Energías Renovables - Análisis Financiero Empresa

May 6, 2024

1 Energías Renovables - Análisis Financiero de la Empresa

Creado por:

- **V. D. Betancourt**

1.1 Introducción

1.1.1 Descripción

El presente proyecto pertenece al ámbito de las **Energías Renovables**. En particular, está enfocado en datos de **Estados Financieros y de Resultados** de una **Empresa**.

1.1.2 Objetivo

El presente proyecto tiene como finalidad estudiar la **Información Financiera Anual de la Empresa**, del **2011** al **2023**, por medio de:

- Generar un *Análisis Exploratorio de Datos*, incluyendo diversas visualizaciones diseñadas en Seaborn.
- Generar *Predicciones* con diferentes modelos, tales como: Redes Neuronales y Regresión Múltiple.

1.2 Settings

[ ]:
```
# Importar Librerías
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

1.3 Carga de Datos y Data Wrangling

Se ha creado un fichero en MS Excel, llamado `'fin_data_energy.xlsx'`, con la **Información Financiera Anual** de la Empresa, disponible en su sitio web, por lo que es *información pública*.

1.3.1 Dataset

El dataset `'fin_data_energy.xlsx'` consta de los siguientes **17 campos**:

Cuentas de Referencia

- `Fecha`: Indica cuándo se recogieron o reportaron los datos financieros.
- `Empresa`: El nombre de la empresa a la que pertenecen los datos financieros. Aunque el Análisis de este proyecto trata de una sola Empresa, el dataset hace distinción entre 2 empresas debido al cambio de nombre de la empresa original, a partir de 2015.

Cuentas del Balance

- `Activo no corriente`: Se refiere a los activos que no se espera que se conviertan en efectivo o que no se usarán dentro de un año. Ejemplos incluyen bienes inmuebles, maquinaria y patentes.
- `Activo corriente`: Incluye todos los activos que se espera que se conviertan en efectivo, se vendan o se consuman en el plazo de un año o menos. Ejemplos comunes son el efectivo, inventarios y cuentas por cobrar.
- `Total Activo`: La suma de los activos corrientes y no corrientes. Representa todos los recursos económicos controlados por la empresa.
- `Patrimonio neto`: También conocido como capital propio, es el valor residual de los activos de la empresa menos sus pasivos. Es básicamente lo que los propietarios de la empresa poseen.
- `Pasivo no corriente`: Deudas o obligaciones financieras que no se espera que sean liquidadas dentro del próximo año fiscal. Ejemplos incluyen préstamos a largo plazo y bonos emitidos.
- `Pasivo corriente`: Deudas o obligaciones que deben ser pagadas dentro de un año. Incluye cosas como cuentas por pagar, deudas a corto plazo y otros pasivos a corto plazo.
- `Total Pasivo`: Representa la suma de los pasivos corrientes y no corrientes.
- `Total Pasivo y Patrimonio`: Es la suma del Patrimonio neto y el Total Pasivo.

Cuentas de Resultados

- `Ingresos de las operaciones`: Los ingresos generados por las actividades principales de la empresa, excluyendo los ingresos extraordinarios.
- `Margen Bruto`: Los ingresos de las operaciones menos el costo de los bienes vendidos. Es una medida de la eficiencia de producción y ventas de la empresa.
- `EBITDA`: Acrónimo de "Earnings Before Interest, Taxes, Depreciation, and Amortization" (Ganancias antes de intereses, impuestos, depreciación y amortización). Es un indicador de la rentabilidad operativa de la empresa antes de ciertos factores financieros y contables.

- **`EBIT`**: Acrónimo de "Earnings Before Interest and Taxes" (Ganancias antes de intereses e impuestos). Similar al EBITDA pero sin excluir la depreciación y la amortización.
- **`Resultado antes de impuestos`**: Las ganancias de la empresa antes de que se hayan deducido los impuestos. Indica la rentabilidad antes de la intervención fiscal.
- **`Resultado consolidado del ejercicio`**: Las ganancias totales de la empresa, incluyendo todas sus subsidiarias, después de deducir gastos e impuestos.
- **`Resultado neto atribuible a la sociedad dominante`**: El beneficio neto que corresponde a la empresa matriz después de considerar los intereses de minoritarios y otros factores. Es la parte del beneficio neto que realmente pertenece a la empresa controladora.

Notas importantes

- La información original se encuentra en **miles de euros**.
- El dataset **`'fin_data_energy.xlsx'`** contiene los datos en **euros**.
- En ese Proyecto/Notebook, se toma la información en euros del dataset **`'fin_data_energy.xlsx'`** y se trabaja en **millones de euros**.
- Se identificó en la información original que el campo **`'Total Pasivo'`** incluía los datos del Pasivo y del Patrimonio. Con la finalidad de ahorrar tiempo, el dataset **`'fin_data_energy.xlsx'`** contiene la separación de esta información. Es decir, ahora **`'Total Pasivo'`** contiene sólo los datos de los Pasivos, se creó una nueva columna para el **`'Patrimonio neto'`**, y el **`'Total Pasivo y Patrimonio'`** es la suma de los 2 campos anteriores.

1.3.2 Carga de Datos

[ ]:
```
import pandas as pd

# Cargar los datos desde un archivo Excel
df = pd.read_excel('fin_data_energy.xlsx')
```

1.3.3 Data Wrangling Básico

Conversión de Fecha

[ ]:
```
# Convertir la columna 'Fecha' a tipo datetime
df['Fecha'] = pd.to_datetime(df['Fecha'])
```

Conversión a Millones

[ ]:
```
# Crear un DataFrame auxiliar para convertir los montos a millones de euros
columns_to_convert = [
    "Activo no corriente", "Activo corriente", "Total Activo", "Patrimonio␣
 ↪neto",
    "Pasivo no corriente", "Pasivo corriente", "Total Pasivo", "Total Pasivo y␣
 ↪Patrimonio",
    "Ingresos de las operaciones", "Margen Bruto", "EBITDA", "EBIT",
    "Resultado antes de impuestos", "Resultado consolidado del ejercicio",
```

```
    "Resultado neto atribuible a la sociedad dominante"
]
```

[ ]:
```
# Dividir cada monto por 1,000,000 para convertir a millones y redondear
df_millions = df.copy()
df_millions[columns_to_convert] = df_millions[columns_to_convert].apply(lambda
 ↪x: (x / 1e6).round(2))
```

[ ]:
```
# Crear respaldos de los DataFrames
df_backup = df.copy()
df_millions_backup = df_millions.copy()
```

[ ]:
```
# Guardar el DataFrame 'df_millions_backup' en un archivo CSV separado por pipes
df_millions_backup.to_csv('fin_data_energy_millions.csv', sep='|', index=False)

# Descargar el archivo en computadora local
#from google.colab import files
#files.download('fin_data_energy_millions.csv')
```

[ ]:
```
# Mostrar los primeros registros del DataFrame en millones para verificar
df_millions.head()
```

[ ]:

	Fecha	Empresa	Activo no corriente	Activo corriente	Total Activo	\
0	2023-12-31	Empresa_1	589.38	705.79	1295.17	
1	2022-12-31	Empresa_1	596.77	720.04	1316.81	
2	2021-12-31	Empresa_1	580.41	768.71	1349.12	
3	2020-12-31	Empresa_1	426.37	720.24	1146.62	
4	2019-12-31	Empresa_1	410.17	364.08	774.25	

	Patrimonio neto	Pasivo no corriente	Pasivo corriente	Total Pasivo	\
0	173.25	573.04	548.88	1121.92	
1	135.77	645.25	535.79	1181.04	
2	148.92	636.64	563.55	1200.19	
3	151.04	535.54	460.04	995.57	
4	155.71	250.78	367.76	618.54	

	Total Pasivo y Patrimonio	Ingresos de las operaciones	Margen Bruto	\
0	1295.17	2293.16	236.30	
1	1316.81	2632.98	143.82	
2	1349.12	1689.98	128.75	
3	1146.62	969.30	114.88	
4	774.25	1043.79	126.57	

	EBITDA	EBIT	Resultado antes de impuestos	\
0	96.13	75.13	45.02	
1	54.14	32.29	12.57	
2	52.94	30.97	4.46	

```
3   66.44  46.79                          34.05
4   73.25  47.02                          33.03

   Resultado consolidado del ejercicio  \
0                                31.38
1                                 7.77
2                                 1.25
3                                30.17
4                                31.34

   Resultado neto atribuible a la sociedad dominante
0                                              29.03
1                                               3.54
2                                               2.84
3                                              26.38
4                                              25.42
```

1.4 Análisis Exploratorio de Datos

1.4.1 Análisis Preliminar

[ ]:
```
# Mostrar
df_millions.head()
```

[ ]:
```
        Fecha    Empresa  Activo no corriente  Activo corriente  Total Activo  \
0 2023-12-31  Empresa_1               589.38            705.79       1295.17
1 2022-12-31  Empresa_1               596.77            720.04       1316.81
2 2021-12-31  Empresa_1               580.41            768.71       1349.12
3 2020-12-31  Empresa_1               426.37            720.24       1146.62
4 2019-12-31  Empresa_1               410.17            364.08        774.25

   Patrimonio neto  Pasivo no corriente  Pasivo corriente  Total Pasivo  \
0           173.25               573.04            548.88       1121.92
1           135.77               645.25            535.79       1181.04
2           148.92               636.64            563.55       1200.19
3           151.04               535.54            460.04        995.57
4           155.71               250.78            367.76        618.54

   Total Pasivo y Patrimonio  Ingresos de las operaciones  Margen Bruto  \
0                    1295.17                      2293.16        236.30
1                    1316.81                      2632.98        143.82
2                    1349.12                      1689.98        128.75
3                    1146.62                       969.30        114.88
4                     774.25                      1043.79        126.57

   EBITDA   EBIT  Resultado antes de impuestos  \
0   96.13  75.13                         45.02
```

```
1   54.14  32.29                          12.57
2   52.94  30.97                           4.46
3   66.44  46.79                          34.05
4   73.25  47.02                          33.03

   Resultado consolidado del ejercicio  \
0                                31.38
1                                 7.77
2                                 1.25
3                                30.17
4                                31.34

   Resultado neto atribuible a la sociedad dominante
0                                              29.03
1                                               3.54
2                                               2.84
3                                              26.38
4                                              25.42
```

[ ]:
```
# Filas y Columnas
df_millions.shape
```

[ ]:
```
(13, 17)
```

[ ]:
```
# Info General por Columna
df_millions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13 entries, 0 to 12
Data columns (total 17 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Fecha                                   13 non-null
datetime64[ns]
 1   Empresa                                 13 non-null     object
 2   Activo no corriente                     13 non-null     float64
 3   Activo corriente                        13 non-null     float64
 4   Total Activo                            13 non-null     float64
 5   Patrimonio neto                         13 non-null     float64
 6   Pasivo no corriente                     13 non-null     float64
 7   Pasivo corriente                        13 non-null     float64
 8   Total Pasivo                            13 non-null     float64
 9   Total Pasivo y Patrimonio               13 non-null     float64
 10  Ingresos de las operaciones             13 non-null     float64
 11  Margen Bruto                            13 non-null     float64
 12  EBITDA                                  13 non-null     float64
 13  EBIT                                    13 non-null     float64
```

```
 14  Resultado antes de impuestos                          13 non-null     float64
 15  Resultado consolidado del ejercicio                   13 non-null     float64
 16  Resultado neto atribuible a la sociedad dominante     13 non-null     float64
dtypes: datetime64[ns](1), float64(15), object(1)
memory usage: 1.9+ KB
```

[ ]:
```
# Variables (Columnas)
df_millions.columns
```

[ ]:
```
Index(['Fecha', 'Empresa', 'Activo no corriente', 'Activo corriente',
       'Total Activo', 'Patrimonio neto', 'Pasivo no corriente',
       'Pasivo corriente', 'Total Pasivo', 'Total Pasivo y Patrimonio',
       'Ingresos de las operaciones', 'Margen Bruto', 'EBITDA', 'EBIT',
       'Resultado antes de impuestos', 'Resultado consolidado del ejercicio',
       'Resultado neto atribuible a la sociedad dominante'],
      dtype='object')
```

1.4.2 Missing Values

[ ]:
```
# Missing Values
print("Total de Missing Values por Columna:")
print(df_millions.isnull().sum())
```

```
Total de Missing Values por Columna:
Fecha                                                 0
Empresa                                               0
Activo no corriente                                   0
Activo corriente                                      0
Total Activo                                          0
Patrimonio neto                                       0
Pasivo no corriente                                   0
Pasivo corriente                                      0
Total Pasivo                                          0
Total Pasivo y Patrimonio                             0
Ingresos de las operaciones                           0
Margen Bruto                                          0
EBITDA                                                0
EBIT                                                  0
Resultado antes de impuestos                          0
Resultado consolidado del ejercicio                   0
Resultado neto atribuible a la sociedad dominante     0
dtype: int64
```

1.4.3 Data Cleaning

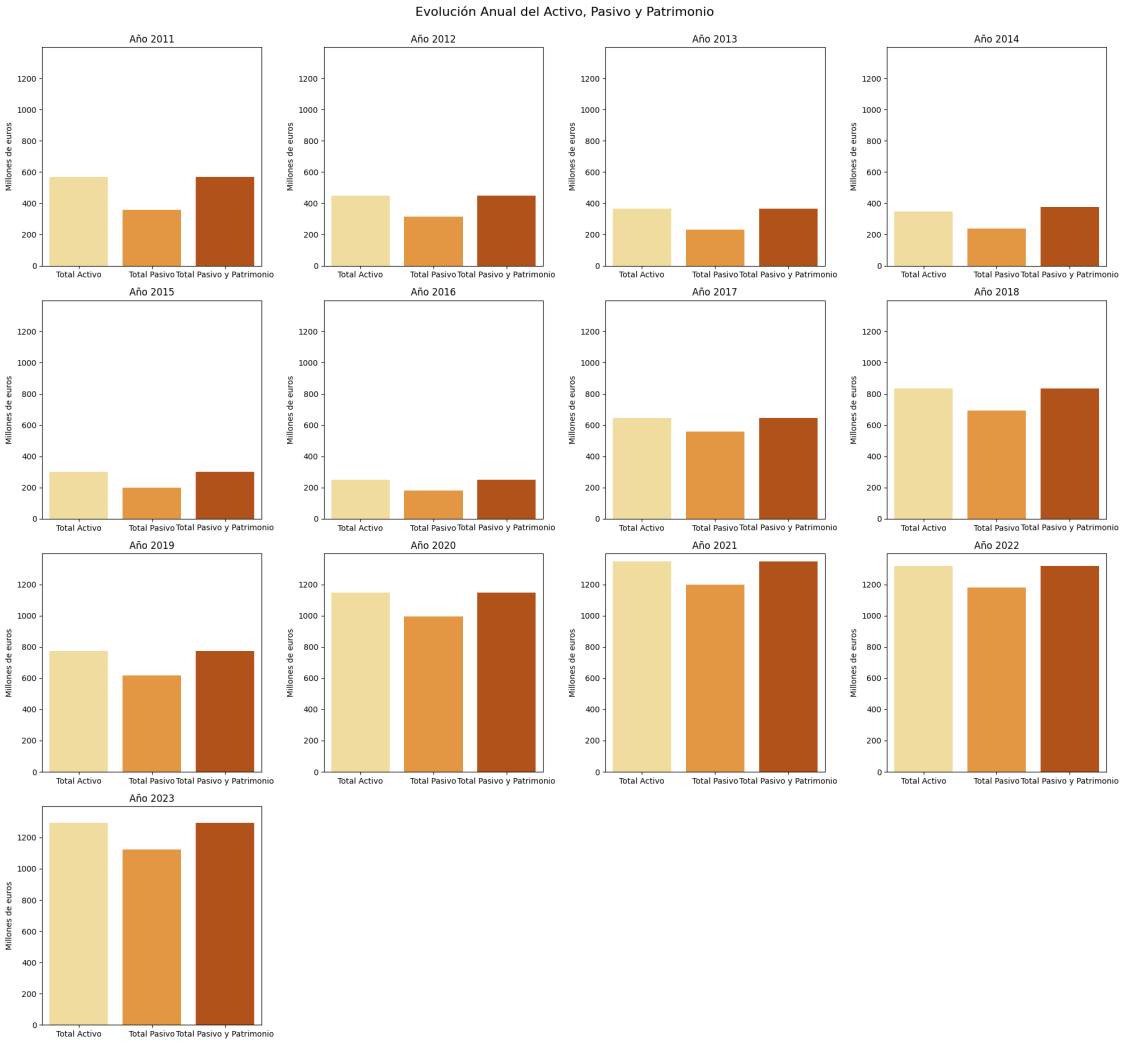
```
[ ]: # Rellenar Missing Values con la Media
     #df_millions.fillna(data.mean(), inplace=True)
```

1.4.4 Barplots

Evolución Anual del Activo, Pasivo y Patrimonio

