# Modeling

**1- Load libraries**

import lasio

import pandas as pd

import math

import matplotlib.pyplot as plt

**2- Load logs used in last session**

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=False)

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)

df.head()

**3- Clean and prepare dataframe for MSE calculations**

len(df)

df2 = df.loc[:, ['TFBA', 'RPMTOTAL','FTAA','FRPI','HDEP']]

df2 = df2.dropna()

df2 = df2.loc[(df2 != 0).all(axis=1)]

len(df2)

df2.head()

**4- Compute MSE**

BS = 17.5

Bit\_Area=(BS\*\*2)\*math.pi/4

factor = 0.35

def calculate\_MSE(row):

return (factor / Bit\_Area) \* (row['TFBA'] + ( 120 \* math.pi \* row['RPMTOTAL'] \* row['FTAA']) / (row['FRPI'] \* 3.28084))

# Apply the function to each row of the dataframe

df2['MSE'] = df2.apply(calculate\_MSE, axis=1)

df2 = df2.dropna()

print(df2)

**5- Plot drilling parameters**

df2=df2.reset\_index()

ax1=plt.subplot2grid((1,5),(0,0),rowspan=1,colspan=1)

ax1.plot("TFBA","HDEP",data=df2)

ax1.invert\_yaxis()

ax1.set\_title('TFBA - WOB')

ax1.grid()

ax2=plt.subplot2grid((1,5),(0,1),rowspan=1,colspan=1)

ax2.plot("RPMTOTAL","HDEP",data=df2)

ax2.invert\_yaxis()

ax2.set\_title('RPMTOTAL - RPM')

ax2.grid()

ax3=plt.subplot2grid((1,5),(0,2),rowspan=1,colspan=1)

ax3.plot("FTAA","HDEP",data=df2)

ax3.invert\_yaxis()

ax3.set\_title('FTAA - Torque')

ax3.grid()

ax4=plt.subplot2grid((1,5),(0,3),rowspan=1,colspan=1)

ax4.plot("FRPI","HDEP",data=df2)

ax4.invert\_yaxis()

ax4.set\_title('FRPI- ROP')

ax4.grid()

ax5=plt.subplot2grid((1,5),(0,4),rowspan=1,colspan=1)

ax5.plot("MSE","HDEP",data=df2, color='red')

ax5.invert\_yaxis()

ax5.set\_title('MSE')

ax5.grid()

**6- Determine if there is a relationship between the parameters**

ax5 = plt.subplot2grid((1,1),(0,0),rowspan=1,colspan=1)

ax5.set\_ylim([0, 50])

ax5.set\_xlim([0, 150])

ax5.scatter(df2['FRPI'], df2['MSE'])

ax5.set\_title('Scatter plot of MSE and FRPI-ROP')

ax5.set\_xlabel('FRPI-ROP')

ax5.set\_ylabel('MSE')

ax5.grid()

ax5 = plt.subplot2grid((1,1),(0,0),rowspan=1,colspan=1)

ax5.scatter(df2['FTAA'], df2['MSE'])

ax5.set\_title('Scatter plot of MSE and FTAA-Torque')

ax5.set\_xlabel('FRPI-ROP')

ax5.set\_ylabel('MSE')

ax5.grid()

ax5 = plt.subplot2grid((1,1),(0,0),rowspan=1,colspan=1)

ax5.scatter(df2['FTAA'], df2['FRPI'])

ax5.set\_title('Scatter plot of FRPI and FTAA-Torque')

ax5.set\_xlabel('FTAA-Torque')

ax5.set\_ylabel('FRPI-ROP')

ax5.grid()

ax5 = plt.subplot2grid((1,1),(0,0),rowspan=1,colspan=1)

ax5.scatter(df2['FTAA'], df2['RPMTOTAL'])

ax5.set\_title('Scatter plot of FRPI and RPMTOTAL')

ax5.set\_xlabel('RPMTOTAL')

ax5.set\_ylabel('FRPI-ROP')

ax5.grid()

ax5 = plt.subplot2grid((1,1),(0,0),rowspan=1,colspan=1)

ax5.scatter(df2['FTAA'], df2['TFBA'])

ax5.set\_title('Scatter plot of FRPI and TFBA')

ax5.set\_xlabel('TFBA')

ax5.set\_ylabel('FRPI-ROP')

ax5.grid()

**7- Code to show a perfect linear regression**

from sklearn.linear\_model import LinearRegression

# Creating the dataset

data = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [2, 3, 4, 5, 6]})

