

Criando Plugins QGIS com pyQGIS

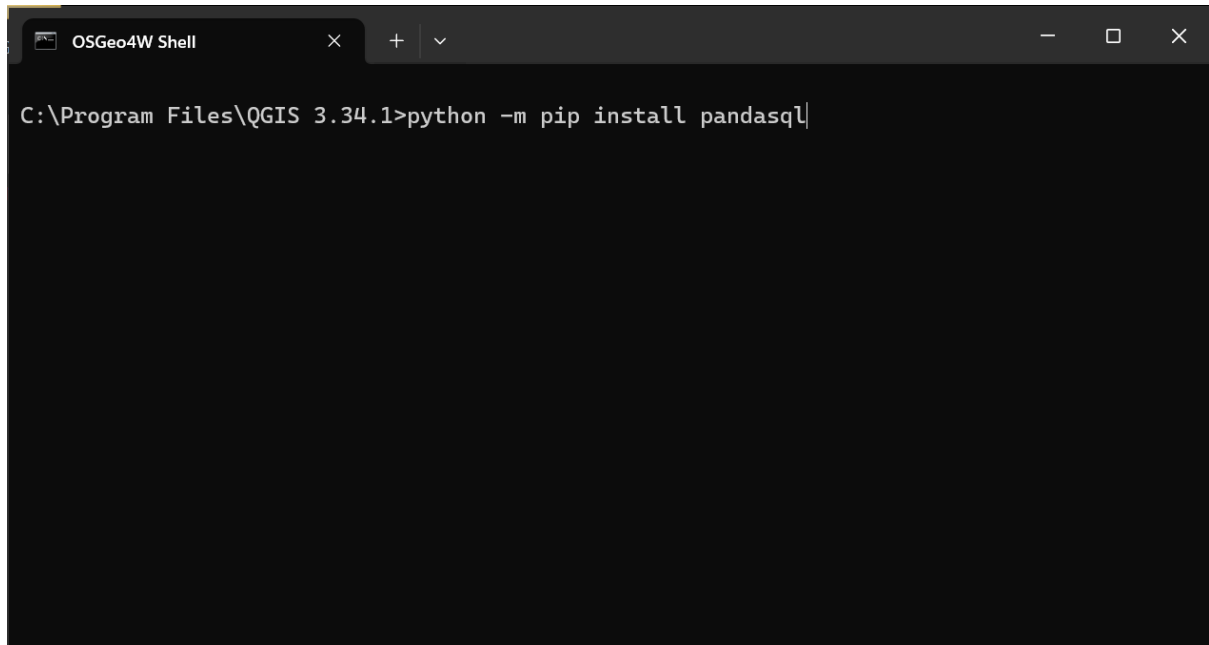
Módulo 4 –Plugin Drillhole1

1 - O ambiente pyQGIS

O QGIS possui um ambiente python dedicado e para instalar novas bibliotecas usamos o shell OSGeo4W. Nesse exemplo de plugin vamos utilizar a biblioteca **pandasql**.

Inicie o shell e digite:

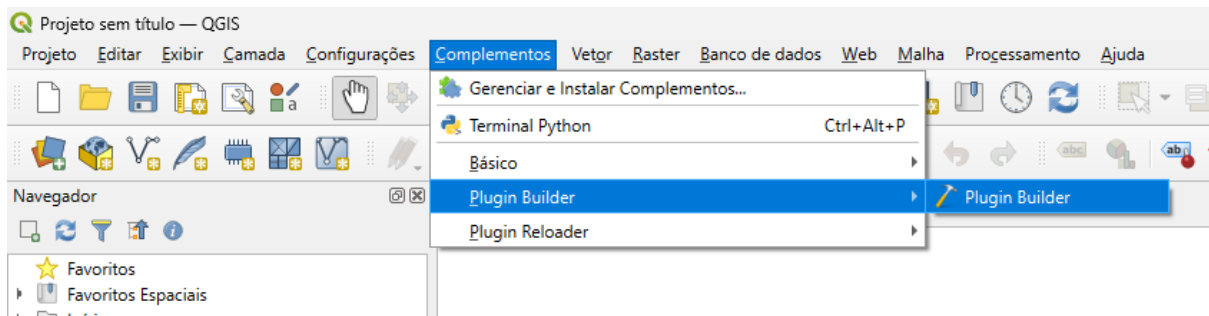
```
python -m pip install pandasql
```

A screenshot of a terminal window titled "OSGeo4W Shell". The window has a dark background and a light-colored text area. The text in the terminal shows the command "C:\Program Files\QGIS 3.34.1>python -m pip install pandasql" being entered. The cursor is at the end of the command. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

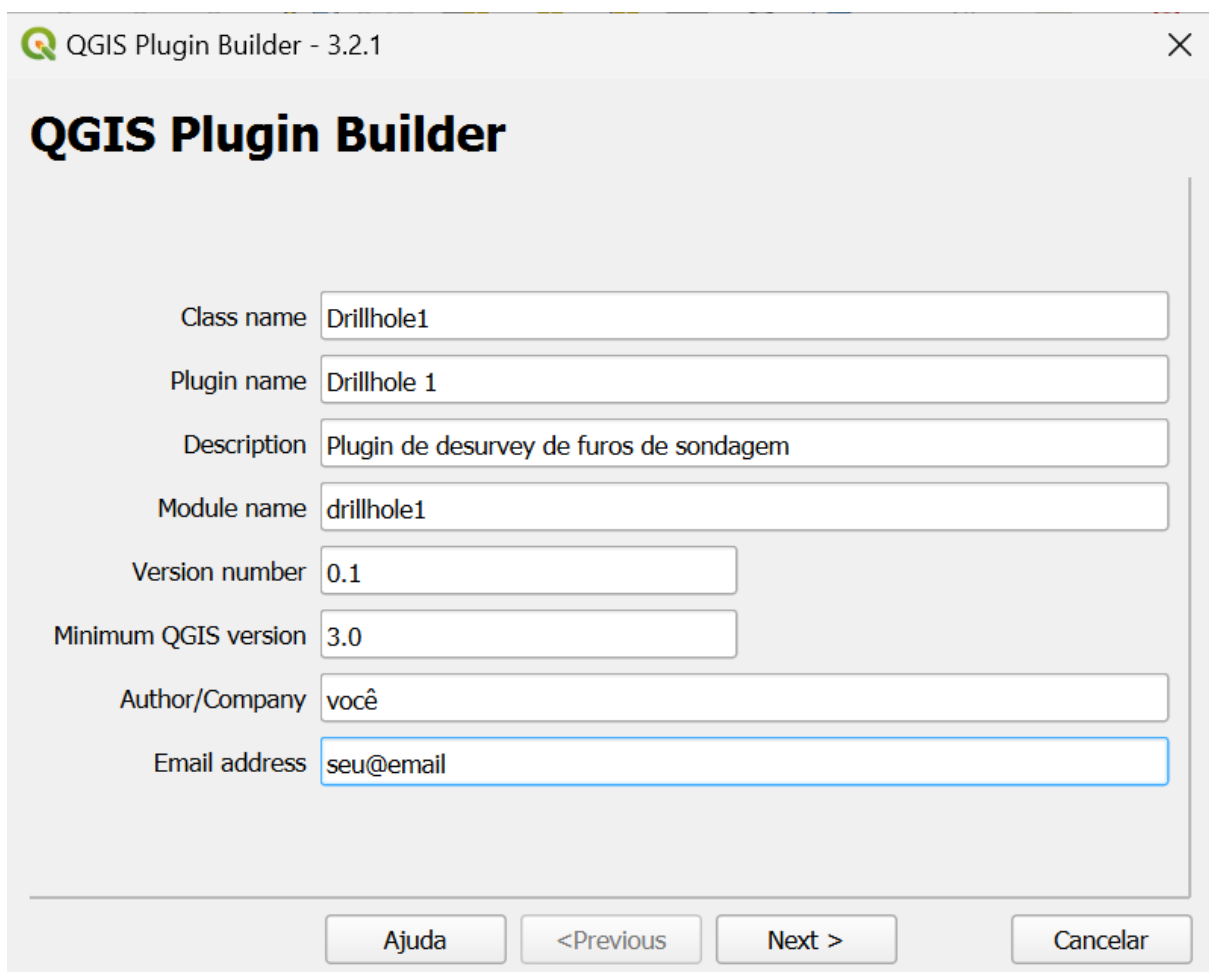
A biblioteca adicional necessária para nosso plugin foi instalada. Procederemos agora com a construção do plugin novo.