```
[ ]: import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     # Cargar el archivo CSV en un DataFrame
     df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

     # Asegurarse de que 'Fecha' está en formato datetime
     df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])

     # Extraer el año de la fecha para facilitar la agrupación
     df_millions['Año'] = df_millions['Fecha'].dt.year

     # Seleccionar solo las columnas necesarias para el gráfico
     data = df_millions.melt(id_vars=['Año'], value_vars=['Total Activo', 'Total␣
      ↪Pasivo', 'Total Pasivo y Patrimonio'])

     # Preparar la figura para los subplots
     plt.figure(figsize=(20, 18))  # Ajusta el tamaño de la figura total aquí según␣
      ↪tus necesidades

     # Definir paleta de colores personalizada o usar una predefinida
     #palette = {'Total Activo': '#D9D9D7', 'Total Pasivo': '#B5AB66', 'Total Pasivo␣
      ↪y Patrimonio': '#ECAD30'}
     # palettes predefinidas: 'deep', 'muted', 'bright', 'pastel', 'dark',␣
      ↪'colorblind', 'husl', 'RdYlBu', 'magma', 'YlOrBr', 'crest', 'rocket_r',␣
      ↪'mako'
     palette = 'YlOrBr'

     # Crear un subplot para cada año
     for i, year in enumerate(sorted(data['Año'].unique()), 1):
         plt.subplot(4, 4, i)  # Ajusta las dimensiones de la cuadrícula según el␣
      ↪número de años/subplots
         sns.barplot(
             data=data[data['Año'] == year],
             x='variable',
             y='value',
             hue='variable',
             palette=palette,
```

```
        dodge=False
    )
    plt.title(f'Año {year}')
    plt.xlabel('')  # Ocultar la etiqueta x
    plt.ylabel('Millones de euros')
    #plt.legend(title='Categoría')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data['value'].max() + 50)

# Ajustar el espaciado entre subplots
plt.tight_layout()

# Añadir un título general
plt.suptitle('Evolución Anual del Activo, Pasivo y Patrimonio', fontsize=16,␣
 ↪y=1.02)

# Mostrar el gráfico
plt.show()
```



Evolución Anual de las Cuentas de Resultados

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el archivo CSV en un DataFrame
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Asegurarse de que 'Fecha' está en formato datetime
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])

# Extraer el año de la fecha para facilitar la agrupación
df_millions['Año'] = df_millions['Fecha'].dt.year
```

```python
# Seleccionar las columnas para las cuentas de resultados
results_columns = [
    "Ingresos de las operaciones",
    "Margen Bruto",
    "EBITDA",
    "EBIT",
    "Resultado antes de impuestos",
    "Resultado consolidado del ejercicio"
]

# Crear un DataFrame con los datos para graficar, transformando a formato largo
data_results = df_millions.melt(id_vars=['Año'], value_vars=results_columns)

# Preparar la figura para los subplots
plt.figure(figsize=(20, 20))  # Ajusta el tamaño de la figura total aquí según␣
 ↪tus necesidades

# Definir paleta de colores personalizada o usar una predefinida
# palettes predefinidas: 'deep', 'muted', 'bright', 'pastel', 'dark',␣
 ↪'colorblind', 'husl', 'RdYlBu', 'magma', 'YlOrBr', 'crest', 'rocket_r',␣
 ↪'mako'
palette = 'muted'
colors = {
    "Ingresos de las operaciones": "#ECAD30",
    "Margen Bruto": "#D9D9D7",
    "EBITDA": "#B5AB66",
    "EBIT": "#788C90",
    "Resultado antes de impuestos": "#73BBA1",
    "Resultado consolidado del ejercicio": "#6CB978"
}

# Crear un subplot para cada año
for i, year in enumerate(sorted(data_results['Año'].unique()), 1):
    plt.subplot(4, 4, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de años/subplots
    sns.barplot(
        data=data_results[data_results['Año'] == year],
        y='variable',
        x='value',
        hue='variable',
        palette=colors, # Usar 'colors' o 'palette'
        dodge=False
    )
    plt.title(f'Año {year}')
    plt.xlabel('Millones de euros')
    plt.ylabel('')
    #plt.legend(title='Categoría')
```

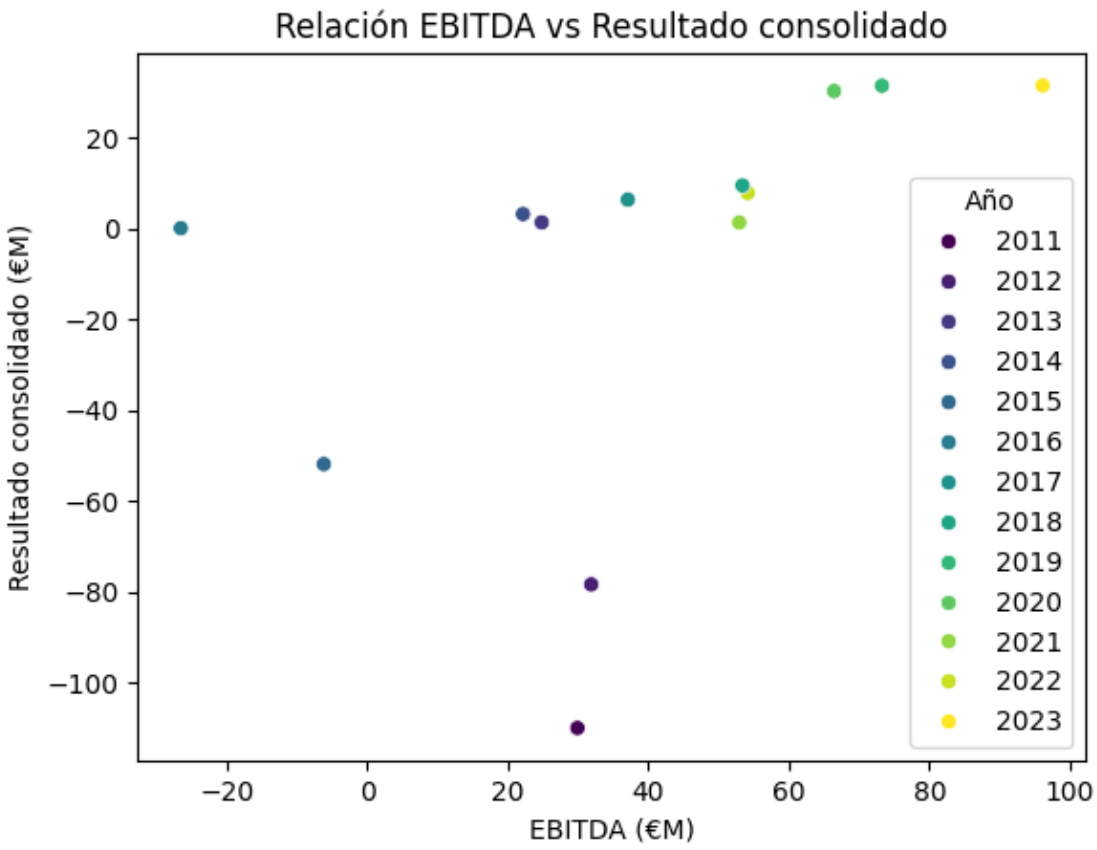
```
    # Ajustar los límites si es necesario para mejor visualización
    plt.xlim(0, data_results['value'].max() * 1.1)

# Ajustar el espaciado entre subplots
plt.tight_layout()

# Añadir un título general
plt.suptitle('Evolución Anual de las Cuentas de Resultados', fontsize=16, y=1.
  ↪02)

# Mostrar el gráfico
plt.show()
```

1.4.5 Scatterplot

EBITDA y Resultado Consolidado Usamos `sns.scatterplot` para crear un **Gráfico de Dispersión (Scatterplot)**. Seleccionamos `"EBITDA"` para el eje X y `"Resultado consolidado del ejercicio"` para el eje Y.

Utilizamos el **año** como la categoría para el color de los puntos, que podría ayudar a visualizar tendencias o cambios a lo largo del tiempo.

[ ]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el archivo CSV en un DataFrame
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Asegurarse de que 'Fecha' está en formato datetime
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])

# Extraer el año de la fecha para facilitar la agrupación
df_millions['Año'] = df_millions['Fecha'].dt.year

# Crear el Scatter plot
scatter_plot = sns.scatterplot(
    x='EBITDA',
    y='Resultado consolidado del ejercicio',
    hue='Año',  # Colorear por año
    data=df_millions,
    palette='viridis',
    legend='full'
)

# Configurar el título y las etiquetas del gráfico
scatter_plot.set_title('Relación EBITDA vs Resultado consolidado')
scatter_plot.set_xlabel('EBITDA (€M)')
scatter_plot.set_ylabel('Resultado consolidado (€M)')

# Mostrar la leyenda
plt.legend(title='Año')

# Mostrar el gráfico
plt.show()
```



Interpretación

Relación entre las Variables

- **Correlación Positiva**: Si los puntos tienden a formar una línea diagonal ascendente de izquierda a derecha, indica una correlación positiva. Esto significa que a medida que el EBITDA aumenta, el Resultado consolidado del ejercicio también tiende a aumentar. Dado que el EBITDA es una medida de rentabilidad antes de intereses, impuestos, depreciación y amortización, un aumento paralelo en el resultado consolidado sugiere una gestión eficiente de los gastos y las deudas.
- **Correlación Negativa**: Una línea diagonal descendente indicaría una correlación negativa, es decir, a medida que el EBITDA aumenta, el Resultado consolidado disminuye. Esto podría indicar problemas como costos de intereses o impuestos inusualmente altos que impactan negativamente las ganancias finales, a pesar de buenos rendimientos operativos.
- **Sin Correlación**: Si los puntos están dispersos sin un patrón claro, significa que no hay una relación lineal aparente entre el EBITDA y el Resultado consolidado del ejercicio. Esto podría ocurrir si otros factores (no capturados por el EBITDA) tienen un impacto significativo en el resultado final.

Impacto del Tiempo (Color por Año)

- **Tendencias a lo Largo del Tiempo**: El color por año puede ayudar a identificar si ha habido cambios en la relación entre EBITDA y resultado consolidado a lo largo del tiempo. Por ejemplo, si los puntos de años recientes tienden a estar más altos o más bajos que en años anteriores, esto puede indicar una mejora o deterioro en la eficiencia operativa o financiera de la empresa.
- **Consistencia Temporal**: Si los colores muestran que los puntos para cada año están agrupados juntos, esto podría indicar que la relación entre las variables es estable a lo largo del tiempo. Si los grupos varían significativamente de un año a otro, esto podría indicar volatilidad o cambios significativos en las operaciones o en el entorno de mercado.

Análisis Adicional

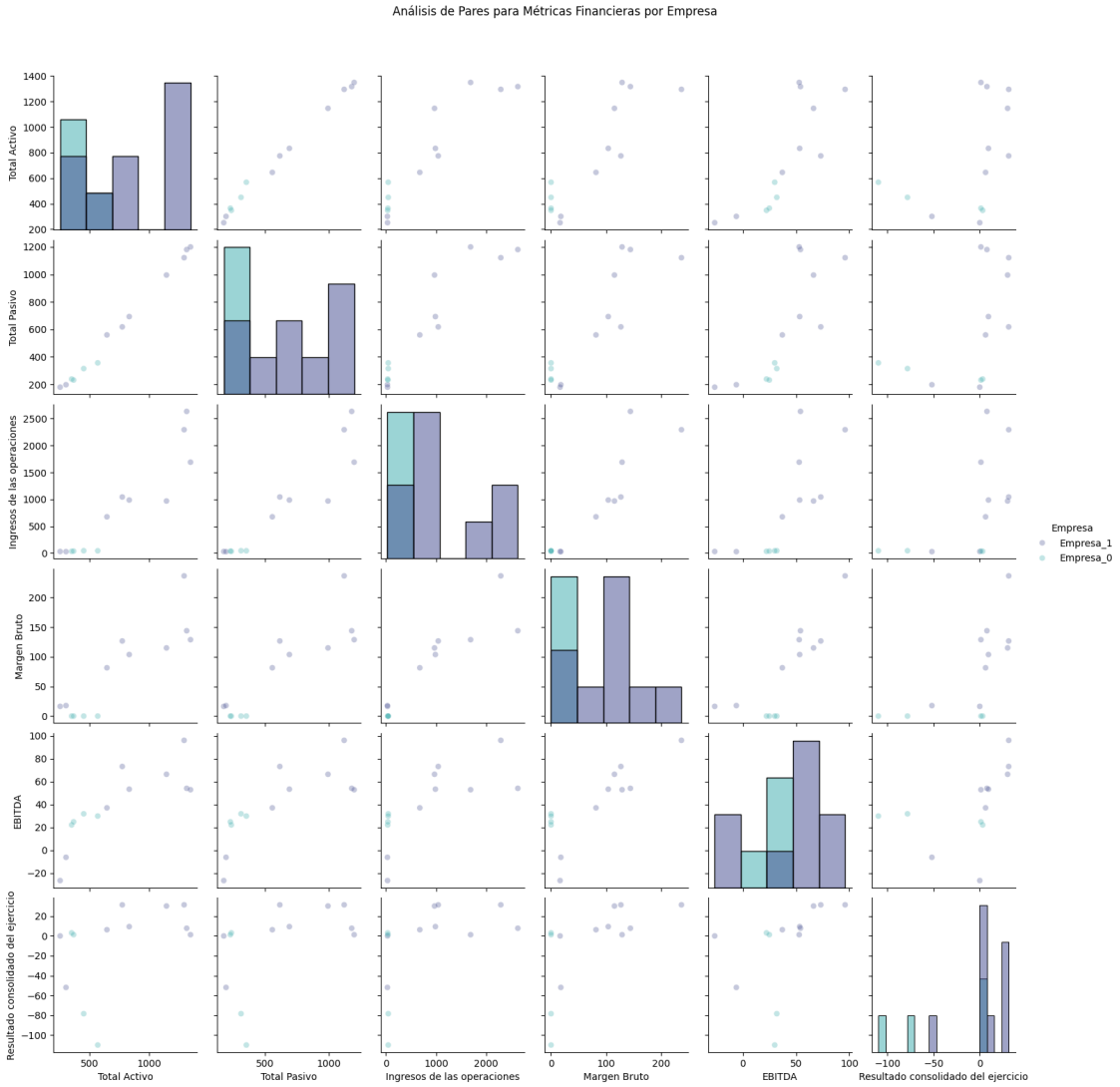
- **Análisis por Segmentos o Divisiones**: Si hay cambios significativos en la estructura empresarial, como adquisiciones o ventas de divisiones, esto también podría afectar la relación entre EBITDA y el resultado consolidado.
- **Influencia de Factores Externos**: Eventos económicos, cambios en la legislación fiscal o fluctuaciones en los mercados financieros también pueden influir en esta relación y deberían considerarse al interpretar los resultados.

1.4.6 Pairplot

Activo, Pasivo, Cuentas de Resultados

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar el archivo CSV en un DataFrame
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Asegurarse de que 'Fecha' está en formato datetime
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])

# Extraer el año de la fecha para facilitar la agrupación
df_millions['Año'] = df_millions['Fecha'].dt.year

# Seleccionar solo algunas columnas numéricas para el análisis
columns_to_plot = [
    "Total Activo", "Total Pasivo",
"Ingresos de las operaciones",
"Margen Bruto",
"EBITDA",
    "Resultado consolidado del ejercicio"
]

# Crear el pair plot
```

```
pair_plot = sns.pairplot(
    df_millions[columns_to_plot + ['Empresa']],
    hue='Empresa',
    palette='mako',  #
    diag_kind='hist',  # Usar 'hist', o 'kde' para gráficos de densidad
    markers='o',  # Tipo de marcador (opcional)
    plot_kws={'alpha': 0.3}  # Transparencia de los puntos
)

# Configurar el título general para el pairplot
pair_plot.fig.suptitle('Análisis de Pares para Métricas Financieras por Empresa', y=1.05)

# Mostrar el gráfico
plt.show()
```



