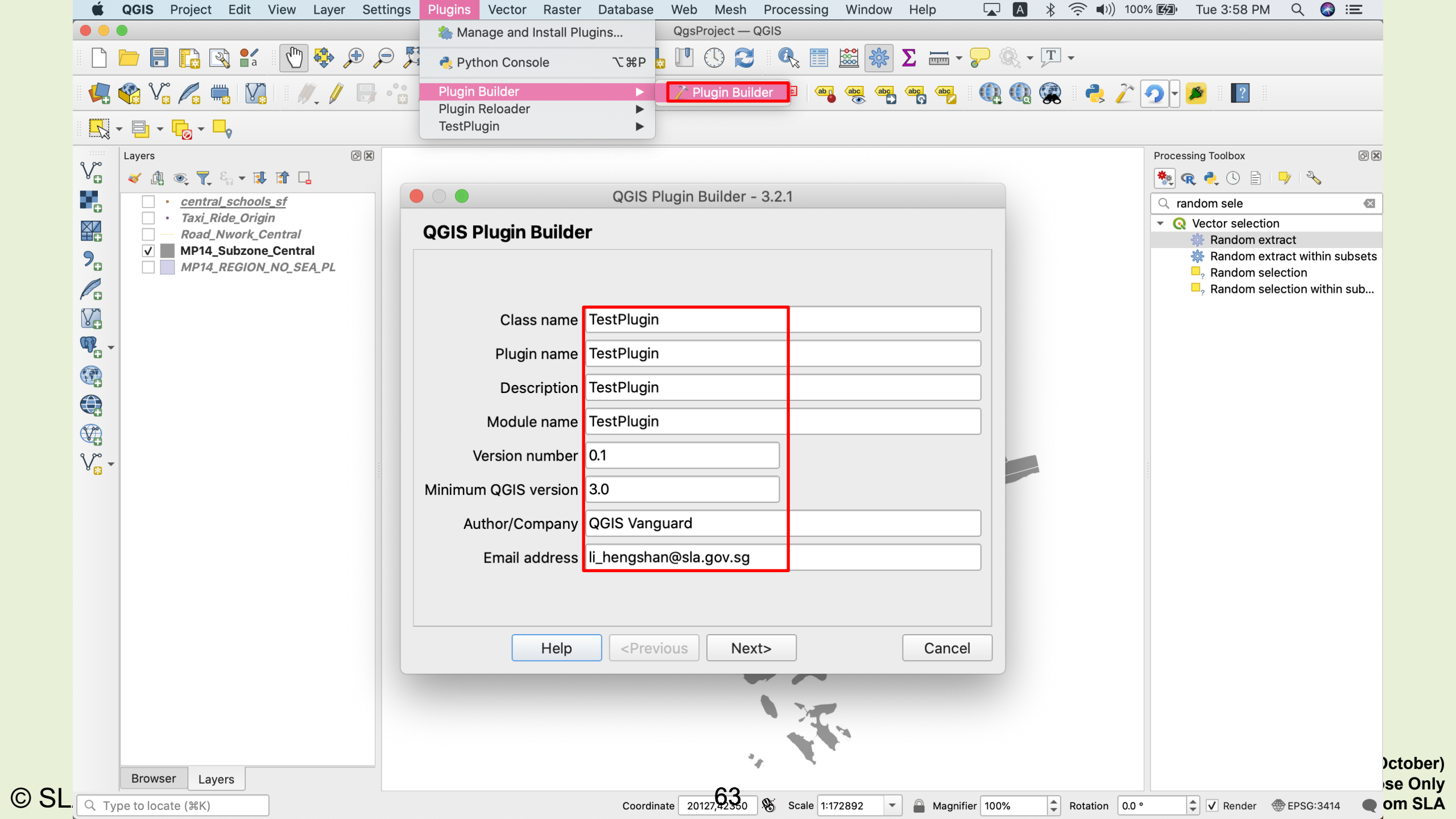
You can write plugins using Python that can range from adding a simple button to sophisticated toolkits.

