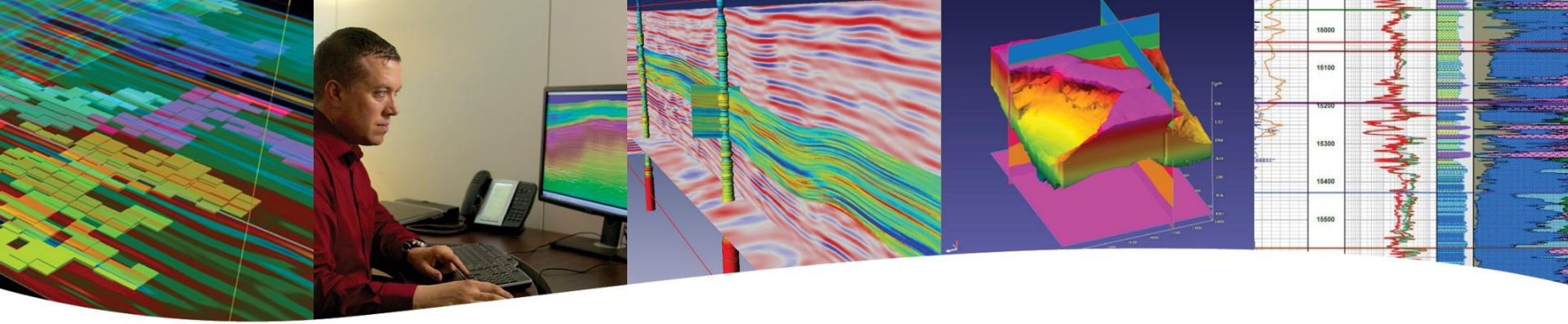


# Machine Learning for Better Wells

Daria Lazareva

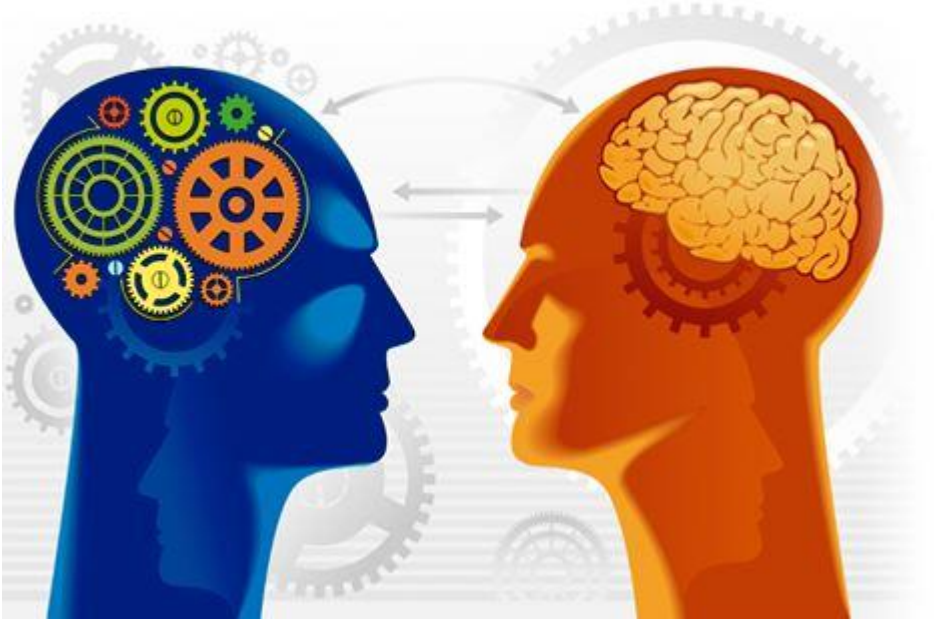


*Paris, 8th October 2019*

CGG GeoSoftware

# Machine Learning for Better Wells

- Facies Classification using Machine Learning
- Sub-clustering in Facies Determination
- Automated Log Editing using Deep Learning



# How does it work?

Well Data

- Views
- Processors
- Console
- Jupyter workflows

## Python Environment

100+ packages including numpy, scipy, matplotlib, keras, theano, plconnect, scikit-learn, pandas, plotly and others

Including IDE jupyter, spyder

Multiple extensions contained within a workspace serving different needs

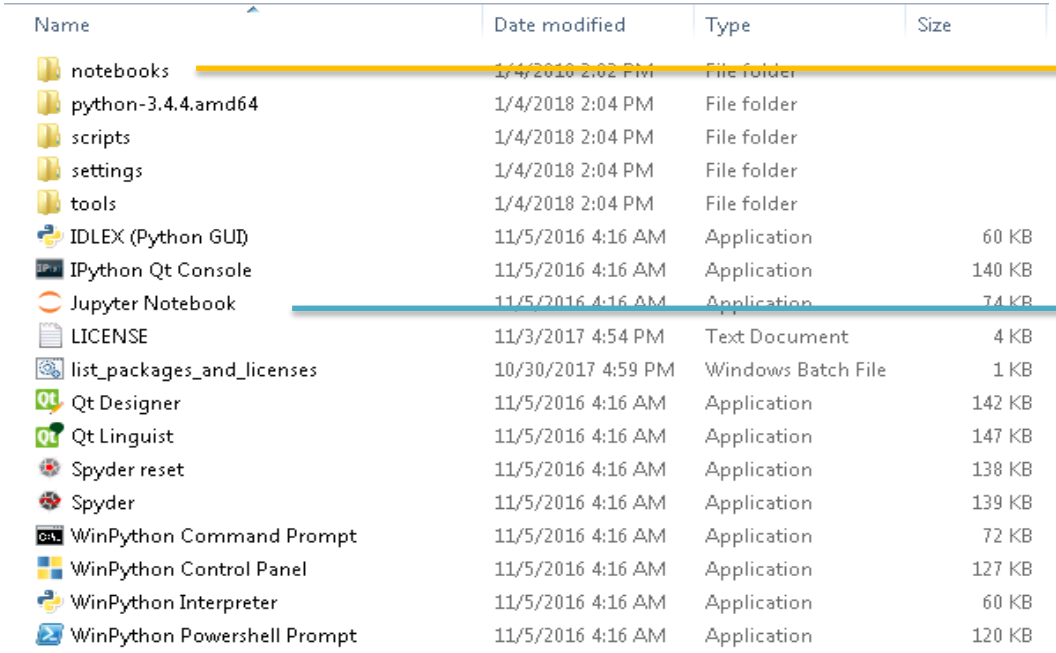
Each extension is a folder in a workspace

Flexible and portable

Can also be packaged as jupyter workflows

All scripts run as external processes and are not restricted by python GIL

# What is inside the distribution? – Jupyter



| Name | Date modified | Type | Size |
|---|---|---|---|
| notebooks | 1/4/2018 2:02 PM | File folder | |
| python-3.4.4.amd64 | 1/4/2018 2:04 PM | File folder | |
| scripts | 1/4/2018 2:04 PM | File folder | |
| settings | 1/4/2018 2:04 PM | File folder | |
| tools | 1/4/2018 2:04 PM | File folder | |
| IDLEX (Python GUI) | 11/5/2016 4:16 AM | Application | 60 KB |
| IPython Qt Console | 11/5/2016 4:16 AM | Application | 140 KB |
| Jupyter Notebook | 11/5/2016 4:16 AM | Application | 74 KB |
| LICENSE | 11/3/2017 4:54 PM | Text Document | 4 KB |
| list_packages_and_licenses | 10/30/2017 4:59 PM | Windows Batch File | 1 KB |
| Qt Designer | 11/5/2016 4:16 AM | Application | 142 KB |
| Qt Linguist | 11/5/2016 4:16 AM | Application | 147 KB |
| Spyder reset | 11/5/2016 4:16 AM | Application | 138 KB |
| Spyder | 11/5/2016 4:16 AM | Application | 139 KB |
| WinPython Command Prompt | 11/5/2016 4:16 AM | Application | 72 KB |
| WinPython Control Panel | 11/5/2016 4:16 AM | Application | 127 KB |
| WinPython Interpreter | 11/5/2016 4:16 AM | Application | 60 KB |
| WinPython Powershell Prompt | 11/5/2016 4:16 AM | Application | 120 KB |

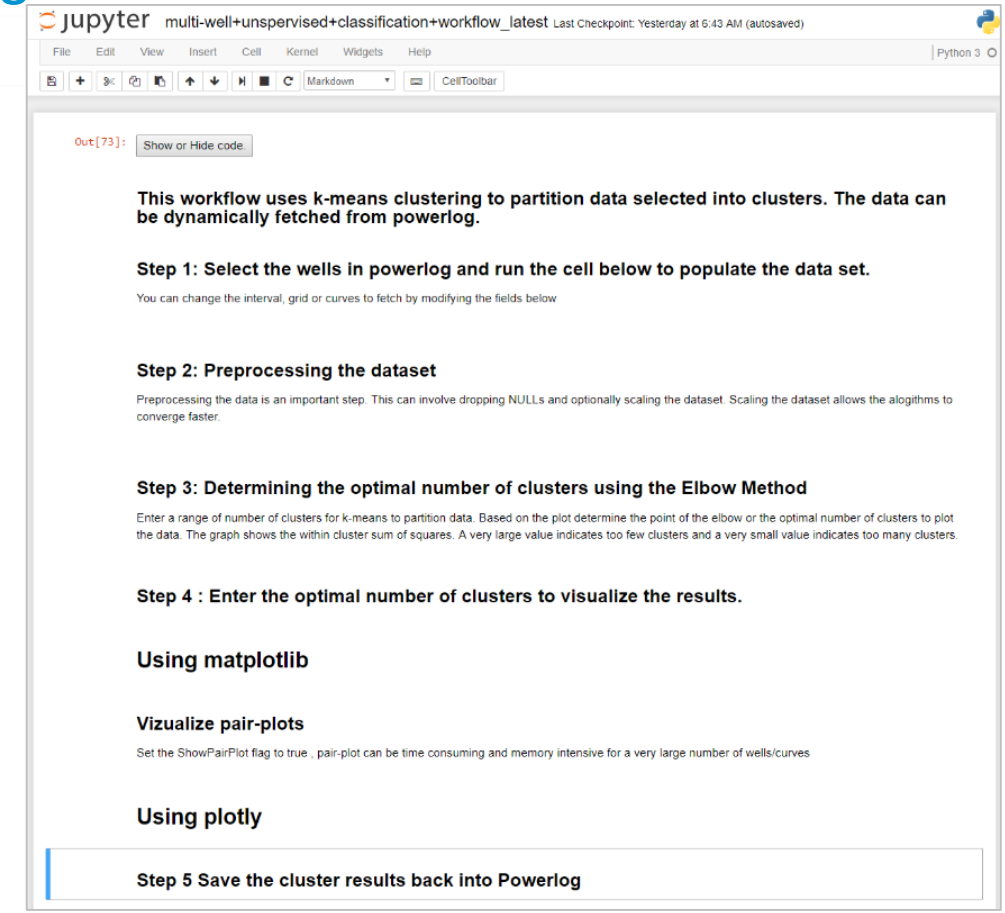
Workflows home directory

Workflows with code and rich text + Visualization

# Facies Classification
# Unsupervised Machine Learning

Passion for Geoscience

# Clustering Workflow



# Select the Wells to be Analyzed

**Step 1: Select the wells in powerlog and run the cell below to populate the data set.**

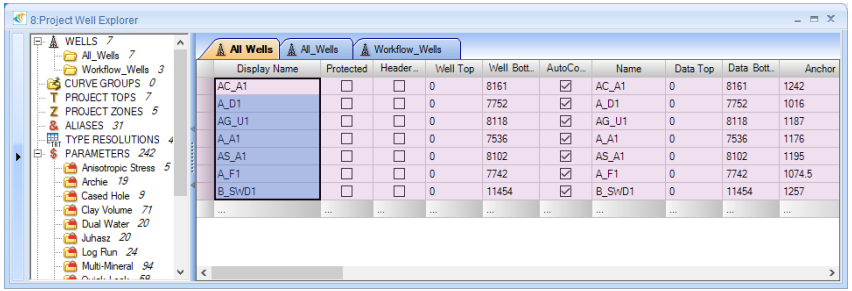
You can change the interval, grid or curves to fetch by modifying the fields below

In [74]:

```
zone = "WorkingInterval"
grid = "Default"
curves_to_fetch = ["&NPHI", "&RHOB", "&GR", "&DT", "&DTS", "&PE"]

import pandas as pd
import plconnect.functions as plf
import sys
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np

current_uwi_list = plf.get_current_selected_wells()
if(len(current_uwi_list) == 0):
    sys.stderr.write("No wells selected in powerlog. Please select a well in the project explorer")


well_data_sets = [] #List of dataframes that contains curve info from the selected wells

for uwi in current_uwi_list:
    df =  plf.get_curveexpressions_for_uwi(uwi, expressions = curves_to_fetch, interval = zone, grid = grid)
    df.insert(0,"uwi",uwi)
    well_data_sets.append(df)


raw_dataset = pd.concat(well_data_sets)
current_uwi = current_uwi_list[0]
```

In [75]:

```
raw_dataset.describe()
```




In [77]:

```
raw_dataset.describe()
```

Out[77]:

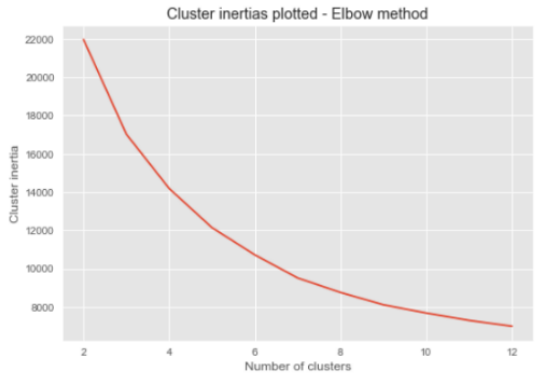
| | DEPTH | &NPHI | &RHOB | &GR | &DT | &DTS | &PE |
|---|---|---|---|---|---|---|---|
| **count** | 21438.000000 | 13694.000000 | 13936.000000 | 21340.000000 | 13881.000000 | 13900.000000 | 13980.000000 |
| **mean** | 8136.784938 | 0.102352 | 2.627881 | 56.886438 | 67.041839 | 117.642525 | 3.714599 |
| **std** | 1266.151535 | 0.068209 | 0.102981 | 53.770362 | 14.051845 | 17.128154 | 13.598955 |
| **min** | 6632.500000 | -0.020400 | 1.990000 | 2.557100 | 44.570000 | 82.102000 | -936.495000 |
| **25%** | 7231.000000 | 0.034000 | 2.533838 | 14.152600 | 52.797000 | 101.927000 | 3.228750 |
| **50%** | 7632.000000 | 0.114605 | 2.653900 | 31.755850 | 66.608000 | 119.077500 | 3.929000 |
| **75%** | 8774.375000 | 0.159000 | 2.715400 | 97.823250 | 80.654000 | 130.023000 | 4.641000 |
| **max** | 11454.000000 | 0.294900 | 2.889394 | 373.859100 | 104.959000 | 191.100000 | 5.559000 |

# Determining Optimum Number of Clusters

### Step 3: Determining the optimal number of clusters using the Elbow Method

Enter a range of number of clusters for k-means to partition data. Based on the plot determine the point of the elbow or the optimal number of clusters to plot the data. The graph shows the within cluster sum of squares. A very large value indicates too few clusters and a very small value indicates too many clusters.

