# Modeling

**1- Load libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

**2- Write a function to compute the Archie equation**

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

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

return sw

**3- Write a function to compute density porosity**

def density\_porosity(input\_density, matrix\_density, fluid\_density):

denpor = (matrix\_density - input\_density) / (matrix\_density - fluid\_density)

return round(denpor, 4)

**4- Load the log data in the file 15\_9-19.csv**

well = pd.read\_csv(“Data/ASCII/15\_9-19.csv", header=0, skiprows=[1])

**5- Determine if there is any missing data**

nan\_counts = df.isna().sum()

print(nan\_counts)

well.describe()

df.replace(-999.00, np.nan, inplace=True)

df.describe()

nan\_counts = df.isna().sum()

print(nan\_counts)

**6- Plot a histogram for gamma ray**

well['GR'].plot(kind='hist', bins=230)

plt.xlim(0, 250)

**7- Calculate the density porosity for this well**

df['PHI'] = density\_porosity(def['RHOB'], 2.65, 1)

df[‘PHI’]

df['PHI'].plot(kind='hist', bins=230)

plt.xlim(0, 1)

**8- Determine if you have all the data required to compute Archie water saturation**

def check\_required\_columns(df, required\_columns):

missing\_columns = [col for col in required\_columns if col not in df.columns]

if missing\_columns:

print(f"Missing columns: {missing\_columns}")

return False

return True

required\_columns = ['PHI', 'RT', 'RW']

if check\_required\_columns(df, required\_columns):

print("All required columns are present.")

else:

print("Some required columns are missing.”)

**9- Calculate water saturation and analyze the quality of the data**

df['SW'] = sw\_archie(df['PHI'], df['RT'], df['RW'], 1, 2, 2)

df[‘SW'].describe()

df['SW\_LIM'] = df['SW'].mask(df['SW']>1, 1)

df[‘SW\_LIM'].describe()

df['SW\_LIM'].plot(kind='hist', bins=230)

plt.ylim(0, 50)

plt.xlim(0, 1)

**10- Define a function to compute shale volume**

def shale\_volume(gamma\_ray, gamma\_ray\_max, gamma\_ray\_min):

vshale = (gamma\_ray - gamma\_ray\_min) / (gamma\_ray\_max - gamma\_ray\_min)

return round(vshale, 4)

**11- Define a function to compute Simandoux water saturation**

def sw\_simandoux(phie, rt, rw, archieA, archieM, archieN, vshale, rshale):

A = (1 - vshale) \* archieA \* rw / (phie \*\* archieM)

B = A \* vshale / (2 \* rshale)

C = A / rt

sw = ((B \*\*2 + C)\*\*0.5 - B) \*\*(2 / archieN)

return sw

**12- Compute Simandoux water saturation for the given well**

well['VSHALE'] = shale\_volume(well['GR'], well['GR'].quantile(q=0.99),

well[‘GR'].quantile(q=0.01))

well['PHIECALC'] = well['PHI'] - (well['VSHALE'] \* 0.3)

well['SW\_SIM'] = sw\_simandoux(well['PHIECALC'], well['RT'], well['RW'], 1, 2, 2, well['VSHALE'],2)

well['SW\_\_SIM\_LIM'] = well['SW\_SIM'].mask(well['SW\_SIM']>1, 1)

**13- Plot all the petrophysical data**

fig, ax = plt.subplots(figsize=(15,10))

#Set up the plot axes

ax1 = plt.subplot2grid((1,7), (0,0), rowspan=1, colspan = 1)

ax2 = plt.subplot2grid((1,7), (0,1), rowspan=1, colspan = 1)

ax3 = plt.subplot2grid((1,7), (0,2), rowspan=1, colspan = 1)

ax4 = plt.subplot2grid((1,7), (0,3), rowspan=1, colspan = 1)

ax5 = ax3.twiny() #Twins the y-axis for the density track with the neutron track

ax6 = plt.subplot2grid((1,7), (0,4), rowspan=1, colspan = 1)

ax7 = ax6.twiny()

ax8 = plt.subplot2grid((1,7), (0,5), rowspan=1, colspan = 1)

ax9 = plt.subplot2grid((1,7), (0,6), rowspan=1, colspan = 1)

# As our curve scales will be detached from the top of the track,

# this code adds the top border back in without dealing with splines

ax10 = ax1.twiny()

ax10.xaxis.set\_visible(False)

ax11 = ax2.twiny()

ax11.xaxis.set\_visible(False)

ax12 = ax3.twiny()

ax12.xaxis.set\_visible(False)