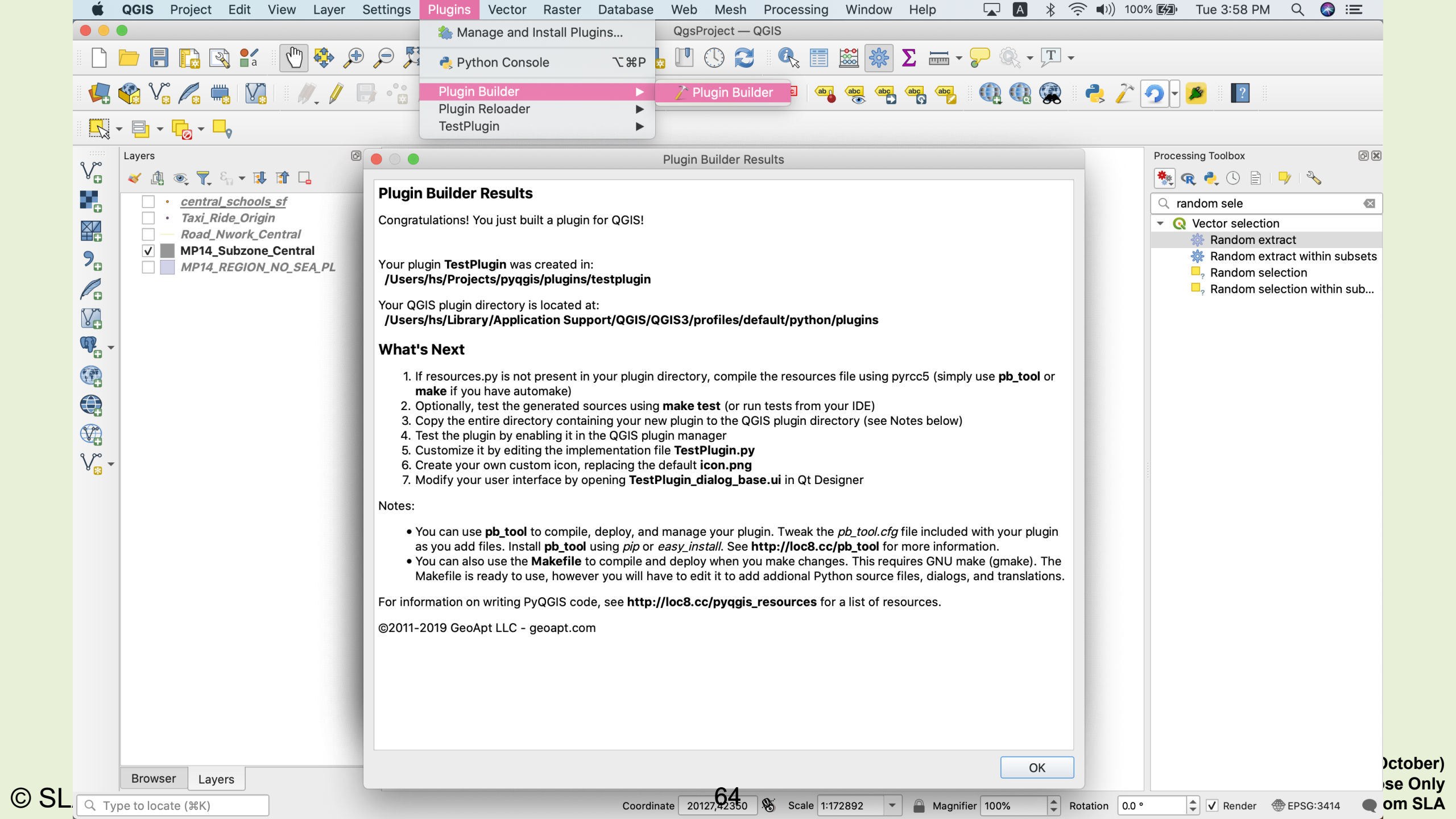


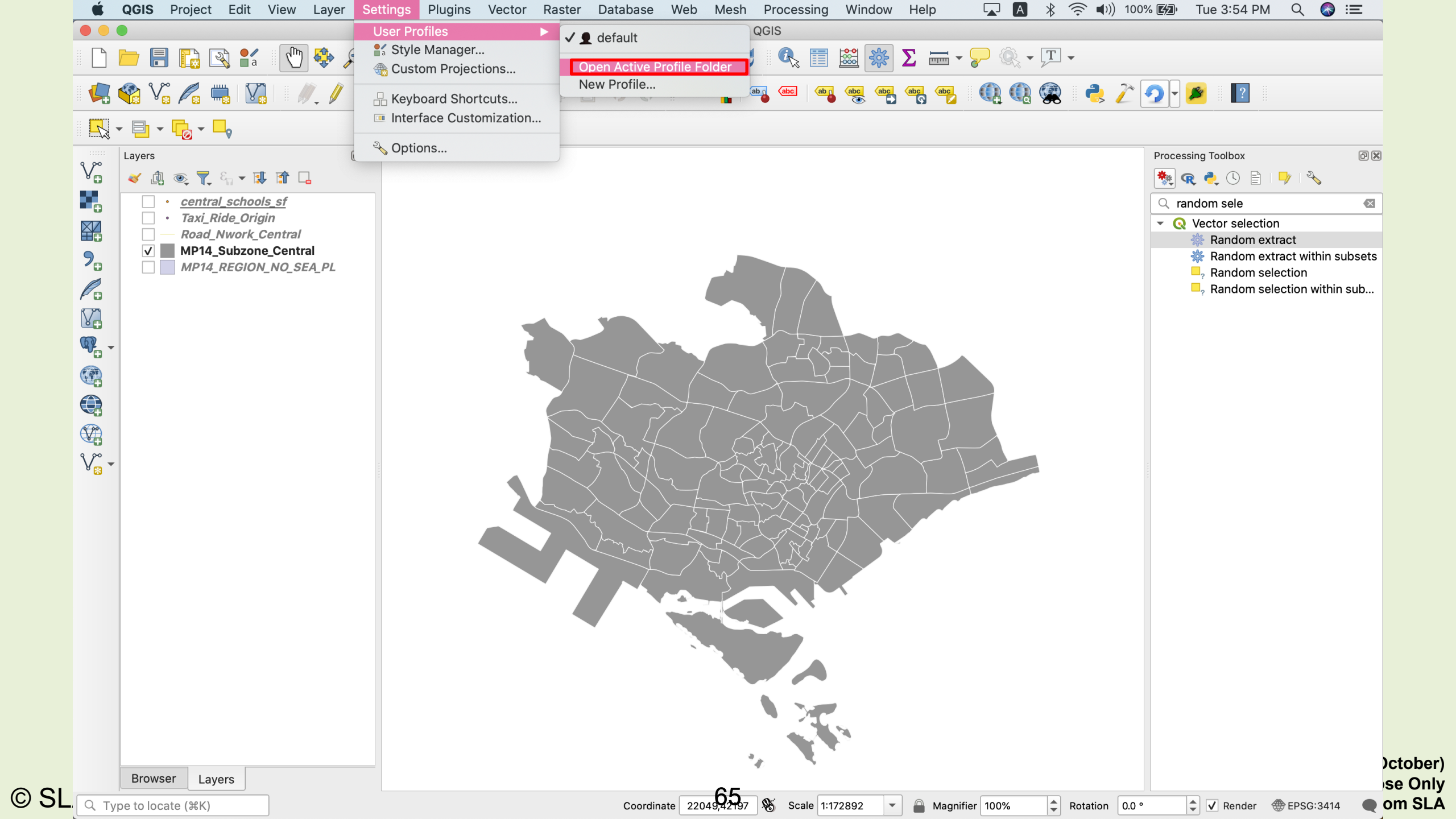









QGIS Project Edit View Layer Settings Plugins Vector Raster Database Web Mesh Processing Window Help
Tue 3:58 PM
Manage and Install Plugins...
Python Console ⌥⌘P
Plugin Builder
Plugin Reloader
TestPlugin
Plugin Builder
QgsProject — QGIS
Layers
central_schools_sf
Taxi_Ride_Origin
Road_Nwork_Central
MP14_Subzone_Central
MP14_REGION_NO_SEA_PL
Plugin Builder Results
Plugin Builder Results
Congratulations! You just built a plugin for QGIS!
Your plugin TestPlugin was created in:
/Users/hs/Projects/pyqgis/plugins/testplugin
Your QGIS plugin directory is located at:
/Users/hs/Library/Application Support/QGIS/QGIS3/profiles/default/python/plugins
What's Next
1. If resources.py is not present in your plugin directory, compile the resources file using pyrcc5 (simply use pb_tool or make if you have automake)
2. Optionally, test the generated sources using make test (or run tests from your IDE)
3. Copy the entire directory containing your new plugin to the QGIS plugin directory (see Notes below)
4. Test the plugin by enabling it in the QGIS plugin manager
5. Customize it by editing the implementation file TestPlugin.py
6. Create your own custom icon, replacing the default icon.png
7. Modify your user interface by opening TestPlugin_dialog_base.ui in Qt Designer
