Entregable Guía 4 – Anteproyecto

Grupo 14: Diana Bayona, Juan Diego Pérez, Jorge Rodríguez y Santiago Gutiérrez

Informe ejecutivo que contempla un resumen dirigido al encargado del proyecto donde se da respuesta a:

1. **Las características de calidad de datos. Una descripción acerca de su completitud, consistencia, claridad, formato.**

La base de datos completa cuenta con un total de 847,960 registros y 5 columnas, luego de incorporar la columna como identificador del cliente: Fecha, Presión, Temperatura, Volumen y Client_ID. Este conjunto de datos consolidado nos permite realizar los diferentes análisis globales o individuales por cliente, facilitando la identificación de patrones de consumo y la evaluación del sistema de distribución de gas.Las mediciones abarcan desde el 14 de enero de 2019 hasta el 31 de diciembre de 2023, cubriendo casi cinco años de registros. (Revisar Anexos)

- **Presión:** La presión del suministro de gas presenta un promedio de 16.07 bar y una desviación estándar de 4.19, lo que refleja una variabilidad moderada en las mediciones. Los valores oscilan entre un mínimo de 2.93 bar y un máximo de 20.31 bar, indicando la presencia de condiciones extremas en ciertos momentos. Sin embargo, la mediana de 17.57 bar y los percentiles cercanos a este valor muestran que, en general, las mediciones de presión se mantienen relativamente estables.
- **Temperatura:** Por otro lado, la temperatura tiene un promedio de 25.19 °C, con una desviación estándar de 3.79 °C, lo que indica que las mediciones están distribuidas de manera relativamente uniforme alrededor de la media. Los valores fluctúan entre un mínimo de -5.26 °C y un máximo de 50.02 °C, evidenciando la capacidad del sistema para operar bajo condiciones climáticas tanto frías como muy cálidas. La mediana, que se sitúa en 25.38 °C, respalda la idea de que la mayoría de las observaciones se concentran cerca del promedio, sugiriendo una estabilidad en la distribución de la temperatura con pocas desviaciones extremas.
- **Volumen:** Finalmente, el volumen de suministro de gas para los diferentes clientes es en promedio de 62.33 m^3, lo cual representa un valor considerable en las mediciones de consumo de gas. Sin embargo, la desviación estándar de 80.50 m^3 sugiere una notable dispersión en los datos, indicando grandes diferencias en los volúmenes registrados, probablemente asociados con variaciones en la demanda o almacenamiento de gas. Los valores oscilan entre 0.00 m^3 y un máximo de 577.41 m^3, lo que refleja una alta variabilidad en el suministro. Cabe destacar que, en al menos el 25% de las observaciones, los volúmenes son 0, posiblemente señalando periodos de inactividad o clientes sin consumo registrado. Por otro lado, el percentil 75% alcanza un volumen de 99.32 m^3, lo que sugiere que la mayoría de las mediciones se encuentran por debajo de este valor, aunque también existen casos aislados con volúmenes mucho mayores.

La auditoría de los datos muestra un conjunto limpio y bien estructurado en la mayoría de las dimensiones evaluadas. No se encontraron valores nulos, duplicados ni registros sin identificadores de cliente, lo que asegura la **integridad** de los datos. La **completitud** también es adecuada, aunque se identificaron dos clientes (Client_16 y Client_8) con posibles problemas de secuencialidad en sus fechas, lo que requiere una revisión adicional. En términos de **coherencia,** las variables de temperatura, presión y volumen están dentro de rangos esperados, sin registros inconsistentes. La **consistencia** muestra una correlación negativa moderada entre volumen y presión (-0.30), lo que es coherente con las leyes de los gases, y no se encontraron relaciones incoherentes entre estas variables. En la validación de **temporalidad**, se detecta que la última fecha registrada es del 31 de diciembre de 2023, indicando que los datos no están actualizados, lo cual podría impactar futuros análisis. Finalmente, el **formato** es correcto, con tipos de datos bien definidos, incluyendo un formato válido para la columna de Fecha. En general, los datos son fiables, pero es necesario resolver la desactualización temporal si así se requiere y revisar los clientes con posibles brechas de fechas.