Interpretación

- **Distribuciones**

  Los gráficos en la ***diagonal*** ayudarán a entender la distribución individual de cada variable. Por ejemplo, un histograma te mostrará la frecuencia de diferentes rangos de valores, mientras que un gráfico KDE te mostrará la densidad de probabilidad.

- **Relaciones Bivariadas**

  Los gráficos ***fuera de la diagonal*** muestran la relación entre pares de variables. Una dispersión en forma de línea ascendente indica una correlación positiva, mientras que una línea descendente indica una correlación negativa. La densidad o dispersión de los puntos puede indicar la variabilidad de la relación.

- **Colores por Categorías**

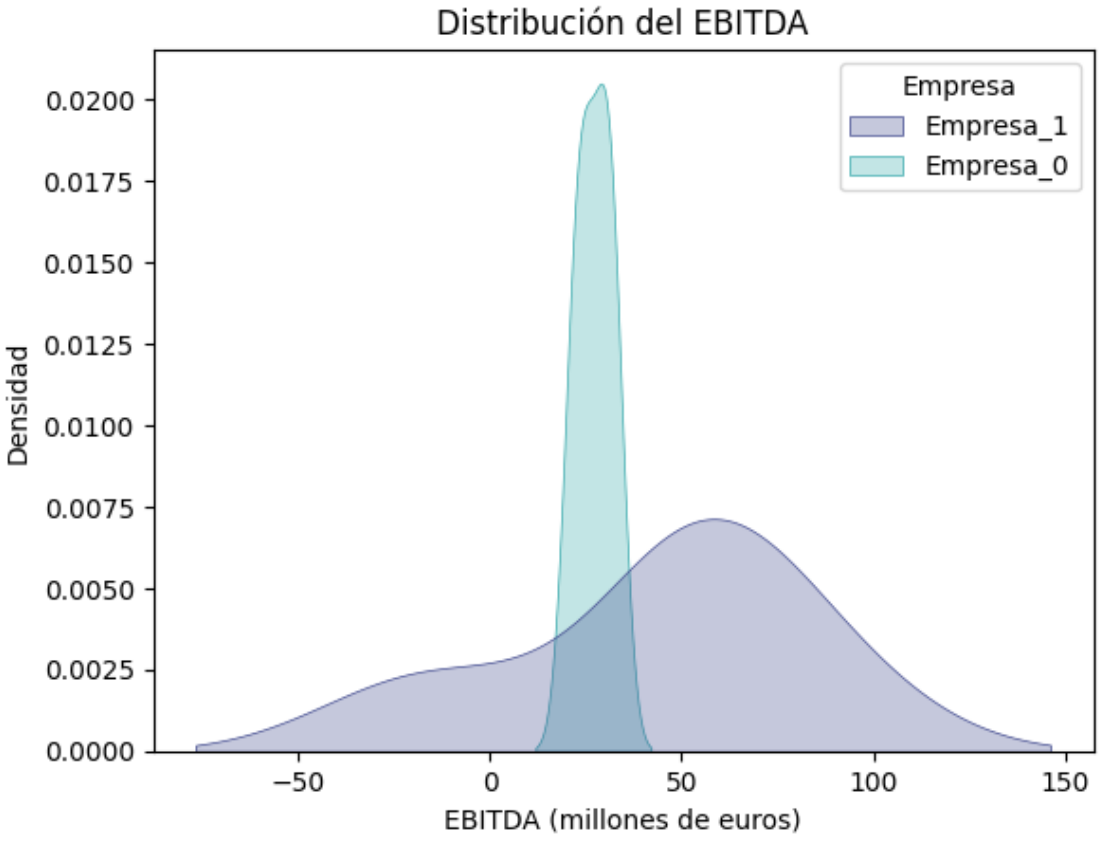
  Usar el **año** como **hue**, ayudará a visualizar cómo estas relaciones y distribuciones pueden haber cambiado con el tiempo.

1.4.7 KDE Plot

EBITDA

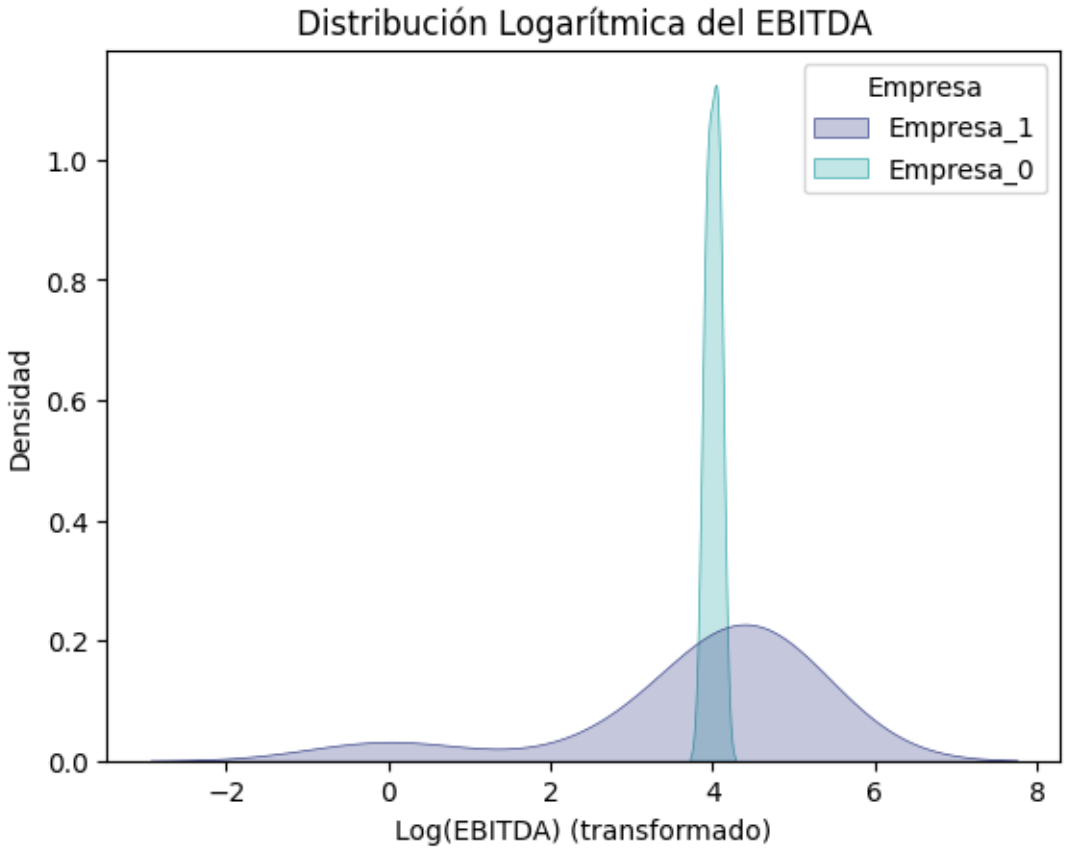
[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Definir los límites basados en los valores mínimos y máximos observados en␣
 ↪tus datos
min_ebitda, max_ebitda = df_millions['EBITDA'].min(), df_millions['EBITDA'].
 ↪max()

# Gráfico KDE
sns.kdeplot(
    data=df_millions,
    x="EBITDA",
    fill=True,
    clip=(min_ebitda - 50, max_ebitda + 50),
    hue="Empresa",
```

```
    palette="mako",
    alpha=0.3,
    linewidth=0.5
)
plt.title('Distribución del EBITDA')
plt.xlabel('EBITDA (millones de euros)')
plt.ylabel('Densidad')
plt.show()
```



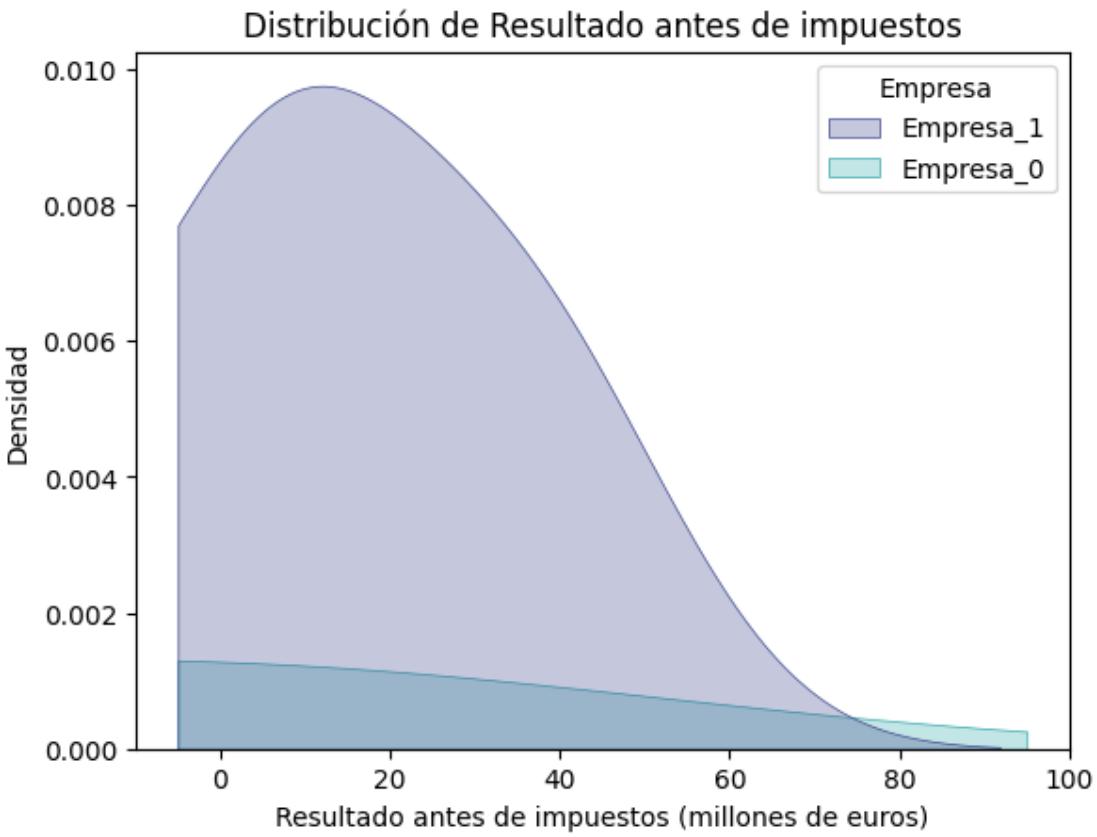
[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
  ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
```

```
df_millions['Año'] = df_millions['Fecha'].dt.year

# Aplicar transformación logarítmica con ajuste para manejar valores negativos
df_millions['EBITDA_log'] = np.log(df_millions['EBITDA'] -␣
 ↪df_millions['EBITDA'].min() + 1)

# Crear Gráfico KDE
sns.kdeplot(
    data=df_millions,
    x="EBITDA_log",
    fill=True,
    hue="Empresa",
    palette="mako",
    alpha=0.3,
    linewidth=0.5
)
plt.title('Distribución Logarítmica del EBITDA')
plt.xlabel('Log(EBITDA) (transformado)')
plt.ylabel('Densidad')
plt.show()
```