Notes:
• You can use pb_tool to compile, deploy, and manage your plugin. Tweak the pb_tool.cfg file included with your plugin as you add files. Install pb_tool using pip or easy_install. See http://loc8.cc/pb_tool for more information.
• You can also use the Makefile to compile and deploy when you make changes. This requires GNU make (gmake). The Makefile is ready to use, however you will have to edit it to add addional Python source files, dialogs, and translations.
For information on writing PyQGIS code, see http://loc8.cc/pyqgis_resources for a list of resources.
©2011-2019 GeoApt LLC - geoapt.com
OK
Processing Toolbox
random sele
Vector selection
Random extract
Random extract within subsets
Random selection
Random selection within sub...
Browser
Layers
Type to locate (⌘K)
Coordinate 20127,42350
Scale 1:172892
Magnifier 100%
Rotation 0.0 °
Render
EPSG:3414



Plugin Folder Structure

PYTHON_PLUGINS_PATH/
MyPlugin/
__init__.py --> *required* . The starting point of the plugin. It has to have the classFactory() method.
mainPlugin.py --> *core code*
metadata.txt --> *required* Contains general info, version, name and some other metadata used by plugins website and plugin infrastructure.
resources.qrc --> *likely useful*
resources.py --> *compiled version, likely useful*
form.ui --> *likely useful*
form.py --> *compiled version, likely useful*