2 - Construindo o esqueleto do Drillhole1 no Plugin Builder 3

Inicie o Plugin Builder:



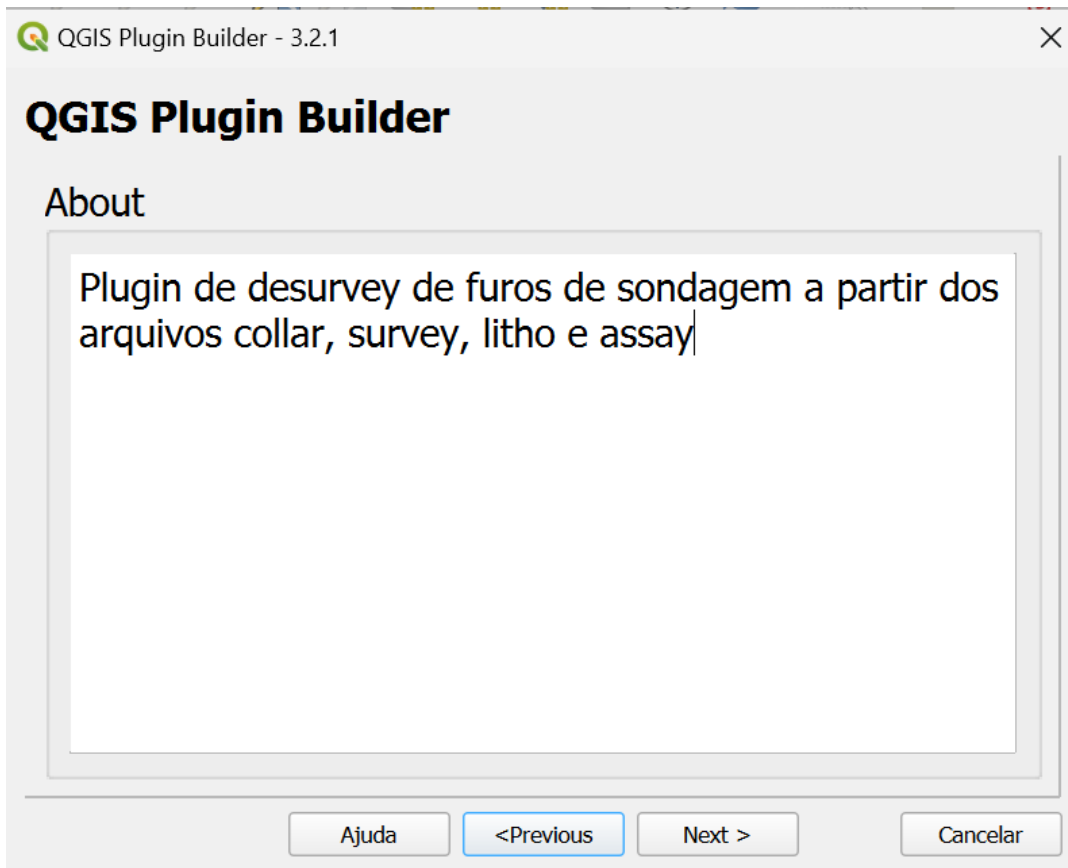
Preencha os campos dos formulários conforme as imagens a seguir:

A screenshot of the 'QGIS Plugin Builder - 3.2.1' dialog box. The title bar shows the QGIS logo and the text 'QGIS Plugin Builder - 3.2.1'. The main area has the title 'QGIS Plugin Builder' in large bold letters. Below the title are several input fields:

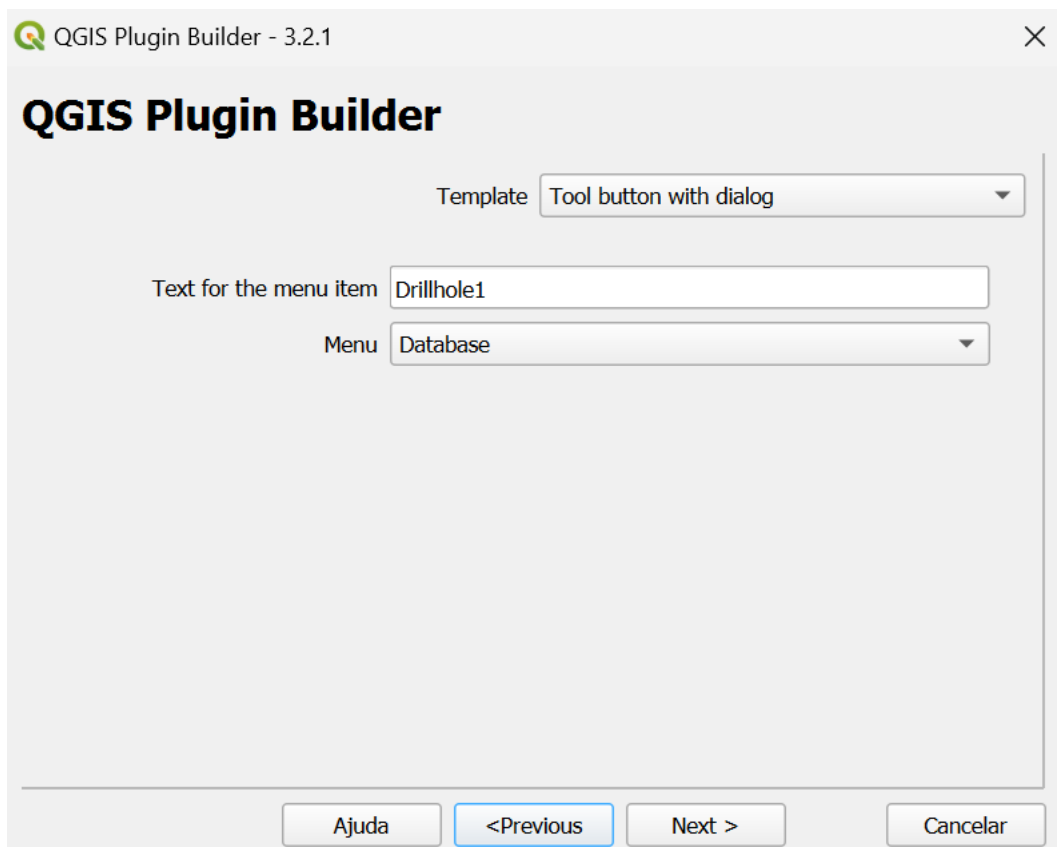
- Class name: Drillhole1
- Plugin name: Drillhole 1
- Description: Plugin de desurvey de furos de sondagem
- Module name: drillhole1
- Version number: 0.1
- Minimum QGIS version: 3.0
- Author/Company: você
- Email address: seu@email

At the bottom of the dialog are four buttons: 'Ajuda', '<Previous', 'Next >', and 'Cancelar'.

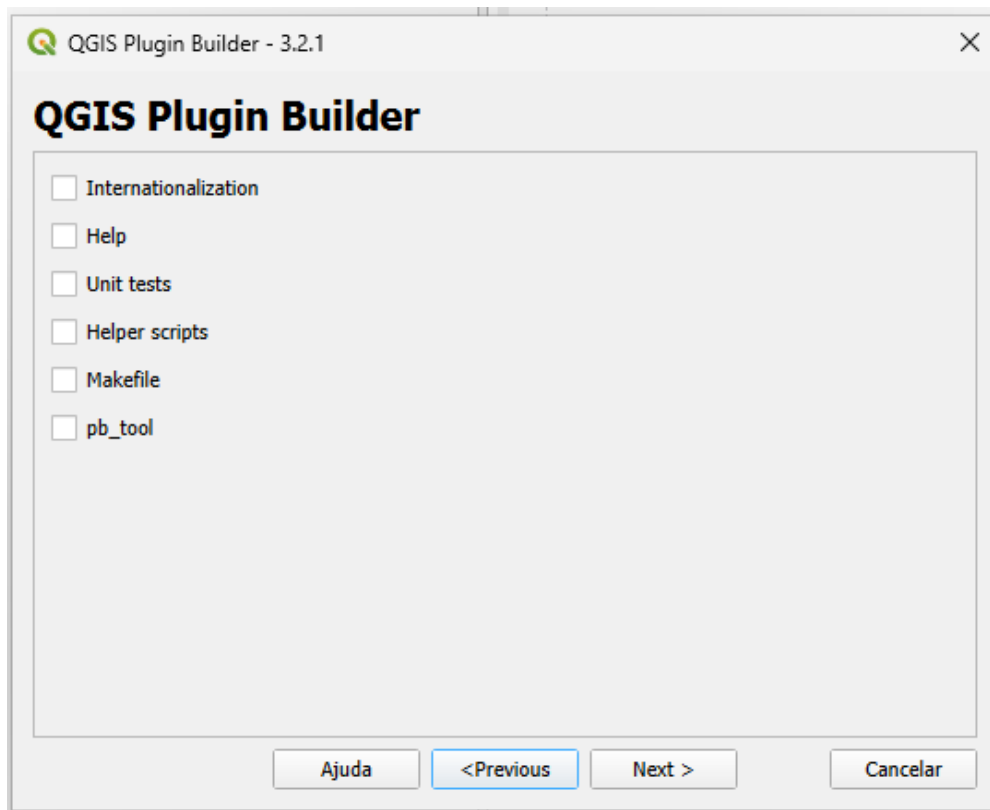
Esse primeiro formulário será usado na criação do arquivo metadata.txt e na definição do nome das classes do plugin.