In [80]:
```
MinCluster = 2
MaxCluster = 13

init = "k-means++" # to use random initialization change it to random


interias = []
for n in range(MinCluster, MaxCluster):
    kmeans = KMeans(n_clusters = n, init= init, random_state = 0)
    kmeans.fit(processed_data.values)
    interias.append(kmeans.inertia_)

plt.style.use('ggplot')
plt.plot(range(MinCluster,MaxCluster), interias)
plt.title("Cluster inertias plotted - Elbow method")
plt.xlabel("Number of clusters")
plt.ylabel("Cluster inertia")
plt.show()
```

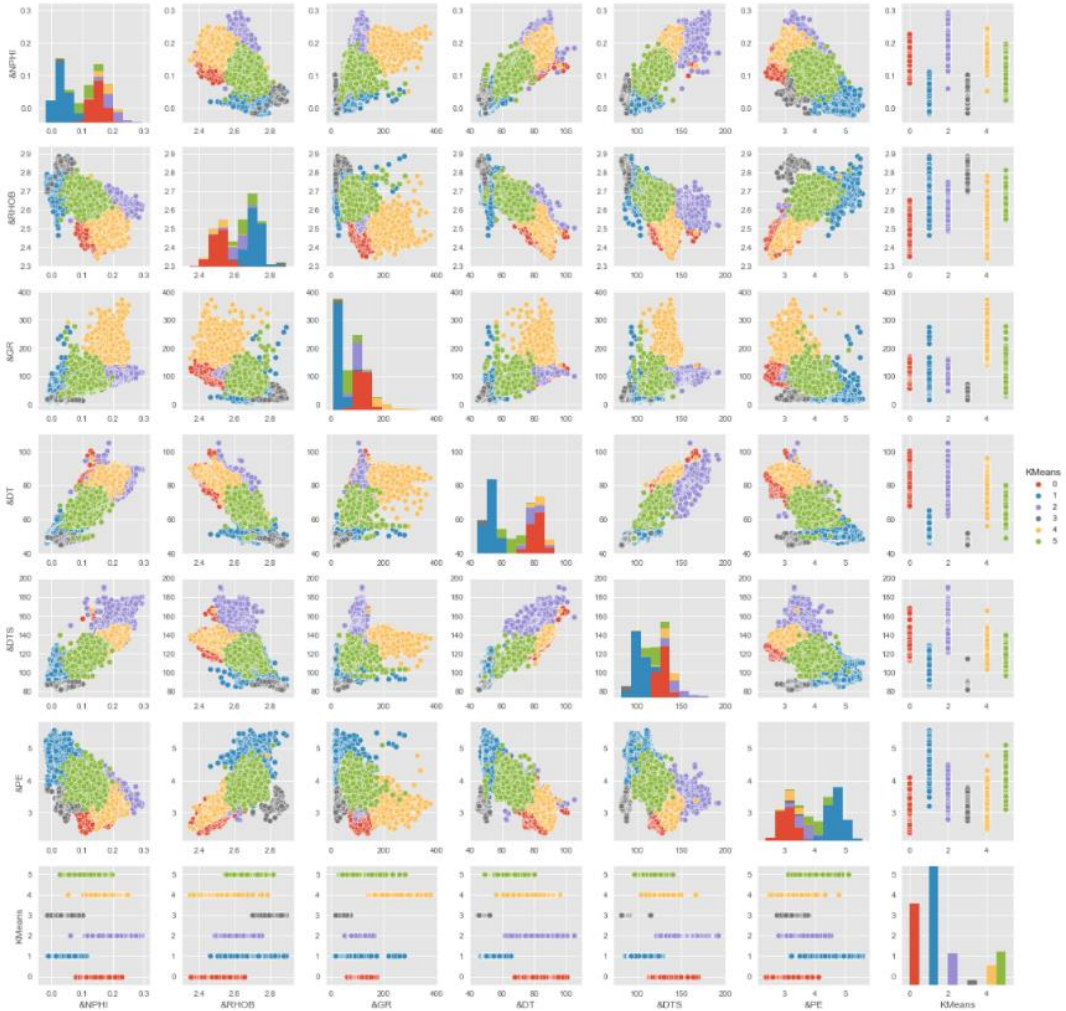



### Step 4 : Enter the optimal number of clusters to visualize the results.

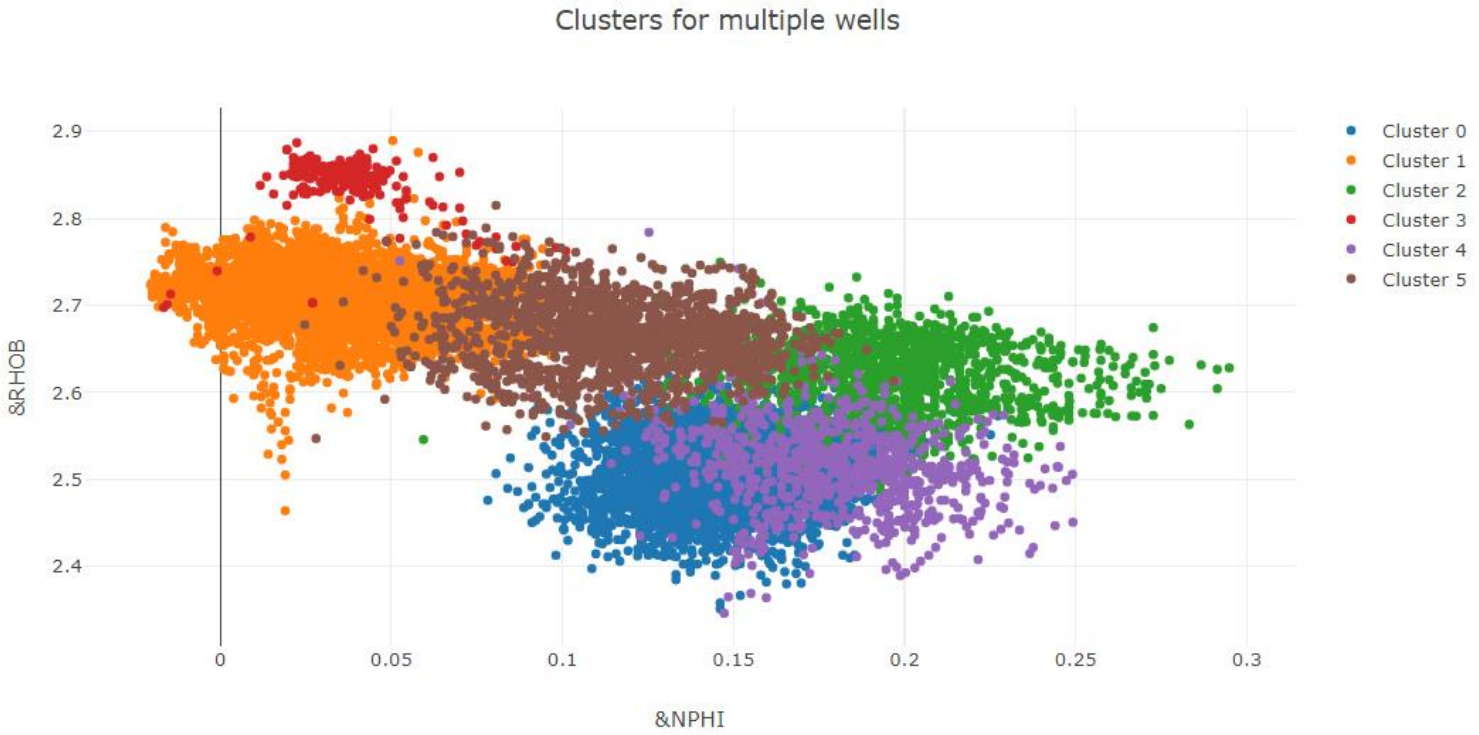
In [81]:
```
NumClusters = 6


cluster_labels = []

kmeans = KMeans(n_clusters = NumClusters, init = init, random_state = 0)
c_values = kmeans.fit_predict(processed_data.values)
title = "Clusters for multiple wells "
print(curves_to_fetch)
#enter the 0 based index of the curve_to_fetch curve for x and y axis. defaulted to 0 and 1
x_axis = 0
y_axis = 1
size = 20

xlabel = curves_to_fetch[x_axis]
ylabel = curves_to_fetch[y_axis]
```

```
['&NPHI', '&RHOB', '&GR', '&DT', '&DTS', '&PE']
```

# Pair-Plots


ΔNPHI
ΔRHOB
ΔGR
ΔDT
ΔDTS
ΔPE
KMeans
ΔNPHI
ΔRHOB
ΔGR
ΔDT
ΔDTS
ΔPE
KMeans

# Crossplot

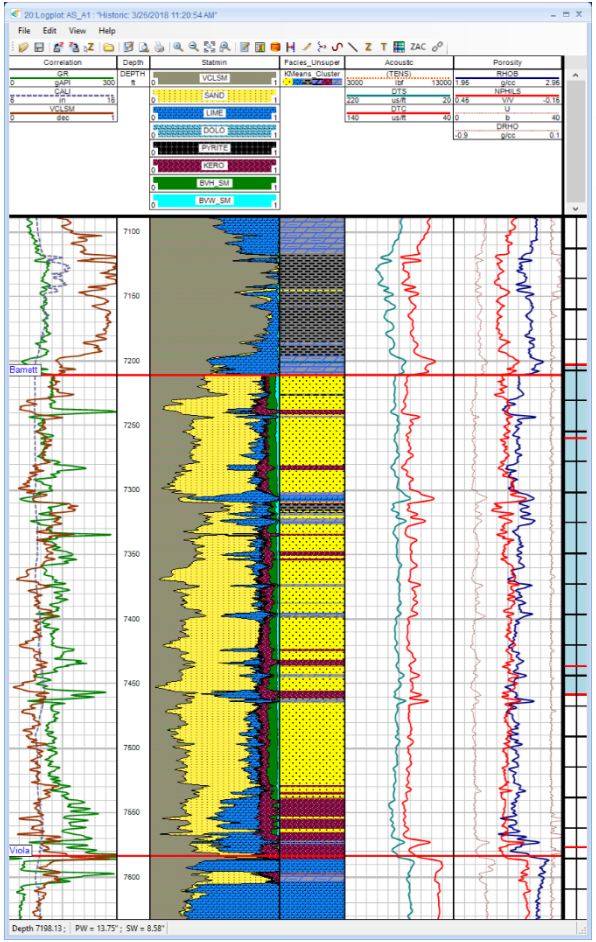

Clusters for multiple wells
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
&RHOB
2.9
2.8
2.7
2.6
2.5
2.4
0
0.05
0.1
0.15
0.2
0.25
0.3
&NPHI

# Write to Database and Display

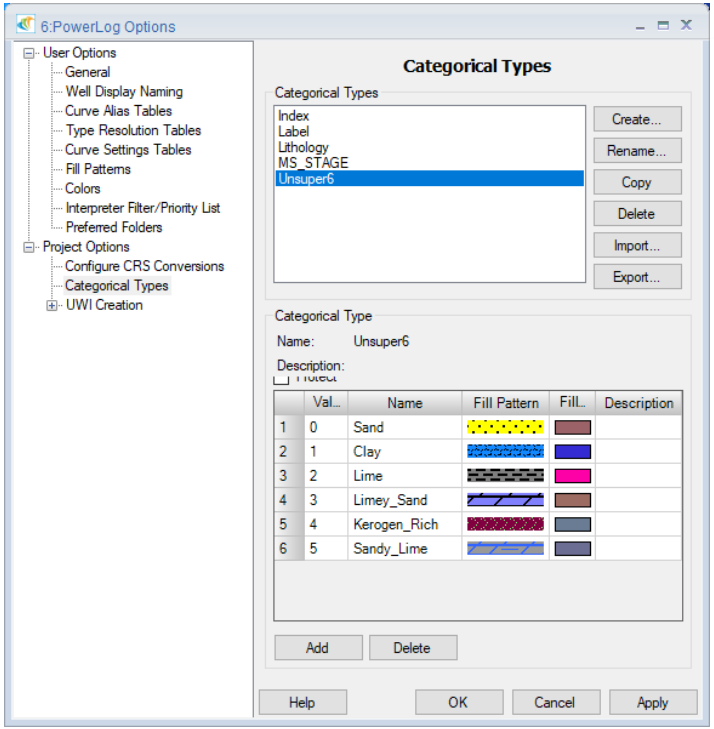
**Step 5 Save the cluster results back into Powerlog**