2. **Las causas para la pertinencia de aplicar o no aplicar los procesos de limpieza. Resultados de tal proceso.**

El análisis inicial de la base de datos reveló un alto nivel de calidad, con ausencia de valores nulos y duplicados en la mayoría de las variables. Sin embargo, durante el proceso de validación, se identificaron **brechas temporales significativas** en los datos de dos clientes específicos (Client_16 y Client_8). Estas brechas, que correspondían a fechas faltantes en las mediciones, comprometían la completitud y consistencia temporal de los datos, afectando la continuidad de las series temporales. (Revisar Anexos)

Pertinencia del Proceso de Limpieza:
Aplicar un proceso de limpieza era crucial para garantizar la integridad y usabilidad de los datos. Las brechas temporales podrían introducir sesgos o errores en análisis posteriores, especialmente en la detección de anomalías y en modelos predictivos que dependen de una secuencia temporal coherente. La limpieza también fue necesaria para cumplir con los requerimientos operativos y asegurar que las decisiones basadas en estos datos sean precisas.

Metodología de Limpieza:
El proceso de limpieza incluyó dos pasos clave:

1. **Identificación y Adición de Fechas Faltantes:** Se detectaron 366 fechas faltantes distribuidas en los registros de Client_16 y Client_8. Estas fechas fueron incorporadas a la base de datos para completar las series temporales.
2. **Imputación de Datos con Interpolación Lineal:** Para las nuevas fechas incorporadas, se utilizaron los valores adyacentes de las variables presión, temperatura y volumen para calcular valores imputados mediante interpolación lineal. Este método fue elegido por su simplicidad y eficiencia en datos continuos, garantizando que los valores imputados sigan las tendencias naturales del sistema.

Resultados del Proceso de Limpieza:

- **Crecimiento del conjunto de datos:** La base de datos pasó de 847,960 a 848,692 registros, reflejando la correcta incorporación de los datos faltantes.
- **Restauración de la Continuidad Temporal**: Las series temporales ahora son completas, sin brechas que puedan afectar el análisis.
- **Mejora en la Consistencia y Completitud:** La limpieza permitió mantener la coherencia entre las variables y asegurar que la base de datos cumpla con los estándares de calidad necesarios para análisis avanzados.

Conclusión:
El proceso de limpieza fue esencial para preparar los datos y garantizar su utilidad. La incorporación de fechas faltantes y la imputación de valores aseguraron una base de datos confiable y alineado con los requerimientos de negocio. Esto permite realizar análisis robustos, optimizando la detección de anomalías y la toma de decisiones basada en datos.

3. **Resultados del proceso de entendimiento de los datos. Conclusión acerca de la coherencia de estos resultados de análisis frente a la problemática de negocio a resolver o los requerimientos encontrados en la guía 2.**

El proceso de entendimiento de los datos permitió identificar patrones clave en las variables principales (presión, temperatura y volumen) que son coherentes con el comportamiento esperado en un sistema de distribución de gas. A continuación, se detallan los hallazgos más relevantes (Revisar Anexos):

Relaciones entre Variables:

- **Presión y Volumen:**

Se observó una correlación negativa entre estas variables, lo cual es consistente con las leyes de los gases. A medida que la presión disminuye, el volumen tiende a incrementarse, lo que refleja un comportamiento típico en la dinámica de distribución de gas en sistemas cerrados.

- **Temperatura y Volumen:**

La correlación positiva entre estas variables muestra que, al aumentar la temperatura, también se incrementa el volumen del gas consumido o almacenado. Este resultado es coherente con el principio de expansión térmica del gas, un factor relevante en condiciones climáticas variables o en procesos industriales sensibles a la temperatura.

Distribución de las Variables:

Los histogramas generados durante el análisis muestran que la temperatura y la presión se distribuyen de manera estable alrededor de valores medios esperados, con ligeras variaciones dependiendo del cliente. El volumen presenta mayor variabilidad, lo que puede ser indicativo de diferentes patrones de consumo entre los clientes, posiblemente relacionados con su nivel de demanda o sus horarios de operación. (Revisar Anexos)

Del análisis individual de cada uno los clientes se pueden observar que cada uno presenta patrones de consumo y operación únicos, reflejados en sus estadísticas descriptivas e histogramas. Por ejemplo, algunos clientes, como Client_3, operan en sistemas de baja presión con consumos significativamente altos, mientras que otros, como Client_5, muestran un consumo más moderado con mayor variabilidad en la temperatura. Las diferencias en el rango de volumen consumido y la presión sugieren que cada cliente podría estar operando bajo condiciones específicas de su entorno o nivel de demanda. Además, los picos en ciertos clientes pueden indicar eventos esporádicos de alto consumo, potencialmente asociados con necesidades operativas puntuales. Estos patrones serán clave para entender mejor el comportamiento individual de cada cliente y para personalizar estrategias de monitoreo y optimización en función de sus características particulares.

Brechas Temporales:

Aunque el conjunto de datos está mayormente completo, la identificación de brechas temporales en Client_16 y Client_8 reveló la necesidad de imputar datos para garantizar la continuidad temporal. La **interpolación lineal** aplicada resolvió estas inconsistencias, asegurando que los análisis posteriores no se vean afectados por registros faltantes.

