Modelos IA para Perforación y Workover

Introducción al uso de IA
Día 2

Peter Kowalchuk - Agosto 2025

Repaso

Que aprendimos ayer

Inteligencia Artificial Analítica

- Supervisada
  - Regresiones
  - Modelos Estadísticos
  - Redes Neurales
- No-supervisada
  - Clasificación

Inteligencia Artificial Generativa

- Modelos de Gran Escala (LLM)
- Algoritmos de búsqueda (RAG)
- Flujos de Trabajo y Agentes

Repaso

- Ambientes de desarrollo
- Ambientes Conda
- IDE - Jupyter Notebooks y Google Collab
- Variables en Python - **types**
    - Numéricas - integer and float
    - String
    - Listas
    - Diccionarios
- For Loop
- If

Agenda

Lo que veremos esta semana

Lunes

Introducción

Inteligencia Artificial

Tecnologias

Manejo de Datos

Ambiente de trabajo (IDE)

Martes

IA Supervisada

Regresiones

Modelos Predictivos

Miércoles

IA Supervisada

Modelos Estadísticos

Jueves

IA No-Supervisada

Clasificación

Generativa

Modelos de Gran Escala

Viernes

IA Generativa

Algoritmos de Búsqueda

Flujos de Trabajo

¿Qué es el IA Supervisado?

IA como Machine Learning

El aprendizaje supervisado es una técnica de inteligencia artificial en la que un modelo **aprende** a partir de un conjunto de **datos etiquetado**, es decir, datos que ya incluyen las respuestas correctas.

El objetivo es que el modelo prediga o clasifique nuevos datos con base en lo aprendido.

Ejemplo

Perforación

Predecir la Tasa de Penetración (ROP) durante la perforación, usando datos históricos.

Datos de entrenamiento (etiquetados):

- Entradas (features):
- Peso sobre la barrena (WOB)
- Velocidad de rotación (RPM)
- Caudal de lodo
- Tipo de formación
- Salida (etiqueta):
- **Tasa de Penetración real (ROP)**

El modelo se entrena con estos datos históricos para luego predecir la ROP en tiempo real durante nuevas operaciones, facilitando la optimización de los parámetros operativos.

¿Qué es la regresión?

IA Supervisada

- Es una técnica estadística que permite modelar la relación entre una variable **dependiente (respuesta)** y una o más variables **independientes (predictoras)**
    - Entender cómo cambia la variable objetivo cuando cambian las variables de entrada
    - Predecir valores futuros basados en datos históricos

- Ejemplo: Predecir la **producción de un pozo (respuesta)** a partir de su **profundidad y presión (predictoras)**

¿Por qué usar regresión en perforación?

Identificar patrones y hacer predicciones

- Predecir la Tasa de Penetración (ROP) a partir de parámetros de perforación
- Estimar el Tiempo No Productivo (NPT) a partir de registros de actividad
- Analizar tendencias de costo vs. profundidad en pozos históricos
- Modelar torque y arrastre en distintas formaciones

Regresión en perforación

Variables comunes en modelos de regresión

- Peso sobre la barrena/mecha (WOB)
- Velocidad de rotación (RPM)
- Torque
- Drag
- Caudal/flujo de lodo
- Tipo de formación
- Ángulo del pozo/Inclinación

Optimización de Workover con Regresión

Predecir la recuperación del pozo después de un trabajo

- Incremento de producción
- Recuperación del nivel de fluidos
- Viabilidad económica de un operación de workover
- Tiempo estimado de retorno a producción estable
- Tasa de declinación posterior al trabajo
- Duración óptima del cierre de pozo durante la intervención
- Costo-beneficio comparativo entre métodos de reparación
- Impacto del trabajo sobre la presión del fondo fluyente

Regresión en workover

Variables comunes en modelos de regresión

- Tasa de producción previa al workover (bbl/d o Mcf/d)
- Agua producida (% de corte de agua)
- Relación gas-petróleo (GOR)
- Tasa de declinación antes del trabajo
- Método de intervención (acidificación, limpieza mecánica, cambio de tubing, fracturamiento)
- Duración del workover
- Tiempo total de shut-in antes y después del trabajo
- Volumen de fluido inyectado
- Presión de inyección o bombeo

Ejemplo Práctico

En Python

```
from sklearn.linear_model import LinearRegression


X = df[['WOB', 'RPM', 'FlowRate']]

y = df['ROP']


model = LinearRegression()

model.fit(X, y)
```