__init__.py

```
def classFactory(iface):
    from .mainPlugin import TestPlugin
    return TestPlugin(iface)
```

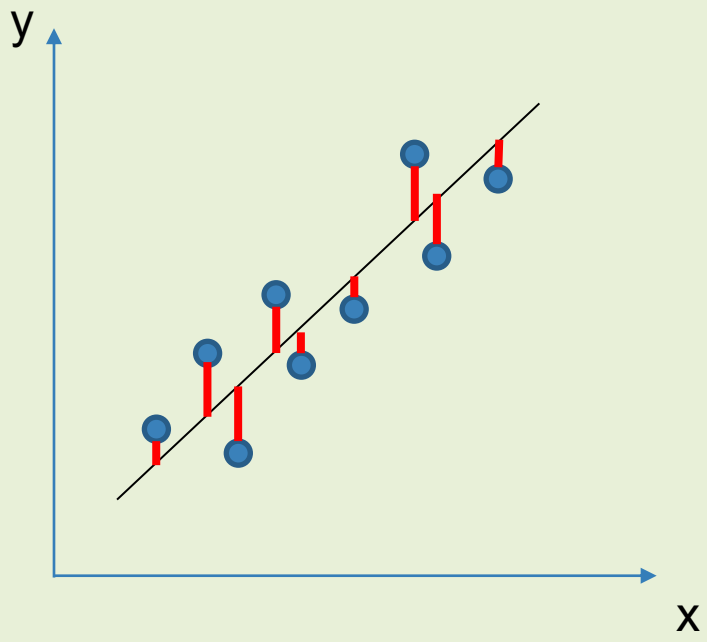
mainPlugin.py

```
from qgis.PyQt.QtGui import *
from qgis.PyQt.QtWidgets import *
# initialize Qt resources from file resources.py
from . import resources
class TestPlugin:

    def __init__(self, iface):
        # save reference to the QGIS interface
        self.iface = iface

    def initGui(self):
        # create action that will start plugin configuration self.action =
        QAction(QIcon(":/plugins/testplug/icon.png"), "Test plugin",
        self.iface.mainWindow()) self.action.setObjectName("testAction")
```

mainPlugin.py

```
        self.action.setStatusTip("This is status tip")
        self.action.triggered.connect(self.run)
        self.iface.addToolBarIcon(self.action)
        self.iface.addPluginToMenu("&Test plugins", self.action)
        self.iface.mapCanvas().renderComplete.connect(self.renderTest)

    def unload(self):
        self.iface.removePluginMenu("&Test plugins", self.action)
        self.iface.removeToolBarIcon(self.action)
        self.iface.mapCanvas().renderComplete.disconnect(self.renderTest)

    def run(self):
        print("TestPlugin: run called!")

    def renderTest(self, painter):
        print("TestPlugin: renderTest called!"))
```