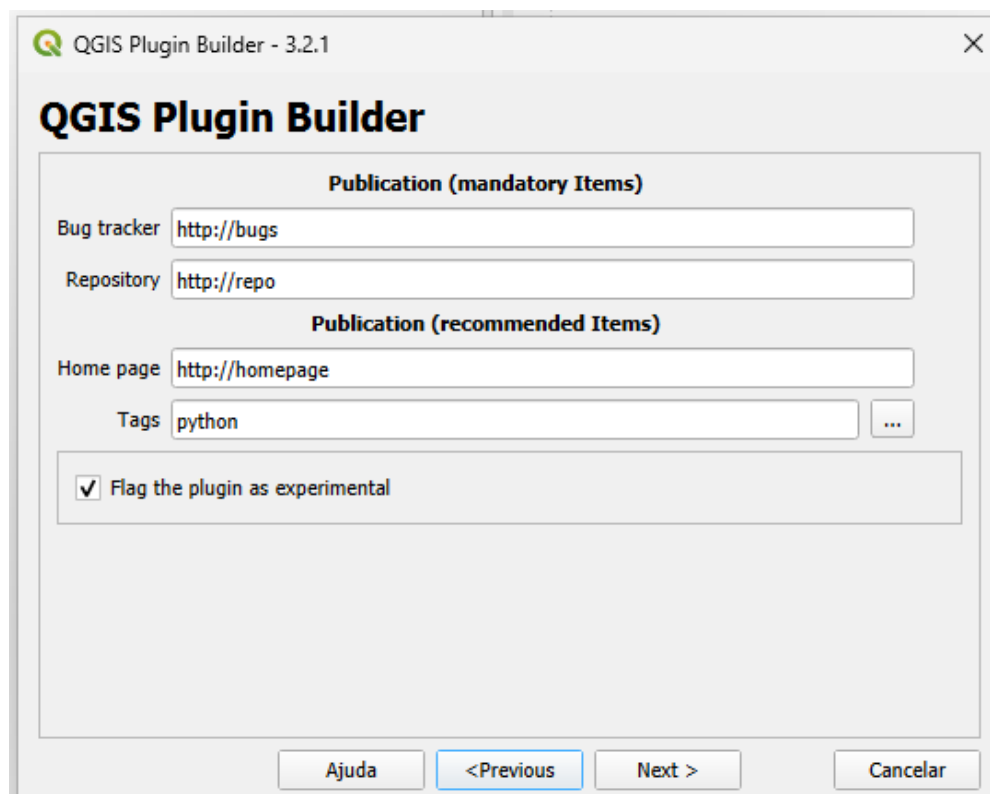
Descrição mais detalhada sobre o plugin que também será colocado no arquivo metadata.txt.



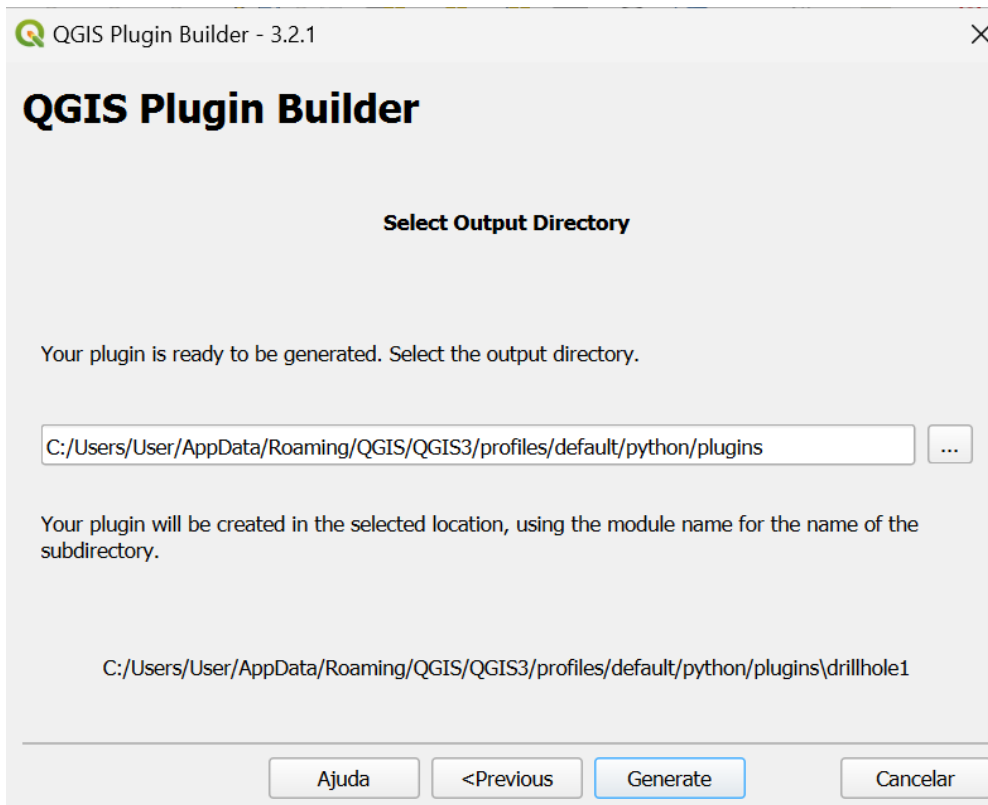
Template (tipo) do plugin, texto que vai aparecer no menu e em qual menu será listado o plugin.



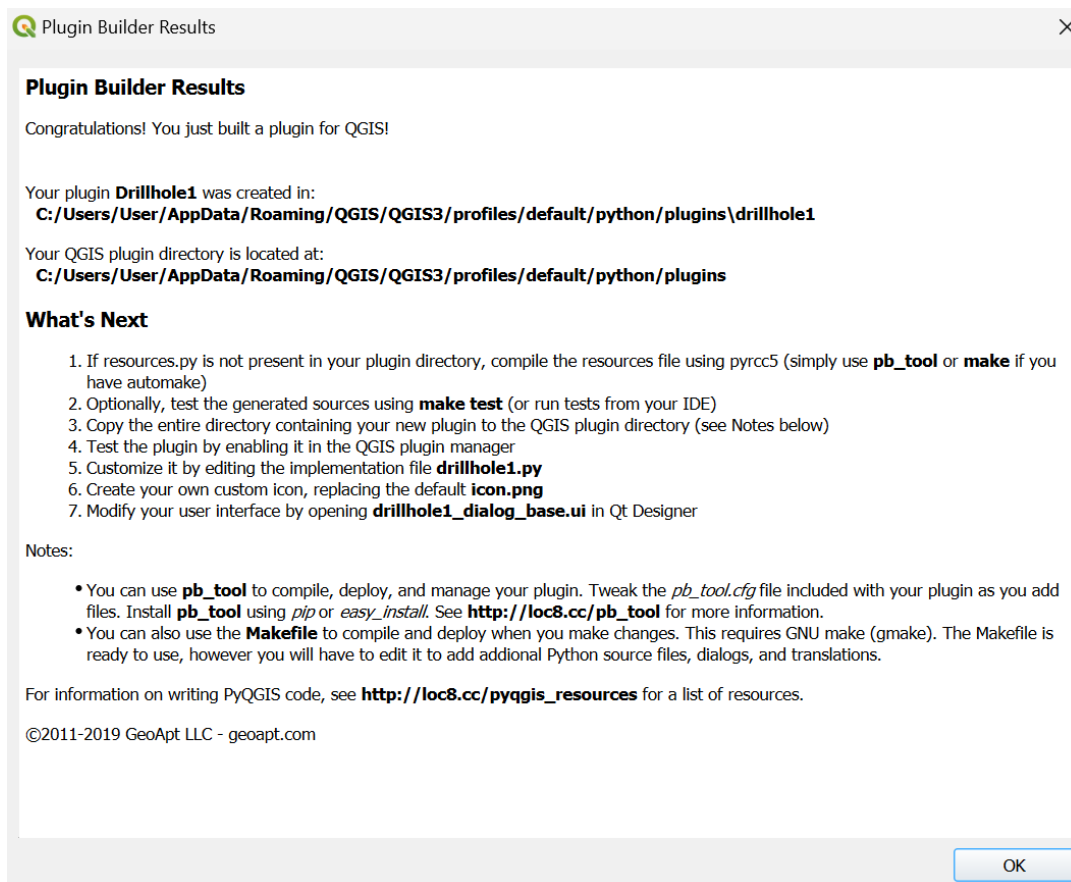
Desmarque todos para esse plugin,



Cheque a caixa de plugin experimental pois não iremos distribuir esse plugin no momento.