Resultado antes de impuestos

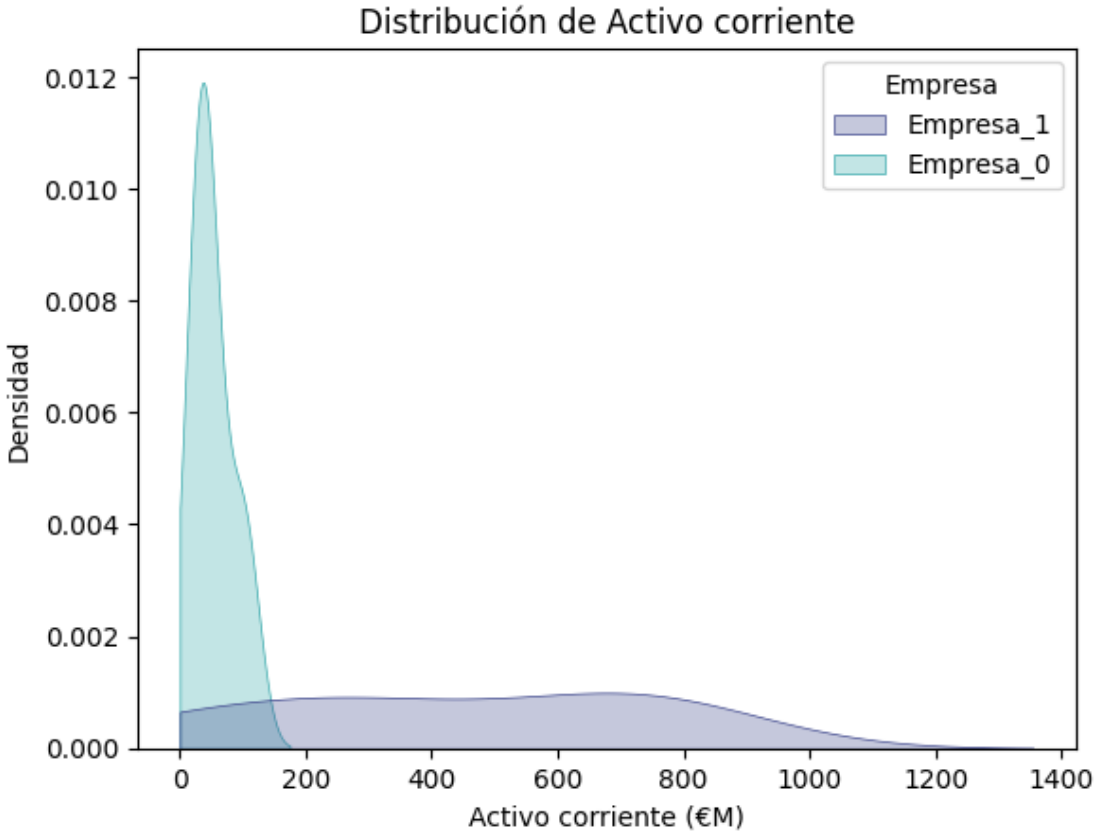
[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Definir los límites basados en los valores mínimos y máximos observados en␣
 ↪tus datos
min_resultado_ai, min_resultado_ai = df_millions['Resultado antes de␣
 ↪impuestos'].min(), df_millions['Resultado antes de impuestos'].max()

# Gráfico KDE
sns.kdeplot(
    data=df_millions,
    x="Resultado antes de impuestos",
    fill=True,
    clip=(min_resultado_ai - 50, min_resultado_ai + 50),
    hue="Empresa",
    palette="mako",
    alpha=0.3,
    linewidth=0.5
)
plt.title('Distribución de Resultado antes de impuestos')
plt.xlabel('Resultado antes de impuestos (millones de euros)')
plt.ylabel('Densidad')
plt.show()
```



Activo Corriente

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Gráfico KDE
sns.kdeplot(
    data=df_millions,
    x="Activo corriente",
    hue="Empresa",
    fill=True,
    common_norm=False,
```

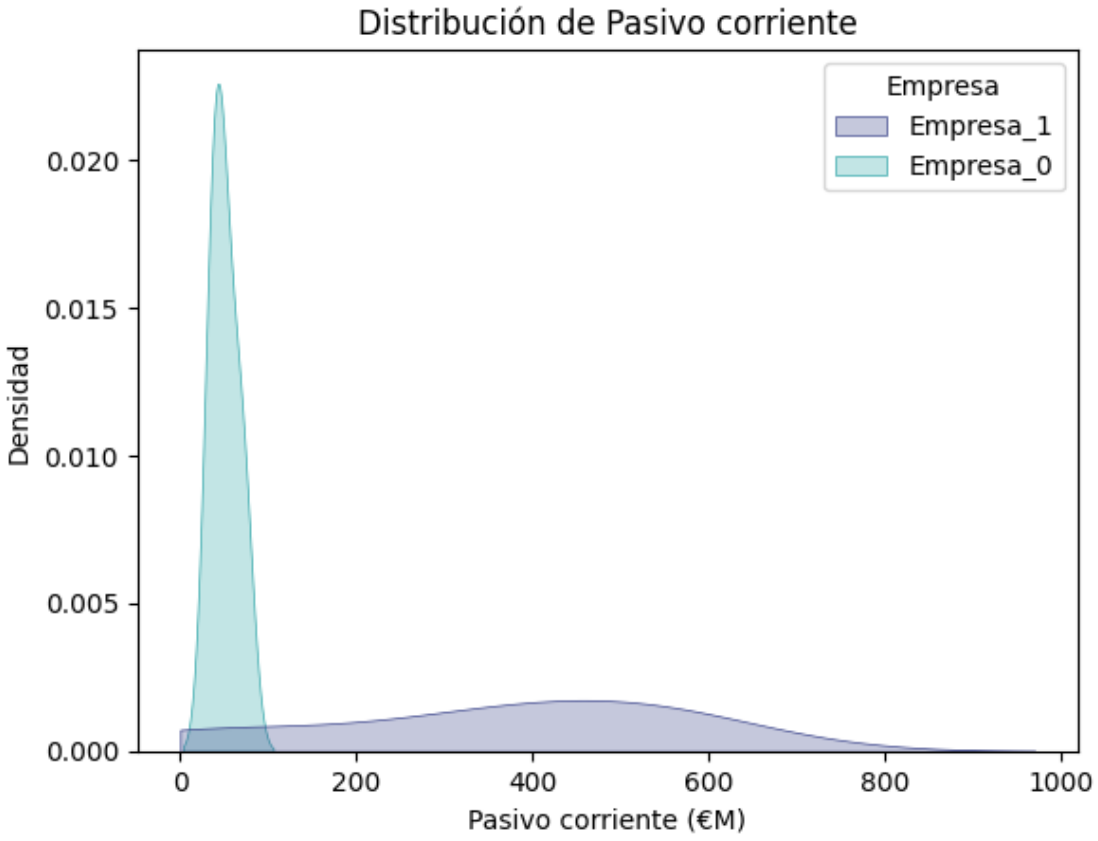
```
    palette="mako",
    alpha=0.3,
    linewidth=0.5,
    clip=(0, None)  # Esto evitará que el KDE vaya a valores negativos
)
plt.title('Distribución de Activo corriente')
plt.xlabel('Activo corriente (€M)')
plt.ylabel('Densidad')
plt.show()
```



Pasivo Corriente

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')
```

```python
# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Gráfico KDE
sns.kdeplot(
    data=df_millions,
    x="Pasivo corriente",
    hue="Empresa",
    fill=True,
    common_norm=False,
    palette="mako",
    alpha=0.3,
    linewidth=0.5,
    clip=(0, None)  # Esto evitará que el KDE vaya a valores negativos
)
plt.title('Distribución de Pasivo corriente')
plt.xlabel('Pasivo corriente (€M)')
plt.ylabel('Densidad')
plt.show()
```

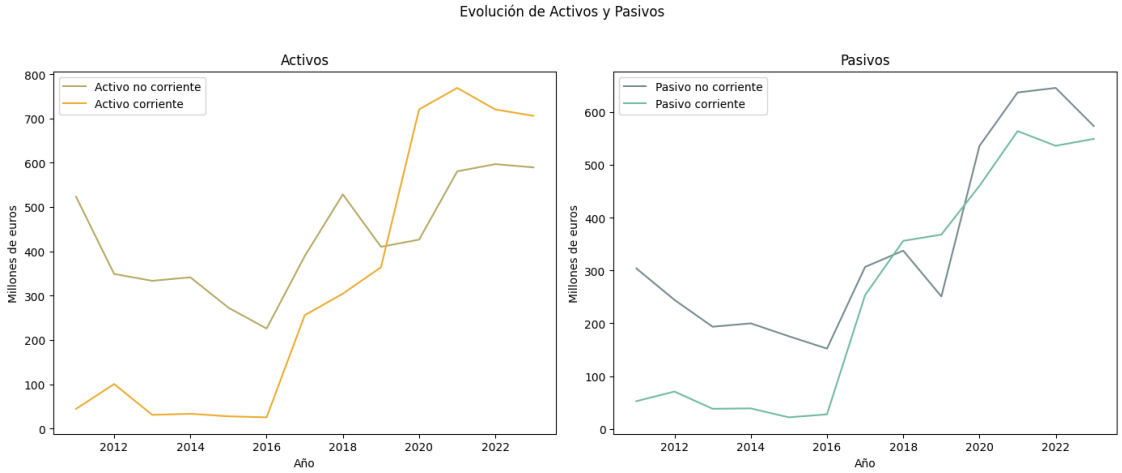


1.4.8 Lineplots

Tendencia de Activos y Pasivos

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Configurar la figura y los ejes para dos subgráficos
fig, axes = plt.subplots(1, 2, figsize=(14, 6))  # 1 fila, 2 columnas
fig.suptitle('Evolución de Activos y Pasivos')

# Variables agrupadas
activos = ["Activo no corriente", "Activo corriente"]
pasivos = ["Pasivo no corriente", "Pasivo corriente"]

# Colores para las líneas de activos y pasivos
activos_colors = ['#B5AB66', '#ECAD30']
pasivos_colors = ['#788C90', '#73BBA1']

# Graficar activos
for variable, color in zip(activos, activos_colors):
    sns.lineplot(data=df_millions, x='Año', y=variable, ax=axes[0],␣
 ↪label=variable, color=color)
axes[0].set_title('Activos')
axes[0].set_ylabel('Millones de euros')
axes[0].set_xlabel('Año')
axes[0].legend()

# Graficar pasivos
for variable, color in zip(pasivos, pasivos_colors):
    sns.lineplot(data=df_millions, x='Año', y=variable, ax=axes[1],␣
 ↪label=variable, color=color)
axes[1].set_title('Pasivos')
axes[1].set_ylabel('Millones de euros')
axes[1].set_xlabel('Año')
axes[1].legend()
```

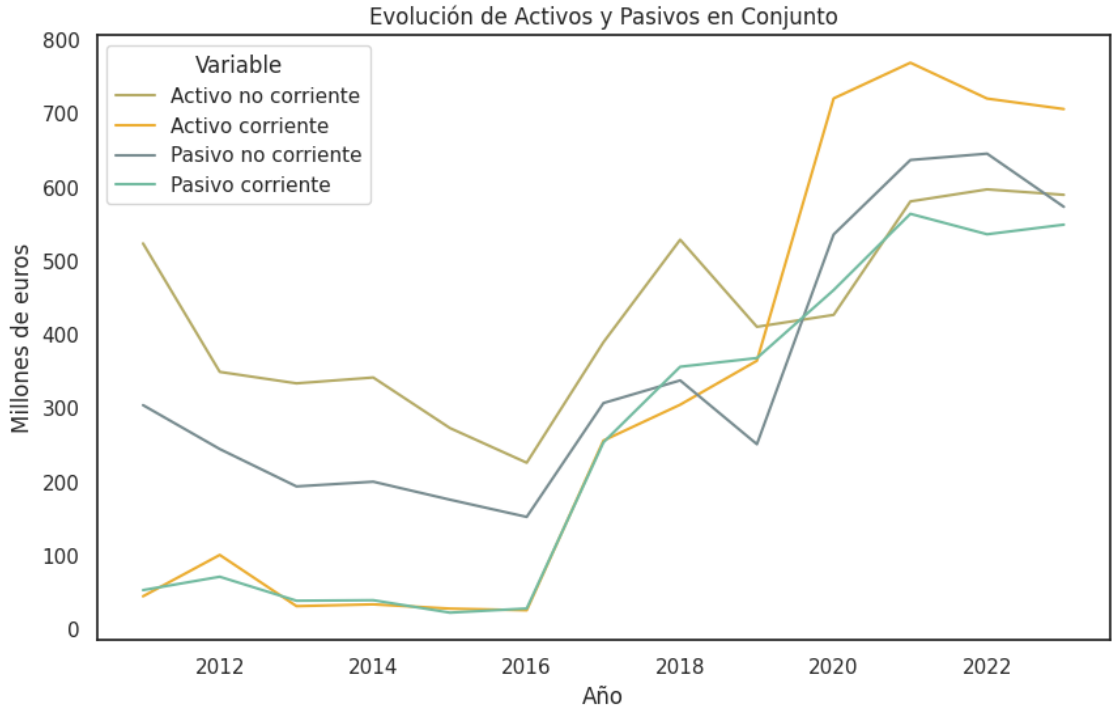
```
# Ajustar layout y mostrar el gráfico
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```



Tendencia de Activos y Pasivos en Conjunto

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Establecer el estilo de Seaborn para un fondo blanco
sns.set(style="white")

# Configurar la figura
plt.figure(figsize=(10, 6))
plt.title('Evolución de Activos y Pasivos en Conjunto')

# Variables y colores
variables = ["Activo no corriente", "Activo corriente", "Pasivo no corriente",␣
 ↪"Pasivo corriente"]
colors = ['#B5AB66', '#ECAD30', '#788C90', '#73BBA1']  # Colores para cada␣
 ↪variable
```

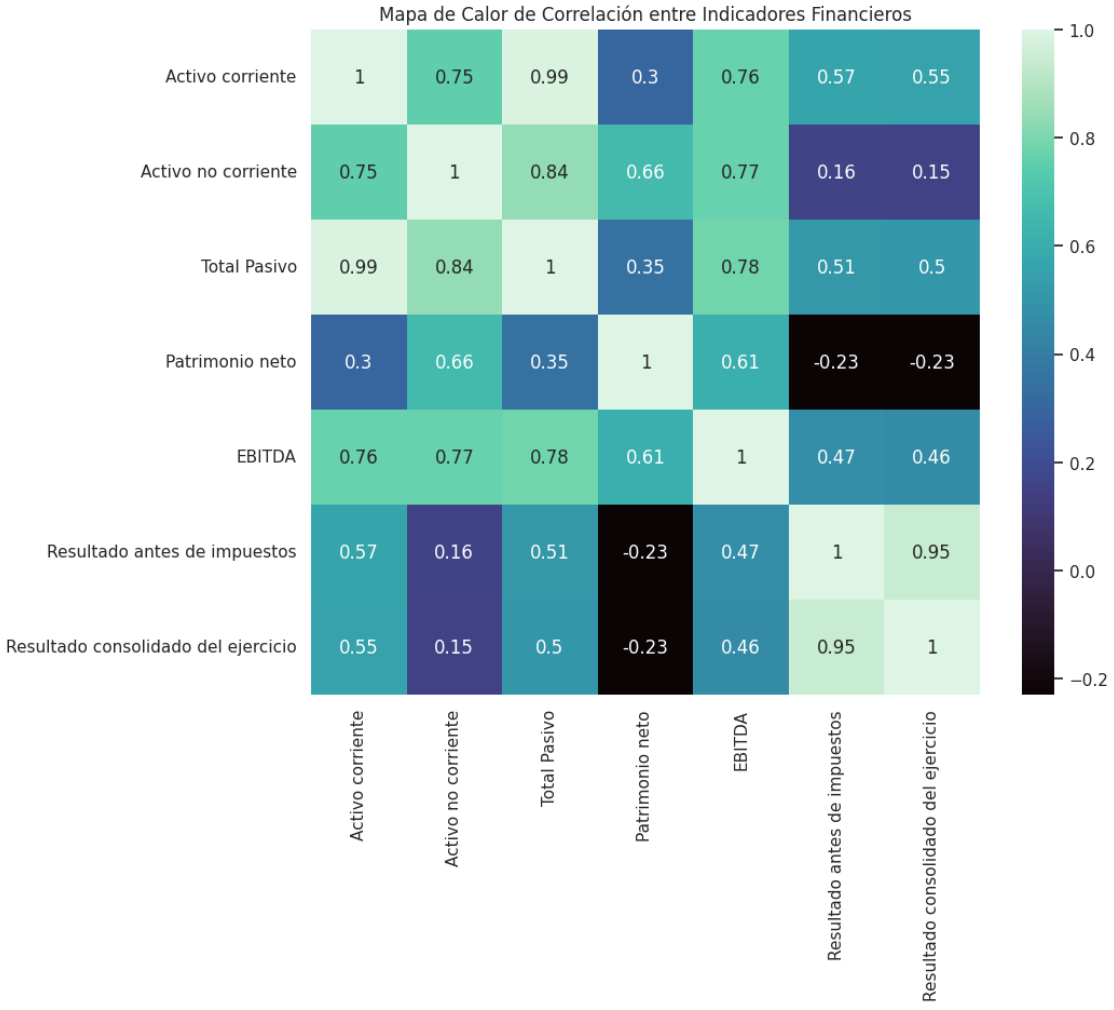
```
# Graficar cada variable
for variable, color in zip(variables, colors):
    sns.lineplot(data=df_millions, x='Año', y=variable, label=variable,␣
 ↪color=color)

plt.xlabel('Año')
plt.ylabel('Millones de euros')
plt.legend(title='Variable')
plt.grid(False)  # Añade una cuadrícula
plt.show()
```