Linear Regression Model



$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

- Coefficient (slope)
- Residuals
- R-squared (variance explained)

$$R^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

OLS Assumptions:

1. The regression model is linear in the coefficients and the error term.
2. The error term has a population mean of zero.
3. All independent variables are uncorrelated with the error term.
4. Observations of the error term are uncorrelated with each other
5. The error term has a constant variance (no heteroscedasticity)
6. No independent variable is a perfect linear function of other explanatory variables (avoid multicollinearity)
7. The error term is normally distributed (optional)

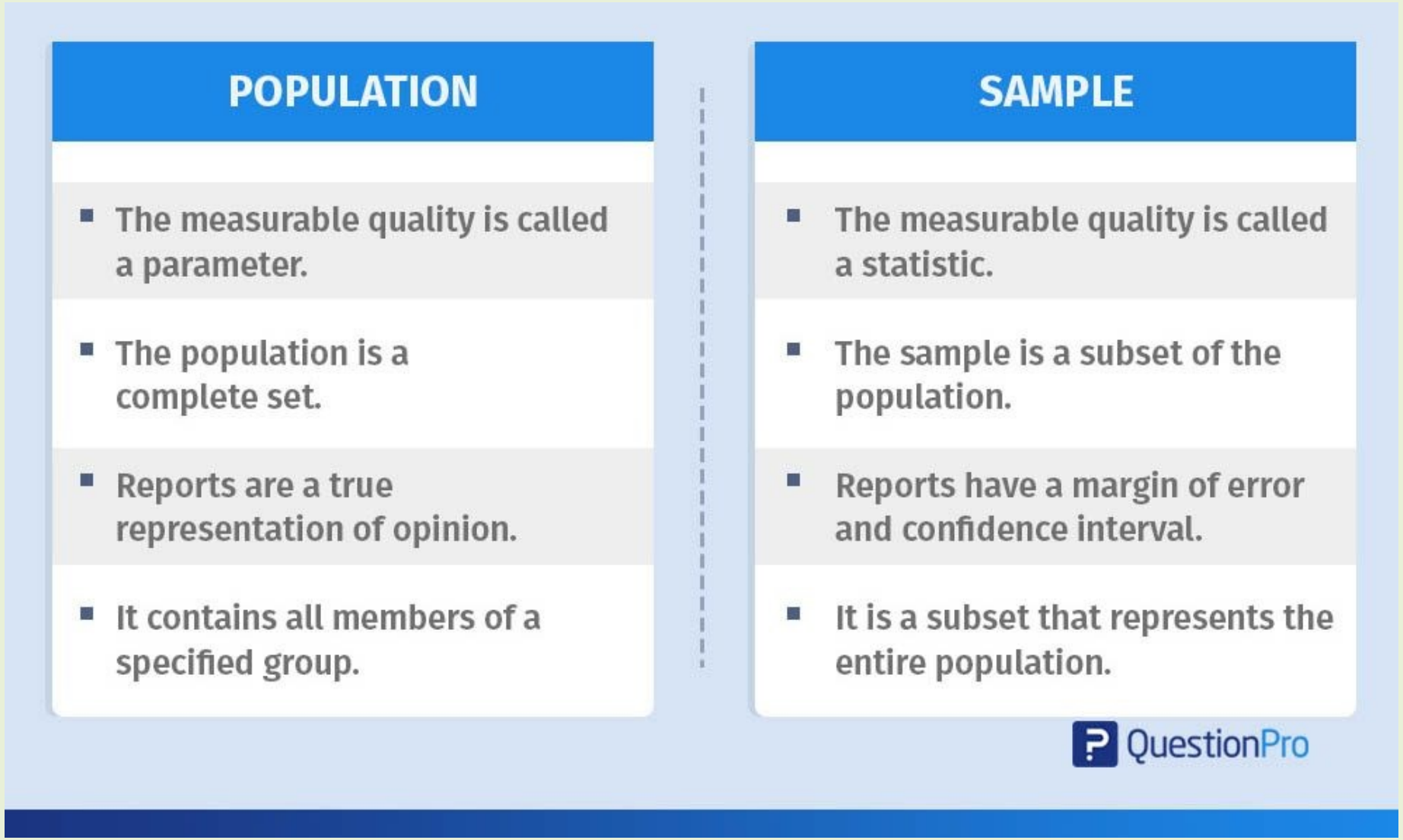
Linear Regression Models

- Ordinary Least Squares (OLS)
- Spatial Auto-Regression
- Geographically Weighted Regression (GWR)

