# Data explorations, event detection, and introduction to statistics

**1- Load the library welly in the conda environment Petrophysics and launch Jupyter Lab**

conda activate DrillingAnalytics

conda list welly

pip install welly

conda list welly

cd ~/DataScienceComodoro2023

Jupyter-Lab

**2- Load welly library in notebook**

from welly import Well

from welly import Curve

**3- Load well log in welly object: 8267\_a0801\_1996\_comp.las**

well=Well.from\_las('Data/LAS/8267\_a0801\_1996\_comp.las')

well

**4- Plot well log data**

tracks =['MD','GR', ['NPHI','RHOB','DRHO'],'DT']

well.plot(tracks=tracks)

**5- How many GR values are in the log and how many are missing?**

well.data[‘GR']

**6- What is the average of GR values?**

gr=well.data['GR']

gr.describe()

**7- What is the average of GR values in a the sand?**

gr=well.data['GR']

gr[2600:3000].describe()

**8- Determine which curves have no data, which have gaps, flatlined and if GR is below 25**

import welly.quality as quality

test = {'Each':[quality.no\_flat,

quality.no\_gaps,

quality.not\_empty

],

'GR':[quality.all\_above(25)

]}

from IPython.display import HTML

data\_qc\_table=well.qc\_table\_html(test)

HTML(data\_qc\_table)

**9- What is the proportion of NaN values in each curve?**

test = {'Each':[quality.no\_flat,

quality.no\_gaps,

quality.not\_empty,

quality.fraction\_not\_nans

],

'GR':[quality.all\_above(25)

]}

data\_qc\_table=well.qc\_table\_html(test)

HTML(data\_qc\_table)

**10- Which curves have spikes?**

test = {'Each':[quality.no\_flat,

quality.no\_gaps,

quality.not\_empty,

quality.fraction\_not\_nans,

quality.no\_spikes(100),

quality.count\_spikes

],

'GR':[quality.all\_above(25)

]}

data\_qc\_table=well.qc\_table\_html(test)

HTML(data\_qc\_table)

**11- Load the gamma ray curve on to a dataframe**

gamma\_dataframe=gr.df

gamma\_dataframe

**12- Load all the curves on to a dataframe**

well\_dataframe=well.df()

well\_dataframe

**13- How many gamma ray readings are there in the dataframe? How many readings should there be for the entire well? If there is a difference, why would that be?**

len(gamma\_dataframe)

len(well\_dataframe)

They are different because the dataframe for the gamma curve includes readings at all depth, while the dataframe with all curves has depth every single depth unit

**14 -Load all the curves into a dataframe, then verify it has the same number of rows as the gamma dataframe**

from welly import Well

import pandas as pd

import matplotlib.pyplot as plt

# Load the LAS file

well = Well.from\_las('Data/LAS/8267\_a0801\_1996\_comp.las')

# Initialize an empty DataFrame to hold all curve data

all\_curves\_df = pd.DataFrame()

# Iterate over all the curves in the well

for curve\_name, curve in well.data.items():

# Access each curve's DataFrame directly

curve\_df = curve.df

# Merge the curve data into the main DataFrame

if all\_curves\_df.empty:

all\_curves\_df = curve\_df # For the first curve, just assign

else:

all\_curves\_df = all\_curves\_df.join(curve\_df, how='outer') # Join on the index (e.g., depth)

# Plot the curves

fig, ax = plt.subplots(figsize=(10, 6))

# Loop through each curve in the DataFrame

for column in all\_curves\_df.columns:

ax.plot(all\_curves\_df.index, all\_curves\_df[column], label=column)

# Add labels and title

ax.set\_xlabel('Depth')

ax.set\_ylabel('Value')

ax.set\_title('Well Log Curves')

# Display the legend

ax.legend(loc='upper right')

# Show the plot

plt.tight\_layout()

plt.show()

len(all\_curves\_df)

**15- Load the las file using lasio and confirm the number of rows is correct in the sand: Data/LAS/8267\_a0801\_1996\_comp.las**

import lasio

las\_file\_path = 'Data/LAS/8267\_a0801\_1996\_comp.las'

well = lasio.read(las\_file\_path)

df=well.df()

len(df)