1.4.9 Heatmap-Correlaciones

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
```

```
df_millions['Año'] = df_millions['Fecha'].dt.year

# Correlación y mapa de calor
corr_matrix = df_millions[['Activo corriente', 'Activo no corriente', 'Total␣
 ↪Pasivo', 'Patrimonio neto', 'EBITDA', 'Resultado antes de impuestos',
       'Resultado consolidado del ejercicio']].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='mako')
plt.title('Mapa de Calor de Correlación entre Indicadores Financieros')
plt.show()
```



[61]:
```
# Extraer Top 10 de Correlaciones
import pandas as pd

# Supongamos que corr_matrix ya está calculada
# Flatten la matriz de correlación y conviértela en DataFrame
```

```
corr_df = corr_matrix.stack().reset_index()
corr_df.columns = ['Variable1', 'Variable2', 'Correlation']

# Eliminar la correlación de una variable consigo misma
corr_df = corr_df[corr_df['Variable1'] != corr_df['Variable2']]

# Eliminar duplicados
corr_df['Variable Pairs'] = corr_df.apply(lambda x: frozenset([x['Variable1'],␣
 ↪x['Variable2']]), axis=1)
corr_df = corr_df.drop_duplicates(subset=['Variable Pairs'])
corr_df.drop('Variable Pairs', axis=1, inplace=True)

# Ordenar por correlación para obtener las más altas y más bajas
top_positive_corr = corr_df.sort_values(by='Correlation', ascending=False).
 ↪head(10)
top_negative_corr = corr_df.sort_values(by='Correlation', ascending=True).
 ↪head(10)
```

[62]:
```
# Mostrar Correlaciones Positivas
print("Top 10 Positive Correlations:")
top_positive_corr
```

```
Top 10 Positive Correlations:
```

[62]:

	Variable1	Variable2	Correlation
2	Activo corriente	Total Pasivo	0.987084
41	Resultado antes de impuestos	Resultado consolidado del ejercicio	0.946926
9	Activo no corriente	Total Pasivo	0.838296
18	Total Pasivo	EBITDA	0.781211
11	Activo no corriente	EBITDA	0.768913
4	Activo corriente	EBITDA	0.764940
1	Activo corriente	Activo no corriente	0.750622
10	Activo no corriente	Patrimonio neto	0.660997
25	Patrimonio neto	EBITDA	0.611445
5	Activo corriente	Resultado antes de impuestos	0.567424

[63]:
```
# Mostrar Correlaciones Negativas
print("\nTop 10 Negative Correlations:")
top_negative_corr
```

```
Top 10 Negative Correlations:
```

[63]:

	Variable1	Variable2	Correlation
27	Patrimonio neto	Resultado consolidado del ejercicio	-0.229499
26	Patrimonio neto	Resultado antes de impuestos	-0.228116
13	Activo no corriente	Resultado consolidado del ejercicio	0.153821
12	Activo no corriente	Resultado antes de impuestos	0.159499
3	Activo corriente	Patrimonio neto	0.296798
17	Total Pasivo	Patrimonio neto	0.347526
34	EBITDA	Resultado consolidado del ejercicio	0.456130
33	EBITDA	Resultado antes de impuestos	0.470061
20	Total Pasivo	Resultado consolidado del ejercicio	0.500675
19	Total Pasivo	Resultado antes de impuestos	0.512482

1.5 Proyecciones

1.5.1 Modelo de Redes Neuronales

Preparación

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import numpy as np

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Filtrar las columnas numéricas necesarias (excluyendo 'Fecha' y otras no␣
 ↪numéricas)
numerical_columns = df_millions.columns.difference(['Fecha', 'Empresa'])
```

[ ]:
```
# Preparar datos de entrada y salida
X = df_millions[numerical_columns]
y = df_millions[numerical_columns]  # Predecimos las mismas columnas que usamos␣
 ↪como entrada
```

```
# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Definición del Modelo Se define un Modelo de Redes Neuronales con **Dropout**.

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Definir el modelo con Dropout
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dropout(0.2),  # Añadir capa de Dropout para reducir el sobreajuste
    Dense(128, activation='relu'),
    Dropout(0.2),  # Otra capa de Dropout
    Dense(64, activation='relu'),
    Dense(y_train.shape[1])  # Capa de salida sin función de activación para␣
 ↪regresión
])

# Compilar el modelo
model.compile(optimizer='adam', loss='mse')

# Mostrar el summary del modelo
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               2176

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0
```

```
 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 16)                1040

=================================================================
Total params: 27984 (109.31 KB)
Trainable params: 27984 (109.31 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Entrenamiento Entrenamiento con **EarlyStopping** para prevenir sobreajuste.

[ ]:
```python
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping para prevenir sobreajuste
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,  # Número de epochs a esperar después de la última mejora
    restore_best_weights=True  # Restaura los pesos del mejor modelo observado
)

# Entrenar el modelo con early stopping
history = model.fit(
    X_train_scaled, y_train,
    epochs=150,
    validation_split=0.1,
    callbacks=[early_stopping]
)
```

```
Epoch 1/150
1/1 [==============================] - 4s 4s/step - loss: 517462.0000 -
val_loss: 378814.0000
Epoch 2/150
1/1 [==============================] - 0s 34ms/step - loss: 517415.9375 -
val_loss: 378795.1875
Epoch 3/150
1/1 [==============================] - 0s 32ms/step - loss: 517316.4375 -
val_loss: 378779.3750
Epoch 4/150
1/1 [==============================] - 0s 37ms/step - loss: 517266.7812 -
val_loss: 378764.6250
Epoch 5/150
1/1 [==============================] - 0s 33ms/step - loss: 517233.5000 -
val_loss: 378750.8438
Epoch 6/150
1/1 [==============================] - 0s 33ms/step - loss: 517149.7188 -
val_loss: 378736.2188
Epoch 7/150
```

```
1/1 [==============================] - 0s 31ms/step - loss: 517126.9375 -
val_loss: 378720.7500
Epoch 8/150
1/1 [==============================] - 0s 40ms/step - loss: 517062.4375 -
val_loss: 378703.2188
Epoch 9/150
1/1 [==============================] - 0s 32ms/step - loss: 517029.0000 -
val_loss: 378684.8750
Epoch 10/150
1/1 [==============================] - 0s 34ms/step - loss: 516897.2812 -
val_loss: 378664.1562
Epoch 11/150
1/1 [==============================] - 0s 32ms/step - loss: 516835.5000 -
val_loss: 378641.7188
Epoch 12/150
1/1 [==============================] - 0s 31ms/step - loss: 516722.5000 -
val_loss: 378616.9375
Epoch 13/150
1/1 [==============================] - 0s 31ms/step - loss: 516621.6250 -
val_loss: 378589.1875
Epoch 14/150
1/1 [==============================] - 0s 31ms/step - loss: 516551.3438 -
val_loss: 378558.4688
Epoch 15/150
1/1 [==============================] - 0s 35ms/step - loss: 516502.0000 -
val_loss: 378525.2500
Epoch 16/150
1/1 [==============================] - 0s 34ms/step - loss: 516293.0625 -
val_loss: 378488.9688
Epoch 17/150
1/1 [==============================] - 0s 36ms/step - loss: 516247.5000 -
val_loss: 378450.0625
Epoch 18/150
1/1 [==============================] - 0s 33ms/step - loss: 516034.0000 -
val_loss: 378408.3750
Epoch 19/150
1/1 [==============================] - 0s 31ms/step - loss: 515784.3750 -
val_loss: 378363.4688
Epoch 20/150
1/1 [==============================] - 0s 32ms/step - loss: 515784.0000 -
val_loss: 378314.4375
Epoch 21/150
1/1 [==============================] - 0s 31ms/step - loss: 515449.8750 -
val_loss: 378260.5938
Epoch 22/150
1/1 [==============================] - 0s 32ms/step - loss: 515361.7188 -
val_loss: 378201.2812
Epoch 23/150
```

```
1/1 [==============================] - 0s 42ms/step - loss: 515073.3438 -
val_loss: 378136.3750
Epoch 24/150
1/1 [==============================] - 0s 34ms/step - loss: 514669.2812 -
val_loss: 378065.3750
Epoch 25/150
1/1 [==============================] - 0s 35ms/step - loss: 514625.3438 -
val_loss: 377987.5000
Epoch 26/150
1/1 [==============================] - 0s 31ms/step - loss: 513855.0000 -
val_loss: 377902.7188
Epoch 27/150
1/1 [==============================] - 0s 33ms/step - loss: 513883.7188 -
val_loss: 377811.4688
Epoch 28/150
1/1 [==============================] - 0s 34ms/step - loss: 513402.0625 -
val_loss: 377712.0000
Epoch 29/150
1/1 [==============================] - 0s 33ms/step - loss: 512724.7812 -
val_loss: 377603.1562
Epoch 30/150
1/1 [==============================] - 0s 36ms/step - loss: 512391.5625 -
val_loss: 377484.8750
Epoch 31/150
1/1 [==============================] - 0s 33ms/step - loss: 512156.1250 -
val_loss: 377356.3750
Epoch 32/150
1/1 [==============================] - 0s 50ms/step - loss: 511473.0625 -
val_loss: 377215.4375
Epoch 33/150
1/1 [==============================] - 0s 43ms/step - loss: 510809.5625 -
val_loss: 377060.4062
Epoch 34/150
1/1 [==============================] - 0s 42ms/step - loss: 509642.1250 -
val_loss: 376889.6875
Epoch 35/150
1/1 [==============================] - 0s 44ms/step - loss: 509117.6562 -
val_loss: 376704.3750
Epoch 36/150
1/1 [==============================] - 0s 32ms/step - loss: 508447.3750 -
val_loss: 376502.9062
Epoch 37/150
1/1 [==============================] - 0s 33ms/step - loss: 507406.3438 -
val_loss: 376284.4062
Epoch 38/150
1/1 [==============================] - 0s 32ms/step - loss: 506169.3438 -
val_loss: 376048.1875
Epoch 39/150
```

```
1/1 [==============================] - 0s 34ms/step - loss: 504791.0625 -
val_loss: 375793.0000
Epoch 40/150
1/1 [==============================] - 0s 37ms/step - loss: 504299.7188 -
val_loss: 375516.5625
Epoch 41/150
1/1 [==============================] - 0s 30ms/step - loss: 501664.0625 -
val_loss: 375217.6875
Epoch 42/150
1/1 [==============================] - 0s 33ms/step - loss: 502138.5000 -
val_loss: 374895.6875
Epoch 43/150
1/1 [==============================] - 0s 36ms/step - loss: 498849.3438 -
val_loss: 374548.0938
Epoch 44/150
1/1 [==============================] - 0s 37ms/step - loss: 497311.7812 -
val_loss: 374171.9688
Epoch 45/150
1/1 [==============================] - 0s 51ms/step - loss: 495168.7188 -
val_loss: 373766.4375
Epoch 46/150
1/1 [==============================] - 0s 32ms/step - loss: 494186.6250 -
val_loss: 373329.5000
Epoch 47/150
1/1 [==============================] - 0s 35ms/step - loss: 490540.3750 -
val_loss: 372857.8438
Epoch 48/150
1/1 [==============================] - 0s 50ms/step - loss: 485743.8750 -
val_loss: 372349.8750
Epoch 49/150
1/1 [==============================] - 0s 33ms/step - loss: 484344.7188 -
val_loss: 371803.6875
Epoch 50/150
1/1 [==============================] - 0s 32ms/step - loss: 481538.1562 -
val_loss: 371218.7500
Epoch 51/150
1/1 [==============================] - 0s 35ms/step - loss: 480884.0000 -
val_loss: 370595.2812
Epoch 52/150
1/1 [==============================] - 0s 33ms/step - loss: 475140.3750 -
val_loss: 369928.5625
Epoch 53/150
1/1 [==============================] - 0s 32ms/step - loss: 473139.6562 -
val_loss: 369219.0312
Epoch 54/150
1/1 [==============================] - 0s 31ms/step - loss: 467066.9375 -
val_loss: 368460.9375
Epoch 55/150
```

```
1/1 [==============================] - 0s 31ms/step - loss: 458357.4062 -
val_loss: 367647.9375
Epoch 56/150
1/1 [==============================] - 0s 32ms/step - loss: 457387.8438 -
val_loss: 366777.4375
Epoch 57/150
1/1 [==============================] - 0s 33ms/step - loss: 452690.1250 -
val_loss: 365848.3438
Epoch 58/150
1/1 [==============================] - 0s 31ms/step - loss: 444962.3750 -
val_loss: 364861.0000
Epoch 59/150
1/1 [==============================] - 0s 37ms/step - loss: 444728.1250 -
val_loss: 363815.7500
Epoch 60/150
1/1 [==============================] - 0s 40ms/step - loss: 437654.2188 -
val_loss: 362707.3125
Epoch 61/150
1/1 [==============================] - 0s 48ms/step - loss: 425341.8438 -
val_loss: 361528.4375
Epoch 62/150
1/1 [==============================] - 0s 34ms/step - loss: 421617.4688 -
val_loss: 360283.8125
Epoch 63/150
1/1 [==============================] - 0s 32ms/step - loss: 412985.9688 -
val_loss: 358966.6875
Epoch 64/150
1/1 [==============================] - 0s 32ms/step - loss: 407591.5000 -
val_loss: 357579.7500
Epoch 65/150
1/1 [==============================] - 0s 31ms/step - loss: 397794.8438 -
val_loss: 356116.8750
Epoch 66/150
1/1 [==============================] - 0s 31ms/step - loss: 393440.3438 -
val_loss: 354577.1562
Epoch 67/150
1/1 [==============================] - 0s 33ms/step - loss: 388084.1250 -
val_loss: 352964.5938
Epoch 68/150
1/1 [==============================] - 0s 34ms/step - loss: 372085.6562 -
val_loss: 351276.4062
Epoch 69/150
1/1 [==============================] - 0s 33ms/step - loss: 355536.8750 -
val_loss: 349507.0000
Epoch 70/150
1/1 [==============================] - 0s 32ms/step - loss: 349294.4375 -
val_loss: 347651.7500
Epoch 71/150
```