A pasta de plugin do sistema (nesse caso em sistema Windows). Clique **Generate** após selecionar o diretório de plugins.

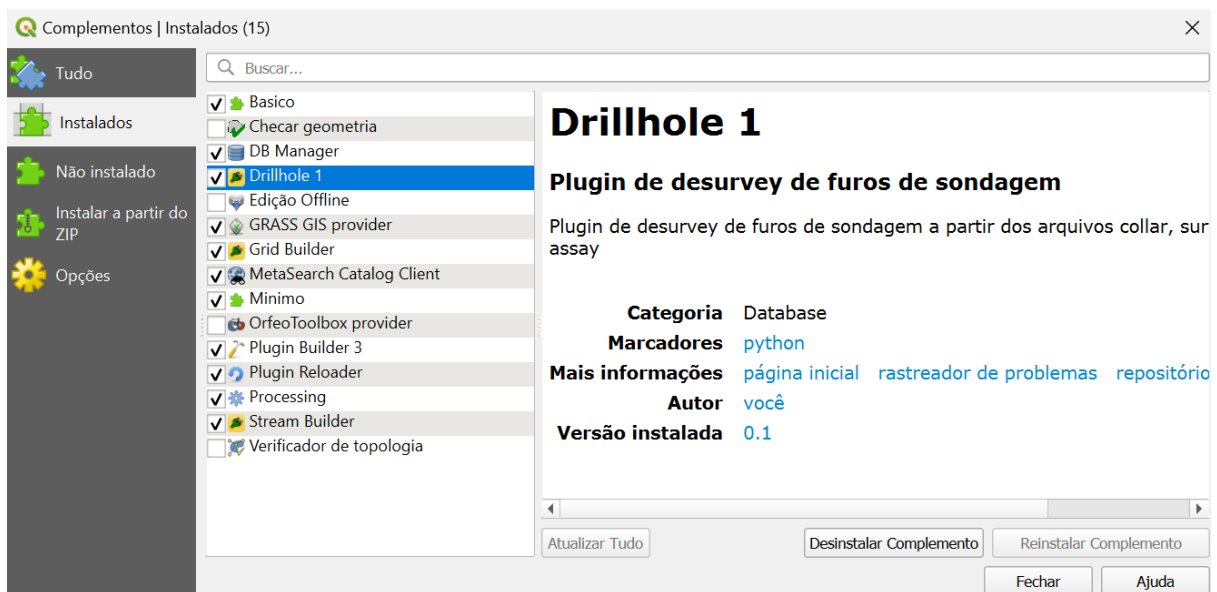


Pronto, os arquivos base de seu plugin foram criados na pasta:

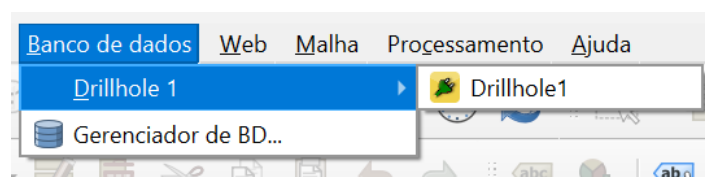
C:/Users/User/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins\drillhole1

Os oito arquivos necessários mais dois arquivos README com instruções do PluginBuilder foram criados automaticamente.

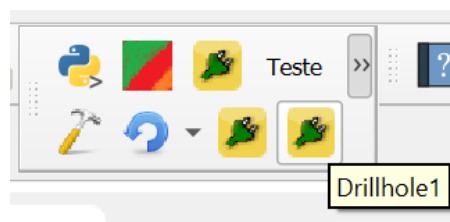
Vamos testar ele iniciando o QGIS e abrindo o **Complementos->Gerenciar e instalar Complementos**. Em Instalados vemos que ele não foi instalado ainda. Marque ele e instale para testarmos.



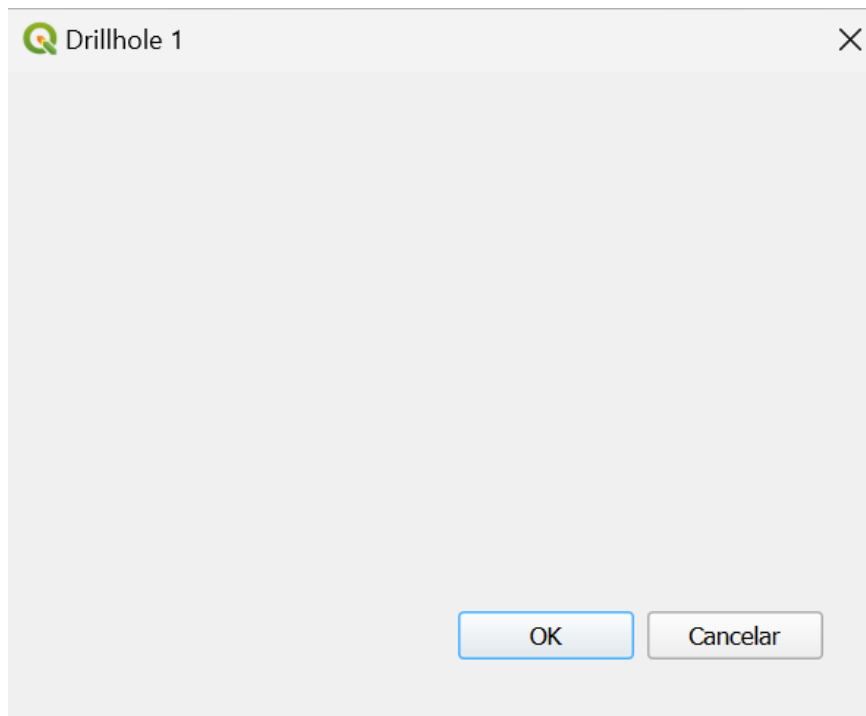
Inicie ele pelo Menu Vetor.



Ou pelo ícone na barra de ferramentas.



Funcionando, mas sem funcionalidade ainda.



Vamos construir a interface gráfica do usuário (GUI) e adicionar a funcionalidade agora na próxima seção.

3 - O plugin Drillhole1

O plugin Drillhole1 tem como objetivo abrir 4 arquivos de dados de furos de sondagem de exploração mineral típicos:

9olar.csv contendo informações de coordenadas da boca do furo em XYZ e profundidade atingida.

- Formato: HOLEID,X,Y,Z,ENDDEPTH

survey.csv contendo informações de direção e mergulho do furo e dos desvios entre boca do furo (collar) e ao longo do mesmo.

- Formato: HOLEID,AT,AZM,DIP

litho.csv contendo informações dos tipos geológicos, domínios e alteração e como estes variam dentro do furo.

- Formato: HOLEID,FROM,TO,DOMAIN,ROCKTYPE,WEATH

survey.csv contendo a análise de teor de um único elemento (au).

- Formato HOLEID,FROM,TO,AU

Os arquivos estão disponíveis em <https://gdatasystems.com/pyqgis/index.php>

Este plugin funciona somente com o formato descrito acima, em um próximo módulo faremos um plugin que nos permitirá formatar livremente os campos com base nos arquivos originais.

Agora execute o QtDesigner para criarmos a nossa interface gráfica de usuário (GUI).

Abra o arquivo **drillhole1_dialog_base.ui** localizado em:

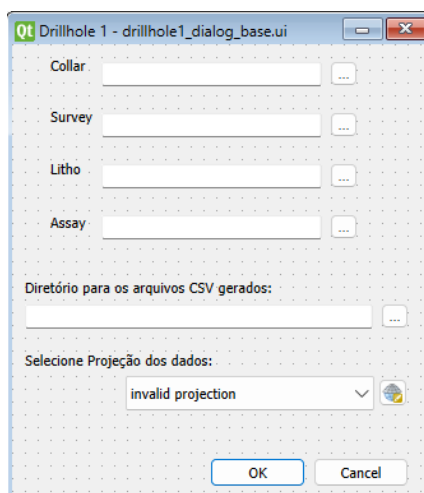
C:\Users\User\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\drillhole1

No primeiro diálogo do programa em **Open**.

Vamos adicionar 5 widgets do tipo Label, 5 widgets do tipo QgsFileWidget e 1 QgsProjectionSelectionWidget. Basta clicar no Widget e arrastar até a janela do diálogo.