Relevancia Frente a la Problemática de Negocio:

Los resultados obtenidos son coherentes con los requerimientos operativos de la empresa. La calidad de los datos después del proceso de limpieza es adecuada para construir modelos predictivos que permitan detectar anomalías en el consumo de gas. Esto es crucial para optimizar el monitoreo del sistema, predecir posibles fallas y garantizar la continuidad del servicio. Además, la estabilidad observada en la relación entre las variables respalda la confianza en el uso de esta base de datos para la toma de decisiones.

Limitaciones Detectadas:

Un punto para considerar es que los datos no están completamente actualizados, con el último registro en diciembre de 2023. Esto puede limitar la capacidad de reflejar situaciones operativas actuales, lo que resalta la importancia de integrar datos más recientes para mantener la relevancia en el análisis y modelado.

Conclusión:
El proceso de entendimiento de los datos muestra un sistema robusto y coherente que puede ser utilizado para abordar la problemática de negocio. Los patrones identificados y la resolución de brechas temporales aseguran que los datos están listos para su uso en análisis avanzados, como la detección de anomalías o el pronóstico de consumo, alineándose perfectamente con los objetivos planteados en la guía inicial.

4. **Anexos técnicos. Nota: estos anexos no se esperan evaluar, permitirán verificar si hay alguna duda frente a informe, por lo que su inclusión en el documento es obligatoria.**

Repositorio: https://github.com/dianabayonap/Proyecto_G14.git

Anexos:

Cargue Información

```
# Ruta del archivo Excel
file_path = 'Datos_Contugas.xlsx'# Asegúrate de que la ruta al archivo sea correcta

# Cargar el archivo Excel y obtener los nombres de las hojas
excel_file = pd.ExcelFile(file_path)
sheet_names = excel_file.sheet_names

# Lista para almacenar los DataFrames
all_data = []

# Cargar y unificar las hojas
for idx, sheet in enumerate(sheet_names, start=1):
    df = pd.read_excel(file_path, sheet_name=sheet)
    df['Client_ID'] = f'Client_{idx}'  # Crear un ID único por cliente
    all_data.append(df)

# Combinar todos los DataFrames en uno solo
datos = pd.concat(all_data, ignore_index=True)
```

```
print("La dimensión de los datos es: " + str(datos.shape))
datos.head(5)
```

```
La dimensión de los datos es: (847960, 5)
```

	Fecha	Presion	Temperatura	Volumen	Client_ID
0	2019-01-14 00:00:00	17.732563	28.209354	20.969751	Client_1
1	2019-01-14 01:00:00	17.747776	28.518614	17.845739	Client_1
2	2019-01-14 02:00:00	17.758916	28.230191	20.975914	Client_1
3	2019-01-14 03:00:00	17.727940	27.811509	20.592299	Client_1
4	2019-01-14 04:00:00	17.746484	27.795293	21.690626	Client_1