```
1/1 [==============================] - 0s 33ms/step - loss: 335145.5625 -
val_loss: 345719.8125
Epoch 72/150
1/1 [==============================] - 0s 34ms/step - loss: 325598.8438 -
val_loss: 343711.8125
Epoch 73/150
1/1 [==============================] - 0s 32ms/step - loss: 314067.6250 -
val_loss: 341630.7188
Epoch 74/150
1/1 [==============================] - 0s 32ms/step - loss: 304834.2188 -
val_loss: 339481.6250
Epoch 75/150
1/1 [==============================] - 0s 33ms/step - loss: 300075.3438 -
val_loss: 337247.0625
Epoch 76/150
1/1 [==============================] - 0s 32ms/step - loss: 281053.5625 -
val_loss: 334938.9375
Epoch 77/150
1/1 [==============================] - 0s 34ms/step - loss: 276702.2812 -
val_loss: 332565.8125
Epoch 78/150
1/1 [==============================] - 0s 33ms/step - loss: 264021.9688 -
val_loss: 330153.0625
Epoch 79/150
1/1 [==============================] - 0s 40ms/step - loss: 241966.9688 -
val_loss: 327700.1562
Epoch 80/150
1/1 [==============================] - 0s 43ms/step - loss: 246821.5000 -
val_loss: 325217.3125
Epoch 81/150
1/1 [==============================] - 0s 33ms/step - loss: 228509.6406 -
val_loss: 322715.5625
Epoch 82/150
1/1 [==============================] - 0s 31ms/step - loss: 215689.5156 -
val_loss: 320223.7500
Epoch 83/150
1/1 [==============================] - 0s 34ms/step - loss: 210456.9375 -
val_loss: 317791.5938
Epoch 84/150
1/1 [==============================] - 0s 31ms/step - loss: 198225.2500 -
val_loss: 315394.5625
Epoch 85/150
1/1 [==============================] - 0s 32ms/step - loss: 189616.8594 -
val_loss: 313093.9375
Epoch 86/150
1/1 [==============================] - 0s 33ms/step - loss: 181609.5000 -
val_loss: 310886.3125
Epoch 87/150
```

```
1/1 [==============================] - 0s 35ms/step - loss: 163783.1094 -
val_loss: 308863.8125
Epoch 88/150
1/1 [==============================] - 0s 38ms/step - loss: 162292.0625 -
val_loss: 306934.4375
Epoch 89/150
1/1 [==============================] - 0s 42ms/step - loss: 148396.6875 -
val_loss: 305081.9375
Epoch 90/150
1/1 [==============================] - 0s 35ms/step - loss: 144205.2656 -
val_loss: 303417.3438
Epoch 91/150
1/1 [==============================] - 0s 32ms/step - loss: 138982.4375 -
val_loss: 301905.5938
Epoch 92/150
1/1 [==============================] - 0s 31ms/step - loss: 138194.8125 -
val_loss: 300577.6562
Epoch 93/150
1/1 [==============================] - 0s 34ms/step - loss: 127527.2188 -
val_loss: 299431.5312
Epoch 94/150
1/1 [==============================] - 0s 31ms/step - loss: 119644.5312 -
val_loss: 298327.4375
Epoch 95/150
1/1 [==============================] - 0s 36ms/step - loss: 110656.8750 -
val_loss: 297233.2812
Epoch 96/150
1/1 [==============================] - 0s 34ms/step - loss: 109375.4453 -
val_loss: 296288.0625
Epoch 97/150
1/1 [==============================] - 0s 32ms/step - loss: 107664.7266 -
val_loss: 295379.6875
Epoch 98/150
1/1 [==============================] - 0s 34ms/step - loss: 103902.7812 -
val_loss: 294482.4062
Epoch 99/150
1/1 [==============================] - 0s 31ms/step - loss: 96762.3984 -
val_loss: 293691.6250
Epoch 100/150
1/1 [==============================] - 0s 32ms/step - loss: 95412.5312 -
val_loss: 293055.5625
Epoch 101/150
1/1 [==============================] - 0s 34ms/step - loss: 92679.1953 -
val_loss: 292533.3125
Epoch 102/150
1/1 [==============================] - 0s 32ms/step - loss: 93299.2969 -
val_loss: 292055.4688
Epoch 103/150
```

```
1/1 [==============================] - 0s 33ms/step - loss: 86085.7109 -
val_loss: 291569.3125
Epoch 104/150
1/1 [==============================] - 0s 33ms/step - loss: 78018.8047 -
val_loss: 291034.4688
Epoch 105/150
1/1 [==============================] - 0s 32ms/step - loss: 76680.7812 -
val_loss: 290460.9688
Epoch 106/150
1/1 [==============================] - 0s 32ms/step - loss: 78075.1562 -
val_loss: 289851.3750
Epoch 107/150
1/1 [==============================] - 0s 32ms/step - loss: 75333.2812 -
val_loss: 289245.5000
Epoch 108/150
1/1 [==============================] - 0s 34ms/step - loss: 71794.6641 -
val_loss: 288587.8750
Epoch 109/150
1/1 [==============================] - 0s 33ms/step - loss: 65027.2500 -
val_loss: 287868.4062
Epoch 110/150
1/1 [==============================] - 0s 33ms/step - loss: 71445.7812 -
val_loss: 287123.7188
Epoch 111/150
1/1 [==============================] - 0s 36ms/step - loss: 73044.7578 -
val_loss: 286277.9375
Epoch 112/150
1/1 [==============================] - 0s 33ms/step - loss: 64556.6523 -
val_loss: 285405.2812
Epoch 113/150
1/1 [==============================] - 0s 33ms/step - loss: 63054.1602 -
val_loss: 284632.4062
Epoch 114/150
1/1 [==============================] - 0s 36ms/step - loss: 69552.8203 -
val_loss: 283861.8438
Epoch 115/150
1/1 [==============================] - 0s 36ms/step - loss: 68413.2812 -
val_loss: 283163.2500
Epoch 116/150
1/1 [==============================] - 0s 41ms/step - loss: 68246.3047 -
val_loss: 282380.5625
Epoch 117/150
1/1 [==============================] - 0s 31ms/step - loss: 63835.9531 -
val_loss: 281590.3750
Epoch 118/150
1/1 [==============================] - 0s 32ms/step - loss: 61647.1797 -
val_loss: 280758.4062
Epoch 119/150
```

```
1/1 [==============================] - 0s 35ms/step - loss: 54832.4219 -
val_loss: 279926.0000
Epoch 120/150
1/1 [==============================] - 0s 32ms/step - loss: 60739.8555 -
val_loss: 279104.1562
Epoch 121/150
1/1 [==============================] - 0s 32ms/step - loss: 59503.2227 -
val_loss: 278158.8438
Epoch 122/150
1/1 [==============================] - 0s 33ms/step - loss: 65143.8047 -
val_loss: 277230.4062
Epoch 123/150
1/1 [==============================] - 0s 31ms/step - loss: 57220.8359 -
val_loss: 276175.6875
Epoch 124/150
1/1 [==============================] - 0s 35ms/step - loss: 57876.3281 -
val_loss: 275052.0938
Epoch 125/150
1/1 [==============================] - 0s 32ms/step - loss: 56697.1445 -
val_loss: 273935.5625
Epoch 126/150
1/1 [==============================] - 0s 38ms/step - loss: 51695.7734 -
val_loss: 272846.5625
Epoch 127/150
1/1 [==============================] - 0s 34ms/step - loss: 59541.5898 -
val_loss: 271751.3750
Epoch 128/150
1/1 [==============================] - 0s 36ms/step - loss: 58878.0430 -
val_loss: 270457.8438
Epoch 129/150
1/1 [==============================] - 0s 33ms/step - loss: 50769.4844 -
val_loss: 269098.6562
Epoch 130/150
1/1 [==============================] - 0s 33ms/step - loss: 51091.7969 -
val_loss: 267871.3750
Epoch 131/150
1/1 [==============================] - 0s 33ms/step - loss: 57129.5742 -
val_loss: 266677.6250
Epoch 132/150
1/1 [==============================] - 0s 33ms/step - loss: 56052.2070 -
val_loss: 265393.2500
Epoch 133/150
1/1 [==============================] - 0s 34ms/step - loss: 49895.6758 -
val_loss: 264194.0625
Epoch 134/150
1/1 [==============================] - 0s 34ms/step - loss: 52918.5508 -
val_loss: 263154.6562
Epoch 135/150
```

```
1/1 [==============================] - 0s 37ms/step - loss: 48880.6484 -
val_loss: 262045.6094
Epoch 136/150
1/1 [==============================] - 0s 36ms/step - loss: 47787.8477 -
val_loss: 261077.2031
Epoch 137/150
1/1 [==============================] - 0s 35ms/step - loss: 44316.5625 -
val_loss: 260244.1875
Epoch 138/150
1/1 [==============================] - 0s 33ms/step - loss: 45929.3828 -
val_loss: 259464.2031
Epoch 139/150
1/1 [==============================] - 0s 42ms/step - loss: 48991.4219 -
val_loss: 258645.6719
Epoch 140/150
1/1 [==============================] - 0s 31ms/step - loss: 48452.5469 -
val_loss: 257910.1250
Epoch 141/150
1/1 [==============================] - 0s 32ms/step - loss: 47111.2109 -
val_loss: 257273.9531
Epoch 142/150
1/1 [==============================] - 0s 33ms/step - loss: 45176.0000 -
val_loss: 256646.5625
Epoch 143/150
1/1 [==============================] - 0s 60ms/step - loss: 47481.9844 -
val_loss: 256086.7344
Epoch 144/150
1/1 [==============================] - 0s 34ms/step - loss: 42667.8398 -
val_loss: 255506.4844
Epoch 145/150
1/1 [==============================] - 0s 34ms/step - loss: 43139.6250 -
val_loss: 254881.8281
Epoch 146/150
1/1 [==============================] - 0s 36ms/step - loss: 48977.8906 -
val_loss: 254318.7812
Epoch 147/150
1/1 [==============================] - 0s 32ms/step - loss: 44110.8477 -
val_loss: 253893.7031
Epoch 148/150
1/1 [==============================] - 0s 33ms/step - loss: 51053.6211 -
val_loss: 253619.4375
Epoch 149/150
1/1 [==============================] - 0s 32ms/step - loss: 41059.3047 -
val_loss: 253352.0312
Epoch 150/150
1/1 [==============================] - 0s 48ms/step - loss: 43432.7812 -
val_loss: 252994.1406
```

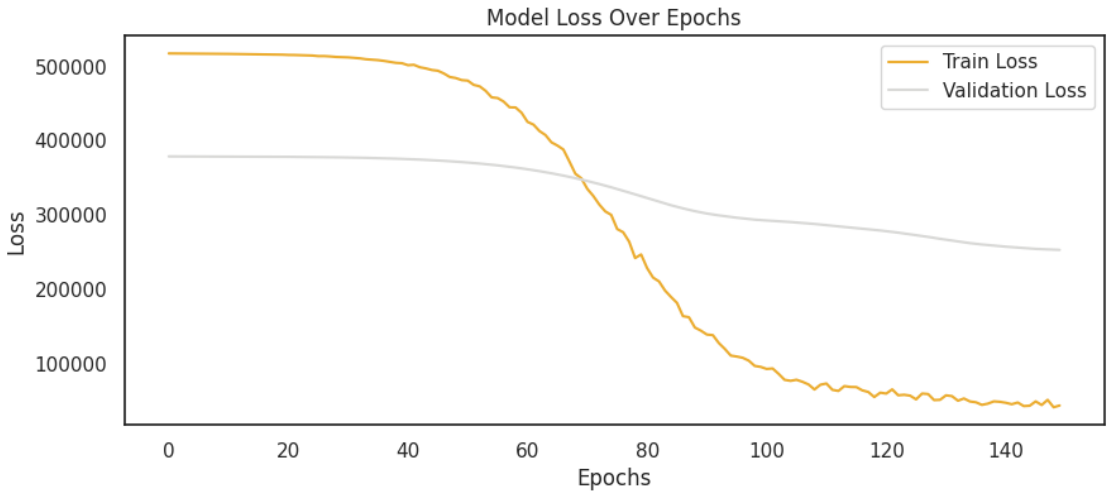
Evaluación

[ ]:
```
# Evaluar el modelo en el conjunto de prueba
test_loss = model.evaluate(X_test_scaled, y_test)
print(f"Test Loss: {test_loss}")
```

```
1/1 [==============================] - 0s 27ms/step - loss: 119426.0000
Test Loss: 119426.0
```

Graficar el Historial de Entrenamiento

[ ]:
```
# Graficar el Historial de Entrenamiento
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.plot(history.history['loss'], label='Train Loss', color='#ECAD30')
plt.plot(history.history['val_loss'], label='Validation Loss', color='#D9D9D7')
plt.title('Model Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

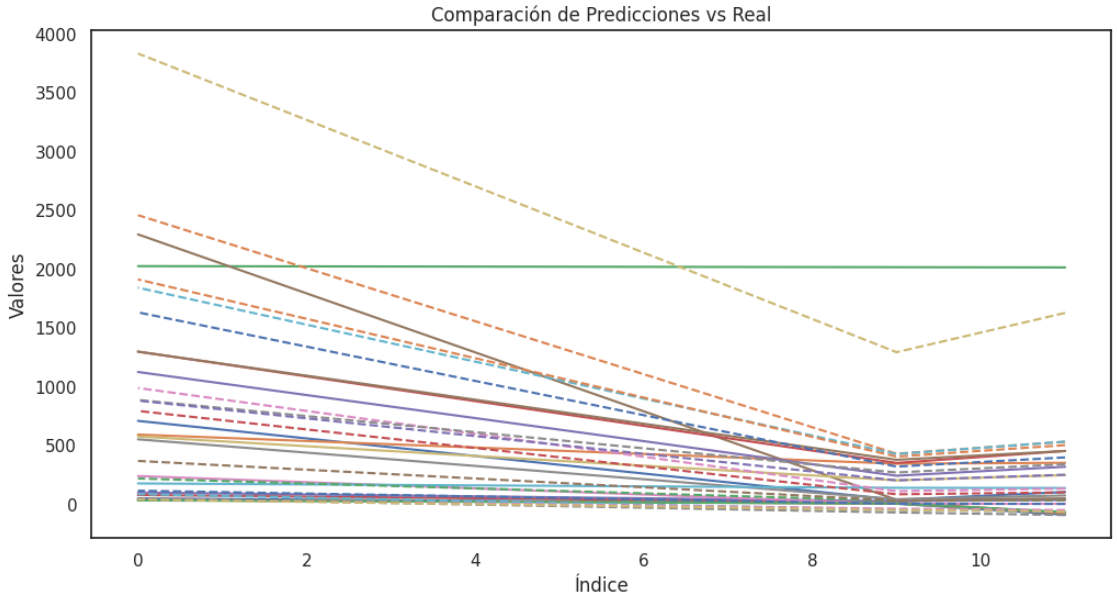


Real vs Predicciones

[ ]:
```
# Comparar Real vs Predicciones
import matplotlib.pyplot as plt

# Asumiendo 'y_pred' como las predicciones del modelo sobre 'X_test'
y_pred = model.predict(X_test_scaled)

plt.figure(figsize=(12, 6))
```

```
plt.plot(y_test.index, y_test, label='Real')
plt.plot(y_test.index, y_pred, label='Predicción', linestyle='--')
plt.title('Comparación de Predicciones vs Real')
plt.xlabel('Índice')
plt.ylabel('Valores')
#plt.legend()
plt.show()
```

```
1/1 [==============================] - 0s 132ms/step
```