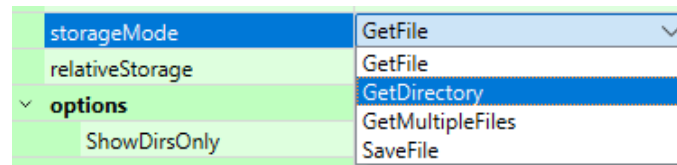


A aparência final da interface deve ser:



Modifique o texto dos campos Label.

No último QgsFileWidget altere o componente **StorageMode** para **GetDirectory**:



Agora altere a propriedade objectName dos campos QgsFileWidget na seguinte ordem.

mQgsFileWidgetCollar

mQgsFileWidgetSurvey

mQgsFileWidgetLitho

mQgsFileWidgetAssay

mQgsFileWidgetDir

Pronto, salve o diálogo e feche o QtDesigner.

Vamos agora editar o arquivo **drillhole1.py** para realizar a tarefa. Vamos ter de adicionar algumas bibliotecas de suporte via **import** e adicionar o código na função **run** que vai fazer a validação inicial dos campos e a criação da camada de pontos de amostragem de sedimento de corrente. Adicionaremos um arquivo **desurvey.py** com as funções separadamente.

As bibliotecas no arquivo **drillhole1.py** serão (adicionar as faltantes):

```
from qgis.PyQt.QtCore import Qsettings, Qtranslator, QCoreApplication, Qvariant
from qgis.PyQt.QtGui import Qicon, Qcolor
from qgis.PyQt.QtWidgets import Qaction, QmessageBox
from qgis.core import QgsProject, QgsVectorLayer, QgsFeature, QgsField, QgsGeometry
from qgis.core import QgsPoint, QgsVectorFileWriter, QgsMarkerSymbol, QgsStyle
# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .drillhole1_dialog import Drillhole1Dialog
import os.path
# Initialize Qt resources from file resources.py
from .desurvey import *
import pandas as pd
from pandasql import sqldf
```

Crie na mesma pasta do plugin o seguinte arquivo com as funções auxiliares **desurvey.py** :

```
import pandas as pd
import numpy as np
#-----
def tresD(co,es):
    linha = pd.DataFrame(columns = ['hole', 'prof', 'x','y','z','dip','az'])
    for I in range(0,len(co.index)):
        x=co.iat[I,2]
        y=co.iat[I,3]
        z=co.iat[I,4]
        di=0
        fur=es.loc[es['hole'] == co.iat[I,0]]
        fur=fur.sort_values('prof')
        ro=fur.shape[0]
        for j in range(0,ro):
            ang=fur.iat[j,6]
```

```

        d=fur.iat[j,1]-di
        di=d+di
        dip=fur.iat[j,5]
        if j>0:
            dip=fur.iat[j,5]-(fur.iat[j,5]-fur.iat[j-1,5])/2
        #--
        deltax=d*np.sin(np.radians(dip))
        r=d*np.cos(np.radians(dip))
        x=round(x+r*np.sin(np.radians(ang)))
        y=round(y+r*np.cos(np.radians(ang)))
        z=round(z+deltax)
        linha.loc[len(linha.index)]=[fur.iat[j,0] ,fur.iat[j,1],x
,y,z,fur.iat[j,5],fur.iat[j,6]]
        #--
        #---
    return linha.sort_values(['hole','prof'])

def calculo(azt,azb,dt,db,dp):
    dt = (90-dt)
    db = (90-db)
    dbt = db-dt
    abt = azb-azt
    d = np.arccos(np.cos(np.radians(dbt)) -
np.sin(np.radians(dt))*np.sin(np.radians(db))*(1-np.cos(np.radians(abt))))
    r = 1
    if d==0:
        r = 1
    else:
        r = 2*np.tan(d/2)/d
    x = 0.5*dp*(np.sin(np.radians(dt))*np.sin(np.radians(azt))+
np.sin(np.radians(db))*np.sin(np.radians(azb)))*r
    y = 0.5*dp*(np.sin(np.radians(dt))*np.cos(np.radians(azt))+
np.sin(np.radians(db))*np.cos(np.radians(azb)))*r
    z = 0.5*dp*(np.cos(np.radians(dt))+np.cos(np.radians(db)))*r
    return [x,y,z]

def calcXYZ(hole,depth,Spts):
    t = Spts.loc[(Spts['hole']==hole) & (Spts['prof'] <=depth)]
    t = t.sort_values('prof',ascending=False)
    b = Spts.loc[(Spts['hole']==hole) & (Spts['prof'] > depth)]
    if b.shape[0] == 0:
        res = calculo(t.iat[0,6], t.iat[0,6], t.iat[0,5], t.iat[0,5], depth-t.iat[0,1])
        return [(t.iat[0,2]+res[0]), (t.iat[0,3]+res[1]), (t.iat[0,4]+res[2])]
    di = b.iat[0,1]- t.iat[0,1]
    rga = t.iat[0,5] - b.iat[0,5]
    stp = rga/di
    dpt = depth-t.iat[0,1]
    pang = stp*dpt
    if b.shape[0] >= 1:
        res = calculo(t.iat[0,6], b.iat[0,6],t.iat[0,5], t.iat[0,5]-pang, dpt)
        return[(t.iat[0,2]+res[0]), (t.iat[0,3]+res[1]), (t.iat[0,4]+res[2])]
    else:
        res = calculo(t.iat[0,6], t.iat[0,6], t.iat[0,5], t.iat[0,5], dpt)
        return [(t.iat[0,2]+res[0]), (t.iat[0,3]+res[1]), (t.iat[0,4]+res[2])]

```

A função run ficará assim:

```

def run(self):
    if self.first_start == True:
        self.first_start = False
        self.dlg = Drillhole1Dialog()

    # show the dialog
    self.dlg.show()
    # Run the dialog event loop
    result = self.dlg.exec_()
    # See if OK was pressed

```

```

if result:
    coll=self.dlg.mQgsFileWidgetCollar.filePath()
    surl=self.dlg.mQgsFileWidgetSurvey.filePath()
    litl=self.dlg.mQgsFileWidgetLitho.filePath()
    asyl=self.dlg.mQgsFileWidgetAssay.filePath()
    dire=self.dlg.mQgsFileWidgetDir.filePath()
    if coll=="" or surl=="" or litl=="" or asyl=="" or dire=="":
        QMessageBox.warning(self.iface.mainWindow(),
            'Erro',
            "Entre todos os campos por favor\nSaindo...")
        return
    col = pd.read_csv (coll,sep=",")
    sur = pd.read_csv (surl,sep=",")
    lit = pd.read_csv (litl,sep=",")
    asy = pd.read_csv (asy1,sep=",")
    query = ''' SELECT col.holeid AS hole, col.x AS x, col.y AS y, col.z AS
z, asy.au AS au, asy."FROM" AS prof,lit."FROM" AS profl, asy."TO" AS toa, lit."TO"
AS tol, (asy."TO"-asy."FROM") AS lena, (lit."TO"-lit."FROM") AS lenl, lit.domain,
lit.rocktype, lit.weath FROM col INNER JOIN asy ON col.HOLEID = asy.HOLEID INNER
JOIN lit ON col.HOLEID = lit.HOLEID WHERE lit."FROM" BETWEEN asy."FROM" AND
(asy."TO"-0.02) ORDER BY hole,prof '''
    dat=sqldf(query, locals())
    query = ''' SELECT col.holeid AS hole,sur.at AS prof, col.x AS x, col.y
AS y, col.z AS z, sur.dip AS dip,sur.azm AS az FROM col INNER JOIN sur ON
col.holeid=sur.holeid WHERE sur.at=0 '''
    collar=sqldf(query, locals())
    collar['dip'] = collar['dip'].apply(lambda x: x*-1)
    query = ''' SELECT holeid AS hole, at AS prof, null AS x, null AS y,
null AS z, dip AS dip,azm FROM sur ORDER BY holeid,prof '''
    station=sqldf(query, locals())
    station['dip'] = station['dip'].apply(lambda x: x*-1)
    crs=self.dlg.mQgsProjectionSelectionWidget.crs()
    Spts=tresD(collar,station)
    for I in range(0,dat.shape[0]):
        dado = calcXYZ(dat.iat[i,0],dat.iat[i,5],Spts)
        dat.iat[i,1] = dado[0]
        dat.iat[i,2] = dado[1]
        dat.iat[I,3] = dado[2]

Spts.to_csv(dire+"/topobase.csv",sep=',',index=False)
temp = QgsVectorLayer("PointZ","desurveyed_intervals","memory")
temp.setCrs(crs)
temp_data = temp.dataProvider()
# Criando os campos necessários
temp_data.addAttribute([QgsField( "hole", Qvariant.String),
                        QgsField( "x", Qvariant.Double),
                        QgsField( "y", Qvariant.Double),
                        QgsField( "z", Qvariant.Double),
                        QgsField( "au", Qvariant.Double),
                        QgsField( "prof", Qvariant.Double),
                        QgsField( "profl", Qvariant.Double),
                        QgsField( "toa", Qvariant.Double),
                        QgsField( "tol", Qvariant.Double),
                        QgsField( "lena", Qvariant.Double),
                        QgsField( "lenl", Qvariant.Double),
                        QgsField( "DOMAIN", Qvariant.String),
                        QgsField( "ROCKTYPE", Qvariant.String),
                        QgsField( "WEATH", Qvariant.String)])

# Atualizando os campos
temp.updateFields()
temp.startEditing()
# Adicionando cada ponto resultante do desurvey na camada ponto 3d
for row in dat.itertuples():
    f = QgsFeature()
    f.setGeometry(QgsGeometry(QgsPoint(row.x,row.y,row.z)))