Tipos de Regresiones

Modelos disponibles

Regresión Lineal Simple

Modela la relación entre una variable dependiente y una variable independiente mediante una línea recta.

$$y = \beta_0 + \beta_1 x + \varepsilon$$

Regresión Lineal Múltiple

Extiende la regresión simple utilizando múltiples variables independientes para predecir la variable dependiente.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Regresión Polinomial

Utiliza polinomios para modelar relaciones no lineales entre variables, permitiendo curvas en lugar de líneas rectas.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \varepsilon$$

Regresión Logística

Utilizada para problemas de clasificación binaria, predice la probabilidad de que ocurra un evento.

$$p = 1 / (1 + e^{-(\beta_0 + \beta_1 x)})$$

Regresión Ridge

Añade regularización L2 para reducir el sobreajuste y manejar la multicolinealidad entre variables.

$$\text{Min: } ||y - X\beta||^2 + \lambda||\beta||^2$$

Regresión Lasso

Utiliza regularización L1 que puede reducir algunos coeficientes a cero, realizando selección automática de variables.

$$\text{Min: } ||y - X\beta||^2 + \lambda||\beta||_1$$

Regresión Elastic Net

Combina las regularizaciones L1 y L2, aprovechando las ventajas tanto de Ridge como de Lasso.

$$\text{Min: } ||y - X\beta||^2 + \lambda_1||\beta||_1 + \lambda_2||\beta||^2$$

Regresión de Poisson

Diseñada para modelar datos de conteo, donde la variable dependiente representa el número de eventos.

$$\log(\mu) = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

Regresión Robusta

Menos sensible a valores atípicos (outliers) que la regresión lineal tradicional, proporcionando estimaciones más estables.

$$\text{Min: } \Sigma\ \rho(r_i) \text{ donde } \rho \text{ es función robusta}$$

¿Qué es la regresión lineal?

Nuestro primer modelo

Es un modelo matemático que busca encontrar la mejor línea recta que describe la relación entre una variable dependiente (Y) y una o más variables independientes (X).

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- Y: Variable objetivo (por ejemplo, ROP)
- X: Variable predictora (por ejemplo, WOB)
- β_0: Intercepto
- β_1: Pendiente (cuánto cambia Y por unidad de cambio en X)
- ε: Error o residuo

¿Qué son los Residuales?

Como sabemos si nuestra modelo es bueno

Los residuales (o errores) son la diferencia entre el valor real y el valor predicho por el modelo.