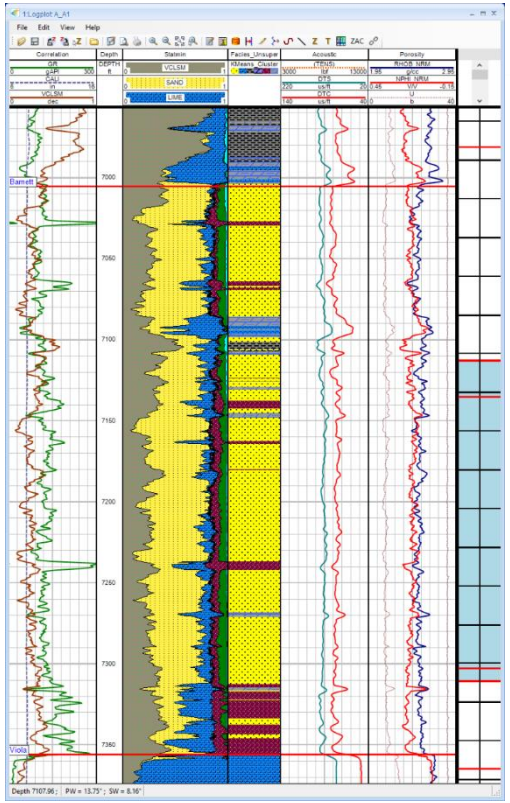
```
SaveData = True
OutCurveName = "KMeans_Cluster"
Verbose = True

from plconnect.exceptions import ExtensionException

if (SaveData):

    #Insert the calculated clusters into the raw_data_set

    raw_dataset[OutCurveName] = c_values

    for uwi in current_uwi_list:
        save_data = raw_dataset[raw_dataset["uwi"] == uwi].drop(curves_to_fetch, axis =1)
        save_data.drop(["uwi"], axis = 1, inplace = True)
        try:
            plf.set_curves_for_uwi(uwi, save_data)

        except ExtensionException as e:
            sys.stderr.write("Error: Could not save clusters for uwi " + uwi)
            sys.stderr.write("\n" + e.args[0])


        if(Verbose):
           print("Saved for uwi " + uwi)
```

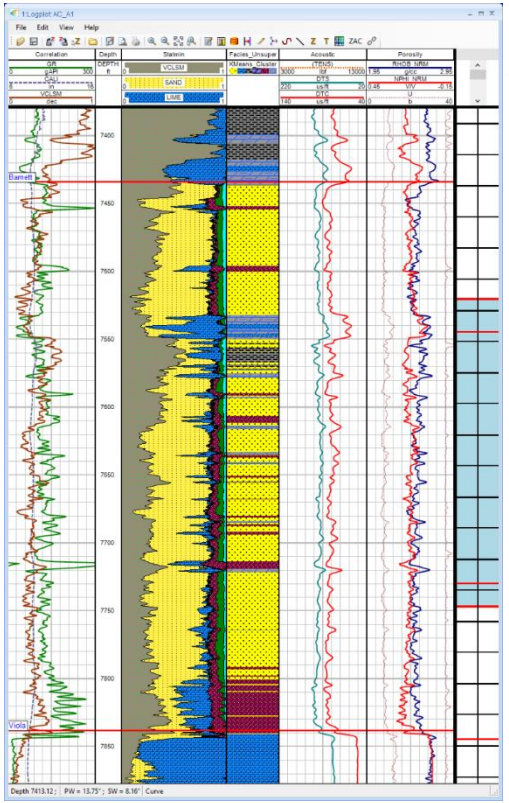
```
Saved for uwi 42121322330000
Saved for uwi 42121322470000
Saved for uwi 42121324910000
Saved for uwi 42439306390000
Saved for uwi 42439309400000
Saved for uwi 42439309560000
Saved for uwi 42439341280000
```



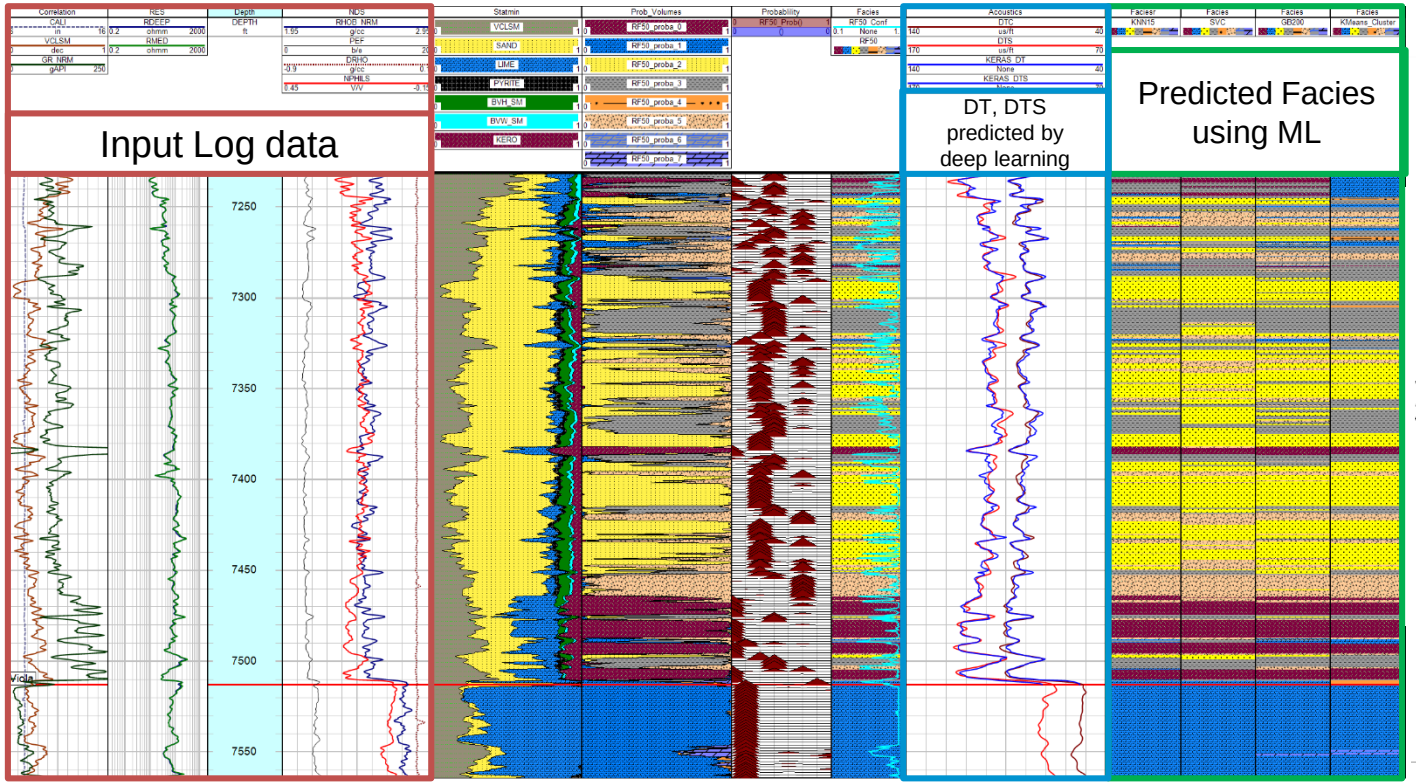
# Build (or Edit) a Category Table

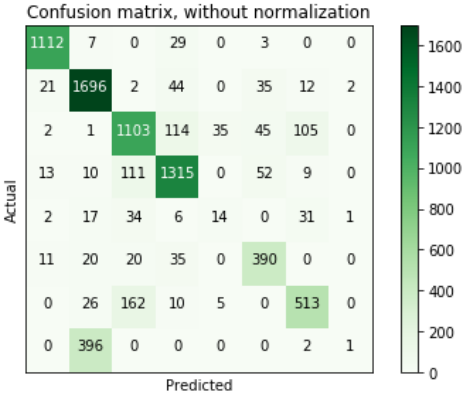




# Supervised Classification with Metrics




| | Accuracy | F1 | Precision | Recall |
|---|---|---|---|---|
| **Accuracy** | 0.811196 | 0.786801 | 0.779591 | 0.811196 |

The K-Means unsupervised clusters provides the supervision

# Export Jupyter Results to .pdf or .html File

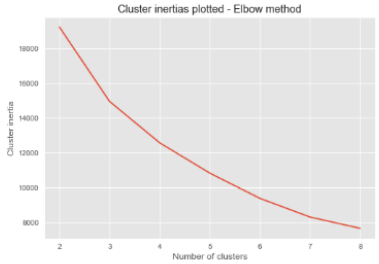
jupyter multi-well+unspervised+classification+workflow_latest Last Checkpoint: Yesterday at 7:19 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Python 3

Code CellToolbar

### Step 3: Determining the optimal number of clusters using the Elbow Method

Enter a range of number of clusters for k-means to partition data. Based on the plot determine the point of the elbow or the optimal number of clusters to plot the data. The graph shows the within cluster sum of squares. A very large value indicates too few clusters and a very small value indicates too many clusters.



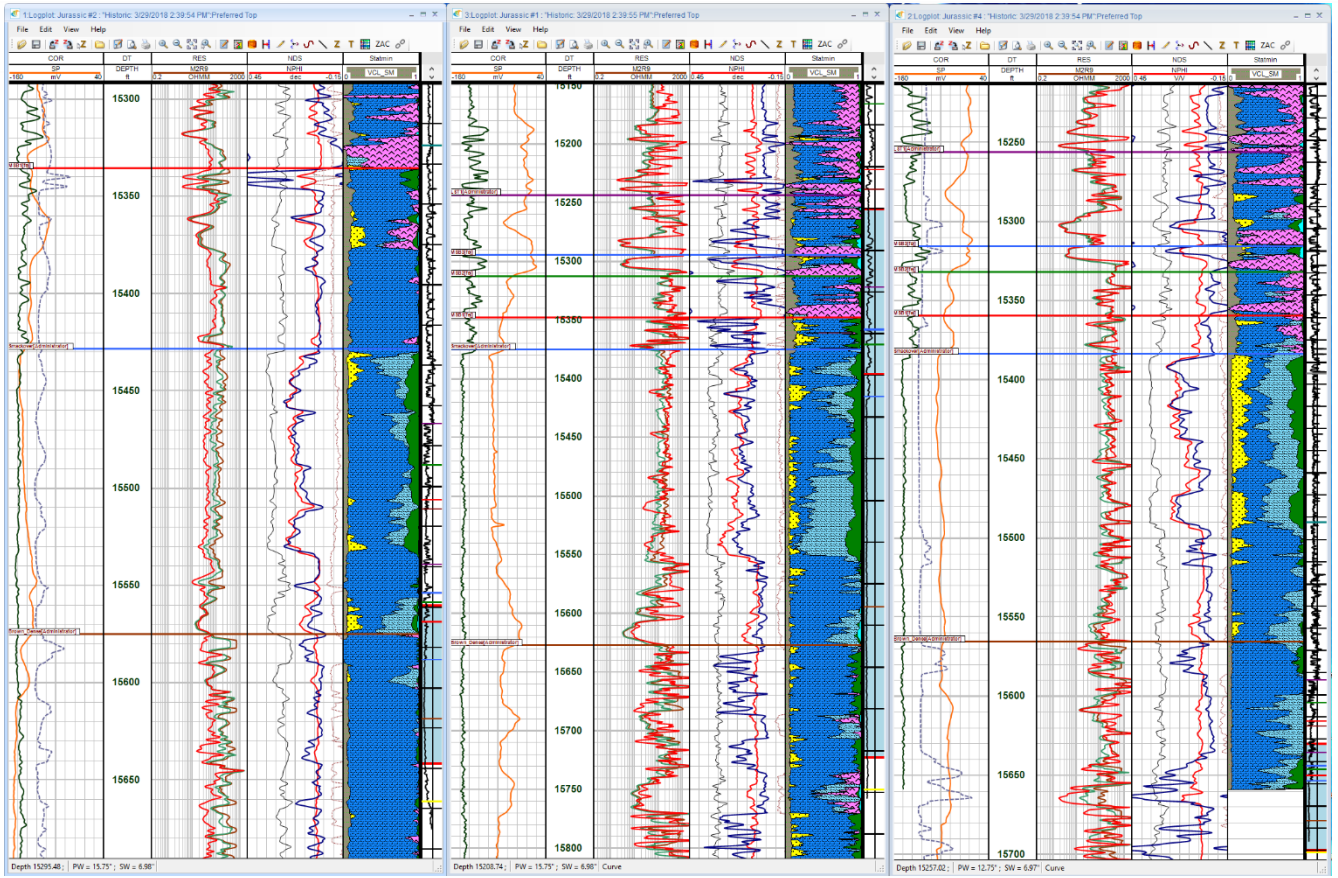

### Step 4 : Enter the optimal number of clusters to visualize the results.

```
['&NPHI', '&RHOB', '&GR', '&DT', '&DTS', '&PE']
```