```

```

f.setAttributes([row.hole,row.x,row.y,row.z,row.au,row.prof,row.profl,row.toa,row.t
ol,row.lena,row.lenl,row.DOMAIN,row.ROCKTYPE,row.WEATH])
temp_data.addFeature(f)

# Aqui gravamos as mudanas das das camadas, adicionamos a camada e
exportamos como arquivo csv
temp.updateExtents()
temp.commitChanges()
QgsProject.instance().addMapLayer(temp)

v_layer = QgsVectorLayer('LineString?crs='+crs.authid(),
'drillholeline', 'memory')
pr = v_layer.dataProvider()
pr.addAttributes([QgsField("hole", QVariant.String)])
v_layer.updateFields()

topo_layer = QgsVectorLayer('PointZ?crs='+crs.authid(), 'collar',
'memory')
prt = topo_layer.dataProvider()
prt.addAttributes([QgsField("hole", QVariant.String)])
topo_layer.updateFields()

base_layer = QgsVectorLayer('PointZ?crs='+crs.authid(),
'base_of_survey', 'memory')
prb = base_layer.dataProvider()
prb.addAttributes([QgsField("hole", QVariant.String)])
base_layer.updateFields()

cabra=""
for index, row in Spts.iterrows():
    if cabra == "_":
        start_point = QgsPoint(row['x'],row['y'],row['z'])
        cabra=row.hole
        f = QgsFeature()
        f.setGeometry(QgsGeometry(start_point))
        f.setAttributes([row.hole])
        prt.addFeature(f)

    else:
        if cabra!=row.hole and index>0:
            end_point = QgsPoint(Spts.loc[(int(index) - 1),
'x'],Spts.loc[(int(index) - 1), 'y'],Spts.loc[(int(index) - 1), 'z'])
            f = QgsFeature()
            f.setGeometry(QgsGeometry(end_point))
            f.setAttributes([row.hole])
            prb.addFeature(f)
            seg = QgsFeature()
            seg.setGeometry(QgsGeometry.fromPolyline([start_point,
end_point]))

            seg.setAttributes([row.hole])
            pr.addFeatures([seg])
            start_point = QgsPoint(row['x'],row['y'],row['z'])
            cabra=row.hole
            f = QgsFeature()
            f.setGeometry(QgsGeometry(start_point))
            f.setAttributes([row.hole])
            prt.addFeature(f)

QgsProject.instance().addMapLayers([v_layer])
QgsProject.instance().addMapLayer(topo_layer)
QgsProject.instance().addMapLayer(base_layer)

renderer = temp.renderer()
symbol1 = QgsMarkerSymbol.createSimple({'name':'dot red','color':
'red','size':0.8})
symbol_layer1 = symbol1.symbolLayer(0)
renderer.setSymbol(symbol1)

```

```

temp.triggerRepaint()

renderer = topo_layer.renderer()
style = QgsStyle.defaultStyle().symbol('topo pop capital')
renderer.setSymbol(style)
renderer.symbol().setSize(2)
topo_layer.triggerRepaint()

renderer = base_layer.renderer()
style = QgsStyle.defaultStyle().symbol('topo pop village')
renderer.setSymbol(style)
renderer.symbol().setSize(1.4)
renderer.symbol().setColor(Qcolor("blue"))
base_layer.triggerRepaint()

self.iface.layerTreeView().refreshLayerSymbology(temp.id())
self.iface.layerTreeView().refreshLayerSymbology(topo_layer.id())
self.iface.layerTreeView().refreshLayerSymbology(base_layer.id())
#gravando o csv do desurvey e finalizando
QgsVectorFileWriter.writeAsVectorFormat(temp,dire+"/desurveyed.csv",
"utf-8",driverName = "CSV" , layerOptions = ['GEOMETRY=AS_XYZ'])
QMessageBox.information(self.iface.mainWindow(),' Pronto ',' Desurvey
executado!')
return

```

Baixe os arquivos CSV em <https://gdatasystems.com/pyqgis/index.php>

Abra o QGIS e o plugin será carregado já com as alterações feitas. Ao iniciarmos o plugin teremos:

Drillhole 1 [X]

Collar ...

Survey ...

Litho ...

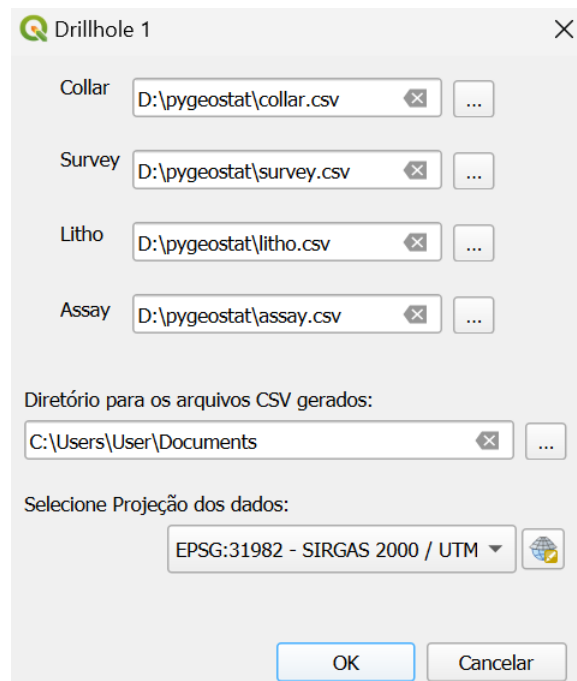
Assay ...

Diretório para os arquivos CSV gerados:
 ...

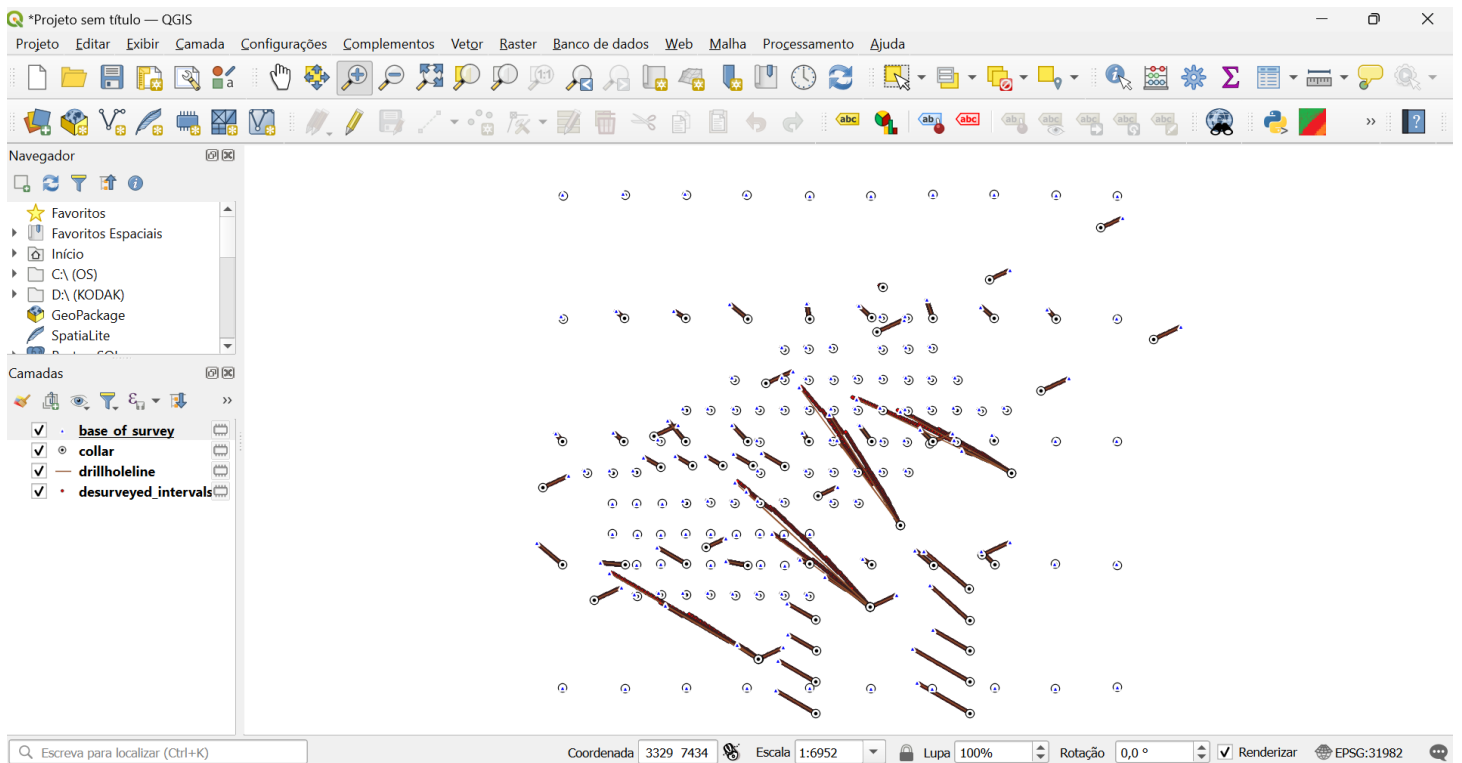
Selecione Projecção dos dados:
 projecção inválida [Globe Icon]