**16- Define a Curve class to hold all the curves in a well log**

class Curve:

def \_\_init\_\_(self, well\_name, well\_location, curve\_name, data):

self.well\_name = well\_name

self.well\_location = well\_location

self.curve\_name = curve\_name

self.data = data

def get\_well\_name(self):

return self.well\_name

def get\_well\_location(self):

return self.well\_location

def get\_curve\_name(self):

return self.curve\_name

def get\_data(self, start=None, end=None):

if start is None and end is None:

return self.data

elif start is None:

return self.data[self.data.index <= end]

elif end is None:

return self.data[self.data.index >= start]

else:

return self.data[(self.data.index >= start) & (self.data.index <= end)]

**17- Instantiate the object with gamma ray data**

curve=Curve("well 1","cerro dragon","gamma ray”,gamma\_dataframe)

**18- Print the name of the well, the location, the curve names and data using attributes**

print(“Well name: "+curve.well\_name+"\nLocation: “+curve.well\_location+"\nCurve name: “+curve.curve\_name+"\nData:\n"+str(curve.data))

**19- Print the name of the well, the location, the curve names and data using methods**

print(“Well name: "+curve.get\_well\_name()+"\nLocation: “+curve.get\_well\_location()+"\nCurve name: “+curve.get\_curve\_name()+"\nData:\n"+str(curve.get\_data()))

**20- Print the name of the well, the location, the curve names and data from 30 to 3000 meters using methods**

print(“Well name: "+curve.get\_well\_name()+"\nLocation: “+curve.get\_well\_location()+"\nCurve name: “+curve.get\_curve\_name()+"\nData:\n"+str(curve.get\_data(30,3000)))

**21- Print the name of the well, the location, the curve names and data from 30 to 3000 meters using attributes**

print(“Well name: "+curve.well\_name+"\nLocation: “+curve.well\_location+"\nCurve name: “+curve.curve\_name+”\nData:\n"+str(curve.data[30:3000]))

**22- Show the use of the rolling method in pandas**

import pandas as pd

# create a sample dataframe

dft = pd.DataFrame({'values': [4.72, 3.89, 3.75, 4.96, 3.62, 4.68, 4.34, 3.63, 3.54, 6.71, 7.89, 3.58, 3.71, 4.19, 3.4, 3.01, 3.33, 4.36, 3.63, 3.61, 3.23, 4.7,

4.57, 2.84, 2.63, 2.86, 2.86, 2.66, 3.04, 3.06, 2.67, 2.57, 2.9, 2.55, 3.33, 2.58, 2.7, 2.38, 2.71, 2.25, 2.05, 3, 3.12, 2.36, 2.76,

3.52, 3.11, 2.94, 2.73, 2.57, 3.35, 3.07, 2.69, 2.95, 3.14, 3.8, 3.26, 2.91, 3.16, 3.23, 3.07, 3.32, 3.56, 3.35, 3.35, 5.31,7.21,

5.14, 4.11, 8.71, 6.28, 6.27, 4.13, 3.11, 4.46, 3.47, 3.33, 2.9, 3.3, 3.28, 3.3, 3.28]})

# define the window size

WS = 5

# calculate the rolling average

dft['rolling\_average'] = dft['values'].rolling(window=WS).mean()

dft.plot()

**23- Load a LAS file into a pandas dataframe**

# Import necessary libraries

import welly

import pandas as pd

import matplotlib.pyplot as plt

# Load the LAS file using welly

las\_file = 'Data/LAS/Sculpin-1\_LWD\_MEM\_2350-3850mMD.las'

well = welly.Well.from\_las(las\_file)

# Copy log data to a pandas DataFrame

df = well.df()

# Filter the Gamma Ray (GR) and Resistivity (P16H, P22H, P28H, P34H, P40H)

# You can modify this to plot specific resistivity curves as needed

selected\_logs = ['GR', 'P16H', 'P22H', 'P28H', 'P34H', 'P40H']

df\_selected = df[selected\_logs].dropna()

# Create the plot

fig, ax = plt.subplots(1, 2, figsize=(10, 8))