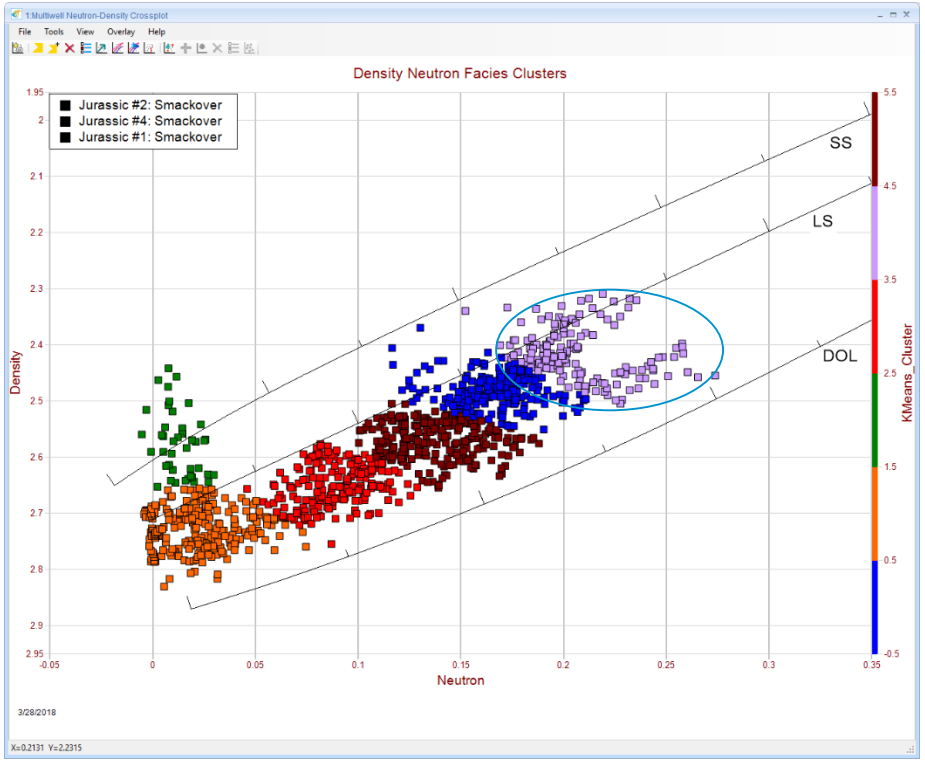
### Using matplotlib

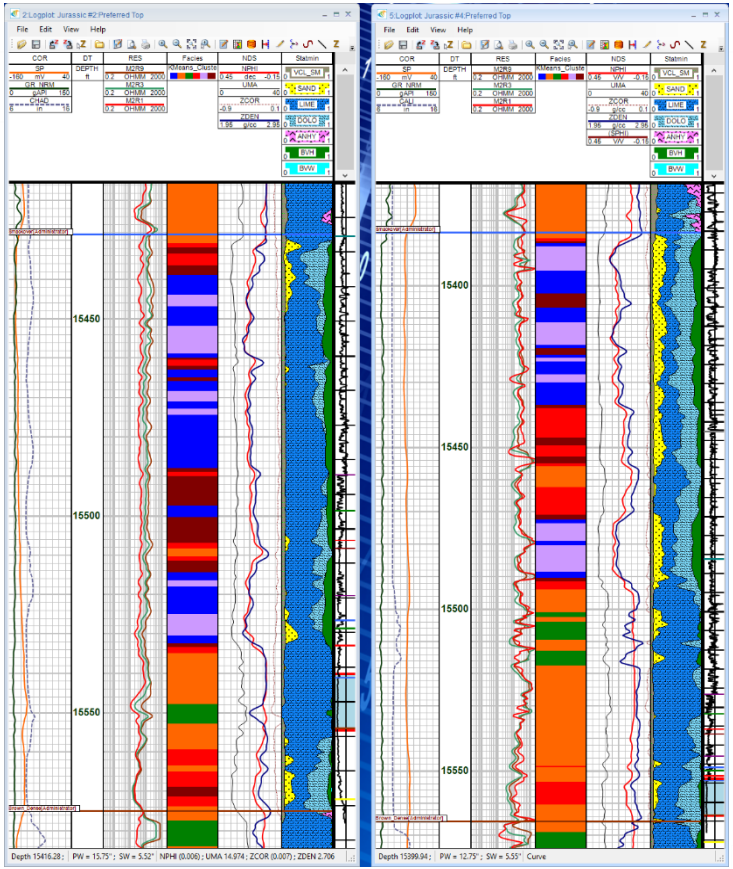
# Sub-Clustering Facies Determination

# Complex Carbonate Example

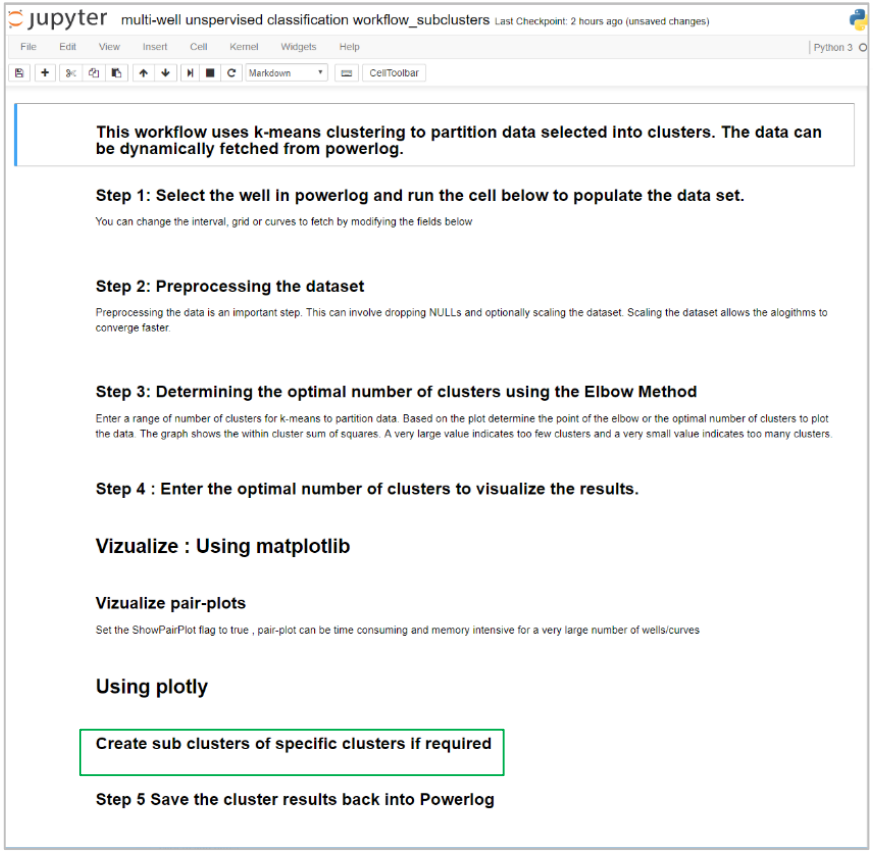


# Clustering Problem



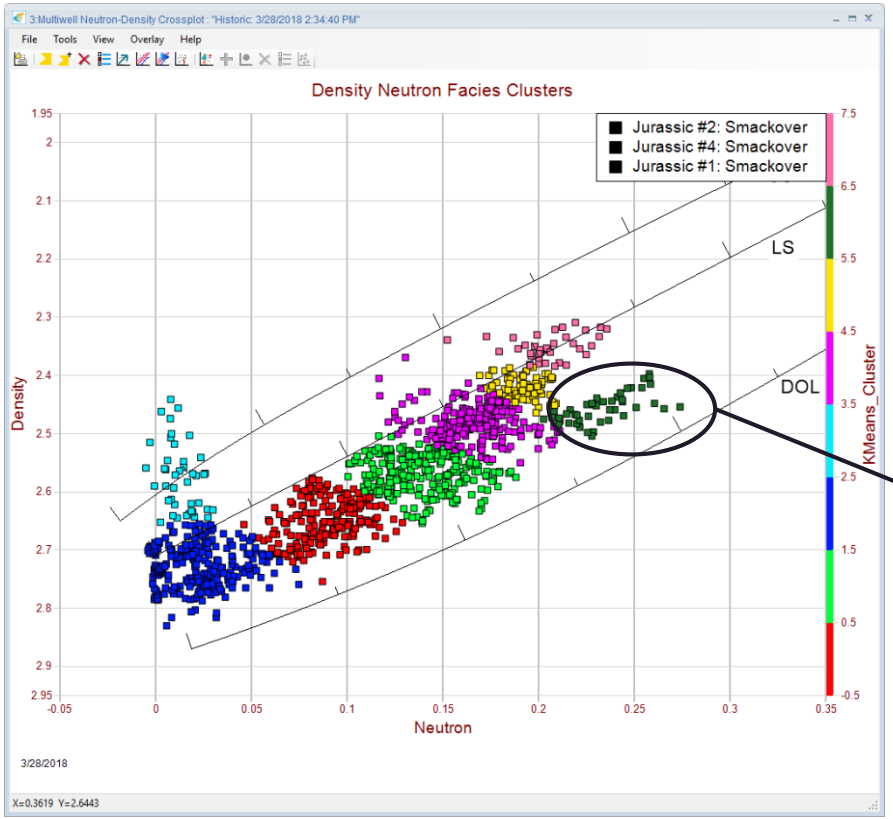
# Sub-Cluster Workflow

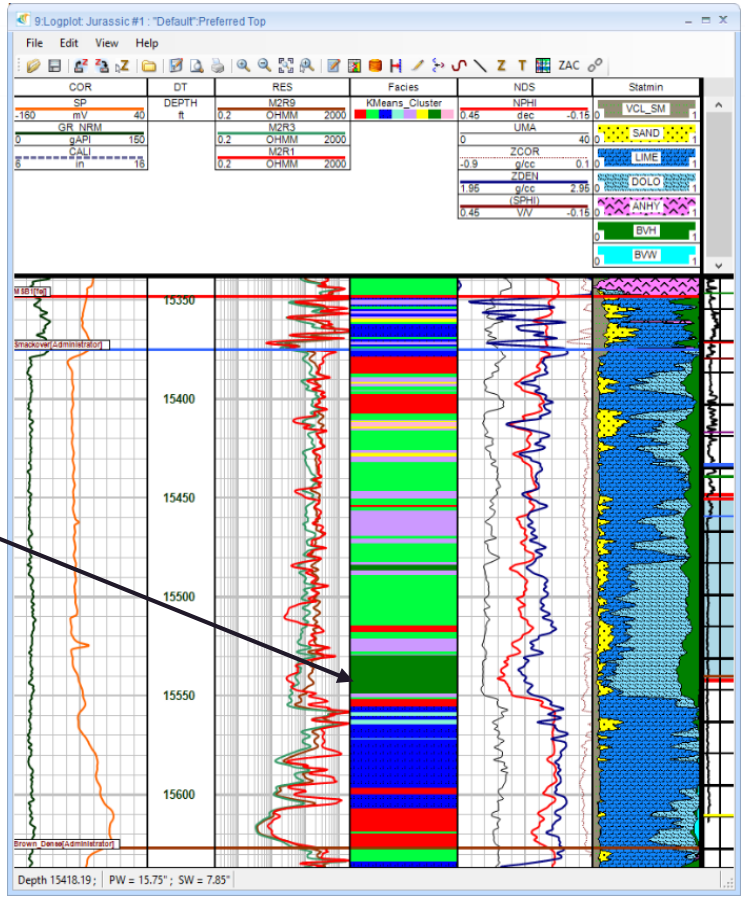


# Sub-Clustering Process and Results

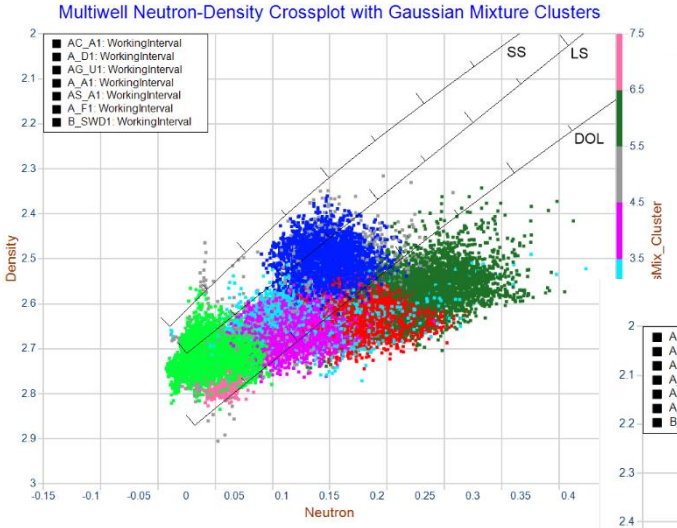
```
In [ ]: PartitionSubclusters = True


        #Enter the cluster number as key, and number of sub clusters as value
        #Format sub_clusters_dict[cluster_number] = how_many_sub_clusters
        #Example sub_clusters[3] = 2 This tells the algorithm to create 2 sub clusters for cluster 3

        sub_clusters_dict = {}

        sub_clusters_dict[0] = 3
```

