Modelos

1- Cargar las librerías en el notebook

```
import lasio
import pandas as pd
import math
import matplotlib.pyplot as plt
```

2- Cargar los registros utilizados en la ultima clase

```
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.la
s',

'210919_IOA_11_BDC-2-04_TD_2047.las','210920_IOA_12_BDC-2-04_TD_2261.las','2
10921_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- Limpiar y preparar el dataframe para hacer cálculos de MSE

```
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- Calcular el 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- Graficar los parámetros de perforación

```
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- Determinar si hay relaciones entre los parámetros

```
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- Producir código presentando una regression lineal perfecta

```
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- Producir código mostrando una regresión interfecta

```
# 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- Mostrar y evaluar la regression entre RPM y 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- Mostrar y evaluar la regression entre TORQUE y 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- Definir una función para evaluar regresiones

```
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- Usar la función para mostrar la regression entre torque y MSE

linear_regression(df2, 'TORQUE', ‘MSE’)

13- Usar la función para mostrar la regression entre WOB y ROP

linear_regression(df2, 'WOB', 'ROP')

14- Producir código para evaluar multiple modelos de regresiones de ROP y escoger el mas preciso, sin utilizar las últimas 3000 líneas

```
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- Escribir código para seleccionar el mejor modelo

```
# 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- Graficar la ROP predicha y la verdadera para los últimos 3000 valores

```
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- Aplicar smoothing a las curvas y graficar de nuevo

```
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()
```