# Plot Gamma Ray in the first track

ax[0].plot(df\_selected['GR'], df\_selected.index, color='green')

ax[0].set\_xlabel('Gamma Ray (GR)')

ax[0].set\_ylabel('Depth (m)')

ax[0].invert\_yaxis() # Depth increases downward

ax[0].grid()

# Plot Resistivity curves in the second track with log scale

for curve in ['P16H', 'P22H', 'P28H', 'P34H', 'P40H']:

ax[1].plot(df\_selected[curve], df\_selected.index, label=curve)

ax[1].set\_xlabel('Resistivity (ohm.m)')

ax[1].set\_xscale('log')

ax[1].invert\_yaxis() # Depth increases downward

ax[1].legend()

ax[1].grid()

# Show the plot

plt.tight\_layout()

plt.show()

**24- Now apply smoothing to all the curves**

# Apply rolling average smoothing to each curve (using window size WS)

WS = 10 # Define window size for smoothing

for curve in selected\_logs:

df\_selected[f'{curve}\_smoothed'] = df\_selected[curve].rolling(window=WS).mean()

# Create the plot

fig, ax = plt.subplots(1, 2, figsize=(10, 64))

# Plot original and smoothed Gamma Ray in the first track

ax[0].plot(df\_selected['GR'], df\_selected.index, color='green', label='GR (original)')

ax[0].plot(df\_selected['GR\_smoothed'], df\_selected.index, color='lime', label='GR (smoothed)', linestyle='--')

ax[0].set\_xlabel('Gamma Ray (GR)')

ax[0].set\_ylabel('Depth (m)')

ax[0].invert\_yaxis() # Depth increases downward

ax[0].grid()

ax[0].legend()

# Plot Resistivity curves (original and smoothed) in the second track with log scale

for curve in ['P16H', 'P22H', 'P28H', 'P34H', 'P40H']:

ax[1].plot(df\_selected[curve], df\_selected.index, label=f'{curve} (original)')

ax[1].plot(df\_selected[f'{curve}\_smoothed'], df\_selected.index, linestyle='--', label=f'{curve} (smoothed)')

ax[1].set\_xlabel('Resistivity (ohm.m)')

ax[1].set\_xscale('log')

ax[1].invert\_yaxis() # Depth increases downward

ax[1].legend()

ax[1].grid()

# Show the plot

plt.tight\_layout()

plt.show()

**25- Load a csv file on to a dataframe, making sure the name of the columns are correct: Swell-1A\_AsciiDrillData\_183.0-5006.csv**

file\_path = 'Data//ASCII//Swell-1A\_AsciiDrillData\_183.0-5006.csv'

df = pd.read\_csv(file\_path)

df.head()

columns=df.columns

file\_path = 'Data//ASCII//Swell-1A\_AsciiDrillData\_183.0-5006.csv'

df = pd.read\_csv(file\_path, skiprows=1)

df.head()

df.columns=columns

df.head()

**26- Determine if there are spikes in the data**

df.plot(subplots=True, figsize=(15,35))

There are Manu negative/vero low value spikes, these can be -999.25 readings

# Count how many values of -999.25 are in each column

missing\_value\_counts = (df == -999.25).sum()

# Print the results

print("Count of -999.25 values per column:") print(missing\_value\_counts)

**27- Change all -999.25 readings to NaN**

import numpy as np

df.replace(-999.25, np.nan, inplace=True)

df.head()

**28- Determine how many NaN readings are there in each curve**

print(“Number of missing readings")

for column in df.columns:

count\_nan = df[column].isna().sum()

print(f"{column}: {count\_nan}”)

**29- Replace all NaN with the average of the before and after value. Compare the before and after**

import pandas as pd

import numpy as np

# Function to replace one NaN with the average of the surrounding values

def replace\_one\_nan\_with\_avg(df, column\_name):

for i in range(1, len(df[column\_name]) - 1):

if pd.isna(df.at[i, column\_name]):

if not pd.isna(df.at[i - 1, column\_name]) and not pd.isna(df.at[i + 1, column\_name]):

df.at[i, column\_name] = (df.at[i - 1, column\_name] + df.at[i + 1, column\_name]) / 2

else:

df.at[i, column\_name] = np.nan

return df

# Store the initial NaN count for each column in a dictionary

initial\_nan\_dic = df.isna().sum().to\_dict()