Auditoria Datos

```
def validate_integrity(datos):
    # Verificar valores nulos
    null_values = datos.isnull().sum()
    print("Valores nulos por columna:")
    print(null_values)

    # Verificar duplicados
    duplicates = datos.duplicated().sum()
    print(f"\nNúmero de filas duplicadas: {duplicates}")
```

```
def validate_completeness(datos):
    # Verificar registros sin Client_ID
    missing_clients = datos['Client_ID'].isnull().sum()
    print(f"Número de registros sin Client_ID: {missing_clients}")

    # Verificar fechas faltantes o no secuenciales para cada cliente
    fechas_completas = datos.groupby('Client_ID')['Fecha'].apply(lambda x: x.is_monotonic_increasing)
    print("\nClientes con fechas no secuenciales o faltantes:")
    print(fechas_completas[~fechas_completas])
```

```
def validate_coherence(datos):
    # Valores esperados para variables específicas
    inconsistent_temps = datos[(datos['Temperatura'] < -50) | (datos['Temperatura'] > 60)]
    inconsistent_pressure = datos[(datos['Presion'] < 0) | (datos['Presion'] > 200)]
    inconsistent_volume = datos[datos['Volumen'] < 0]

    print(f"Registros con Temperatura inconsistente: {len(inconsistent_temps)}")
    print(f"Registros con Presión inconsistente: {len(inconsistent_pressure)}")
    print(f"Registros con Volumen inconsistente: {len(inconsistent_volume)}")
```

```
def validate_consistency(datos):
    # Comprobar consistencia de relaciones, ejemplo: Volumen vs Presion
    correlation = datos[['Volumen', 'Presion']].corr().iloc[0, 1]
    print(f"Correlación entre Volumen y Presión: {correlation:.2f}")

    # Verificar si existen valores fuera de los intervalos esperados simultáneamente
    unexpected_relations = datos[(datos['Presion'] > 100) & (datos['Volumen'] < 10)]
    print(f"Número de relaciones inconsistente Volumen vs Presión: {len(unexpected_relations)}")
```

```
def validate_format(datos):
    # Verificar formato de la columna de fecha
    try:
        datos['Fecha'] = pd.to_datetime(datos['Fecha'])
        print("Formato de fechas es válido.")
    except Exception as e:
        print(f"Formato de fechas inválido: {e}")

    # Verificar tipos de datos
    print("\nTipos de datos por columna:")
    print(datos.dtypes)
```

```
def validate_timeliness(datos):
    max_date = datos['Fecha'].max()
    print(f"Última fecha registrada: {max_date}")

    if max_date < pd.Timestamp.now() - pd.Timedelta(days=7):
        print("Advertencia: Los datos no están actualizados.")
```

```
def validate_data(datos):
    print("Validación de Integridad:")
    validate_integrity(datos)

    print("\nValidación de Completitud:")
    validate_completeness(datos)

    print("\nValidación de Coherencia:")
    validate_coherence(datos)

    print("\nValidación de Consistencia:")
    validate_consistency(datos)

    print("\nValidación de Temporalidad:")
    validate_timeliness(datos)

    print("\nValidación de Formato:")
    validate_format(datos)
```

```
validate_data(datos)
```

```
Validación de Integridad:
Valores nulos por columna:
Fecha          0
Presion        0
Temperatura    0
Volumen        0
Client_ID      0
dtype: int64

Número de filas duplicadas: 0

Validación de Completitud:
Número de registros sin Client_ID: 0

Clientes con fechas no secuenciales o faltantes:
Client_ID
Client_16    False
Client_8     False
Name: Fecha, dtype: bool

Validación de Coherencia:
Registros con Temperatura inconsistente: 0
Registros con Presión inconsistente: 0
Registros con Volumen inconsistente: 0

Validación de Consistencia:
Correlación entre Volumen y Presión: -0.30
Número de relaciones inconsistente Volumen vs Presión: 0

Validación de Temporalidad:
Última fecha registrada: 2023-12-31 23:00:00
Advertencia: Los datos no están actualizados.

Validación de Formato:
Formato de fechas es válido.

Tipos de datos por columna:
Fecha          datetime64[ns]
Presion               float64
Temperatura           float64
Volumen               float64
Client_ID              object
dtype: object
```

```
def check_date_sequence(datos, client_id):
    client_data = datos[datos['Client_ID'] == client_id].sort_values('Fecha')
    client_data['Diferencia'] = client_data['Fecha'].diff().dt.total_seconds() / 3600  # Diferencia en horas
    print(f"Secuencia de fechas para {client_id}:\n")
    print(client_data[['Fecha', 'Diferencia']])
    print("\nRegistros con diferencias significativas:\n")
    print(client_data[client_data['Diferencia'] > 1])  # Ajusta este valor según lo esperado
```

```
check_date_sequence(datos, 'Client_16')
check_date_sequence(datos, 'Client_8')
```

```
Secuencia de fechas para Client_16:

                     Fecha  Diferencia
634906 2019-01-14 00:00:00         NaN
634907 2019-01-14 01:00:00         1.0
634908 2019-01-14 02:00:00         1.0
634909 2019-01-14 03:00:00         1.0
634910 2019-01-14 04:00:00         1.0
...                    ...         ...
678048 2023-12-31 19:00:00         1.0
678049 2023-12-31 20:00:00         1.0
678050 2023-12-31 21:00:00         1.0
678051 2023-12-31 22:00:00         1.0
678052 2023-12-31 23:00:00         1.0

[43147 rows x 2 columns]

Registros con diferencias significativas:
```

```python
def find_missing_dates(datos, client_id, freq='h'):
    client_data = datos[datos['Client_ID'] == client_id].sort_values('Fecha')
    expected_dates = pd.date_range(start=client_data['Fecha'].min(),
                                   end=client_data['Fecha'].max(), freq=freq)
    actual_dates = client_data['Fecha']
    missing_dates = expected_dates.difference(actual_dates)
    print(f"Fechas faltantes para {client_id}:")
    print(missing_dates)
```

```python
find_missing_dates(datos, 'Client_16')
find_missing_dates(datos, 'Client_8')
```

