Exploración de datos, detección de eventos e introducción a estadística

1-Cargar la librería welly en el ambient DrillingAnalytics y ejecutar Jupyter-Lab desde el directorio ~/DataScienceComodoro2023

```
conda activate DrillingAnalytics
conda list welly
pip install welly
conda list welly
cd ~/DataScienceComodoro2023
Jupyter-Lab
```

2- Cargar la libreria welly en el notebook

```
from welly import Well
from welly import Curve
```

3- Cargar el registro en welly: 8267_a0801_1996_comp.las

```
well=Well.from_las('Data/LAS/8267_a0801_1996_comp.las')
well
```

4- Gráficar las datos en el registro

```
tracks =['MD','GR', ['NPHI','RHOB','DRHO'],'DT']
well.plot(tracks=tracks)
```

5- Cuantos valores de GR existen, y cuantas lecturas están faltantes?

```
well.data[‘GR']
```

6- Cual es el promedio de los valores de GR?

```
gr=well.data['GR']
gr.describe()
```

7- Cual es el promedio de los valores de GR en la arena?

```
gr=well.data['GR']
gr[2600:3000].describe()
```

8- Determinar si las curvas están vacías, tienen gaps, están planas y si GR es menor que 25

```
import welly.quality as quality

test = {'Each':[quality.no_flat,
                quality.no_gaps,
```

```
            quality.not_empty
        ],
        'GR':[quality.all_above(25)
        ]}

from IPython.display import HTML
data_qc_table=well.qc_table_html(test)
HTML(data_qc_table)
```

9- Cual es la proporción de valores NaN en cada una de las curvas?

```
test = {'Each':[quality.no_flat,
            quality.no_gaps,
            quality.not_empty,
            quality.fraction_not_nans
        ],
        'GR':[quality.all_above(25)
        ]}

data_qc_table=well.qc_table_html(test)
HTML(data_qc_table)
```

10- Cuales curvas tienen picos?

```
test = {'Each':[quality.no_flat,
            quality.no_gaps,
            quality.not_empty,
            quality.fraction_not_nans,
            quality.no_spikes(100),
            quality.count_spikes
        ],
        'GR':[quality.all_above(25)
        ]}

data_qc_table=well.qc_table_html(test)
HTML(data_qc_table)
```

11- Cargar la curva de gamma ray a un dataframe

```
gamma_dataframe=gr.df
gamma_dataframe
```

12- Cargar today has curvas a un dataframe

```
well_dataframe=well.df()
well_dataframe
```

13- Cuantas lecturas hay en el dataframe de gamma ray y cuantas en el dataframe del pozo completo? Si hay una diferencia, explique porqué?

```
len(gamma_dataframe)

len(well_dataframe)
```

14- Definir una clase llamada Curve para guardar las curvas de un registro

```
class Curve:
    def __init__(self, well_name, well_location, curve_name, data):
        self.well_name = well_name
        self.well_location = well_location
        self.curve_name = curve_name
        self.data = data

    def get_well_name(self):
        return self.well_name

    def get_well_location(self):
        return self.well_location

    def get_curve_name(self):
        return self.curve_name

    def get_data(self, start=None, end=None):
        if start is None and end is None:
            return self.data
        elif start is None:
            return self.data[self.data.index <= end]
        elif end is None:
            return self.data[self.data.index >= start]
        else:
            return self.data[(self.data.index >= start) & (self.data.index <= end)]
```

15- Instanciar el objeto con los data de gamma ray

```
curve=Curve("pozo 1","cerro dragon","gamma ray”,gamma_dataframe)
```

16- Imprimir el nombre del pozo, la locación, el nombre de la curva y la data usando atributos

```
print("Nombre del pozo: "+curve.well_name+"\nLocación:
"+curve.well_location+"\nNombre de la curva: “+curve.curve_name+"\nData:
\n"+str(curve.data))
```

17- Imprimir el nombre del pozo, la locación, el nombre de la curva y la data usando métodos

```
print("Nombre del pozo: "+curve.get_well_name()+"\nLocación:
"+curve.get_well_location()+"\nNombre de la curva: “+curve.get_curve_name()+"\nData:
\n"+str(curve.get_data()))
```

18- Imprimir el nombre del pozo, la locación, el nombre de la curva y la data de 30 a 3000 metros usando métodos

```
print("Nombre del pozo: "+curve.get_well_name()+"\nLocación:
"+curve.get_well_location()+"\nNombre de la curva: “+curve.get_curve_name()+"\nData:
\n"+str(curve.get_data(30,3000)))
```

19- Imprimir el nombre del pozo, la locación, el nombre de la curva y la data de 30 a 3000 metros usando atributos

```
print("Nombre del pozo: "+curve.well_name+"\nLocación:
"+curve.well_location+"\nNombre de la curva: “+curve.curve_name+"\nData:
\n"+str(curve.data[30:3000]))
```

20- Cargar el archivo las usando lasio y comprobar que el numero de lineas es la correcta: Data/LAS/8267_a0801_1996_comp.las

```
import lasio

las_file_path = 'Data/LAS/8267_a0801_1996_comp.las'

well = lasio.read(las_file_path)

df=well.df()
df[30:3000]
```