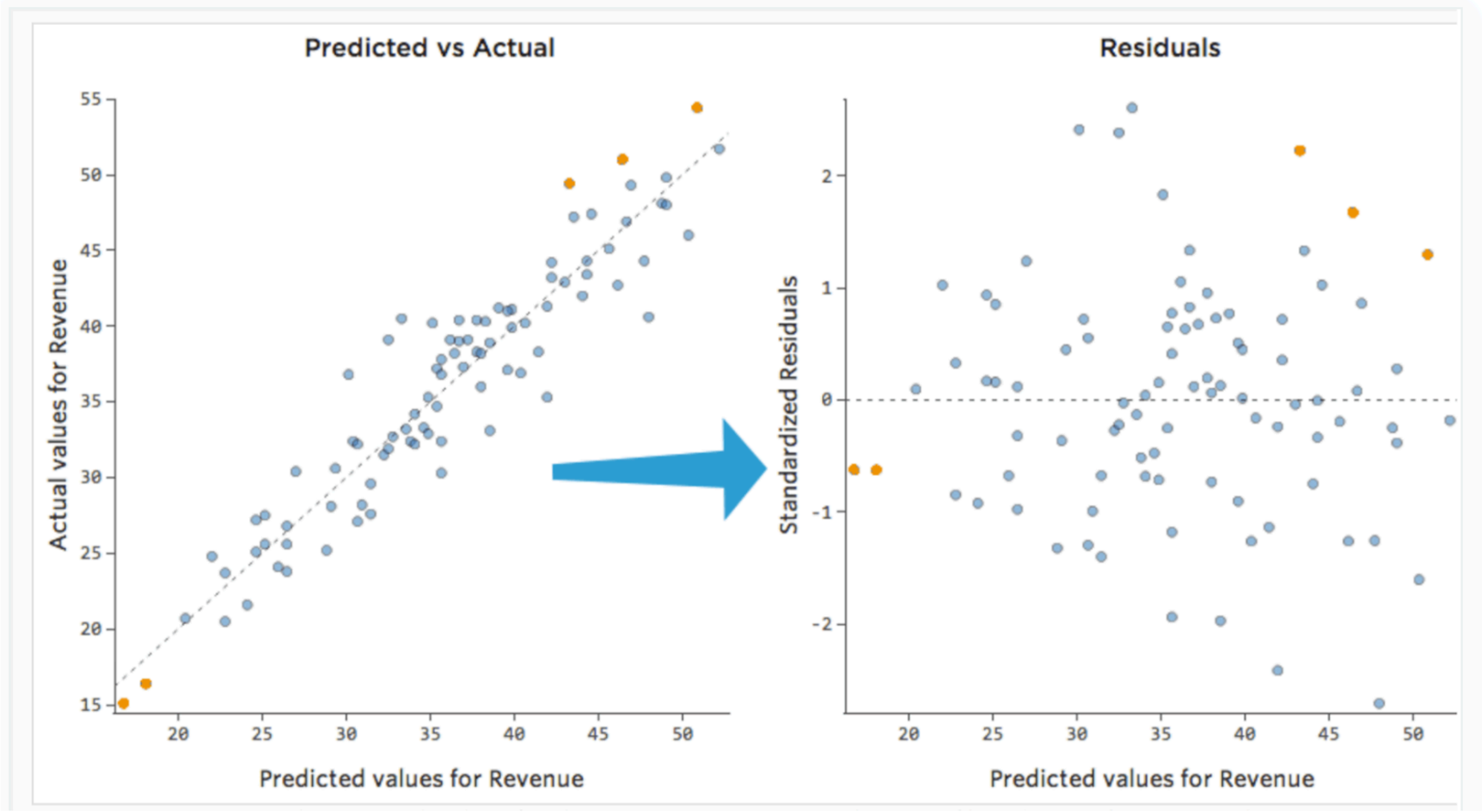
$$\text{Residuo} = Y_{\text{real}} - Y_{\text{predicho}}$$

Un buen modelo tiene residuales pequeños y distribuidos aleatoriamente.

Se pueden gráficas para evaluar el comportamiento del modelo.

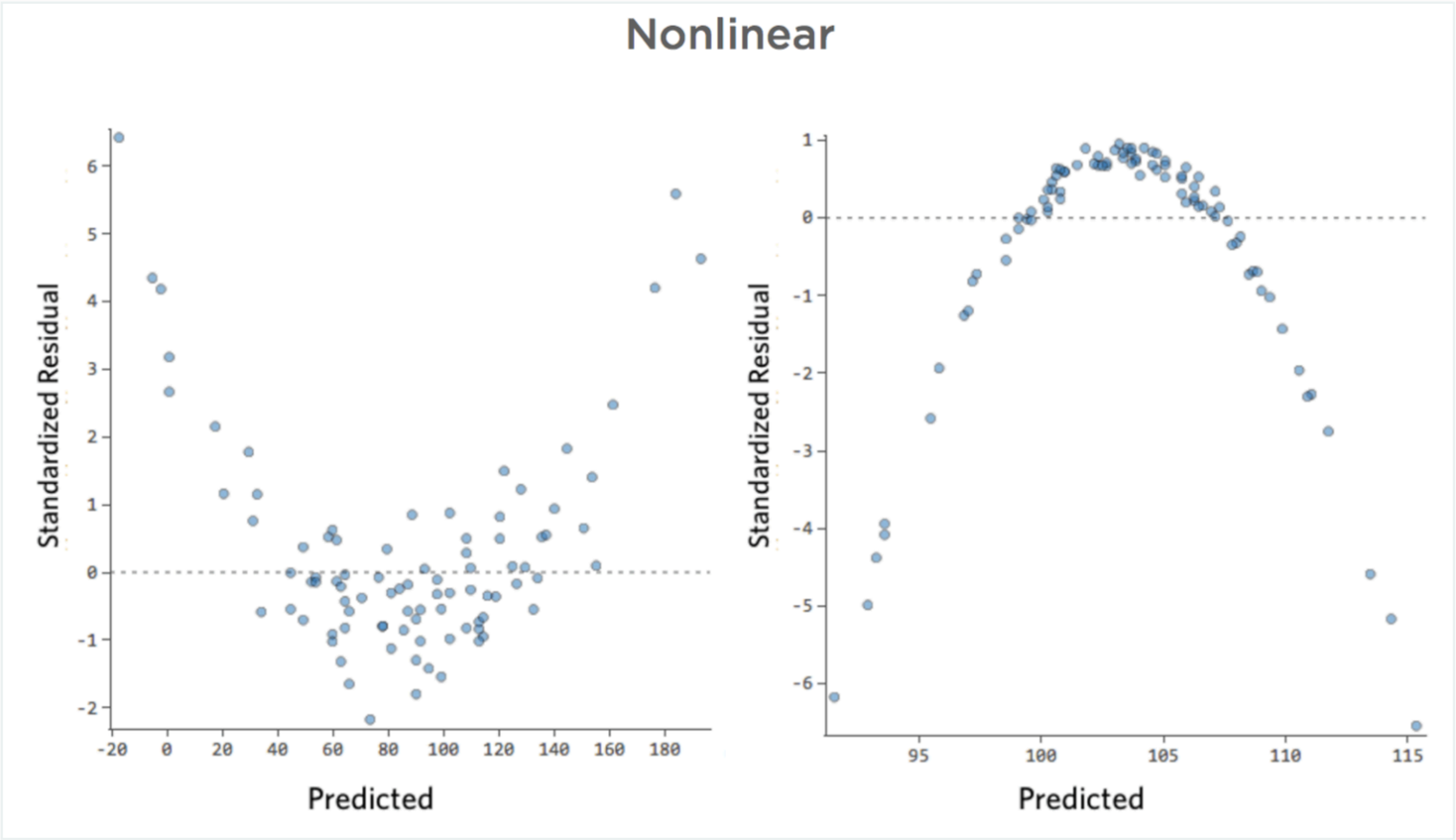
¿Qué son los Residuales?