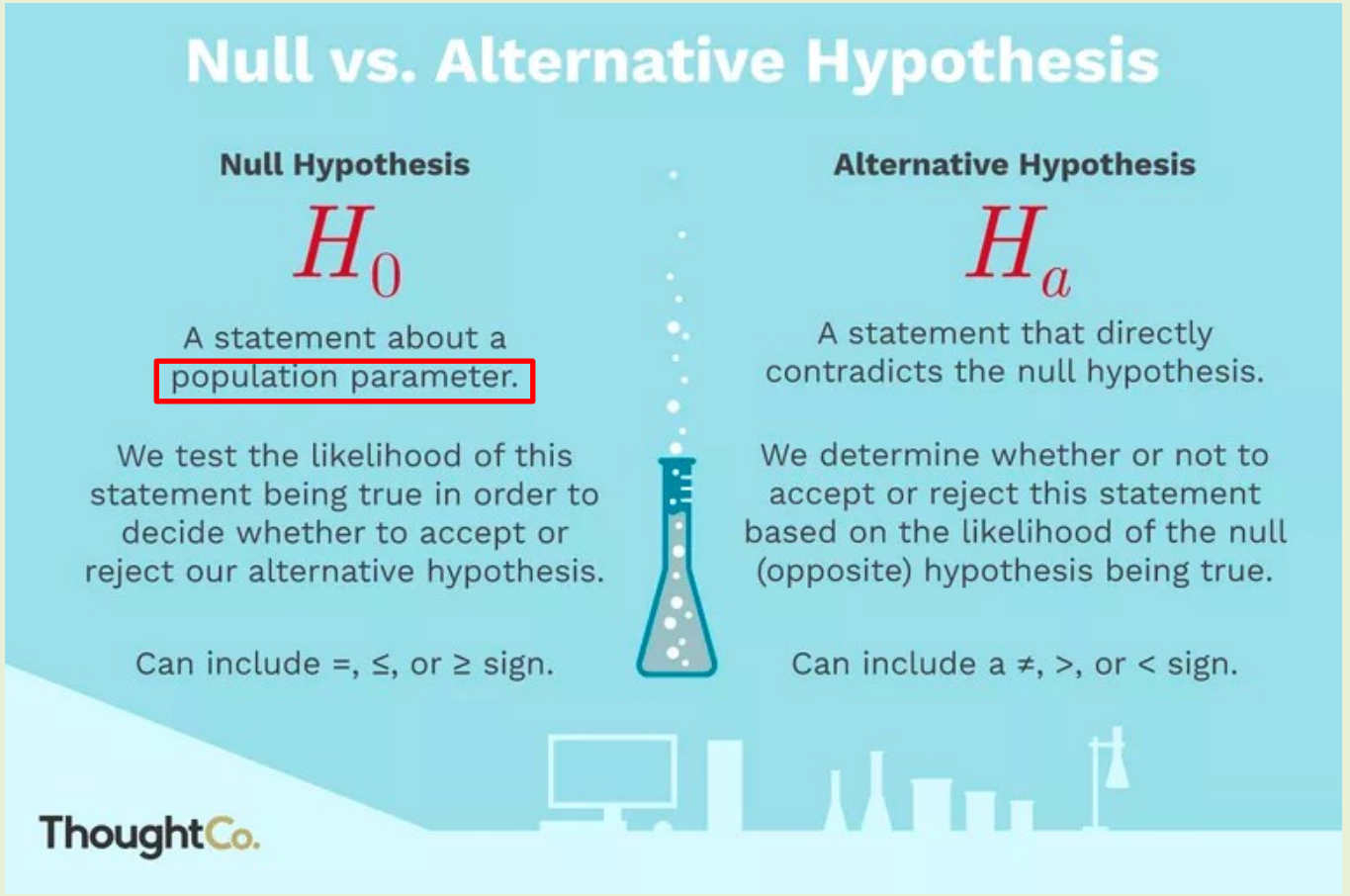
Spatial Autocorrelation

- Why?
  - Regression
  - Residuals

- What is autocorrelation?