21- Cargar el csv archivo las a un dataframe, teniendo cuidado de que el nombre de las columnas sea correcto: Swell-1A_AsciiDrillData_183.0-5006.csv

```
file_path = 'Data//ASCII//Swell-1A_AsciiDrillData_183.0-5006.csv'

df = pd.read_csv(file_path, skiprows=1)

df.head()



columns=df.columns

file_path = 'Data//ASCII//Swell-1A_AsciiDrillData_183.0-5006.csv'

df = pd.read_csv(file_path, skiprows=1)

df.head()



df.columns=columns
df.head()
```

22- Determinar si hay spikes en la data

```
df.plot(subplots=True, figsize=(15,35))
```

23- Convertir los -999.25 a NaN

```
import numpy as np
```

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

24- Determinar cuantos NaN hay en cada un de las curvas

```
print("Numero de datos faltantes")
for column in df.columns:
    count_nan = df[column].isna().sum()
    print(f"{column}: {count_nan}")
```

25- Eliminar los NaN reemplazándolos con el promedio del valor anterior y siguiente. Comparar los datos antes y después de la eliminación de los NaN

```
def replace_one_nan_with_avg(df, column_name):
    for i in range(1, len(df[column_name])-1):
        if pd.isna(df[column_name][i]):
            if not pd.isna(df[column_name][i-1]) and not pd.isna(df[column_name][i+1]):
                df[column_name][i] = (df[column_name][i-1] + df[column_name][i+1]) / 2
            else:
                df[column_name][i] = np.nan
    return df

for column in columns:
    replace_one_nan_with_avg(df,column)
df.head()


for column in df.columns:
    count_nan = df[column].isna().sum()
    print(f"Column: {column}: new NaN {count_nan}, old NaN {initial_nan_dic[column]}")
```

26- Eliminar los NaN reemplazándolos con el promedio del valor anterior y siguiente, teniendo en cuenta que pueden aver mas de un NaN consecutivo. Comparar los datos antes y después de la eliminación de los NaN

```
def replace_many_nan_with_avg(df, column_name):
    for i in range(len(df[column_name])):
        if pd.isna(df[column_name][i]):
            prev_value = None
            next_value = None
            for j in range(i-1, -1, -1):
                if not pd.isna(df[column_name][j]):
                    prev_value = df[column_name][j]
                    break
            for j in range(i+1, len(df[column_name])):
                if not pd.isna(df[column_name][j]):
                    next_value = df[column_name][j]
                    break
            if prev_value is None and next_value is None:
                df[column_name][i] = np.nan
            elif prev_value is None:
                df[column_name][i] = next_value
            elif next_value is None:
```

```
                df[column_name][i] = prev_value
            else:
                df[column_name][i] = (prev_value + next_value) / 2
    return df

for column in columns:
    replace_many_nan_with_avg(df,column)

for column in df.columns:
    count_nan = df[column].isna().sum()
    print(f"Column: {column}: new NaN {count_nan}, old NaN {initial_nan_dic[column]}")


df.head()

df.plot(subplots=True, figsize=(15,35))
```

27- Demostrar el use del método rolling average de pandas

```
import pandas as pd

# create a sample dataframe
dft = pd.DataFrame({'values': [4.72, 3.89, 3.75, 4.96, 3.62, 4.68, 4.34, 3.63, 3.54, 6.71,
7.89, 3.58, 3.71, 4.19, 3.4, 3.01, 3.33, 4.36, 3.63, 3.61, 3.23, 4.7,
                    4.57, 2.84, 2.63, 2.86, 2.86, 2.66, 3.04, 3.06, 2.67, 2.57, 2.9, 2.55,
3.33, 2.58, 2.7, 2.38, 2.71, 2.25, 2.05, 3, 3.12, 2.36, 2.76,
                    3.52, 3.11, 2.94, 2.73, 2.57, 3.35, 3.07, 2.69, 2.95, 3.14, 3.8, 3.26,
2.91, 3.16, 3.23, 3.07, 3.32, 3.56, 3.35, 3.35, 5.31,7.21,
                    5.14, 4.11, 8.71, 6.28, 6.27, 4.13, 3.11, 4.46, 3.47, 3.33, 2.9, 3.3,
3.28, 3.3, 3.28]})

# define the window size
WS = 5

# calculate the rolling average
dft['rolling_average'] = dft['values'].rolling(window=WS).mean()

dft.plot()
```

28- Filtrar todas las curvas usando rolling average

```
# define the window size
WS = 20

dfs=pd.DataFrame()
for column in columns:
    dfs[column] = df[column].rolling(window=WS).mean()

dfs.plot(subplots=True, figsize=(15,35))
```

29- Cargar archivos LAS en tiempo para una corrida completa

```
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=True)
    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)
```

30- Mostrar la curva de profundidad

```
ax=df.plot(y='HDEP',use_index=True)
ax.invert_yaxis()
plt.show()
```