Residuales demuestran un buen modelo



¿Qué son los Residuales?

Residuales demuestran un mal modelo



Evaluar si una Regresión es Buena

Utilizando indicadores de calidad

- **Coeficiente de determinación (R^2)**
    - Valor entre 0 y 1 que indica qué porcentaje de la variabilidad de Y es explicado por X.
    - Ejemplo: $R^2 = 0.85 \rightarrow$ el 85% de la variación en Y está explicada por el modelo.
- Análisis de residuales
    - Gráficar residuales vs. valores predichos.
        - Patrón aleatorio: modelo adecuado
        - Forma de U o patrón: falta de ajuste o variable omitida
- **Error Cuadrático Medio (MSE / RMSE)**
    - Cuantifica el promedio de los errores al cuadrado.
    - Cuanto más bajo, mejor el modelo.
- Pruebas estadísticas (p-valores, F-test)
    - Evalúan si los coeficientes del modelo son significativos.

Predecir Tasa de Penetración (ROP)

Ejemplo Aplicado en Perforación

Objetivo: Predecir Tasa de Penetración (ROP)

Variables:

- X: WOB, RPM, Caudal de lodo
- Y: ROP

Resultados:

- $R^2 = 0.88$ → muy buen ajuste
- Residuales aleatorios → sin sesgo
- RMSE bajo → predicciones precisas

Modelo útil para optimizar parámetros de perforación en tiempo real

Manejo de datos

Conceptos requeridos para implementar modelos

- Pandas
- Cargar datos
  - Data files CSV, LAS y WITSML
- Fusion/concatenación de datos
- Agrupación de datos

Dos nuevas estructuras de control

En python, pero también en otros lenguajes

- If… else… elif
- While loop

If statement

If… else…

```
ROP = 35  # Current rate of penetration in meters/hour
threshold = 30  # Threshold for ROP

if ROP > threshold:
    print("ROP exceeds safe limit. Adjust drilling parameters.")
else:
    print("ROP is within safe limits.")
```

If statement

If… elif… else…

```
ROP = 35  # Current rate of penetration in meters/hour
threshold_high = 40  # Upper threshold for ROP
threshold_low = 30   # Lower threshold for ROP

if ROP > threshold_high:
    print("ROP is too high! Adjust drilling parameters immediately.")
elif threshold_low <= ROP <= threshold_high:
    print("ROP is within the optimal range.")
else:
    print("ROP is too low. Increase drilling speed.")
```