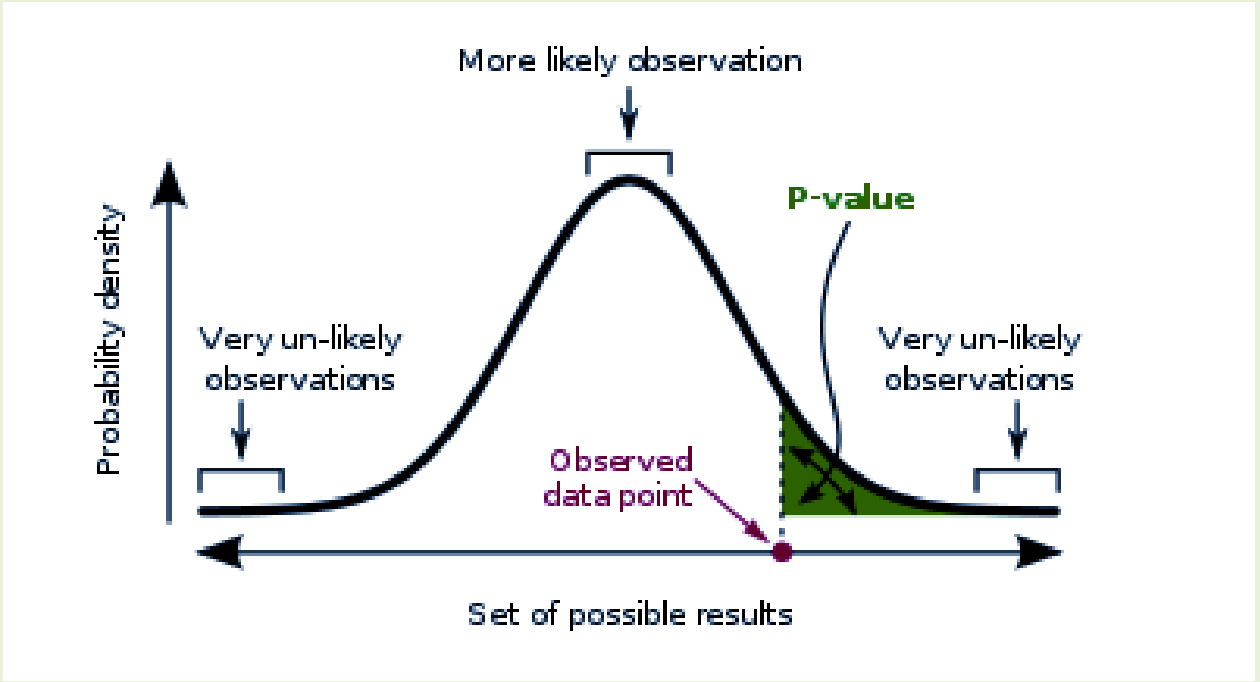


Null vs. Alternative Hypothesis



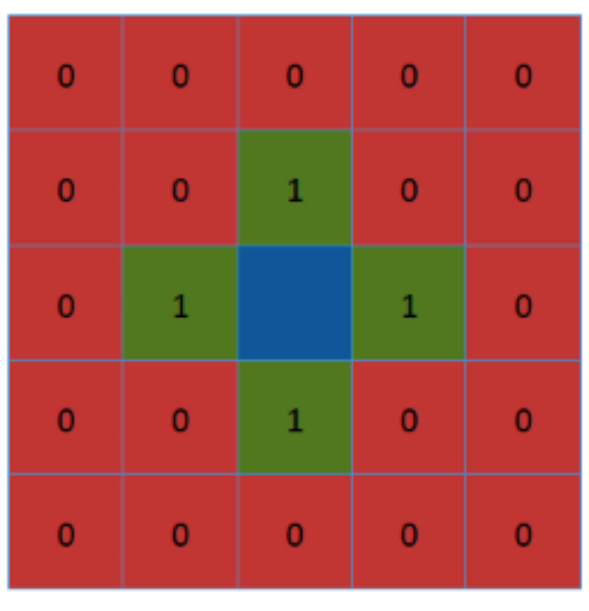
p-Value

- The probability that, when the null hypothesis is true, the statistical summary is equal to or greater than the actual observed results.