31- Determinar si la curva de profundidad incrementa constantemente

```
df['HDEP'].is_monotonic_increasing
```

32- Gráficar la curva de profundidad donde no incrementa constantemente

```
ax = df.loc['00:00:00.22-09-21':, 'HDEP'].plot()
ax.invert_yaxis()
plt.xticks(rotation=45)
plt.show()
```

33- Cuales son los rangos normales de RPM?

```
1) Graficar la distribución de RPM:
df['RPMTOTAL'].plot(kind='kde')

2) Eliminamos la distribución alrededor de cero:
df_new = df.loc[df['RPMTOTAL'] > 0]
df_new['RPMTOTAL'].plot(kind='kde')

3) Graficamos con la librería sklearn

from sklearn.neighbors import KernelDensity

# Extract the RPMTOTAL column
rpmtotal = df_new['RPMTOTAL'].values.reshape(-1, 1)

# Fit the kernel density estimator
kde = KernelDensity(kernel='gaussian', bandwidth=8).fit(rpmtotal)
```

```
# Create a range of values to evaluate the estimator
x = np.linspace(rpmtotal.min(), rpmtotal.max(), 1000).reshape(-1, 1)

# Evaluate the estimator at the given values
y = np.exp(kde.score_samples(x))

# Plot the KDE distribution
plt.plot(x, y)
plt.xlabel('RPMTOTAL')
plt.ylabel('Density')
plt.title('KDE Distribution of RPMTOTAL')
plt.show()

4) Calculamos los valores normales:
from sklearn.cluster import KMeans

# Fit the KMeans estimator
kmeans = KMeans(n_clusters=5,n_init=10).fit(rpmtotal)

# Get the cluster labels
labels = kmeans.labels_

# Get the indices of the clusters
cluster_indices = [np.where(labels == i)[0] for i in range(kmeans.n_clusters)]

# Segregate the clusters into smaller arrays
clusters = [rpmtotal[indices] for indices in cluster_indices]

# Plot the normal distributions for each array
for i, cluster in enumerate(clusters):
    mu, std = cluster.mean(), cluster.std()
    x = np.linspace(mu - 3 * std, mu + 3 * std, 1000)
    y = np.exp(-0.5 * ((x - mu) / std) ** 2) / (std * np.sqrt(2 * np.pi))
    plt.plot(x, y, label=f'Cluster {i+1}')
plt.xlabel('RPMTOTAL')
plt.ylabel('Density')
plt.title('Normal Distributions of RPMTOTAL Clusters')
plt.legend()
plt.show()
```

34- Graphicar una distribución normal y mostrar los rangos de confianza de 95%

```
from scipy import stats

def plot_normal_distribution(mean, std, ci):
    x_axis = np.arange(0, 260, 0.001)
    pdf = stats.norm.pdf(x_axis, mean, std)

    fig, ax = plt.subplots()
    fig.set_figwidth(15) # Set the width of the figure to 10 inches
    ax.plot(x_axis, pdf)
```

```
    std_lim = stats.norm.ppf(1 - (1 - ci) / 2) # 95% CI
    low = mean - std_lim * std
    high = mean + std_lim * std

    ax.fill_between(x_axis, pdf, where=(low < x_axis) & (x_axis < high))
    ax.text(low, 0, f'{low:.2f}', ha='center',rotation=45)
    ax.text(high, 0, f'{high:.2f}', ha='center',rotation=45)

    plt.show()

plot_normal_distribution(150, 40, 0.90)
```

35- Obtener los parámetros estadísticos para los valores normales de RPM

```
for i, cluster in enumerate(clusters):
    mu, std = cluster.mean(), cluster.std()
    print(f'Cluster {i+1}: Mean = {mu:.2f}, Standard Deviation = {std:.2f}')
```

36- Graficar las distribuciones normales para los valores normales de RPM usando un rango de confianza de 95%

```
for i, cluster in enumerate(clusters):
    plot_normal_distribution(cluster.mean(), cluster.std(), 0.95)
```

37- Graficar todos los valores normales de RPM en una sola gráfica

```
def plot_normal_distribution(mean, std, ci, color):
    x_axis = np.arange(0, 260, 0.001)
    pdf = stats.norm.pdf(x_axis, mean, std)

    ax.plot(x_axis, pdf, color=color)

    std_lim = stats.norm.ppf(1 - (1 - ci) / 2) # 95% CI
    low = mean - std_lim * std
    high = mean + std_lim * std

    ax.fill_between(x_axis, pdf, where=(low < x_axis) & (x_axis < high))
    ax.text(low, 0.015, f'{low:.2f}', ha='left', fontweight='bold', rotation=45)
    ax.text(high, 0.015, f'{high:.2f}', ha='center', fontweight='bold', rotation=45)

fig, ax = plt.subplots(figsize=(25, 5))

colors = ['red', 'green', 'blue', 'orange', 'purple', 'brown', 'pink', 'gray', 'olive', 'cyan']

for i, cluster in enumerate(clusters):
    plot_normal_distribution(cluster.mean(), cluster.std(), 0.95, colors[i % len(colors)])

plt.show()
```