# Data modeling predictions

**1- Install the dlisio library in the petrophysics dev environment**

pip install dlisio

**2- Import the dlisio library**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import dlisio

**3- Load the first log in the DLIS file NLOG\_LIS\_LAS\_7857\_FMS\_DSI\_MAIN\_LOG.DLIS**

f, \*tail = dlis.load(‘Data\\DLIS\\NLOG\_LIS\_LAS\_7857\_FMS\_DSI\_MAIN\_LOG.DLIS’)

**4- How many logs are there in this DLIS file?**

print(f)

print(tail)

**5- Show the contents of the log**

f.describe()

**6- Show the number of log headers in the file**

origin, \*origin\_tail = f.origins

print(len(origin\_tail))

Example of unpacking:

a=[1,2,3,4]

c,\*b=a

b

[2, 3, 4]

**7- Show the number of log headers in the file**

origin**.**describe()

**8- Show the frames in the log**

for frame in f.frames:

# Search through the channels for the index and obtain the units

for channel in frame.channels:

if channel.name == frame.index:

depth\_units = channel.units

print(f'Frame Name: \t\t {frame.name}')

print(f'Index Type: \t\t {frame.index\_type}')

print(f'Depth Interval: \t {frame.index\_min} - {frame.index\_max} {depth\_units}')

print(f'Depth Spacing: \t\t {frame.spacing} {depth\_units}')

print(f'Direction: \t\t {frame.direction}')

print(f'Num of Channels: \t {len(frame.channels)}')

print(f'Channel Names: \t\t {str(frame.channels)}')

print(‘\n\n')

**9- Show the tools ran in this log**

def summary\_dataframe(object, \*\*kwargs):

# Create an empty dataframe

df = pd.DataFrame()

# Iterate over each of the keyword arguments

for i, (key, value) in enumerate(kwargs.items()):

list\_of\_values = []

# Iterate over each parameter and get the relevant key

for item in object:

# Account for any missing values.

try:

x = getattr(item, key)

list\_of\_values.append(x)

except:

list\_of\_values.append('')

continue

# Add a new column to our data frame

df[value]=list\_of\_values

# Sort the dataframe by column 1 and return it

return df.sort\_values(df.columns[0])

tools = summary\_dataframe(f.tools, name='Name', description='Description')

tools

**10- Show the parameters for the sonic tool**

dsstb = f.object('TOOL', 'DSSTB')

dsstb\_params = summary\_dataframe(dsstb.parameters, name='Name', long\_name='Long Name', values='Values')

dsstb\_params

**11- Load and plot sonic data for a frame**

frame1 = f.object('FRAME','60B')

dtc = f.object('CHANNEL', 'PWF4')

# Print out the properties of the channel/curve

print(f'Name: \t\t{dtc.name}')

print(f'Long Name: \t{dtc.long\_name}')

print(f'Units: \t\t{dtc.units}')

print(f'Dimension: \t{dtc.dimension}') #if >1, then data is an array

curves = frame1.curves()

depth = curves['TDEP'] \* 0.00254

dtco = curves['DTCO']

dtsm = curves['DTSM']

stc\_mono = curves['SPR4']

wf\_mono = curves['PWF4']

print(f'{depth.min()} - {depth.max()}')

plt.plot(depth, dtco)

plt.plot(depth, dtsm)

plt.ylim(40, 240)

**12- Plot sonic semblance**

fig, axes = plt.subplots(figsize=(7,10))

ax1 = plt.subplot2grid((1, 1), (0,0))

ax2 = ax1.twiny()

ax1.imshow(stc\_mono, interpolation='bilinear', aspect='auto',

cmap=plt.cm.jet, vmin=0, vmax=100,

extent=[40, 240, depth.min(), depth.max()])

#Setting up the display depth range. Note that the

# depths need to be deepest first and shallowest second

ax1.set\_ylim(depth.max(), depth.min())

ax2.plot(dtco, depth, color='black')

ax2.plot(dtsm, depth, color='brown')

ax2.set\_xlim(40, 240)

plt.show()