cut selected cells

```
Fechas faltantes para Client_16:
DatetimeIndex(['2019-01-19 08:00:00', '2019-01-27 05:00:00',
               '2019-03-13 03:00:00', '2019-03-13 04:00:00',
               '2019-03-13 05:00:00', '2019-03-13 06:00:00',
               '2019-03-13 07:00:00', '2019-03-13 08:00:00',
               '2019-03-13 09:00:00', '2019-03-13 10:00:00',
               ...
               '2023-10-23 16:00:00', '2023-10-23 17:00:00',
               '2023-10-23 18:00:00', '2023-10-23 19:00:00',
               '2023-10-23 20:00:00', '2023-10-23 21:00:00',
               '2023-10-23 22:00:00', '2023-11-04 07:00:00',
               '2023-11-09 06:00:00', '2023-11-10 03:00:00'],
              dtype='datetime64[ns]', length=366, freq=None)
Fechas faltantes para Client_8:
DatetimeIndex(['2019-01-19 08:00:00', '2019-01-27 05:00:00',
               '2019-03-13 03:00:00', '2019-03-13 04:00:00',
               '2019-03-13 05:00:00', '2019-03-13 06:00:00',
               '2019-03-13 07:00:00', '2019-03-13 08:00:00',
               '2019-03-13 09:00:00', '2019-03-13 10:00:00',
               ...
               '2023-10-23 16:00:00', '2023-10-23 17:00:00',
               '2023-10-23 18:00:00', '2023-10-23 19:00:00',
               '2023-10-23 20:00:00', '2023-10-23 21:00:00',
               '2023-10-23 22:00:00', '2023-11-04 07:00:00',
               '2023-11-09 06:00:00', '2023-11-10 03:00:00'],
              dtype='datetime64[ns]', length=366, freq=None)
```