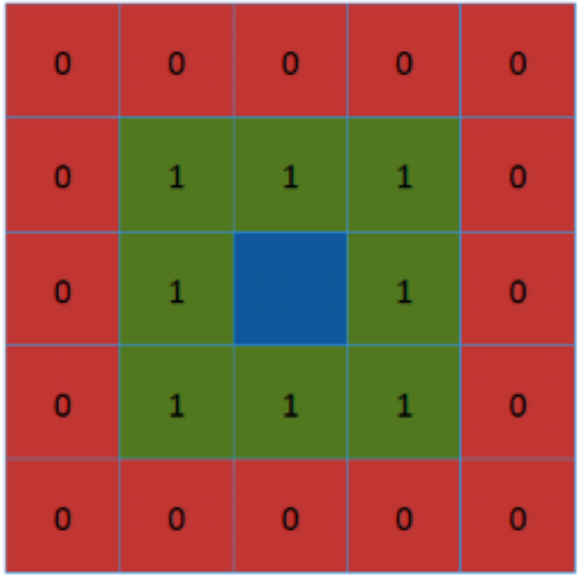


https://data-flair.training/blogs/python-statistics/

Spatial Weights



Rook Weights



Queen Weights

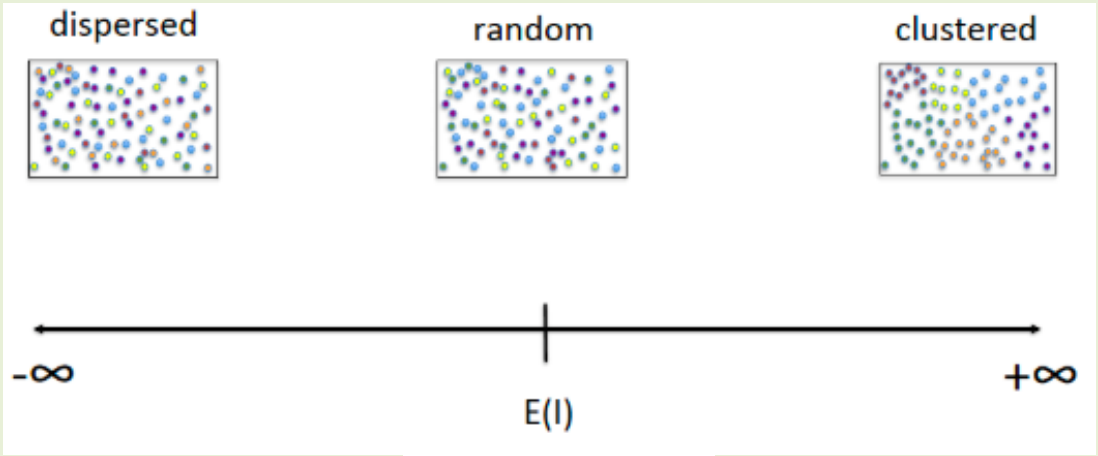
Spatial Autocorrelation

- How to test spatial autocorrelation?
  - Moran's I
  - Variogram
- What it can be used to. Modelling
  - Kriging
  - GAM
- Spatial regression

Moran's I

$$I = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}(x_i - \bar{x})(x_j - \bar{x})}{\left(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j}\right)\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Global Moran's *I*



$$E(I) = \frac{-1}{n-1}$$

$$Z = \frac{I - E_I}{\sqrt{V(I)}}$$

Local Moran's I

$$I_i = \frac{x_i - \bar{X}}{S_i^2} \sum_{j=1}^{n} w_{i,j}(x_j - \bar{X})$$

$$S_i^2 = \frac{\sum_{j=1}^{n}(x_j - \bar{X})^2}{n-1}$$

$$Z_{I_i} = \frac{I_i - \mathrm{E}[I_i]}{\sqrt{V[I_i]}}$$

$$\mathrm{E}[I_i] = -\frac{\sum_{j=1}^{n} w_{i,j}}{n-1}$$

$$V[I_i] = \mathrm{E}[I_i^2] - \mathrm{E}[I_i]^2$$

Local Moran's *I*