While loop

```python
import pandas as pd

# Sample gamma ray values (measured in API units)
gamma_ray_values = [85, 90, 95, 105, 110, 108, 102, 99, 95, 90]

# Convert the list into a Pandas Series for better handling
gamma_ray_series = pd.Series(gamma_ray_values)

# Initialize variables
index = 0
above_100 = False

# Loop through the gamma ray series
while index < len(gamma_ray_series):
    value = gamma_ray_series[index]

    if not above_100 and value > 100:
        print(f"Gamma ray went above 100 at index {index}, value: {value}")
        above_100 = True
    elif above_100 and value < 100:
        print(f"Gamma ray dropped below 100 at index {index}, value: {value}")
        above_100 = False

    index += 1
```

Un nuevo type

Classes in Python

```python
import pandas as pd

class Well:
    def __init__(self, name, location, TD, reservoir):
        self.name = name                # Well name
        self.location = location        # Well location
        self.TD = TD                    # Total depth of the well
        self.reservoir = reservoir      # Reservoir type: oil or gas
        self.data = {}                  # Dictionary to store runs and drilling data

    # Method to write drilling data for a specific run
    def write_data(self, run, depth, WOB, RPM, TRQ, Gamma):
        # Create a DataFrame to store the run data
        run_data = pd.DataFrame({
            'Depth (m)': depth,
            'WOB (kN)': WOB,
            'RPM': RPM,
            'Torque (Nm)': TRQ,
            'Gamma Ray (API)': Gamma
        })
        # Store the run data in the dictionary using the run number as the key
        self.data[run] = run_data
```

```python
    # Method to read drilling data for a specific run
    def read_data(self, run):
        # Check if the run data exists
        if run in self.data:
            print(f"Data for run {run}:\n")
            print(self.data[run])
        else:
            print(f"No data available for run {run}")

    # Method to display well information
    def display_info(self):
        print(f"Well Name: {self.name}")
        print(f"Location: {self.location}")
        print(f"Total Depth (TD): {self.TD} meters")
        print(f"Reservoir Type: {self.reservoir}")
```

Clases y Objetos

Programación Orientada a Objetos

¿Qué es una Clase en Python?

Clase: Una descripción del plan para crear objetos (instancias). Define los atributos (datos) y métodos (funciones) que tendrán los objetos.

Atributos: Variables que almacenan información sobre el objeto (por ejemplo, nombre, ubicación, profundidad en un pozo).

Métodos: Funciones que definen el comportamiento o acciones del objeto (por ejemplo, calcular, actualizar o recuperar datos).

Un nuevo type

Classes en Python

```
well_1 = Well(name="Well A", location="Gulf of Mexico", TD=3500, reservoir="oil")

# Sample drilling data for run 100 (arrays for depth, WOB, RPM, TRQ, and Gamma Ray)
depth_100 = [100, 200, 300, 400, 500]
WOB_100 = [10, 12, 14, 13, 15]  # in kN
RPM_100 = [120, 130, 125, 135, 140]  # in rotations per minute
TRQ_100 = [1500, 1600, 1550, 1650, 1700]  # in Newton meters
Gamma_100 = [80, 85, 90, 110, 95]  # in API units

# Write the drilling data for run 100
well_1.write_data(run=100, depth=depth_100, WOB=WOB_100, RPM=RPM_100, TRQ=TRQ_100, Gamma=Gamma_100)

# Sample drilling data for run 200
depth_200 = [600, 700, 800, 900, 1000]
WOB_200 = [16, 17, 18, 17, 19]
RPM_200 = [150, 155, 160, 158, 162]
TRQ_200 = [1750, 1800, 1850, 1820, 1900]
Gamma_200 = [100, 110, 120, 115, 130]

# Write the drilling data for run 200
well_1.write_data(run=200, depth=depth_200, WOB=WOB_200, RPM=RPM_200, TRQ=TRQ_200, Gamma=Gamma_200)

# Display well information
well_1.display_info()

# Read and display the data for run 100
well_1.read_data(run=100)

# Read and display the data for run 200
well_1.read_data(run=200)
```

Un nuevo type

Como determinar el type

```
type(object)
```

```
x = 42
print(type(x))  # Output: <class 'int'>

df = pd.DataFrame()  # Assuming pandas is imported
print(type(df))  # Output: <class 'pandas.core.frame.DataFrame'>
```

Clases y Objetos

Programación Orientada a Objetos

Características Importantes:

- **Instanciación:**
    - Crear un objeto a partir de una clase. Cada objeto es una instancia de la clase
- **Constructor (método __init__):**
    - Es un método especial que inicializa un objeto con atributos al momento de ser creado.
- **Encapsulamiento:**
    - Consiste en agrupar datos (atributos) y métodos dentro de una sola entidad (la clase), lo que permite un código más organizado y protege los datos internos.