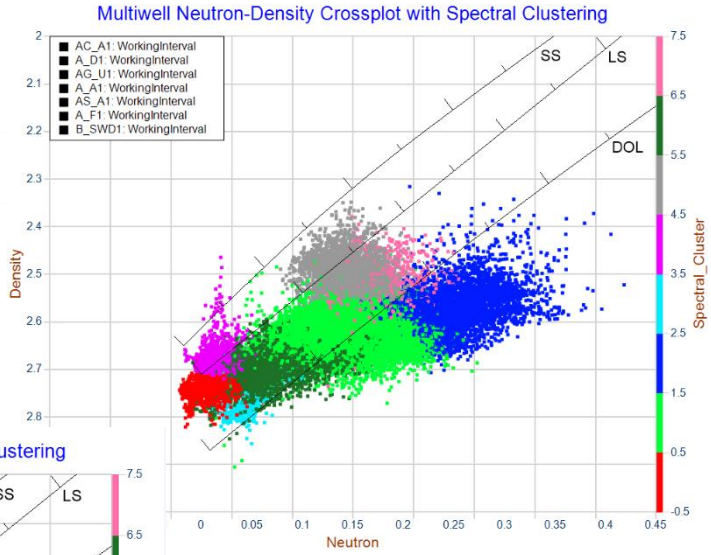
# Sub-Clustering Results

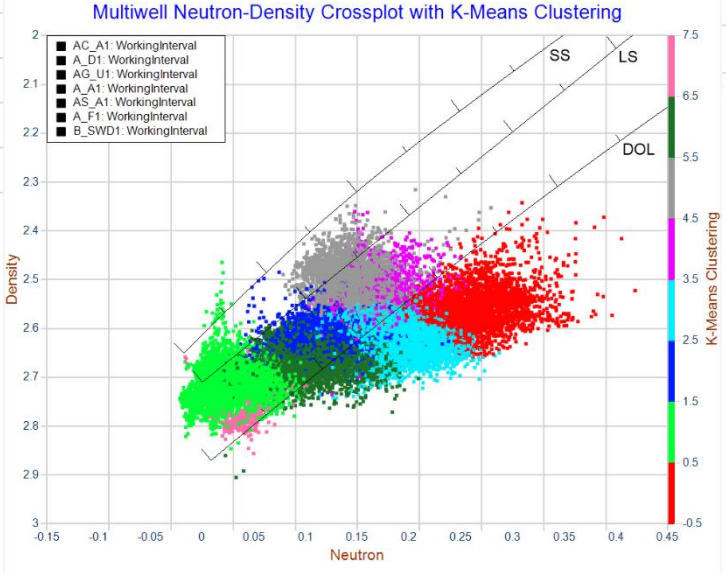


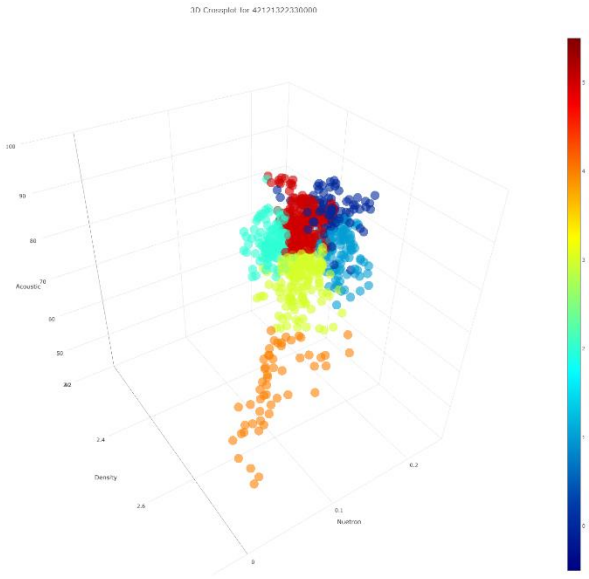
# Crossplots of Clustering Types



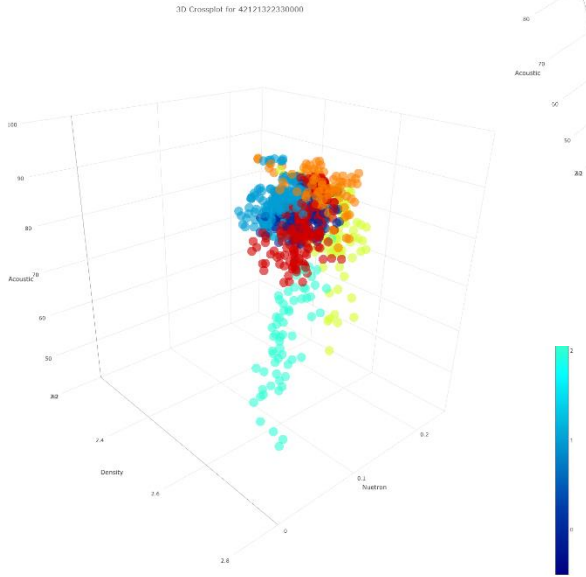



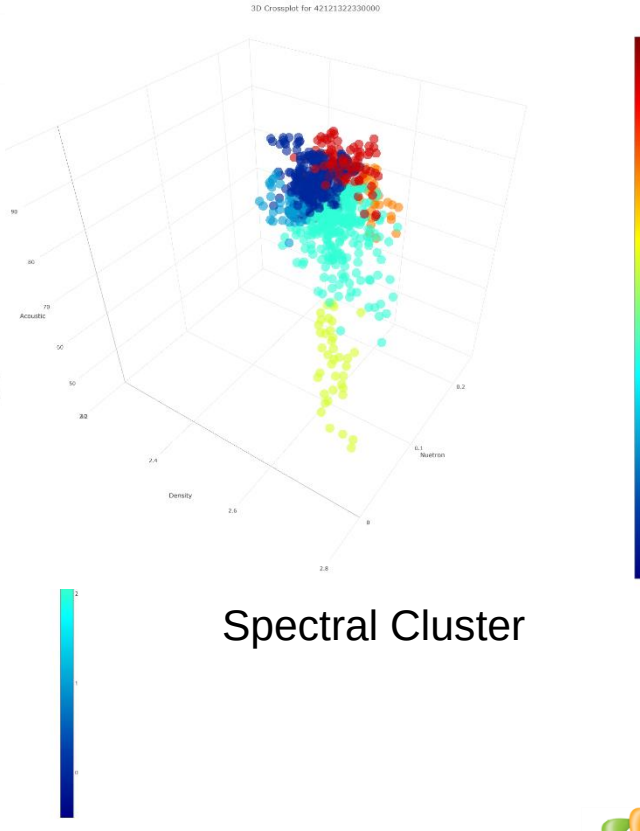
# 3D Crossplots of Clustering Results




K-Means




Gaussian Mixture




Spectral Cluster

# Automated Log Editing

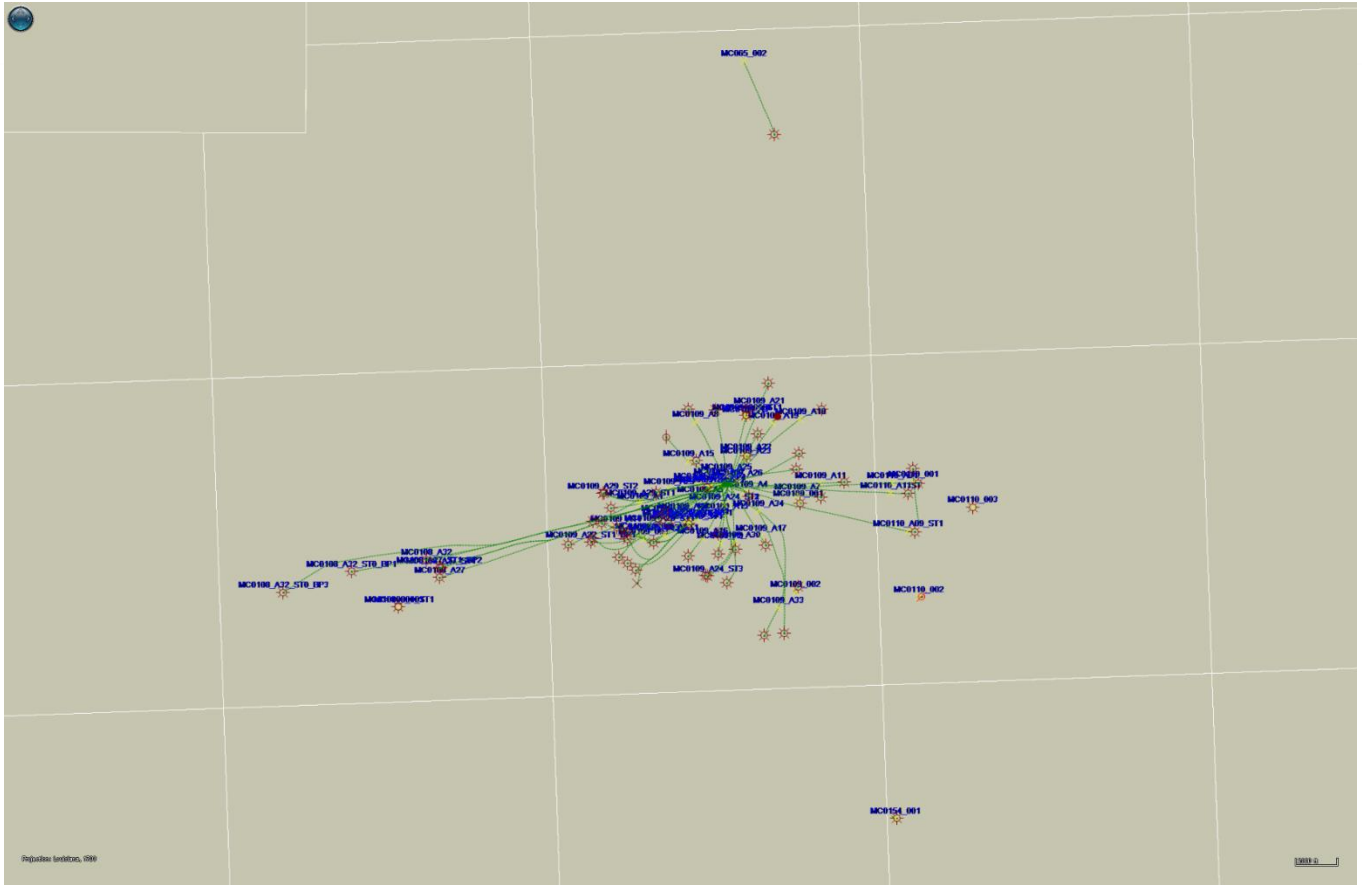
Chiran Ranganathan, Fred Jenson, Joe Johnston

CGG
Passion for Geoscience

# Why Automate Log Editing?

The objective of automated log editing is to greatly reduce the amount of time spent in repetitive mundane operations preparing data for meaningful analysis.