# Replace NaNs in specified columns

for column in columns: # 'columns' should be defined elsewhere with the list of columns

replace\_one\_nan\_with\_avg(df, column)

# Display the original and new NaN counts

for column in df.columns:

count\_nan = df[column].isna().sum()

print(f"Column: {column}: new NaN {count\_nan}, old NaN {initial\_nan\_dic[column]}")

**30- Replace all NaN with the average of the before and after value considering they could also be NaN values. Compare the before and after**

def replace\_many\_nan\_with\_avg(df, column\_name):

for i in range(len(df[column\_name])):

if pd.isna(df[column\_name][i]):

prev\_value = None

next\_value = None

for j in range(i-1, -1, -1):

if not pd.isna(df[column\_name][j]):

prev\_value = df[column\_name][j]

break

for j in range(i+1, len(df[column\_name])):

if not pd.isna(df[column\_name][j]):

next\_value = df[column\_name][j]

break

if prev\_value is None and next\_value is None:

df[column\_name][i] = np.nan

elif prev\_value is None:

df[column\_name][i] = next\_value

elif next\_value is None:

df[column\_name][i] = prev\_value

else:

df[column\_name][i] = (prev\_value + next\_value) / 2

return df

for column in columns:

replace\_many\_nan\_with\_avg(df,column)

for column in df.columns:

count\_nan = df[column].isna().sum()

print(f"Column: {column}: new NaN {count\_nan}, old NaN {initial\_nan\_dic[column]}")

**31- Load time LAS files for an entire run**

def read\_las\_files(file\_list):

dfs=pd.DataFrame()

for file in file\_list:

well = lasio.read('Data/LAS/'+file)

df = well.df()

dfs=pd.concat([dfs, df], ignore\_index=True)

return dfs

file\_list=['210915\_IOA\_07\_BDC-2-04\_TD\_695.las','210916\_IOA\_08\_BDC-2-04\_TD\_695.las','210917\_IOA\_09\_BDC-2-04\_TD\_1171.las','210918\_IOA\_10\_BDC-2-04\_TD\_1551.las',

'210919\_IOA\_11\_BDC-2-04\_TD\_2047.las','210920\_IOA\_12\_BDC-2-04\_TD\_2261.las','210921\_IOA\_13\_BDC-2-04\_TD\_2327.las','210922\_IOA\_14\_BDC-2-04\_TD\_2462.las',

'210923\_IOA\_15\_BDC-2-04\_TD\_2516.las','210924\_IOA\_16\_BDC-2-04\_TD\_2516.las']

df=read\_las\_files(file\_list)

**32- Show the depth curve**

ax=df.plot(y='HDEP',use\_index=True)

ax.invert\_yaxis()

plt.show()

**33- Determine if the depth curve increases constantly**

df[‘HDEP'].is\_monotonic\_increasing

**34- Are the depth curves is FE logs increasing constantly?**

import welly

import pandas as pd

import warnings

# Load the LAS file using welly

las\_file = 'Data/LAS/Sculpin-1\_LWD\_MEM\_2350-3850mMD.las'

well = welly.Well.from\_las(las\_file)

# Copy log data to a pandas DataFrame

df = well.df()

# Filter the Gamma Ray (GR) and Resistivity (P16H, P22H, P28H, P34H, P40H)

selected\_logs = ['GR', 'P16H', 'P22H', 'P28H', 'P34H', 'P40H']

df\_selected = df[selected\_logs].dropna()

df\_selected.columns

df\_selected.index

df\_selected.index.is\_monotonic\_increasing

**35- Define a function that computes water saturation using Archie, and test is for some readings**

def sw\_archie(porosity, rt, rw, archieA=1, archieM=2, archieN=2):

sw = ((archieA / (porosity \*\* archieM)) \* (rw / rt)) \*\* (1 / archieN)

return sw

sw\_value = sw\_archie(porosity=0.25, rt=2.5, rw=0.1)

sw\_value = sw\_archie(porosity=0.25, rt=2.5, rw=0.1, archieA=1.2, archieM=1.8, archieN=2.5)