# Creating the model

model = LinearRegression()

# Fitting the model

model.fit(data[['x']], data['y'])

# Getting the R-squared value

r\_sq = model.score(data[['x']], data['y'])

print('Coefficient of determination:', r\_sq)

# Predicting the response

y\_pred = model.predict(data[['x']])

print('Predicted response:', y\_pred, sep='\n')

# Plotting the actual versus predicted results

x = data['x']

y = data['y']

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred, color='blue', linewidth=3)

plt.title('Actual vs Predicted')

plt.xlabel('X')

plt.ylabel('Y')

plt.show()

**8- Code showing a non-perfect linear regression**

# Creating the dataset

data = pd.DataFrame({'x': [1, 1.5, 2, 3, 3.5, 4, 4.9, 5.5, 5.7, 6.3, 7, 8], 'y': [2, 1.7, 3, 3.5, 4, 5, 5, 5.6, 6.4, 6.1, 7, 8]})

# Creating the model

model = LinearRegression()

# Fitting the model

model.fit(data[['x']], data['y'])

# Getting the R-squared value

r\_sq = model.score(data[['x']], data['y'])

print('Coefficient of determination:', r\_sq)

# Predicting the response

y\_pred = model.predict(data[['x']])

print('Predicted response:', y\_pred, sep='\n')

# Plotting the actual versus predicted results

x = data['x']

y = data['y']

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred, color='blue', linewidth=3)

plt.title('Actual vs Predicted')

plt.xlabel('X')

plt.ylabel('Y')

# Plotting the residuals

residuals = y - y\_pred

plt.figure()

plt.scatter(x, residuals, color='red')

plt.axhline(y=0, color='black', linestyle='--')

plt.title('Residual Plot')

plt.xlabel('X')

plt.ylabel('Residuals')

plt.show()

**9- Show and evaluate a regression between RPM and ROP**

# Creating the dataset

data = df2

# Creating the model

model = LinearRegression()

# Fitting the model

model.fit(data[['RPM']], data['ROP'])

# Getting the R-squared value

r\_sq = model.score(data[['RPM']], data['ROP'])

print('Coefficient of determination:', r\_sq)

# Predicting the response

y\_pred = model.predict(data[['RPM']])

print('Predicted response:', y\_pred, sep='\n')

# Plotting the actual versus predicted results

x = data['RPM']

y = data['ROP']

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred, color='blue', linewidth=3)

plt.title('Actual vs Predicted')

plt.xlabel('WOB')

plt.ylabel('ROP')

# Plotting the residuals

residuals = y - y\_pred

plt.figure()

plt.scatter(x, residuals, color='red')

plt.axhline(y=0, color='black', linestyle='--')

plt.title('Residual Plot')

plt.xlabel('WOB')

plt.ylabel('Residuals')

plt.show()

**10- Show and evaluate a regression between TORQUE and MSE**

# Creating the dataset

data = df2

# Creating the model

model = LinearRegression()

# Fitting the model

model.fit(data[['TORQUE']], data['MSE'])

# Getting the R-squared value

r\_sq = model.score(data[['TORQUE']], data['MSE'])

print('Coefficient of determination:', r\_sq)

# Predicting the response

y\_pred = model.predict(data[['TORQUE']])

print('Predicted response:', y\_pred, sep='\n')

# Plotting the actual versus predicted results

x = data['TORQUE']

y = data['MSE']

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred, color='blue', linewidth=3)

plt.title('Actual vs Predicted')

plt.xlabel('TORQUE')

plt.ylabel('MSE')

# Plotting the residuals

residuals = y - y\_pred

plt.figure()

plt.scatter(x, residuals, color='red')

plt.axhline(y=0, color='black', linestyle='--')

plt.title('Residual Plot')

plt.xlabel('TORQUE')

plt.ylabel('Residuals')

plt.show()

**11- Define a function to evaluate regressions**

def linear\_regression(data: pd.DataFrame, torque\_col: str, mse\_col: str) -> None:

# Creating the model

model = LinearRegression()

# Fitting the model

model.fit(data[[torque\_col]], data[mse\_col])

# Getting the R-squared value

r\_sq = model.score(data[[torque\_col]], data[mse\_col])

print('Coefficient of determination:', r\_sq)

# Predicting the response

y\_pred = model.predict(data[[torque\_col]])

print('Predicted response:', y\_pred, sep='\n')

# Plotting the actual versus predicted results

x = data[torque\_col]

y = data[mse\_col]