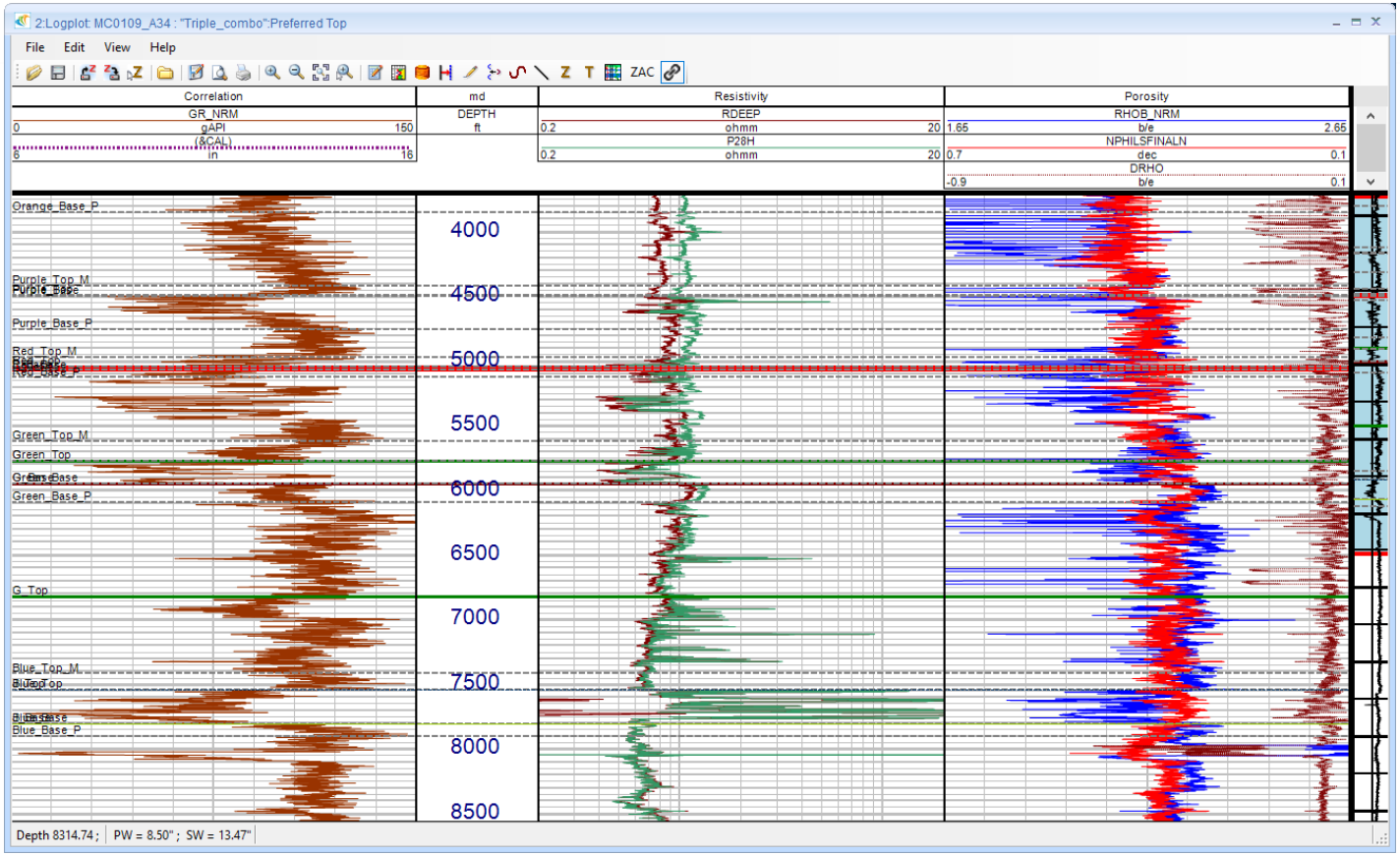
# The Area of Interest



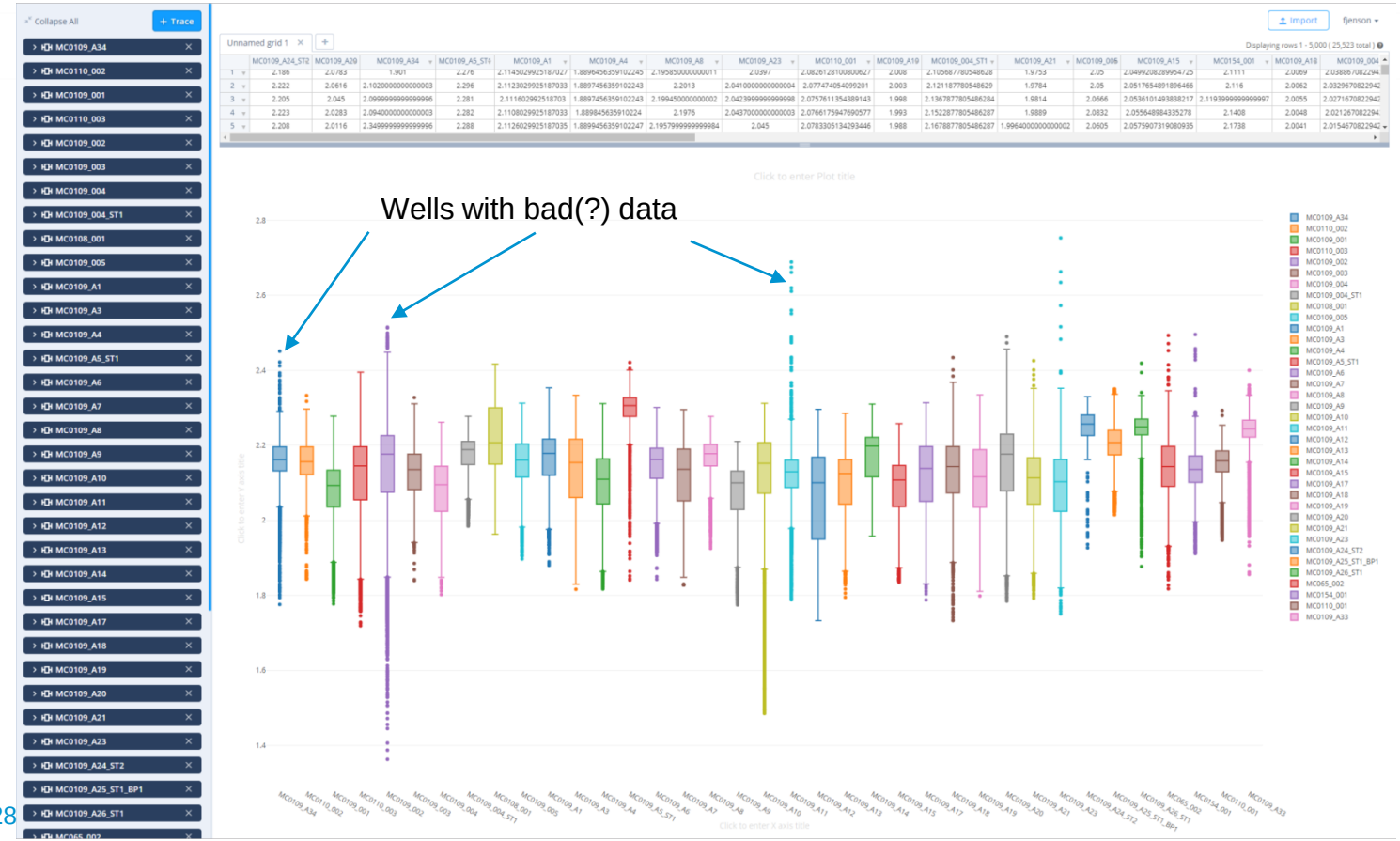
# The Automated Editing Workflow

- Prep the data
- Review the data
- Anomaly detection and flag curve generation
- Pay flag creation
- Create nulled flag curve for use in synthetic generation
- Generate synthetic curve over non-nulled intervals
- Merge synthetic curve with measured curve controlled by anomaly flags

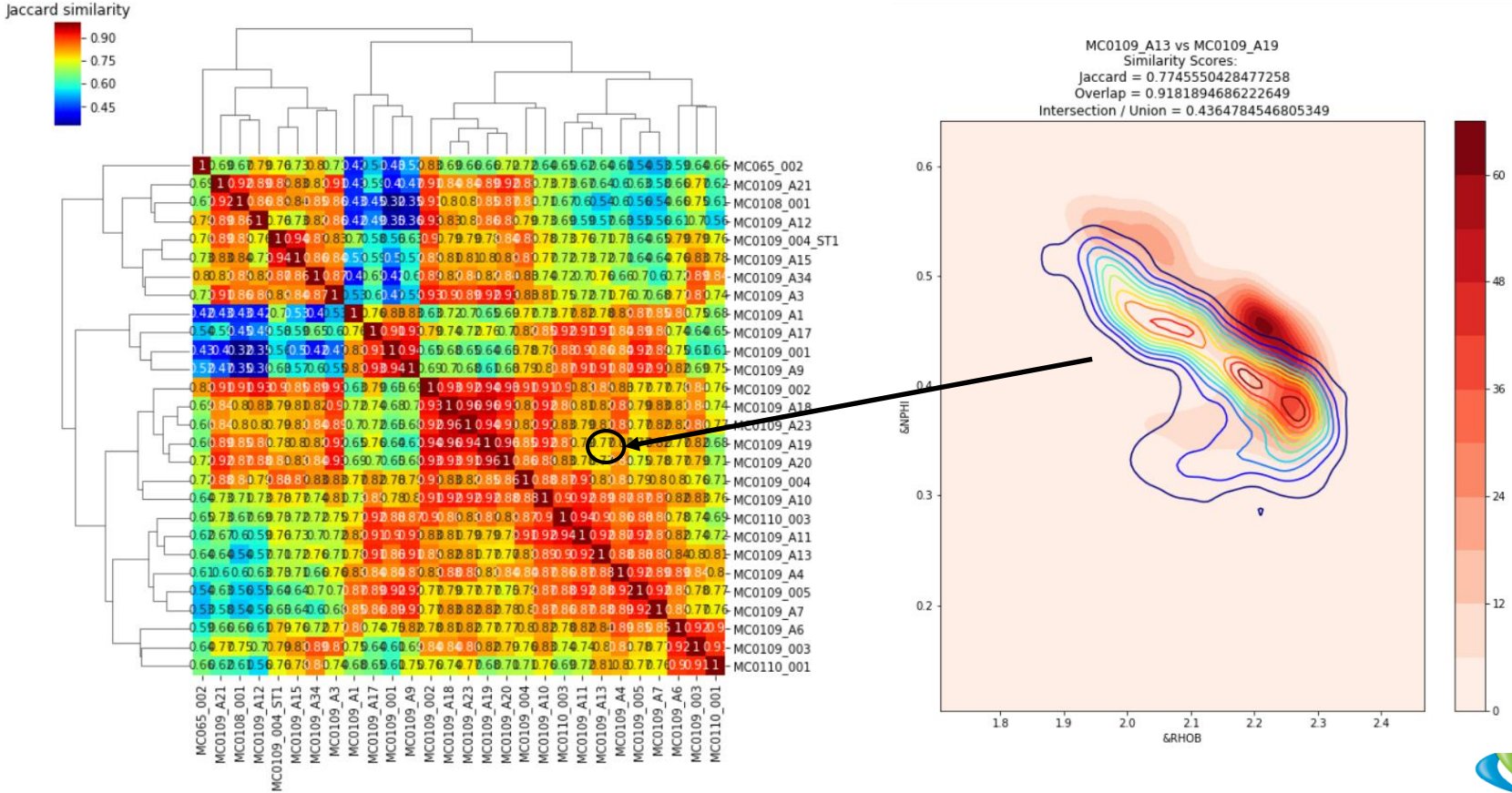
# Log Data Example



# Examine the Density Curve Data



# Similarity Analysis



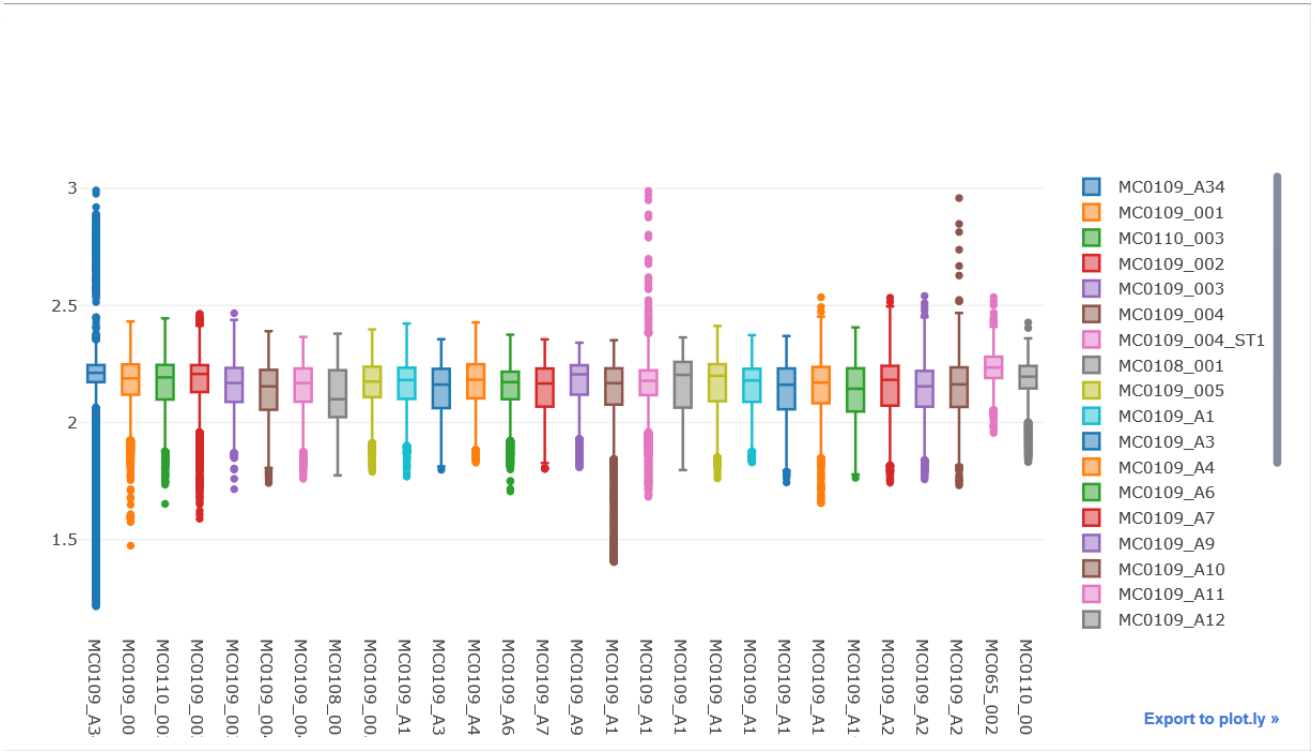
# Jupyter used for Machine Learning Workflows

## Multi-Well Boxplot Outlier Detection

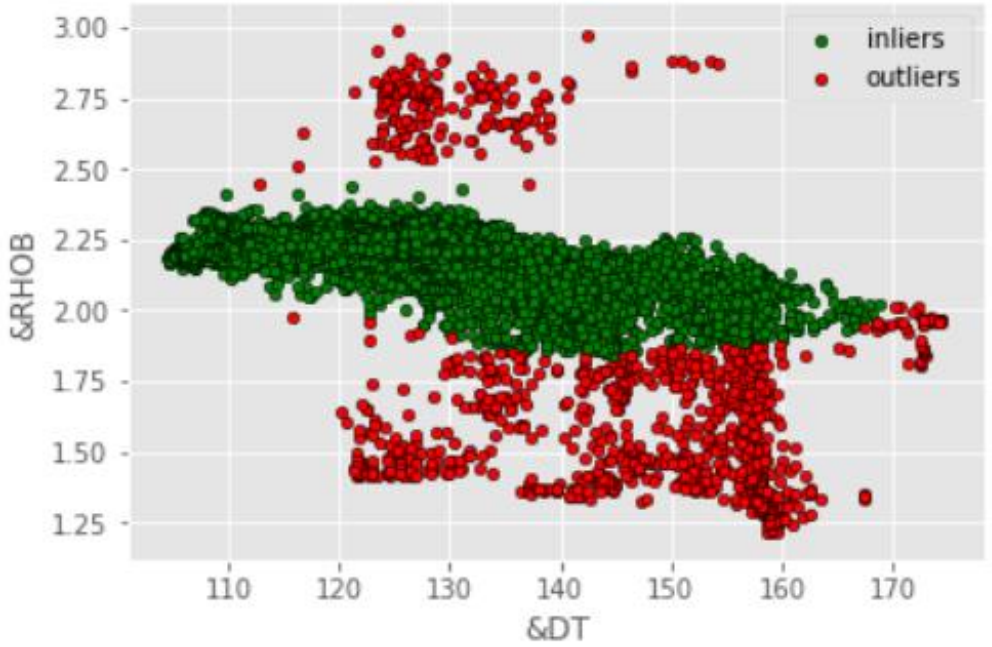
In [ ]:
```
%matplotlib inline
import plconnect.functions as plf
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plconnect.functions as plf
```

In [ ]:
```
uwi_list = plf.get_current_selected_wells()
all_frames = []
all_uwis = []
all_wellnames = []

curve_name =  "&RHOB"
outlier_flag = "Boxplot_Flag"

zone = "WorkingInterval"
grid = "Default"
```

# Anomaly Detection, Boxplots with Whiskers



# Spatial Clustering Technique




There are parameters set to control the number of outliers detected using spatial clustering. The main parameters are

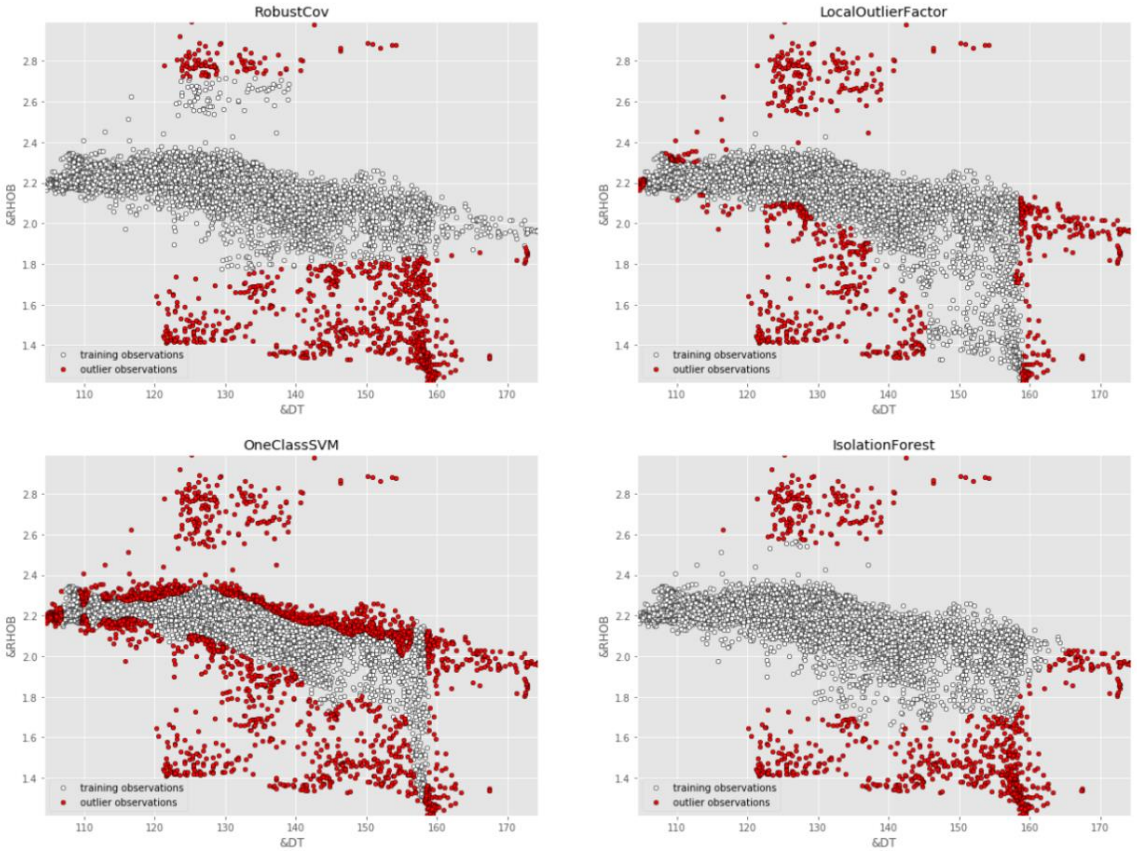
**Eps** The maximum distance between two samples for them to be considered in the same neighborhood.