Crear DataFrame con Predicciones

[ ]:
```
import pandas as pd

# Crear DataFrame de predicciones
pred_df = pd.DataFrame(y_pred, columns=numerical_columns)

# Asumiendo que las fechas de predicción siguen inmediatamente después de la␣
 ↪última fecha en df_millions
last_date = df_millions['Fecha'].max()
pred_dates = pd.date_range(start=last_date + pd.DateOffset(years=1),␣
 ↪periods=len(pred_df), freq='A')

# Añadir columnas de 'Fecha' y 'Empresa'
pred_df['Fecha'] = pred_dates
pred_df['Empresa'] = "Empresa_1"  # O el nombre de la empresa que corresponda
```

```
# Reordenar las columnas para que coincidan con df_millions
pred_df = pred_df[['Fecha', 'Empresa'] + [col for col in pred_df.columns if col␣
 ↪not in ['Fecha', 'Empresa']]]
```

Concatenar DataFrames

[ ]:
```
# Concatenar df_millions y pred_df
combined_df = pd.concat([df_millions, pred_df], ignore_index=True)

# Mostrar
combined_df.tail()
```

[ ]:

	Fecha	Empresa	Activo no corriente	Activo corriente	Total Activo	Patrimonio neto	Pasivo no corriente	Pasivo corriente	Total Pasivo	Total Pasivo y Patrimonio	Ingresos de las operaciones	Margen Bruto	EBITDA	EBIT	Resultado antes de impuestos	Resultado consolidado del ejercicio
11	2012-12-31	Empresa_0	348.950000	100.440000	449.400000	134.350000	244.210000	70.840000	315.050000	449.400000	45.580000	0.000000	31.870000	-79.210000	-93.270000	-78.360000
12	2011-12-31	Empresa_0	523.360000	44.300000	567.660000	211.160000	303.820000	52.680000	356.500000	567.660000	44.380000	0.000000	29.910000	-129.410000	-141.070000	-109.940000
13	2024-12-31	Empresa_1	336.831299	129.725067	529.198486	33.038521	247.366272	96.756195	394.873718	499.455841	527.864380	31.283142	-1.439542	-65.935631	-53.056591	-96.000374
14	2025-12-31	Empresa_1	266.487061	107.302788	423.321320	28.008167	199.533264	80.960732	316.599121	401.873230	426.273621	25.650326	0.870352	-50.799030	-41.351448	-73.708931
15	2026-12-31	Empresa_1	883.691772	985.614502	1839.635010	364.976105	878.238220	790.540100	1629.530029	1909.702026	2456.962158	215.255905	111.703011	81.608116	38.918011	47.051598

```
    Resultado neto atribuible a la sociedad dominante          Año
11                                        -68.980000  2012.000000
12                                       -102.460000  2011.000000
13                                        -66.153656  1624.662109
14                                        -51.939289  1290.458984
15                                         35.793549  3832.902100
```

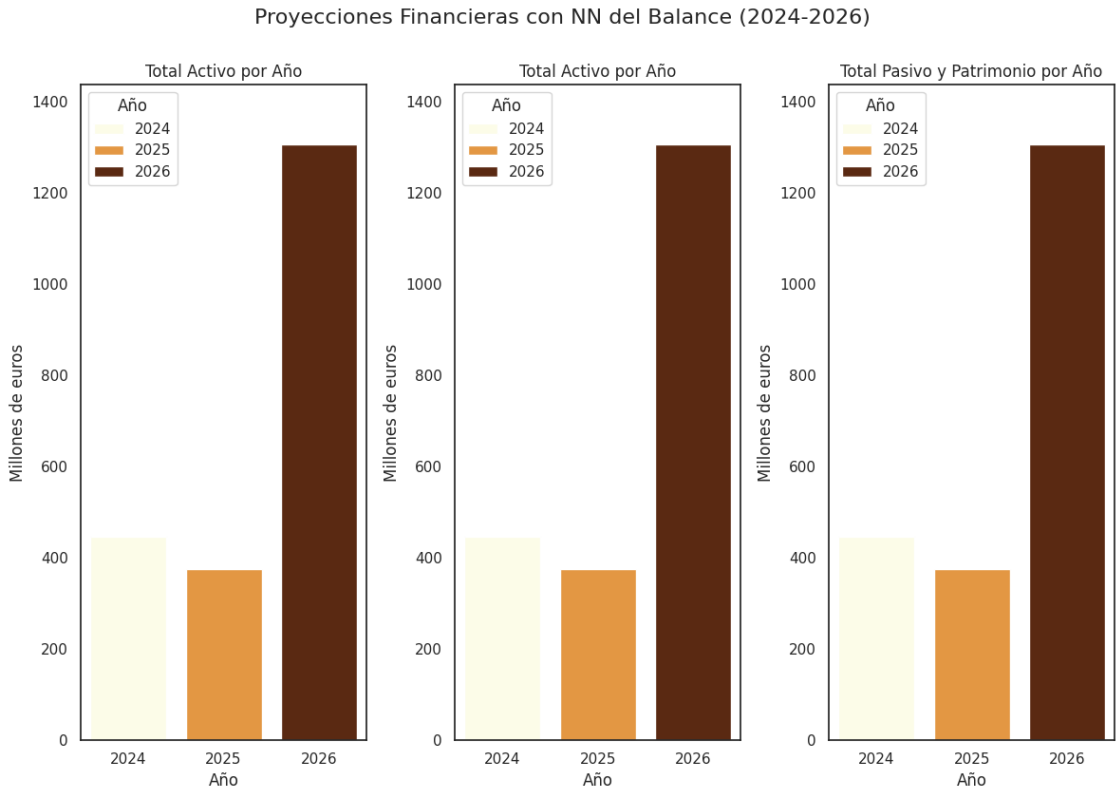
Exportar Predicciones a CSV

[ ]:
```
# Exportar DataFrame de predicciones
pred_df.to_csv('fin_data_energy_pred_only_nn.csv', sep='|', index=False)

# Exportar DataFrame combinado
combined_df.to_csv('fin_data_energy_millions_con_pred_nn.csv', sep='|',␣
 ↪index=False)
```

Visualizar Predicciones

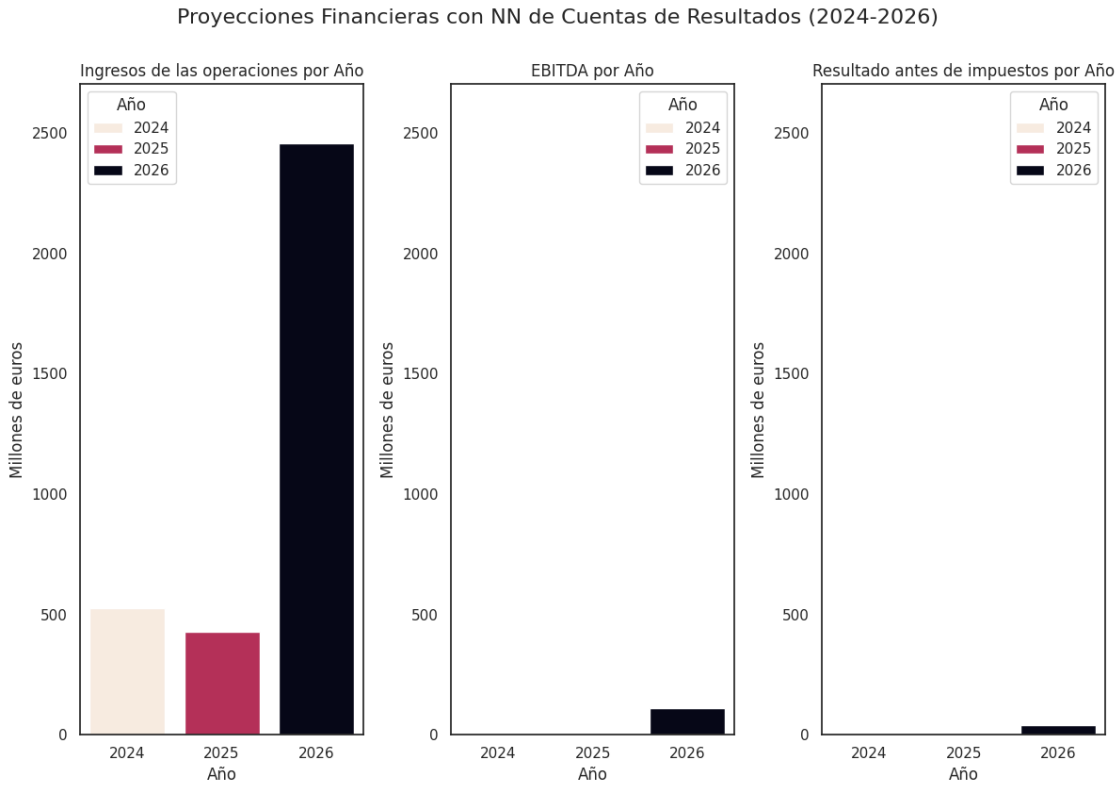
[ ]:
```
# Visualizar Predicciones del Balance
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Asegurarse de que 'Fecha' está en formato datetime y extraer el año
combined_df['Fecha'] = pd.to_datetime(combined_df['Fecha'])
combined_df['Año'] = combined_df['Fecha'].dt.year.astype('int32')

# Filtrar los datos para los años 2024, 2025, 2026
years_of_interest = [2024, 2025, 2026]
combined_df = combined_df[combined_df['Año'].isin(years_of_interest)]

# Variables de interés para el gráfico
variables_of_interest = ['Total Activo', 'Total Activo', 'Total Pasivo y␣
 ↪Patrimonio']

# Crear un DataFrame con los datos para graficar, transformando a formato largo
data_to_plot = combined_df.melt(id_vars=['Año'],␣
 ↪value_vars=variables_of_interest)

# Preparar la figura para los subplots
plt.figure(figsize=(12, 8))  # Ajusta el tamaño de la figura total aquí según␣
 ↪tus necesidades

# Definir paleta de colores personalizada o usar una predefinida
palette = 'YlOrBr'
colors = {
    "Total Activo": "#ECAD30",
```

```
    "Total Activo": "#D9D9D7",
    "Total Pasivo y Patrimonio": "#B5AB66",
    "Resultado antes de impuestos": "#73BBA1",
}

# Crear un subplot para cada variable de interés
for i, variable in enumerate(variables_of_interest, 1):
    plt.subplot(1, 3, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de variables/subplots
    sns.barplot(
        data=data_to_plot[data_to_plot['variable'] == variable],
        x='Año',
        y='value',
        hue='Año',
        palette=palette,
        dodge=False
    )
    plt.title(f'{variable} por Año')
    plt.xlabel('Año')
    plt.ylabel('Millones de euros')
    plt.legend(title='Año')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data_to_plot['value'].max() * 1.1)

# Ajustar el espaciado entre subplots y añadir un título general
plt.tight_layout()
plt.suptitle('Proyecciones Financieras con NN del Balance (2024-2026)',␣
 ↪fontsize=16, y=1.05)

# Mostrar el gráfico
plt.show()
```



[ ]:
```
# Visualizar Predicciones de Cuentas de Resultados
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Asegurarse de que 'Fecha' está en formato datetime y extraer el año
combined_df['Fecha'] = pd.to_datetime(combined_df['Fecha'])
combined_df['Año'] = combined_df['Fecha'].dt.year.astype('int32')

# Filtrar los datos para los años 2024, 2025, 2026
years_of_interest = [2024, 2025, 2026]
combined_df = combined_df[combined_df['Año'].isin(years_of_interest)]

# Variables de interés para el gráfico
variables_of_interest = ['Ingresos de las operaciones', 'EBITDA', 'Resultado␣
 ↪antes de impuestos']

# Crear un DataFrame con los datos para graficar, transformando a formato largo
data_to_plot = combined_df.melt(id_vars=['Año'],␣
 ↪value_vars=variables_of_interest)

# Preparar la figura para los subplots
```

```
plt.figure(figsize=(12, 8))  # Ajusta el tamaño de la figura total aquí según␣
 ↪tus necesidades

# Definir paleta de colores personalizada o usar una predefinida
palette = 'rocket_r'
colors = {
    "Ingresos de las operaciones": "#ECAD30",
    "Margen Bruto": "#D9D9D7",
    "EBITDA": "#B5AB66",
    "EBIT": "#788C90",
    "Resultado antes de impuestos": "#73BBA1",
    "Resultado consolidado del ejercicio": "#6CB978"
}

# Crear un subplot para cada variable de interés
for i, variable in enumerate(variables_of_interest, 1):
    plt.subplot(1, 3, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de variables/subplots
    sns.barplot(
        data=data_to_plot[data_to_plot['variable'] == variable],
        x='Año',
        y='value',
        hue='Año',
        palette=palette,
        dodge=False
    )
    plt.title(f'{variable} por Año')
    plt.xlabel('Año')
    plt.ylabel('Millones de euros')
    plt.legend(title='Año')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data_to_plot['value'].max() * 1.1)

# Ajustar el espaciado entre subplots y añadir un título general
plt.tight_layout()
plt.suptitle('Proyecciones Financieras con NN de Cuentas de Resultados␣
 ↪(2024-2026)', fontsize=16, y=1.05)

# Mostrar el gráfico
plt.show()
```



1.5.2 Modelo de Regresión Múltiple

Preparación

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

# Asegúrate de cargar el DataFrame si aún no está cargado
df_millions = pd.read_csv('fin_data_energy_millions.csv', sep='|')
```

[ ]:
```
# Convertir la columna 'Fecha' a tipo datetime y extraer el año si no se ha␣
 ↪hecho previamente
df_millions['Fecha'] = pd.to_datetime(df_millions['Fecha'])
df_millions['Año'] = df_millions['Fecha'].dt.year

# Asumiendo que df_millions ya está cargado y las columnas de interés␣
 ↪seleccionadas
numerical_columns = df_millions.columns.difference(['Fecha', 'Empresa'])
```

[ ]:
```
# Preparar datos de entrada y salida
X = df_millions[numerical_columns]
y = df_millions[numerical_columns]  # Estamos prediciendo las mismas columnas

# Dividir datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Escalar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Definición del Modelo

[ ]:
```
# Crear y entrenar el modelo de Regresión Múltiple
model = LinearRegression()
```

Entrenamiento

[ ]:
```
# Entrenar el modelo
model.fit(X_train_scaled, y_train)
```

[ ]:
```
LinearRegression()
```

Generar Predicciones

[ ]:
```
# Hacer predicciones en el conjunto de prueba
y_pred = model.predict(X_test_scaled)
```