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred, color='blue', linewidth=3)

plt.title('Actual vs Predicted')

plt.xlabel(torque\_col)

plt.ylabel(mse\_col)

# Plotting the residuals

residuals = y - y\_pred

plt.figure()

plt.scatter(x, residuals, color='red')

plt.axhline(y=0, color='black', linestyle='--')

plt.title('Residual Plot')

plt.xlabel(torque\_col)

plt.ylabel('Residuals')

plt.show()

**12- Use the function to show the regression between torque and MSE**

linear\_regression(df2, 'TORQUE', ‘MSE’)

**13- Use the function to show the regression between WOB y ROP**

linear\_regression(df2, 'WOB', 'ROP')

**14- Code to evaluate multiple regressions models for ROP, select the best model (do not use the last 3000 lines)**

df3=df2[:-3000]

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.preprocessing import PolynomialFeatures

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.metrics import mean\_squared\_error, r2\_score

# Split the data into training and testing sets

X = df3[['WOB', 'RPM', 'TORQUE', 'HDEP']]

y = df3['ROP']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the models

models = [LinearRegression(), Ridge(alpha=1.0), Ridge(alpha=10.0), Ridge(alpha=100.0), DecisionTreeRegressor(), RandomForestRegressor(), SVR()]

model\_names = ['Linear Regression', 'SVM','Ridge Regression (alpha=10.0)', 'Ridge Regression (alpha=100.0)', 'Decision Tree', 'Random Forest']

for model, name in zip(models, model\_names):

if name == 'SVM':

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

else:

poly = PolynomialFeatures(degree=2)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

model.fit(X\_train\_poly, y\_train)

y\_pred = model.predict(X\_test\_poly)

if name == 'SVM':

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'{name} MSE: {mse:.2f}, R2: {r2:.2f}')

# Plot true ROP vs predicted

plt.scatter(y\_test, y\_pred)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=4)

plt.xlabel('True ROP')

plt.ylabel('Predicted ROP')

plt.title(name)

plt.show()

**15- Code to select the best model**

# Split the data into training and testing sets

X = df3[['WOB', 'RPM', 'TORQUE', 'HDEP']]

y = df3['ROP']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the models

models = [LinearRegression(), Ridge(alpha=1.0), Ridge(alpha=10.0), Ridge(alpha=100.0), DecisionTreeRegressor(), RandomForestRegressor(), SVR()]

model\_names = ['Linear Regression', 'SVM','Ridge Regression (alpha=10.0)', 'Ridge Regression (alpha=100.0)', 'Decision Tree', 'Random Forest']

# Select the model with the lowest R2

min\_r2 = float('inf')

best\_model = None

for model, name in zip(models, model\_names):

if name == 'SVM':

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

else:

poly = PolynomialFeatures(degree=2)

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

model.fit(X\_train\_poly, y\_train)

y\_pred = model.predict(X\_test\_poly)

if name == 'SVM':

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'{name} MSE: {mse:.2f}, R2: {r2:.2f}')

if r2 < min\_r2:

min\_r2 = r2

best\_model = model

**16- Plot predicted ROP and the real ROP for the last 3000 data points**

df4=df2[-3000:]

# Predict ROP using values from a dataframe df4

X\_new = df4[['WOB', 'RPM', 'TORQUE', 'HDEP']]

y\_pred = best\_model.predict(X\_new)

# Add the predicted values to df4 as a new column called ROP\_pred

df4['ROP\_pred'] = y\_pred

fig, ax = plt.subplots(1, 1, figsize=(10, 40))

ax.plot("ROP","HDEP",data=df4, color='red')

ax.invert\_yaxis()

ax.set\_title('ROP')

ax.plot("ROP\_pred","HDEP",data=df4, color='green')

ax.invert\_yaxis()

ax.set\_title('ROP\_Pred')

plt.show()

**17- Apply smoothing to the curves**

df4=df4.drop(‘TIME',axis=1)

df4.head()

# define the window size

WS = 10

dfs=pd.DataFrame()

for column in df4.columns:

dfs[column] = df4[column].rolling(window=WS).mean()

fig, ax = plt.subplots(1, 1, figsize=(10, 40))

ax.plot("ROP","HDEP",data=dfs, color='red')

ax.invert\_yaxis()

ax.set\_title('ROP')

ax.plot("ROP\_pred","HDEP",data=dfs, color='green')

ax.invert\_yaxis()

ax.set\_title('ROP\_Pred')

plt.show()