Clases y Objetos

Programación Orientada a Objetos

```python
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def start(self):
        print(f"{self.make} {self.model} is starting...")

# Creating an instance of the Car class
my_car = Car('Jeep', 'Wrangler', 2018)
my_car.start()
```

Clases y Objetos

Programación Orientada a Objetos

```python
import pandas as pd

# Creating a simple DataFrame using Pandas DataFrame class
data = {
    'Well Name': ['Well A', 'Well B', 'Well C'],
    'Depth (m)': [3500, 3200, 3000],
    'Pressure (psi)': [5000, 4800, 4600],
    'Reservoir': ['Oil', 'Gas', 'Oil']
}

# Create a DataFrame object (an instance of the DataFrame class)
df = pd.DataFrame(data)

# Display the DataFrame
print("DataFrame:")
print(df)

# Using methods from the DataFrame class

# Method 1: Get basic information about the DataFrame
print("\nInfo about the DataFrame:")
print(df.info())

# Method 2: Get statistics about numerical columns
print("\nStatistical Summary:")
print(df.describe())

# Method 3: Filter rows where Reservoir is 'Oil'
oil_wells = df[df['Reservoir'] == 'Oil']
print("\nWells with Oil Reservoir:")
print(oil_wells)
```

Clases y Objetos

Pandas library

```
DataFrame:
  Well Name  Depth (m)  Pressure (psi) Reservoir
0    Well A       3500            5000       Oil
1    Well B       3200            4800       Gas
2    Well C       3000            4600       Oil

Info about the DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Well Name       3 non-null      object
 1   Depth (m)       3 non-null      int64
 2   Pressure (psi)  3 non-null      int64
 3   Reservoir       3 non-null      object
dtypes: int64(2), object(2)
memory usage: 224.0+ bytes
```

```
Statistical Summary:
         Depth (m)  Pressure (psi)
count     3.000000        3.000000
mean   3233.333333     4800.000000
std     250.000000      200.000000
min    3000.000000     4600.000000
25%    3100.000000     4700.000000
50%    3200.000000     4800.000000
75%    3350.000000     4900.000000
max    3500.000000     5000.000000

Wells with Oil Reservoir:
  Well Name  Depth (m)  Pressure (psi) Reservoir
0    Well A       3500            5000       Oil
2    Well C       3000            4600       Oil
```

Clases y Objetos

Lasio library

```python
import lasio

# Load the LAS file using lasio's LASFile class
las = lasio.read("example.las")

# Display basic information about the LAS file
print("Well Information:")
print(las.well)  # Access well information (metadata)

# Display curve information (data for each log)
print("\nCurve Information:")
print(las.curves)  # Access the curve information

# Display a portion of the data (log values)
print("\nLog Data (first 5 rows):")
print(las.data[:5])  # Print first 5 rows of log data

# Extract specific log data (e.g., Depth and Gamma Ray)
depth = las['DEPT']  # Access the depth log
gamma_ray = las['GR']  # Access the gamma ray log

# Display the first 5 values for Depth and Gamma Ray
print("\nDepth and Gamma Ray (first 5 values):")
for d, gr in zip(depth[:5], gamma_ray[:5]):
    print(f"Depth: {d:.2f} m, Gamma Ray: {gr:.2f} API")
```

```
Well Information:
Mnemonic    Value          Unit  Description
----------  -------------  ----  ------------------------------------------------
STRT        1000.0000      M     START DEPTH
STOP        2000.0000      M     STOP DEPTH
STEP        0.5000         M     STEP
NULL        -999.2500      M     NULL VALUE
WELL        EXAMPLE WELL         WELL NAME
...         ...            ...   ...

Curve Information:
Mnemonic  Unit  Value
--------  ----  ------------------------------------
DEPT      M     DEPTH
GR        API   GAMMA RAY
RES       OHMM  RESISTIVITY

Log Data (first 5 rows):
[[ 1.00000000e+03  4.50000000e+01  1.00000000e+02]
 [ 1.00050000e+03  4.80000000e+01  9.80000000e+01]
 [ 1.00100000e+03  5.10000000e+01  9.60000000e+01]
 [ 1.00150000e+03  5.30000000e+01  9.40000000e+01]
 [ 1.00200000e+03  5.60000000e+01  9.20000000e+01]]

Depth and Gamma Ray (first 5 values):
Depth: 1000.00 m, Gamma Ray: 45.00 API
Depth: 1000.50 m, Gamma Ray: 48.00 API
Depth: 1001.00 m, Gamma Ray: 51.00 API
Depth: 1001.50 m, Gamma Ray: 53.00 API
Depth: 1002.00 m, Gamma Ray: 56.00 API
```