Análisis Descriptivo

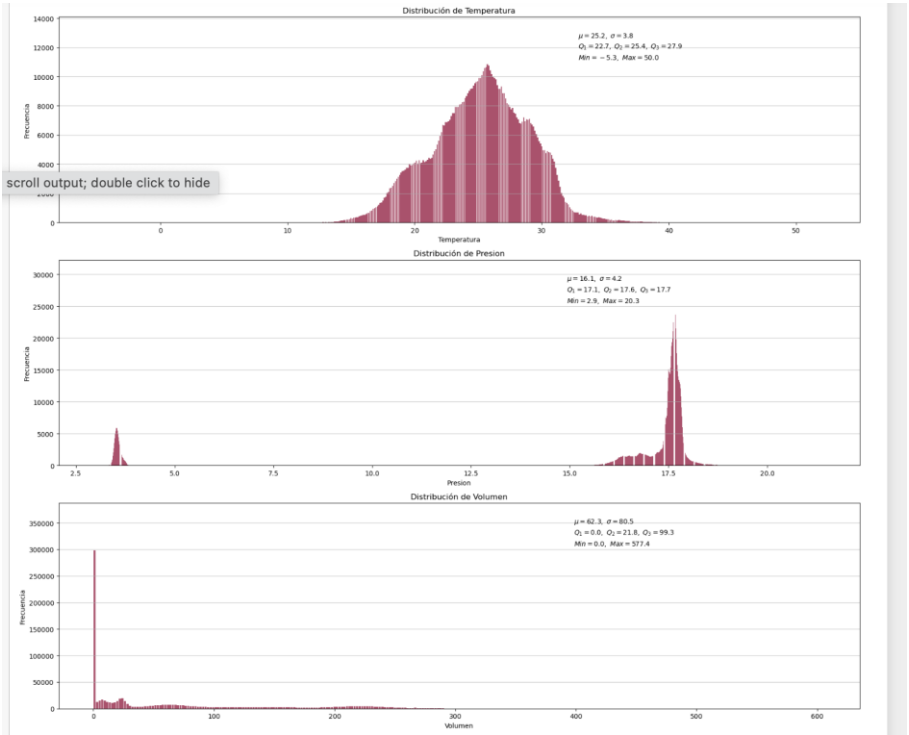
```python
datos.describe()
```

	Fecha	Presion	Temperatura	Volumen
count	847960	847960.000000	847960.000000	847960.000000
mean	2021-07-08 22:08:19.659418112	16.072957	25.198239	62.328206
min	2019-01-14 00:00:00	2.934873	-5.257899	0.000000
25%	2020-04-13 23:00:00	17.097350	22.693027	0.000000
50%	2021-07-07 06:00:00	17.570449	25.379859	21.773567
75%	2022-10-06 02:15:00	17.694254	27.886244	99.319649
max	2023-12-31 23:00:00	20.307852	50.019853	577.413425
std	NaN	4.186408	3.790497	80.498112

```python
def descriptive_stats_per_client(datos):
    clients = datos['Client_ID'].unique()  # Obtener IDs únicos de los clientes

    descriptive_summary = {}  # Diccionario para almacenar los resultados

    for client in clients:
        client_data = datos[datos['Client_ID'] == client]  # Filtrar datos por cliente
        descriptive_summary[client] = client_data[['Temperatura', 'Presion', 'Volumen']].describe()  # Resumen estad

        print(f"Descriptive statistics for {client}:\n")
        print(descriptive_summary[client])
        print("\n" + "-"*50 + "\n")

    return descriptive_summary

# Llama a la función con tu DataFrame
stats_per_client = descriptive_stats_per_client(datos)  # Cambia por el DataFrame que contenga tus datos
```

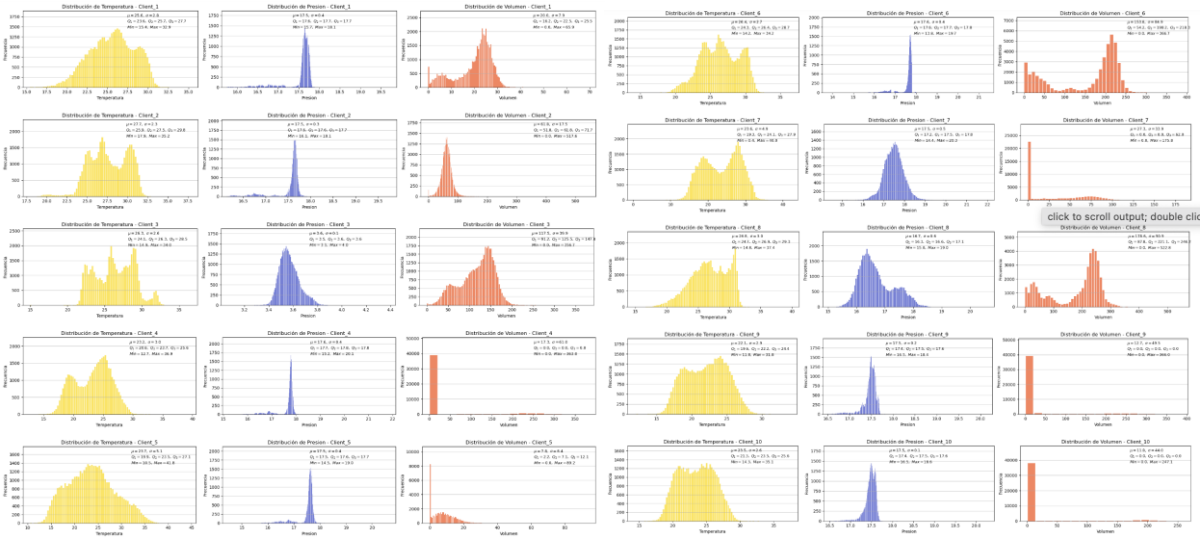
```
Descriptive statistics for Client_1:

        Temperatura       Presion       Volumen
count  43412.000000  43412.000000  43412.000000
mean      25.575853     17.535934     19.976401
std        2.756246      0.358310      7.939171
min       15.401803     15.742337      0.000000
25%       23.587235     17.590781     16.248989
50%       25.711326     17.651439     22.470213
75%       27.695776     17.711107     25.533223
max       32.869112     18.074274     65.936644

--------------------------------------------------

Descriptive statistics for Client_2:

        Temperatura       Presion       Volumen
count  41382.000000  41382.000000  41382.000000
mean      27.673040     17.526440     61.819045
std        2.345356      0.318984     17.495571
min       17.884059     16.129015      0.000000
25%       25.859054     17.581711     51.809989
50%       27.493743     17.637486     61.814487
75%       29.825196     17.675036     71.742183
max       35.208346     18.106402    517.564868
```