# Gamma Ray track

ax1.plot("GR", "DEPTH", data = well, color = "green")

ax1.set\_xlabel("Gamma")

ax1.xaxis.label.set\_color("green")

ax1.set\_xlim(0, 200)

ax1.set\_ylabel("Depth (m)")

ax1.tick\_params(axis='x', colors="green")

ax1.spines["top"].set\_edgecolor("green")

ax1.title.set\_color('green')

ax1.set\_xticks([0, 50, 100, 150, 200])

# Resistivity track

ax2.plot("RT", "DEPTH", data = well, color = "red")

ax2.set\_xlabel("Resistivity")

ax2.set\_xlim(0.2, 2000)

ax2.xaxis.label.set\_color("red")

ax2.tick\_params(axis='x', colors="red")

ax2.spines["top"].set\_edgecolor("red")

ax2.set\_xticks([0.1, 1, 10, 100, 1000])

ax2.semilogx()

# Density track

ax3.plot("RHOB", "DEPTH", data = well, color = "red")

ax3.set\_xlabel("Density")

ax3.set\_xlim(1.95, 2.95)

ax3.xaxis.label.set\_color("red")

ax3.tick\_params(axis='x', colors="red")

ax3.spines["top"].set\_edgecolor("red")

ax3.set\_xticks([1.95, 2.45, 2.95])

# Sonic track

ax4.plot("DT", "DEPTH", data = well, color = "purple")

ax4.set\_xlabel("Sonic")

ax4.set\_xlim(140, 40)

ax4.xaxis.label.set\_color("purple")

ax4.tick\_params(axis='x', colors="purple")

ax4.spines["top"].set\_edgecolor("purple")

# Neutron track placed ontop of density track

ax5.plot("NPHI", "DEPTH", data = well, color = "blue")

ax5.set\_xlabel('Neutron')

ax5.xaxis.label.set\_color("blue")

ax5.set\_xlim(0.45, -0.15)

ax5.set\_ylim(4150, 3500)

ax5.tick\_params(axis='x', colors="blue")

ax5.spines["top"].set\_position(("axes", 1.08))

ax5.spines["top"].set\_visible(True)

ax5.spines["top"].set\_edgecolor("blue")

ax5.set\_xticks([0.45, 0.15, -0.15])

# Porosity track

ax6.plot("PHI", "DEPTH", data = well, color = "black")

ax6.set\_xlabel("Total PHI")

ax6.set\_xlim(0.5, 0)

ax6.xaxis.label.set\_color("black")

ax6.tick\_params(axis='x', colors="black")

ax6.spines["top"].set\_edgecolor("black")

ax6.set\_xticks([0, 0.25, 0.5])

# Porosity track

ax7.plot("PHIECALC", "DEPTH", data = well, color = "blue")

ax7.set\_xlabel("Effective PHI")

ax7.set\_xlim(0.5, 0)

ax7.xaxis.label.set\_color("blue")

ax7.tick\_params(axis='x', colors="blue")

ax7.spines["top"].set\_position(("axes", 1.08))

ax7.spines["top"].set\_visible(True)

ax7.spines["top"].set\_edgecolor("blue")

ax7.set\_xticks([0, 0.25, 0.5])

# Sw track

ax8.plot("SW\_LIM", "DEPTH", data = well, color = "black")

ax8.set\_xlabel("SW - Archie")

ax8.set\_xlim(0, 1)

ax8.xaxis.label.set\_color("black")

ax8.tick\_params(axis='x', colors="black")

ax8.spines["top"].set\_edgecolor("black")

ax8.set\_xticks([0, 0.5, 1])

# Sw track

ax9.plot("SW\_SIM", "DEPTH", data = well, color = "blue")

ax9.set\_xlabel("SW - Simandoux")

ax9.set\_xlim(0, 1)

ax9.xaxis.label.set\_color("blue")

ax9.tick\_params(axis='x', colors="blue")

ax9.spines["top"].set\_edgecolor("blue")

ax9.set\_xticks([0, 0.5, 1])

# Common functions for setting up the plot can be extracted into

# a for loop. This saves repeating code.

for ax in [ax1, ax2, ax3, ax4, ax6, ax8, ax9]:

ax.set\_ylim(4150, 3500)

ax.grid(which='major', color='lightgrey', linestyle='-')

ax.xaxis.set\_ticks\_position("top")

ax.xaxis.set\_label\_position("top")

ax.spines["top"].set\_position(("axes", 1.02))

plt.tight\_layout()

**14- Load data from file Data/ASCII/L0509WellData.csv**

df **=** pd**.**read\_csv(“Data/ASCII/L0509WellData.csv", header**=**0)

df.describe()