**13- Load the file Data\\ASCII\\Xeek\_train\_subset\_clean.csv**

df = pd.read\_csv('Data\\ASCII\\Xeek\_train\_subset\_clean.csv').tail(1000)

df.head()

**14- Spilt data in training and test**

train\_df, test\_df = train\_test\_split(df, test\_size=0.3, random\_state=42)

train\_df = train\_df.reset\_index(drop=True)

test\_df = test\_df.reset\_index(drop=True)

**15- Select only the gamma, density neutron data**

X\_train = train\_df[['DEPTH\_MD','CALI', 'RDEP', 'RHOB', 'GR', 'NPHI', 'PEF']].fillna(0)

y\_train = train\_df['DTC'].fillna(0)

X\_train.head()

y\_train.head()

X\_test = test\_df[['DEPTH\_MD','CALI', 'RDEP', 'RHOB', 'GR', 'NPHI', 'PEF']].fillna(0)

y\_test = test\_df['DTC'].fillna(0)

y\_test.head()

**16- Calculate a predicted DT with a linear regression model**

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

y\_pred\_lin = lin\_reg.predict(X\_test)

lin\_mse = mean\_squared\_error(y\_test, y\_pred\_lin)

print(f"Linear Regression MSE: {lin\_mse}”)

**17- Plot real vs predicted DT**

plt.figure(figsize=(6, 30))

plt.plot(y\_test, y\_test.index, label='Actual DTC', color='blue')

plt.plot(y\_pred\_lin, y\_test.index, label='Predicted DTC', color='red', linestyle='--')

plt.gca().invert\_yaxis() # Invert the depth axis

plt.xlabel('DTC')

plt.ylabel('Depth (MD)')

plt.title('Oilfield Log: Predicted vs Actual DTC')

plt.legend()

plt.grid(True)

plt.show()

**18- Calculate a predicted DT with a random forrest model**

rf = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

y\_pred\_rf = rf.predict(X\_test)

rf\_mse = mean\_squared\_error(y\_test, y\_pred\_rf)

print(f"Random Forest MSE: {rf\_mse}")

**19- Plot real vs predicted DT**

plt.figure(figsize=(6, 30))

plt.plot(y\_test, y\_test.index, label='Actual DTC', color='blue')

plt.plot(y\_pred\_lin, y\_test.index, label='Linear Predicted DTC', color='red', linestyle='--')

plt.plot(y\_pred\_rf, y\_test.index, label='Random Forrest Predicted DTC', color='green', linestyle='--')

plt.gca().invert\_yaxis() # Invert the depth axis

plt.xlabel('DTC')

plt.ylabel('Depth (MD)')

plt.title('Oilfield Log: Predicted vs Actual DTC')

plt.legend()

plt.grid(True)

plt.show()

**20- Calculate a predicted DT with a gradient boosting model**

gb = GradientBoostingRegressor(n\_estimators=100, random\_state=42)

gb.fit(X\_train, y\_train)

y\_pred\_gb = gb.predict(X\_test)

gb\_mse = mean\_squared\_error(y\_test, y\_pred\_gb)

print(f"Gradient Boosting MSE: {gb\_mse}")

**21- Plot real vs predicted DT**

plt.figure(figsize=(6, 30))

plt.plot(y\_test, y\_test.index, label='Actual DTC', color='blue')

plt.plot(y\_pred\_lin, y\_test.index, label='Linear Predicted DTC', color='red', linestyle='--')

plt.plot(y\_pred\_rf, y\_test.index, label='Random Forrest Predicted DTC', color='green', linestyle='--')

plt.plot(y\_pred\_gb, y\_test.index, label='Gradient Boosting Predicted DTC', color='lime', linestyle='--')

plt.gca().invert\_yaxis() # Invert the depth axis

plt.xlabel('DTC')

plt.ylabel('Depth (MD)')

plt.title('Oilfield Log: Predicted vs Actual DTC')

plt.legend()

plt.grid(True)

plt.show()