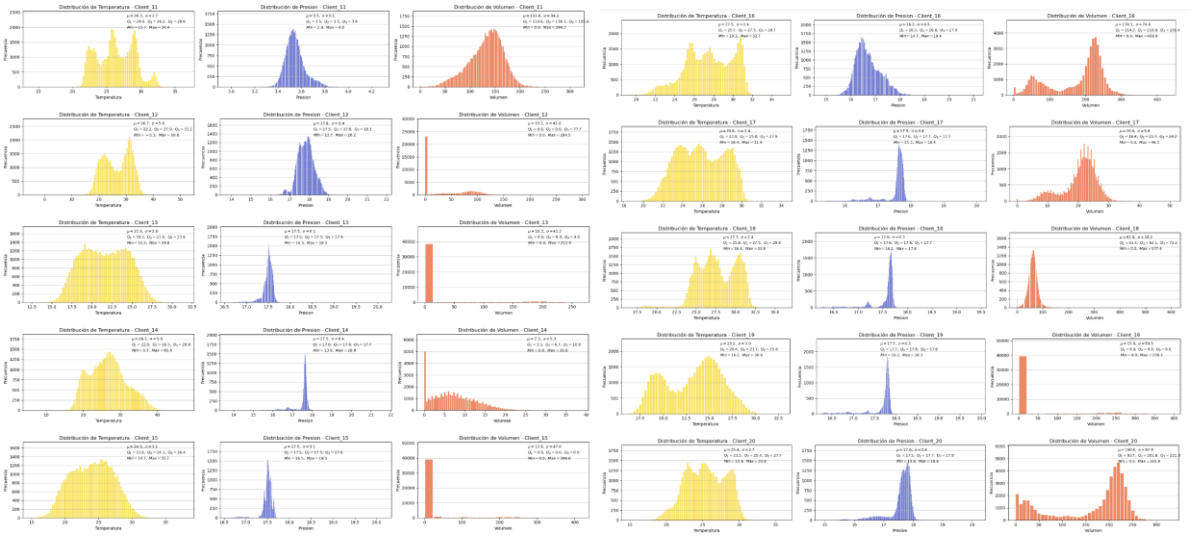
```python
import matplotlib.pyplot as plt

def descriptive(datos):
    plt.figure(figsize=(18, 15.5))  # Ajuste del tamaño de la figura
    selected_vars = ['Temperatura', 'Presion', 'Volumen']  # Variables específicas

    for i, var in enumerate(selected_vars):
        plt.subplot(3, 1, i + 1)  # Solo 3 gráficas en filas separadas
        if len(datos[var].unique()) > 3:
            dataHist = datos[var]
            n, bins, patches = plt.hist(x=dataHist, bins='auto', color='#98002E', alpha=0.7, rwidth=0.85)
            maxfreq = n.max()
            maxval = dataHist.min() + (dataHist.max() - dataHist.min()) * 0.69
            plt.grid(axis='y', alpha=0.75)
            plt.xlabel(f'{var}')
            plt.ylabel('Frecuencia')
            plt.title(f'Distribución de {var}')
            plt.text(maxval, maxfreq * 1.17, f'$\mu={round(dataHist.mean(), 1)},$ $\sigma={round(dataHist.std(), 1)}
            plt.text(maxval, maxfreq * 1.1, f'$Q_1={round(dataHist.quantile(q=0.25), 1)},$ $Q_2={round(dataHist.quan
            plt.text(maxval, maxfreq * 1.03, f'$Min={round(dataHist.min(), 1)},$ $Max={round(dataHist.max(), 1)}$')
            plt.ylim(ymax=maxfreq * 1.3)
            plt.xlim(xmax=dataHist.max() * 1.1)
        else:
            data = datos.groupby(var).size()
            labels = data.keys()
            colors = ['#E31B23', '#532E63']
            plt.pie(x=data, autopct="%.1f%%", labels=labels, pctdistance=0.6, startangle=120, textprops={'fontsize':
            plt.title((f"Proporciones de {var}").format(var), fontsize=18)

        plt.tight_layout()  # Ajusta la posición de las gráficas

    plt.show()
descriptive(datos)
```