OK Cancelar

Entre o caminho para os arquivos collar, survey, litho e assay. Defina o diretório onde os arquivos csv de resultado serão e escolha uma projeção UTM qualquer (os dados são em metros, mas não são referenciados a Datum nenhum).



Clique ok e aguarde uns 50 segundos para o processamento dos dados e geração dos arquivos e camadas temporárias resultantes. Ao concluir termos no QGIS:



A camada **desurveyed_intervals** representa o resultado do desurvey com todos os intervalos reprojados em X, Y e Z e seus atributos, ver tabela de atributos abaixo:

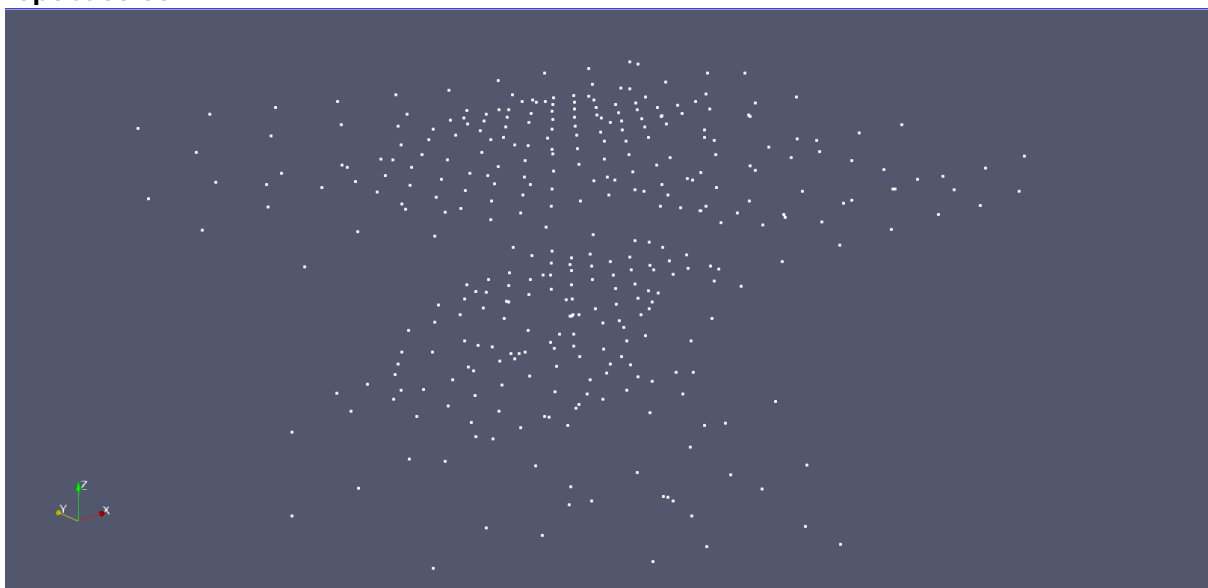
	hole	x	y	z	au	prof	profl	toa	tol	lena	lenl	DOMAIN	ROCKTYPE	WEATH
1	M001	3999	7499	727	0	0	0	2,5	2,5	2,5	2,5	D2	AvT	OX
2	M001	3998,99716594...	7499,00163624...	724,500002855...	0,71	2,5	2,5	5	5	2,5	2,5	D2	AvT	OX
3	M001	3998,98866377...	7499,00654496...	722,000022846...	0,96	5	5	7,5	7,5	2,5	2,5	D2	AvT	OX
4	M001	3998,97449357...	7499,01472613...	719,500077106...	0,48	7,5	7,5	10	10	2,5	2,5	D2	AvT	OX
5	M001	3998,95465543...	7499,02617969...	717,000182769...	1,42	10	10	12,5	12,5	2,5	2,5	D2	AvT	OX
6	M001	3998,92914947...	7499,04090557...	714,500356970...	2,16	12,5	12,5	15	15	2,5	2,5	D2	AvT	OX
7	M001	3998,89797588...	7499,05890365...	712,000616842...	1,34	15	15	17,5	17,5	2,5	2,5	D2	AvT	OX
8	M001	3998,86113487...	7499,08017381...	709,500979518...	0,39	17,5	17,5	20	20	2,5	2,5	D2	AvT	OX
9	M001	3998,81862669...	7499,10471592...	707,001462131...	0,43	20	20	22,5	22,5	2,5	2,5	D2	AvT	OX
10	M001	3998,77045163...	7499,13252980...	704,502081811...	0,87	22,5	22,5	25	25	2,5	2,5	D2	AvT	OX
11	M001	3998,71661003...	7499,16361527...	702,002855690...	0,23	25	25	27,5	27,5	2,5	2,5	D2	AvT	OX
12	M001	3998,65710225...	7499,19797210...	699,503800896...	0,34	27,5	27,5	30	30	2,5	2,5	D2	AvT	OX
13	M001	3998,59192870...	7499,23560007...	697,004934558...	0,94	30	30	32,5	32,5	2,5	2,5	D2	AvT	OX
14	M001	3998,52108983...	7499,27649891...	694,506273803...	1,42	32,5	32,5	35	35	2,5	2,5	D2	AvT	OX
15	M001	3998,44458612...	7499,32066834...	692,007835756...	0,88	35	35	37,5	37,5	2,5	2,5	D2	AvT	OX
16	M001	3998,36241810...	7499,36810807...	689,509637542...	0,26	37,5	37,5	40	40	2,5	2,5	D2	AvT	OX
17	M001	3998,27458633...	7499,41881777...	687,011696282...	0,5	40	40	42,5	42,5	2,5	2,5	D2	AvT	MX
18	M001	3998,18109141...	7499,47279709...	684,514029098...	0	42,5	42,5	45	45	2,5	2,5	D2	AvT	MX

A camada **collar** é posição da boca do furo e a camada **base_of_survey** a base levantada do furo.

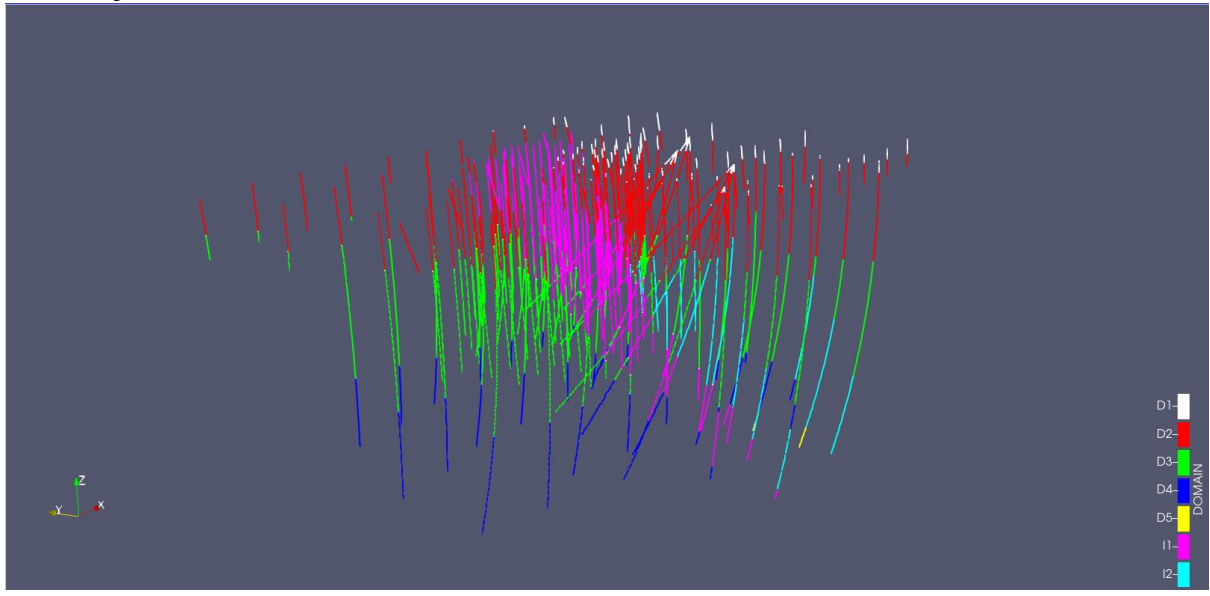
A camada **drillholeline** é a projeção em planta do furo.