**min_samples** The minimum number of samples to be a neighborhood.

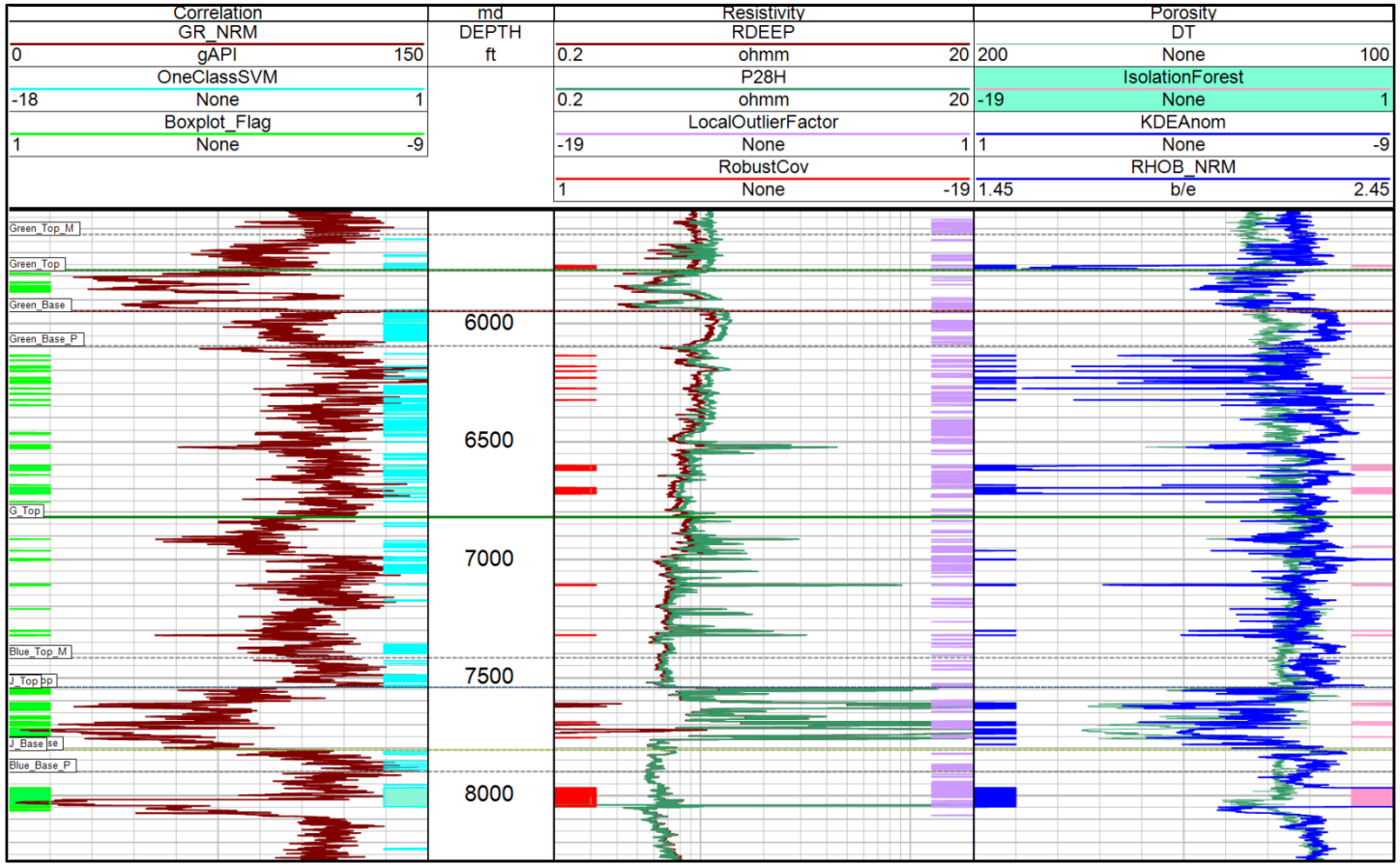
There are other parameters of interest that can be adjusted to optimize outlier detection including clustering algorithms.

# Outlier Detections Methods

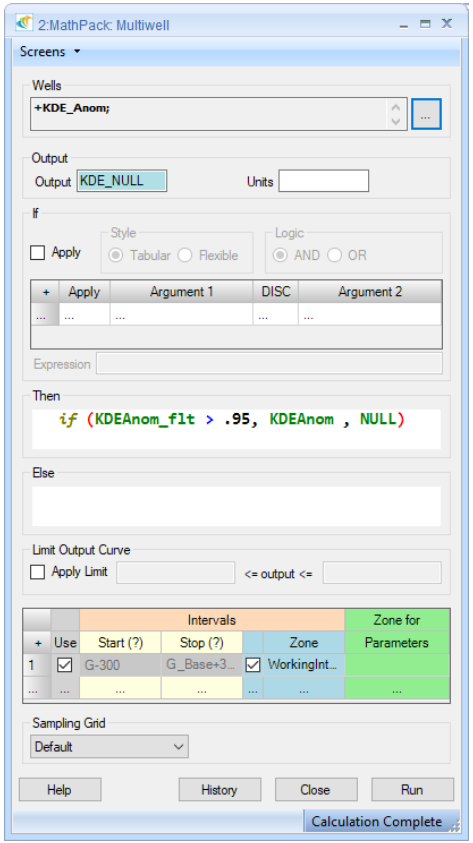
Each of these outlier detection methods have parameters that can be tuned for optimum results.



# Flag Curves


Correlation
GR_NRM
0 gAPI 150
OneClassSVM
-18 None 1
Boxplot_Flag
1 None -9
md
DEPTH
ft
Resistivity
RDEEP
0.2 ohmm 20
P28H
0.2 ohmm 20
LocalOutlierFactor
-19 None 1
RobustCov
1 None -19
Porosity
DT
200 None 100
IsolationForest
-19 None 1
KDEAnom
1 None -9
RHOB_NRM
1.45 b/e 2.45
Green_Top_M
Green_Top
Green_Base
Green_Base_P
G_Top
Blue_Top_M
J_Top pp
J_Base se
Blue_Base_P
6000
6500
7000
7500
8000

# Synthetic Curve Generation




```
#Create a directory to save the trained model and enter its path below
MODEL_SAVE_DIRECTORY = r"C:\CCGpy35\pl_pydistribution_3.5.4.5Qt5\Models"
DNN_MODEL_NAME = "keras_RHOB_predictions"
MODEL_SUFFIX = "_dlmodel"

if not os.path.exists(MODEL_SAVE_DIRECTORY):
    os.makedirs(MODEL_SAVE_DIRECTORY)

uwi_list = []
dataset = None
#Define the independent variables and the dependent variable. The last variable is the target
#curves_to_fetch = ["&NPHI","&RHOB","&PE", "&RILD", "BVW_SM","VCLSM", "&DT", "&DTS"]
curves_to_fetch = [ "&NPHI","&GR", "log10(&RILD)", "DPTH", "&RHOB"]
#Define the set above with depth included, cleaning up any &/$

curves_to_fetch_prediction = copy.deepcopy(curves_to_fetch)
curves_to_fetch_prediction.pop(-1)


column_names = ["DEPTH"]
for curve_name in curves_to_fetch:
    column_names.append(curve_name)


#training_zones = ["Barnett", "Marble_Falls"] #Add more zones that do not overlap to filter to higher quality training data
training_zones = ["WorkingInterval"]

#interval1 = WellInterval(Start = "G_MKR", Stop = "J_Base")

#grid = "Default"  #consider using Same As:&DT for training
grid = "Same As:KDE_Null"


SelectTrainingWellsFromAGroup = True
TrainingWellsGroupName = "KDE_Anom"  #If SelectTrainingWellsFromAGroup=True then this well group will be used
```

# Synthetic Curve Prediction

## Predictions Step 1: Load the model to predict

In [19]:
```
from keras.models import load_model, model_from_json
regressor_path = os.path.join(MODEL_SAVE_DIRECTORY, DNN_MODEL_NAME +MODEL_SUFFIX + ".json_model")
weights_path = os.path.join(MODEL_SAVE_DIRECTORY, DNN_MODEL_NAME +MODEL_SUFFIX + ".h5")

dl_model_json = None
with open(regressor_path,"r") as f:
    dl_model_json = f.read()

model_loaded = model_from_json(dl_model_json)
model_loaded.load_weights(weights_path)

sc_X_path = os.path.join(MODEL_SAVE_DIRECTORY, DNN_MODEL_NAME +"_preproc" + ".scaling")
sc_X = pickle.load(open(sc_X_path, 'rb'))

print("Model loaded")
```

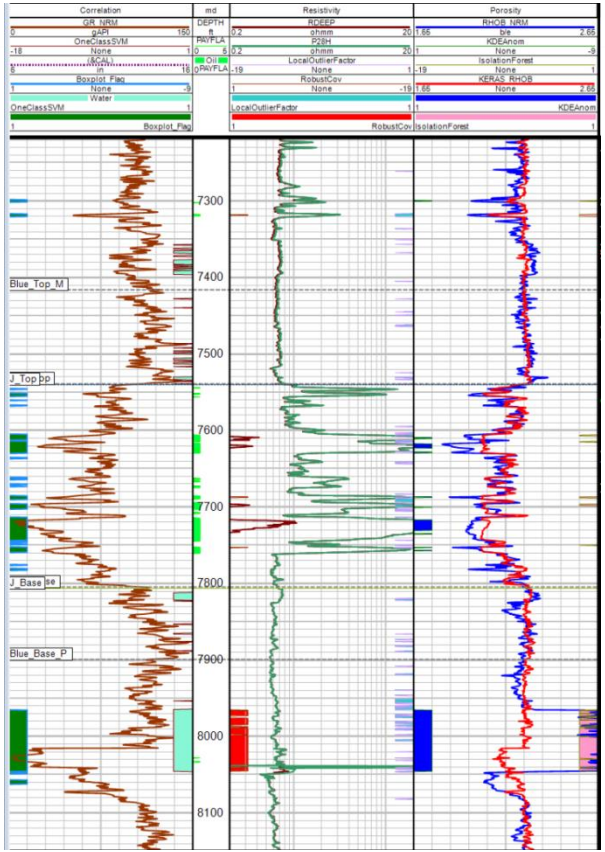
```
Model loaded
```

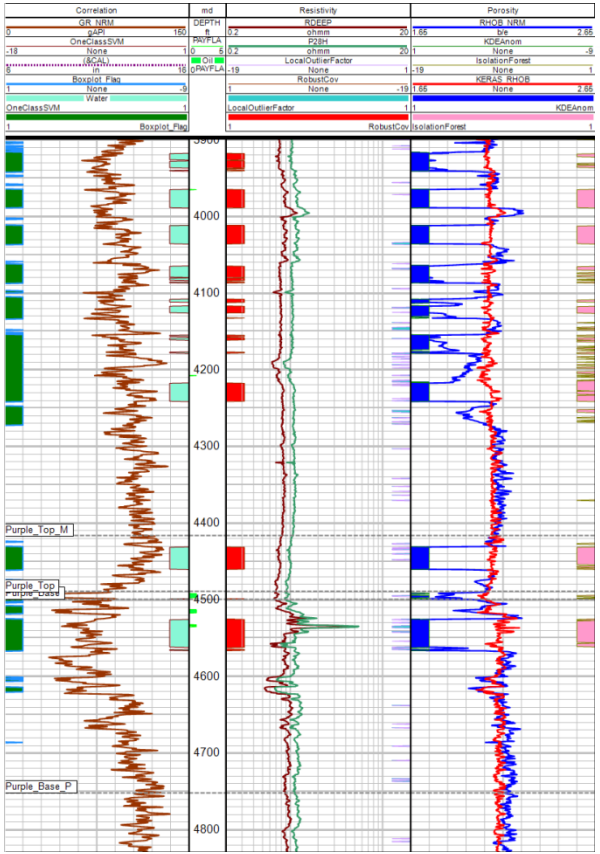
## Predictions Step 2 :Select the well(s) in Powerlog to make a prediction