Numpy y Pandas

Librerias de datos

- Numpy
    - Matrices (XD) para computación numerica

- Pandas
    - Contruida con Numpy
    - Series (1D) y Dataframes (2D) para manipulación y análisis de datos

Numpy

Matris multidimensional

```python
import numpy as np

# Creating an array
arr = np.array([1, 2, 3])

# Performing element-wise addition
arr2 = arr + 10  # Output: array([11, 12, 13])

# Matrix multiplication
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = np.dot(A, B)  # Matrix multiplication result
```

Pandas Sequence

Multiple tipos de datos

```python
import pandas as pd

# Create a Pandas Series with different data types
data = [10, 3.14, 'Hello', True, None]
index = ['A', 'B', 'C', 'D', 'E']
series = pd.Series(data, index=index)

# Display the Series
print(series)
```

Pandas Sequence

Multiple cálculos numéricos

```python
import pandas as pd
import numpy as np

# Create a Pandas Series with drilling depths (in meters)
drilling_depths_meters = pd.Series([1500, 2000, 2500, 3000, 3500], index=['Well A', 'Well B', 'Well C', 'Well D', 'Well E'])

# Conversion factor: 1 meter = 3.28084 feet
conversion_factor = 3.28084

# Apply NumPy multiplication to convert meters to feet
drilling_depths_feet = drilling_depths_meters * conversion_factor

# Display the original and converted Series
print("Original Drilling Depths (meters):")
print(drilling_depths_meters)
print("\nConverted Drilling Depths (feet):")
print(drilling_depths_feet)
```

Pandas Sequence

Multiple cálculos numéricos

```python
import pandas as pd
import numpy as np

# Create a Pandas Series with drilling depths (in meters)
well_depths = pd.Series([1500, 2000, 2500, 3000, 3500], index=['Well A', 'Well B', 'Well C', 'Well D', 'Well E'])

# Define constants
overburden_density = 2300  # kg/m^3 (average overburden density)
g = 9.81  # gravity in m/s^2

# Calculate overburden pressure using the formula: Pressure = Density * Depth * g
overburden_pressure = overburden_density * well_depths * g

# Display the calculated overburden pressures (in Pascals)
print("Overburden Pressure (Pa) for each well:")
print(overburden_pressure)
```

Qué es una Pandas DataFrame?

Estructura de datos etiquetada en 2 dimensiones

- Similar a una tabla en una base de datos o una hoja de cálculo de Excel.
- Características clave:
    - Filas y columnas: Los datos están organizados en filas y columnas con etiquetas.
    - Datos heterogéneos: Puede contener distintos tipos de datos (por ejemplo, enteros, cadenas, decimales).
    - Operaciones flexibles: Soporta operaciones como filtrado, combinación, ordenamiento y agrupamiento.
    - Indexado: Tanto las filas como las columnas tienen etiquetas (índices).

Qué es una Pandas DataFrame?

Column 1	Column 2	Column 3
Row 0	Data	Data
Row 1	Data	Data
Row 2	Data	Data

Pandas Dataframe

Multiple cálculus numéricos

```
import pandas as pd

# Create a DataFrame to represent well sections and their parameters
data = {
    'Well Section': ['Surface', 'Intermediate', 'Production'],
    'Depth (meters)': [0, 2000, 4000],
    'Mud Weight (ppg)': [8.5, 10.0, 12.0],  # pounds per gallon
    'Pore Pressure (psi)': [0, 8000, 15000],  # in psi
}

# Create DataFrame
well_df = pd.DataFrame(data)

# Convert mud weight from ppg to psi (1 ppg ≈ 0.4335 psi per foot of depth)
# Approximate depth is used to calculate mud weight in psi
well_df['Mud Weight (psi)'] = well_df['Mud Weight (ppg)'] * 0.4335 * well_df['Depth (meters)']

# Calculate the difference between mud weight and pore pressure
well_df['Pressure Control (psi)'] = well_df['Mud Weight (psi)'] - well_df['Pore Pressure (psi)']

# Display the DataFrame
print(well_df)
```