Além dessas camadas foram gerados dois arquivos csv. O arquivo **desurveyed.csv** com o resultado do desurvey e o arquivo **topobase.csv** com os pontos de medidas do survey no furo (estação). Abrindo estes arquivos com o Paraview usando o filtro TableToPoint teremos:

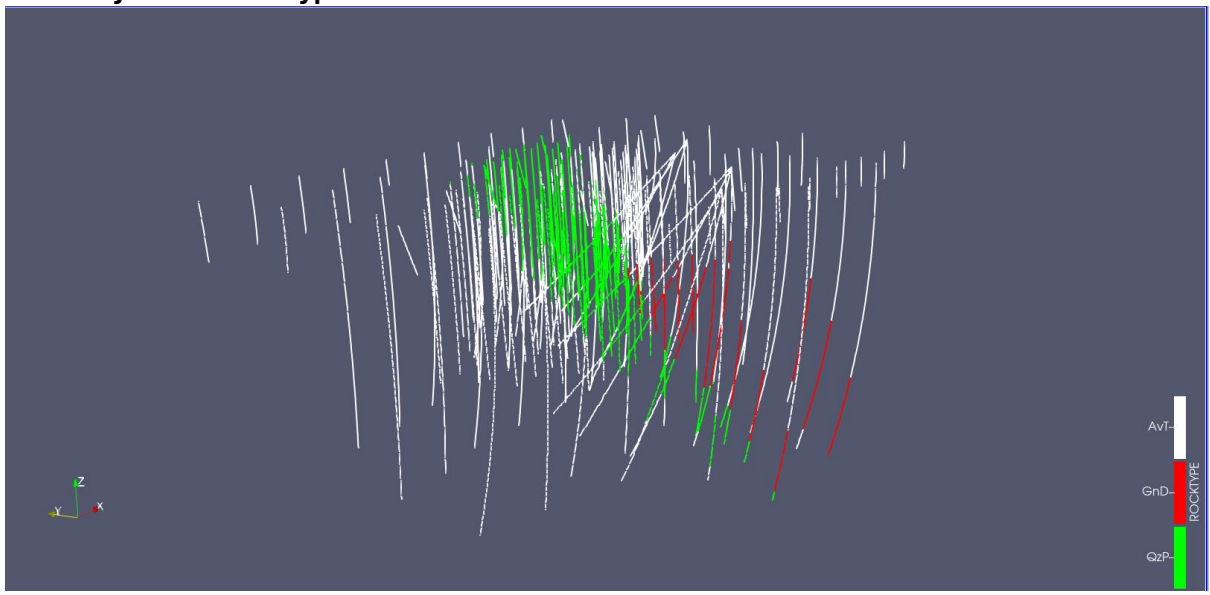
topobase.csv



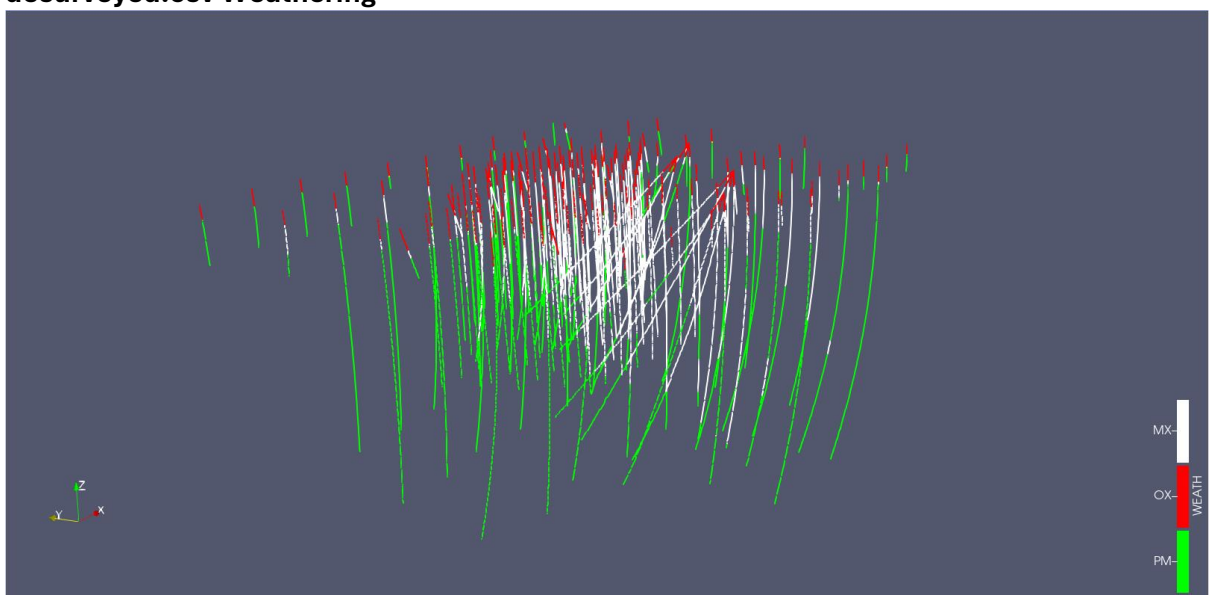
desurveyed.csv Domains



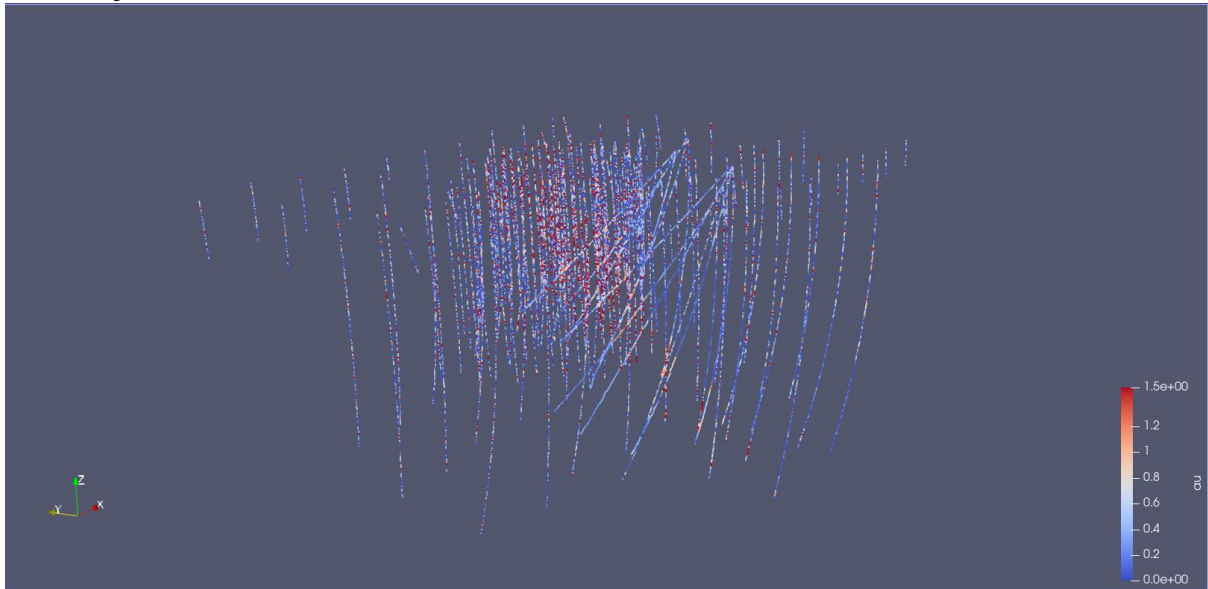
desurveyed.csv Rocktype



desurveyed.csv Weathering



desurveyed.csv Au



Revisitaremos este plugin num módulo mais adiante implementando mais flexibilidade dos dados originais.

No próximo módulo vamos criar um plugin que executa processamento em imagens do Sentinel2.

Até lá!