In [20]:
```
prediction_wells = plf.get_current_selected_wells()
print(prediction_wells)

from plconnect.exceptions import ExtensionException

OUTCURVENAME = "KERAS_RHOB"

prediction_zone  = "WorkingInterval"
prediction_grid = "Default"
```

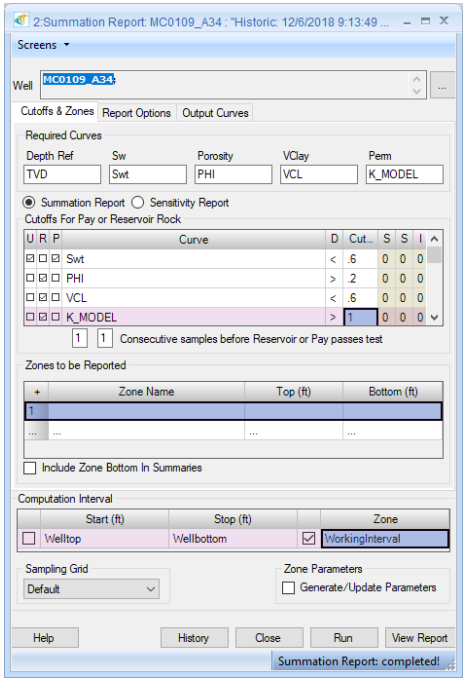


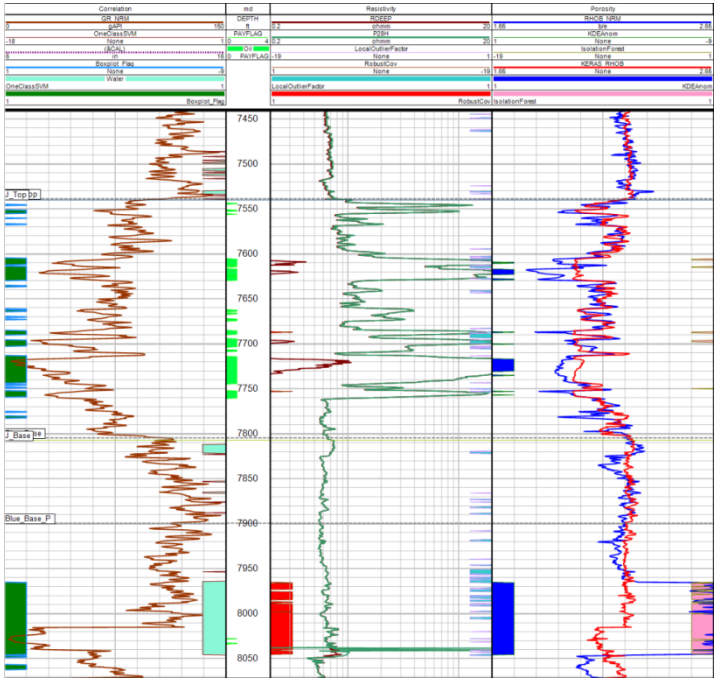
# Keras Generated RHOB Curve



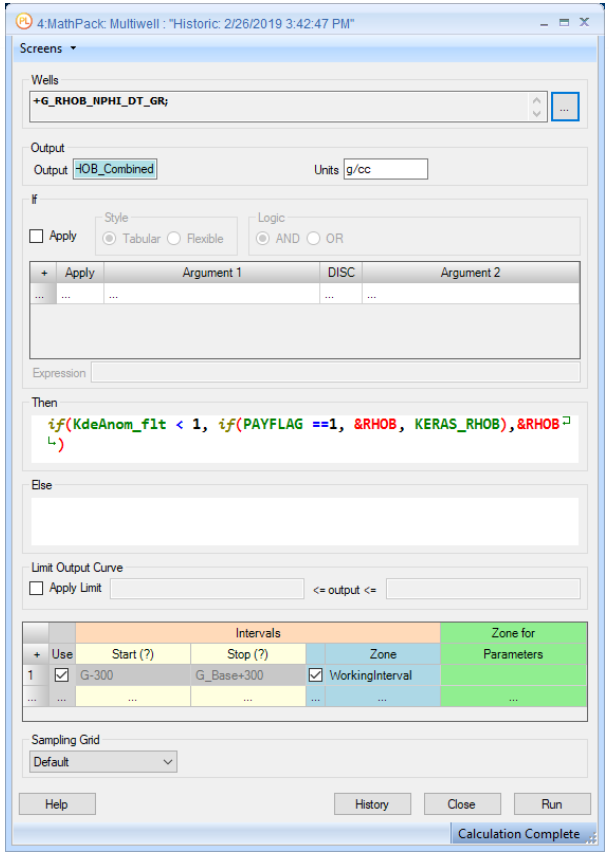
# Pay Zone Protection

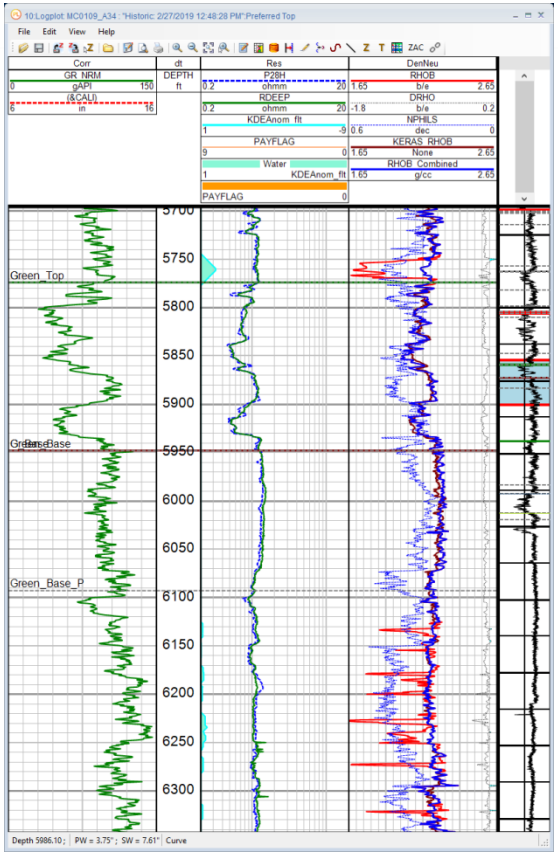


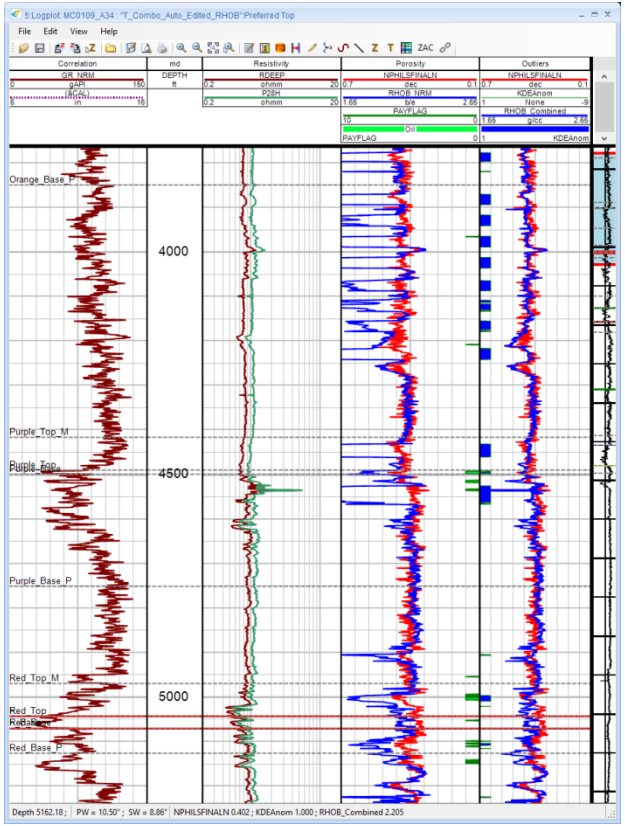

Productive intervals often show up as anomalies with outlier detection packages. A pay flag can mediate this issue.

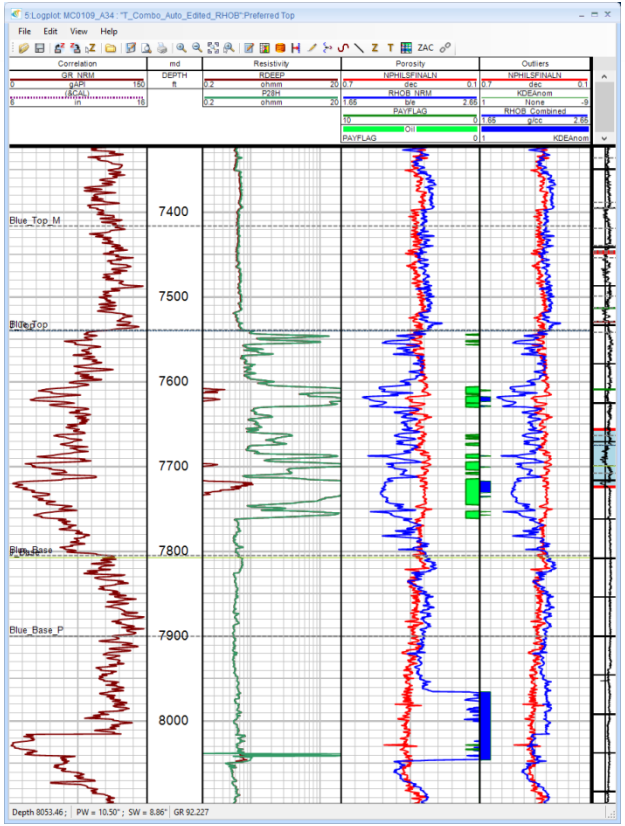
# Creating the Combined Curved



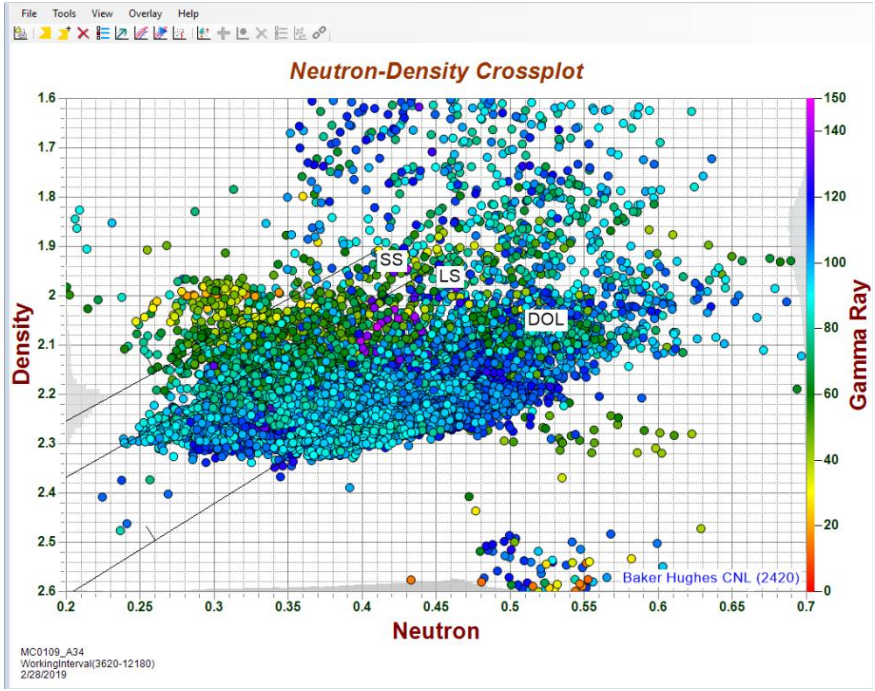



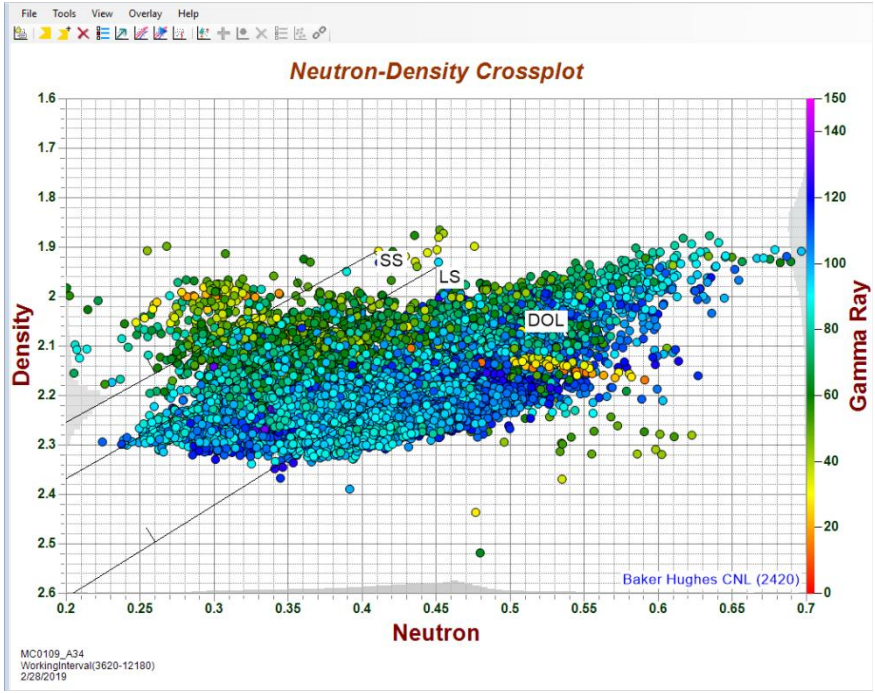
# Combined RHOB Curve



# Original versus Merged Data



# Summary

- Machine learning workflows can greatly reduce the time required to correct curve data when dealing with significant numbers of wells
- Eliminating repetitive mundane tasks is a good thing
- These workflows can be saved and used in other areas
- It is a lot more efficient to create valid log data with Jupyter Workflows than manually editing curves on large numbers of wells

# Thank You



CGG
Passion for Geoscience