Pandas Dataframe

Multiple cálculus numéricos

```
import pandas as pd

# Create a DataFrame to represent well sections and their parameters
data = {
    'Well Section': ['Surface', 'Intermediate', 'Production', 'Deep Production', 'Final Section'],
    'Depth (meters)': [0, 2000, 4000, 6000, 8000],
    'Mud Weight (ppg)': [8.5, 10.0, 12.0, 14.0, 16.0],  # pounds per gallon
    'Pore Pressure (psi)': [0, 8000, 15000, 25000, 35000],  # in psi
}

# Create DataFrame
well_df = pd.DataFrame(data)

# Convert mud weight from ppg to psi (1 ppg ≈ 0.4335 psi per foot of depth)
# Approximate depth is used to calculate mud weight in psi
well_df['Mud Weight (psi)'] = well_df['Mud Weight (ppg)'] * 0.4335 * well_df['Depth (meters)']

# Calculate the difference between mud weight and pore pressure
well_df['Pressure Control (psi)'] = well_df['Mud Weight (psi)'] - well_df['Pore Pressure (psi)']

# Identify sections at risk of blowout
well_df['Blowout Risk'] = well_df['Pressure Control (psi)'] < 0

# Display the DataFrame
print(well_df)

# Find sections at risk of blowout
blowout_risk_sections = well_df[well_df['Blowout Risk']]
print("\nSections at Risk of Blowout:")
print(blowout_risk_sections[['Well Section', 'Depth (meters)', 'Pressure Control (psi)']])
```

```
iimport pandas as pd

# Create a DataFrame comparing average drilling parameters for two wells
data = {
    'Well Section': ['Surface', 'Intermediate', 'Production', 'Final Section'],
    'Well A: Average Depth (meters)': [0, 2000, 4000, 6000],
    'Well A: Average Rate of Penetration (m/h)': [10, 12, 8, 6],  # meters per hour
    'Well A: Average Weight on Bit (kN)': [50, 80, 120, 100],  # kilonewtons
    'Well A: Average Mud Weight (ppg)': [8.5, 10.0, 12.0, 14.0],  # pounds per gallon
    'Well A: Average Torque (Nm)': [1000, 1200, 1500, 1300],  # Newton meters
    'Well A: Average Pressure (psi)': [0, 8000, 15000, 25000],  # pounds per square inch

    'Well B: Average Depth (meters)': [0, 2100, 3900, 5800],
    'Well B: Average Rate of Penetration (m/h)': [9, 14, 7, 5],  # meters per hour
    'Well B: Average Weight on Bit (kN)': [55, 85, 110, 90],  # kilonewtons
    'Well B: Average Mud Weight (ppg)': [8.0, 11.0, 13.0, 15.0],  # pounds per gallon
    'Well B: Average Torque (Nm)': [950, 1250, 1400, 1200],  # Newton meters
    'Well B: Average Pressure (psi)': [0, 8200, 14800, 24000],  # pounds per square inch
}

# Create DataFrame
comparison_df = pd.DataFrame(data)

# Convert average mud weights from ppg to kg/m^3 for additional analysis
comparison_df['Well A: Average Mud Weight (kg/m^3)'] = comparison_df['Well A: Average Mud Weight (ppg)'] * 119.826
comparison_df['Well B: Average Mud Weight (kg/m^3)'] = comparison_df['Well B: Average Mud Weight (ppg)'] * 119.826

# Calculate differences between Well A and Well B
comparison_df['Depth Difference (meters)'] = comparison_df['Well A: Average Depth (meters)'] - comparison_df['Well B: Average Depth (meters)']
comparison_df['ROP Difference (m/h)'] = comparison_df['Well A: Average Rate of Penetration (m/h)'] - comparison_df['Well B: Average Rate of
Penetration (m/h)']
comparison_df['WOB Difference (kN)'] = comparison_df['Well A: Average Weight on Bit (kN)'] - comparison_df['Well B: Average Weight on Bit
(kN)']
comparison_df['Mud Weight Difference (ppg)'] = comparison_df['Well A: Average Mud Weight (ppg)'] - comparison_df['Well B: Average Mud
Weight (ppg)']
comparison_df['Torque Difference (Nm)'] = comparison_df['Well A: Average Torque (Nm)'] - comparison_df['Well B: Average Torque (Nm)']
comparison_df['Pressure Difference (psi)'] = comparison_df['Well A: Average Pressure (psi)'] - comparison_df['Well B: Average Pressure (psi)']

# Display the DataFrame
print(comparison_df)
```