**22- Plot curves against depth**

pred\_df=test\_df

pred\_df.head()

pred\_df['Pred DTC lin']=y\_pred\_lin

pred\_df['Pred DTC rf']=y\_pred\_rf

pred\_df['Pred DTC gb']=y\_pred\_gb

pred\_df.head()

pred\_df = pred\_df.set\_index('DEPTH\_MD',drop=True)

pred\_df.head()

pred\_df = pred\_df.sort\_index(ascending=True)

pred\_df.head()

pred\_df = pred\_df.reset\_index(drop=False)

pred\_df.head()

import matplotlib.pyplot as plt

# Set up the figure and axes

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

# Set up the plot axes

ax1 = plt.subplot2grid((1, 4), (0, 0), rowspan=1, colspan=1) # Gamma Ray track

ax2 = plt.subplot2grid((1, 4), (0, 1), rowspan=1, colspan=1) # Density track

ax3 = plt.subplot2grid((1, 4), (0, 2), rowspan=1, colspan=1) # Neutron track

ax4 = plt.subplot2grid((1, 4), (0, 3), rowspan=1, colspan=1) # Sonic track for DTC and predicted curves

# Gamma Ray track

ax1.plot(pred\_df["GR"],pred\_df['DEPTH\_MD'], color="green") # Ensure using index for depth

ax1.set\_xlabel("Gamma Ray")

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.set\_ylim(pred\_df['DEPTH\_MD'].max(), pred\_df['DEPTH\_MD'].min()) # Invert depth axis

ax1.grid(which='major', color='lightgrey', linestyle='-')

# Density track

ax2.plot(pred\_df["RHOB"],pred\_df['DEPTH\_MD'], color="red") # Ensure using index for depth

ax2.set\_xlabel("Density (RHOB)")

ax2.set\_xlim(1.95, 2.95)

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

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

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

ax2.set\_ylim(pred\_df['DEPTH\_MD'].max(), pred\_df['DEPTH\_MD'].min()) # Invert depth axis

ax2.grid(which='major', color='lightgrey', linestyle='-')

# Neutron track

ax3.plot(pred\_df["NPHI"],pred\_df['DEPTH\_MD'], color="blue") # Ensure using index for depth

ax3.set\_xlabel("Neutron (NPHI)")

ax3.set\_xlim(0.45, -0.15)

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

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

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

ax3.set\_ylim(pred\_df['DEPTH\_MD'].max(), pred\_df['DEPTH\_MD'].min()) # Invert depth axis

ax3.grid(which='major', color='lightgrey', linestyle='-')

# Sonic track (for DTC and predicted DTC curves)

ax4.plot(pred\_df["DTC"],pred\_df['DEPTH\_MD'], label='DTC', color="lime") # Ensure using index for depth

ax4.plot(pred\_df["Pred DTC lin"],pred\_df['DEPTH\_MD'], label='Pred DTC lin', color="orange", linestyle='--')

ax4.plot(pred\_df["Pred DTC rf"],pred\_df['DEPTH\_MD'], label='Pred DTC rf', color="cyan", linestyle='--')

ax4.plot(pred\_df["Pred DTC gb"],pred\_df['DEPTH\_MD'], label='Pred DTC gb', color="magenta", linestyle='--')

ax4.set\_xlabel("Sonic (DTC)")

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

ax4.set\_xlim(140, 40) # Reverse scale for DTC

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

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

ax4.set\_ylim(pred\_df['DEPTH\_MD'].max(), pred\_df['DEPTH\_MD'].min()) # Invert depth axis

ax4.grid(which='major', color='lightgrey', linestyle='-')

ax4.legend()

# Adjust layout

plt.tight\_layout()

plt.show()