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

**15- Produce a histogram of the gamma curve**

mean = df['GR'].mean()

p5 = df['GR'].quantile(0.05)

p95 = df['GR'].quantile(0.95)

plt.figure(figsize=(6,4), dpi=300)

df['GR'].plot(kind='hist', bins=30, color='red', alpha=0.5, edgecolor='black')

plt.xlabel('Gamma Ray', fontsize=14)

plt.ylabel('Frequency', fontsize=14)

plt.xlim(0,200)

plt.axvline(mean, color='blue', label='mean')

plt.axvline(p5, color='green', label='5th Percentile')

plt.axvline(p95, color='purple', label='95th Percentile')

plt.legend()

plt.show()

**16- Plot histograms for all curves**

cols\_to\_plot = list(df)

cols\_to\_plot.remove("DEPTH")

rows = 3

cols = 2

fig=plt.figure(figsize=(10,10))

for i, feature in enumerate(cols\_to\_plot):

ax=fig.add\_subplot(rows,cols,i+1)

df[feature].hist(bins=20,ax=ax,facecolor='green', alpha=0.6)

ax.set\_title(feature+" Distribution")

ax.set\_axisbelow(True)

ax.grid(color='whitesmoke')

plt.tight\_layout()

plt.show()

**17- Plot a Density Porosity cross-plot**

df.plot(kind="scatter", x="NPHI", y="RHOB", c="GR",

colormap="YlOrRd\_r", ylim=(3,2))

**18- Normalize data to use in modeling - resistivity is usually not normalized. Load data from well**

data = pd.read\_csv(‘Data/ASCII/VolveWells.csv’)

data.head()

data.describe()

**19- How many wells are there in the data dataframe**

data[‘WELL'].unique()

**20- Describe the data for each well**

wells = data.groupby('WELL')

wells.head()

wells.min()

wells.max()

wells.describe()

**21- Plot a distribution of GR for each well**

fig, ax = plt.subplots(figsize=(8,6))

for label, df in wells:

df.GR.plot(kind ='kde', ax=ax, label=label)

plt.xlim(0, 200)

plt.grid(True)

plt.legend()

plt.savefig('before\_normalisation.png', dpi=300)

**22- Prepare data for normalization**

It is possible that datasets can contain erroneous values which may affect the minimum and the maximum values within a curve. Therefore, some interpreters prefer to base their normalization parameters on percentiles.

In this example, I have used the 5th and 95th percentiles.

The first step is to calculate the percentile (or quantile as pandas refers to it) by grouping the data by wells and then applying the .quantile method to a specific column. In this case, GR. The quantile function takes in a decimal value, so a value of 0.05 is equivalent to the 5th percentile and 0.95 is equivalent to the 95th percentile.

5 percentile

gr\_percentile\_05 = data.groupby('WELL')['GR'].quantile(0.05)

print(gr\_percentile\_05)

data['05\_PERC'] = data['WELL'].map(gr\_percentile\_05)

data.describe()

95 percentile

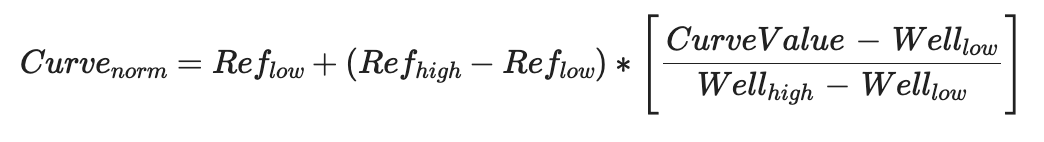
gr\_percentile\_95 = data.groupby(‘WELL')['GR'].quantile(0.95)

gr\_percentile\_95

data['95\_PERC'] = data['WELL'].map(gr\_percentile\_95)

data.describe()

**23- Compute a normalization curve using the formula:**



def normalise(curve, ref\_low, ref\_high, well\_low, well\_high):

return ref\_low + ((ref\_high - ref\_low) \* ((curve - well\_low) / (well\_high - well\_low)))

key\_well\_low = 25

key\_well\_high = 110

data['GR\_NORM'] = data.apply(lambda x: normalise(x['GR'], key\_well\_low, key\_well\_high, x['05\_PERC'], x['95\_PERC']), axis=1)

**24- Plot the normalized gamma curve**

fig, ax = plt.subplots(figsize=(8,6))

for label, df in wells:

df.GR\_NORM.plot(kind ='kde', ax=ax, label=label)

plt.xlim(0, 200)

plt.grid(True)

plt.legend()

plt.savefig('after\_normalisation.png', dpi=300)