scroll output; double click to hide

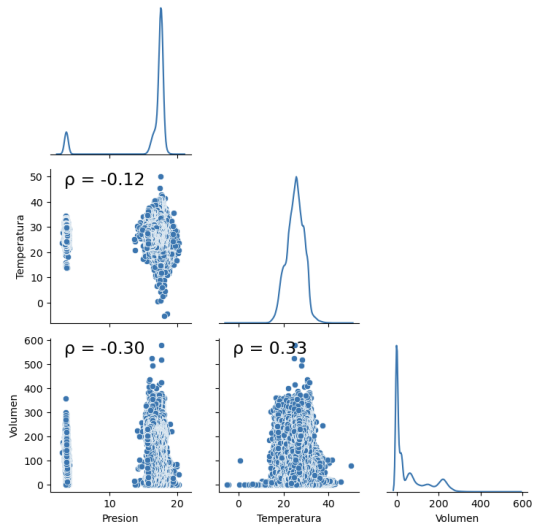
```python
def descriptive_per_client(datos):
    clients = datos['Client_ID'].unique()  # Obtener IDs únicos de los clientes

    for client in clients:
        client_data = datos[datos['Client_ID'] == client]  # Filtrar datos por cliente
        client_data.loc[:,'Volumen'] = round(client_data['Volumen'],1)
        plt.figure(figsize=(20, 3.5))  # Ajuste del tamaño de la figura
        selected_vars = ['Temperatura', 'Presion', 'Volumen']  # Variables específicas

        for i, var in enumerate(selected_vars):
            plt.subplot(1, 3, i + 1)  # Gráficas en filas separadas
            colors = ['#ffd800', '#394acb', '#fb4c16']
            if len(client_data[var].unique()) > 3:
                dataHist = client_data[var]
                n, bins, patches = plt.hist(x=dataHist, bins='auto', color=colors[i], alpha=0.7, rwidth=0.85)
                maxfreq = n.max()
                maxval = dataHist.min() + (dataHist.max() - dataHist.min()) * 0.69
                plt.grid(axis='y', alpha=0.75)
                plt.xlabel(f'{var}')
                plt.ylabel('Frecuencia')
                plt.title(f'Distribución de {var} - {client}')
                plt.text(maxval, maxfreq * 1.2, f'$\mu={round(dataHist.mean(), 1)},$ $\sigma={round(dataHist.std(),
                plt.text(maxval, maxfreq * 1.1, f'$Q_1={round(dataHist.quantile(q=0.25), 1)},$ $Q_2={round(dataHist.
                plt.text(maxval, maxfreq * 1.0, f'$Min={round(dataHist.min(), 1)},$ $Max={round(dataHist.max(), 1)}$
                plt.ylim(ymax=maxfreq * 1.3)
                plt.xlim(xmax=dataHist.max() * 1.1)
            else:
                data = client_data.groupby(var).size()
                labels = data.keys()
                colors = ['#E31B23', '#532E63']
                plt.pie(x=data, autopct="%.1f%%", labels=labels, pctdistance=0.6, startangle=120, textprops={'fontsi
                plt.title((f"Proporciones de {var} - {client}").format(var), fontsize=18)

            plt.tight_layout()  # Ajusta la posición de las gráficas

        plt.show()

descriptive_per_client(datos)
```





```
cats = ['Client_ID', 'Fecha']
cont = list(set(datos.columns) - set(cats))
## Correlogramas
def corrfunc(x, y, ax=None, **kws):
    r, _ = pearsonr(x, y)
    ax = ax or plt.gca()
    ax.annotate(f'p = {r:.2f}', xy=(.1, .9), xycoords=ax.transAxes, fontsize= 17)
g = sns.PairGrid(datos[cont], diag_sharey=False, corner=True)
g.map_lower(sns.scatterplot)
g.map_diag(sns.kdeplot)
g.map_lower(corrfunc)
```

```
<seaborn.axisgrid.PairGrid at 0x1d00e4f15a0>
```



Tratamiento de datos

```
def find_missing_dates(datos, client_id, freq='h'):
    client_data = datos[datos['Client_ID'] == client_id].sort_values('Fecha')
    expected_dates = pd.date_range(start=client_data['Fecha'].min(),
                                   end=client_data['Fecha'].max(), freq=freq)
    actual_dates = client_data['Fecha']
    missing_dates = expected_dates.difference(actual_dates)
    return missing_dates
```

```
# Diccionario para almacenar las fechas faltantes por cliente
missing_dates_dict = {}

# Identificar y guardar fechas faltantes para cada cliente con problemas
for client_id in ['Client_16', 'Client_8']:
    missing_dates_dict[client_id] = find_missing_dates(datos, client_id)
```

```
def add_missing_dates(datos, client_id, missing_dates):
    # Crear un DataFrame con las fechas faltantes
    missing_data = pd.DataFrame({'Fecha': missing_dates, 'Client_ID': client_id})

    # Unir con el DataFrame original y ordenar por cliente y fecha
    datos = pd.concat([datos, missing_data], ignore_index=True).sort_values(by=['Client_ID', 'Fecha'])
    return datos

# Agregar fechas faltantes para cada cliente
for client_id, missing_dates in missing_dates_dict.items():
    datos = add_missing_dates(datos, client_id, missing_dates)
```

```
# Imputación general de valores
datos['Presion'] = datos['Presion'].interpolate(method='linear')
datos['Temperatura'] = datos['Temperatura'].interpolate(method='linear')
datos['Volumen'] = datos['Volumen'].interpolate(method='linear')
```

```
validate_completeness(datos)
```

```
Número de registros sin Client_ID: 0

Clientes con fechas no secuenciales o faltantes:
Series([], Name: Fecha, dtype: bool)
```

```
print("La dimensión de los datos es: " + str(datos.shape))
```

```
La dimensión de los datos es: (848692, 5)
```