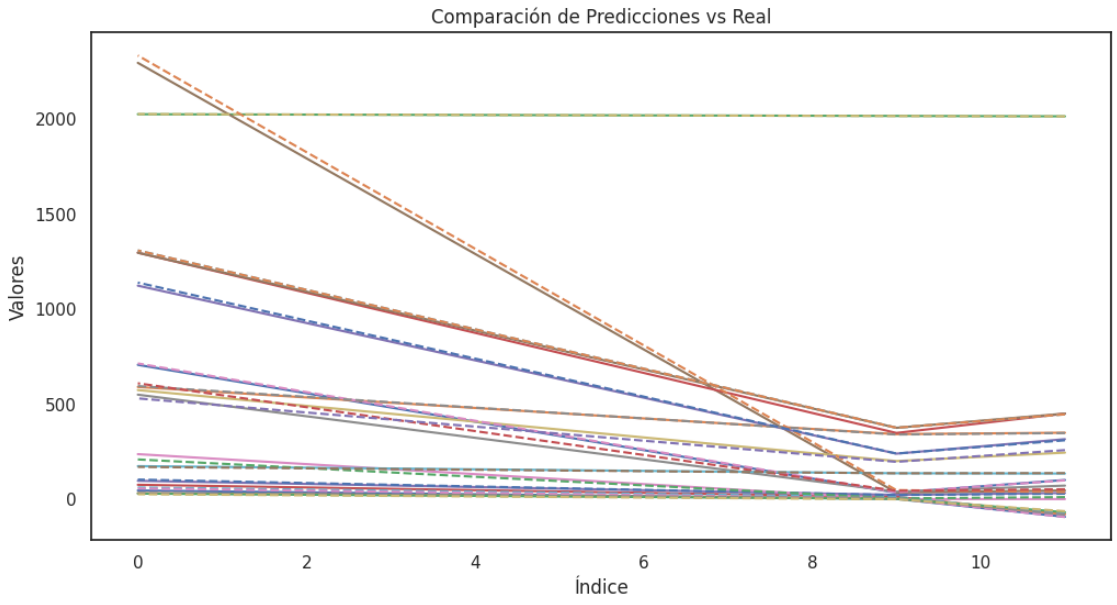
Evaluación

[ ]:
```
# Calcular el MSE para evaluar el modelo
mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {mse}")

# Opcional: Mostrar los coeficientes del modelo
print("Model coefficients:\n")
for i, col in enumerate(numerical_columns):
    print(f"{col}: {model.coef_[0][i]}")
```

```
Test MSE: 220.82969100792556
Model coefficients:

Activo corriente: 96.14694474780546
Activo no corriente: -78.49037219784806
Año: 19.09914138182017
EBIT: 11.477726367921205
EBITDA: 3.281634966209372
```

```
Ingresos de las operaciones: -3.782206714039277
Margen Bruto: -3.179887079982446
Pasivo corriente: 42.2232866810938
Pasivo no corriente: 52.58582757725171
Patrimonio neto: 16.482437584481538
Resultado antes de impuestos: -15.757562424447352
Resultado consolidado del ejercicio: -7.034473172237018
Resultado neto atribuible a la sociedad dominante: 10.426034819742338
Total Activo: 48.4024812437815
Total Pasivo: 48.3555217090933
Total Pasivo y Patrimonio: 48.4024812437815
```

[ ]:
```python
# Comparar Real vs Predicciones
import matplotlib.pyplot as plt

# Asumiendo que tienes 'y_pred' como las predicciones del modelo sobre 'X_test'
y_pred = model.predict(X_test_scaled)

plt.figure(figsize=(12, 6))
plt.plot(y_test.index, y_test, label='Real')
plt.plot(y_test.index, y_pred, label='Predicción', linestyle='--')
plt.title('Comparación de Predicciones vs Real')
plt.xlabel('Índice')
plt.ylabel('Valores')
#plt.legend()
plt.show()
```



Crear DataFrame con Predicciones

[ ]:
```
# Convertir las predicciones en DataFrame
pred_df = pd.DataFrame(y_pred, columns=numerical_columns)

# Añadir la columna 'Fecha' y 'Empresa'
# Suponiendo que las predicciones son para los años 2024, 2025, 2026 (puedes␣
 ↪ajustar según tus necesidades)
pred_df['Fecha'] = pd.date_range(start='2024-01-01', periods=len(pred_df),␣
 ↪freq='A')  # Frecuencia Anual
pred_df['Empresa'] = "Empresa_1"  # Suponemos que todas las predicciones son␣
 ↪para "Empresa_1"

# Asegurar que las columnas 'Fecha' y 'Empresa' estén al principio para␣
 ↪coincidir con df_millions
pred_df = pred_df[['Fecha', 'Empresa'] + [col for col in pred_df.columns if col␣
 ↪not in ['Fecha', 'Empresa']]]
```

Concatenar DataFrames

[ ]:
```
# Concatenar pred_df con df_millions
combined_df = pd.concat([df_millions, pred_df], axis=0).reset_index(drop=True)

# Verificar el nuevo DataFrame
combined_df.tail()  # Mostrar las últimas filas para ver las predicciones
```

[ ]:

	Fecha	Empresa	Activo no corriente	Activo corriente	Total Activo	\
11	2012-12-31	Empresa_0	348.950000	100.440000	449.400000	
12	2011-12-31	Empresa_0	523.360000	44.300000	567.660000	
13	2024-12-31	Empresa_1	347.180079	98.392838	445.570832	
14	2025-12-31	Empresa_1	339.378772	35.591845	374.981587	
15	2026-12-31	Empresa_1	594.067959	713.037590	1307.111554	

	Patrimonio neto	Pasivo no corriente	Pasivo corriente	Total Pasivo	\
11	134.350000	244.210000	70.840000	315.050000	
12	211.160000	303.820000	52.680000	356.500000	
13	136.496769	257.437182	51.629641	309.058570	
14	135.595933	195.770786	43.617491	239.388970	
15	169.094963	529.340463	608.691738	1138.023622	

	Total Pasivo y Patrimonio	Ingresos de las operaciones	Margen Bruto	\
11	449.400000	45.580000	0.000000	
12	567.660000	44.380000	0.000000	
13	445.570832	36.758252	10.377905	
14	374.981587	47.068598	3.679902	
15	1307.111554	2331.587041	208.464954	

EBITDA	EBIT	Resultado antes de impuestos	\

```
11   31.870000  -79.210000                    -93.270000
12   29.910000 -129.410000                   -141.070000
13   28.428585  -78.825171                    -94.461664
14   21.228726   11.055984                      3.586123
15  102.556357   61.101804                     56.073789

    Resultado consolidado del ejercicio  \
11                            -78.360000
12                           -109.940000
13                            -81.179772
14                              5.435255
15                             31.490532

    Resultado neto atribuible a la sociedad dominante          Año
11                                         -68.980000  2012.000000
12                                        -102.460000  2011.000000
13                                         -63.582164  2011.786543
14                                          -0.963339  2013.553997
15                                          25.459432  2023.688764
```

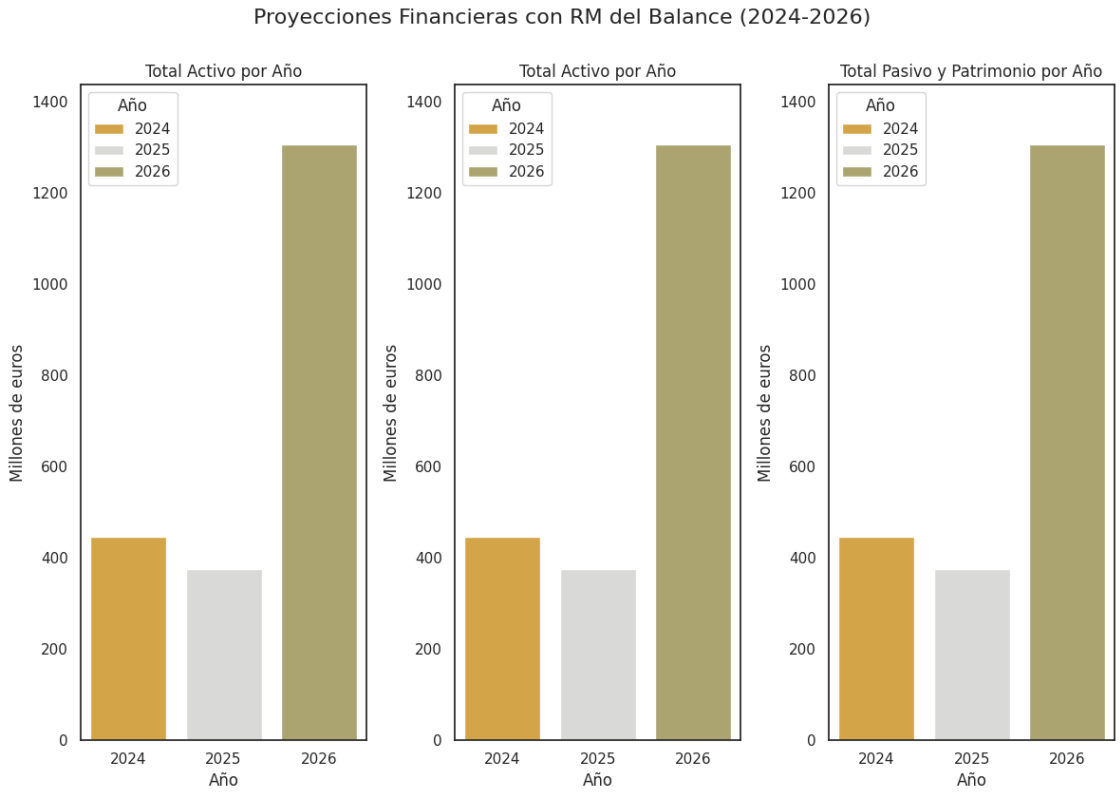
Exportar Predicciones a CSV

[ ]:
```
# Exportar DataFrame de predicciones
pred_df.to_csv('fin_data_energy_pred_only_rm.csv', sep='|', index=False)

# Exportar DataFrame combinado
combined_df.to_csv('fin_data_energy_millions_con_pred_rm.csv', sep='|',␣
 ↪index=False)
```

Visualizar Predicciones

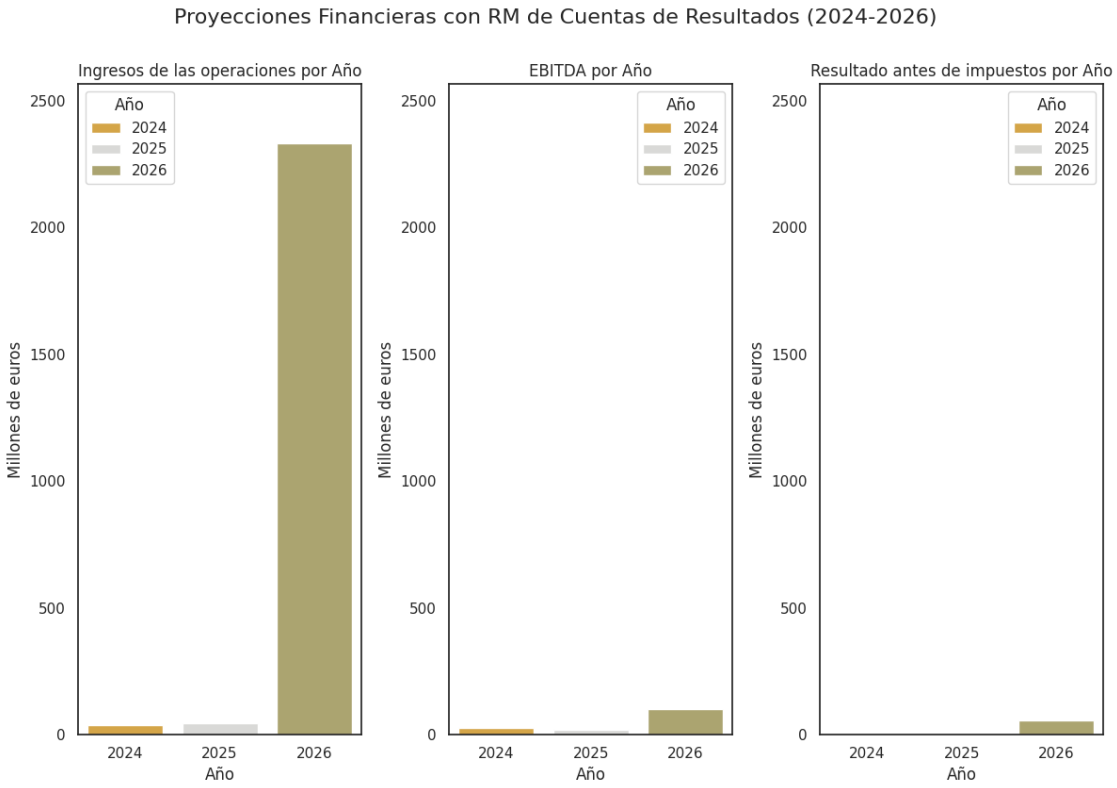
[ ]:
```
# Visualizar Predicciones del Balance
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Asegurarse de que 'Fecha' está en formato datetime y extraer el año
combined_df['Fecha'] = pd.to_datetime(combined_df['Fecha'])
combined_df['Año'] = combined_df['Fecha'].dt.year.astype('int32')

# Filtrar los datos para los años 2024, 2025, 2026
years_of_interest = [2024, 2025, 2026]
combined_df = combined_df[combined_df['Año'].isin(years_of_interest)]

# Variables de interés para el gráfico
variables_of_interest = ['Total Activo', 'Total Activo', 'Total Pasivo y␣
 ↪Patrimonio']
```

```
# Crear un DataFrame con los datos para graficar, transformando a formato largo
data_to_plot = combined_df.melt(id_vars=['Año'],␣
 ↪value_vars=variables_of_interest)

# Preparar la figura para los subplots
plt.figure(figsize=(12, 8))  # Ajusta el tamaño de la figura total aquí según␣
 ↪tus necesidades

# Definir paleta de colores personalizada o usar una predefinida
palette = 'mako'
colors = {
    2024: "#ECAD30",
    2025: "#D9D9D7",
    2026: "#B5AB66"
}

# Crear un subplot para cada variable de interés
for i, variable in enumerate(variables_of_interest, 1):
    plt.subplot(1, 3, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de variables/subplots
    sns.barplot(
        data=data_to_plot[data_to_plot['variable'] == variable],
        x='Año',
        y='value',
        hue='Año',
        palette=colors,
        dodge=False
    )
    plt.title(f'{variable} por Año')
    plt.xlabel('Año')
    plt.ylabel('Millones de euros')
    plt.legend(title='Año')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data_to_plot['value'].max() * 1.1)

# Ajustar el espaciado entre subplots y añadir un título general
plt.tight_layout()
plt.suptitle('Proyecciones Financieras con RM del Balance (2024-2026)',␣
 ↪fontsize=16, y=1.05)

# Mostrar el gráfico
plt.show()
```



[ ]:
```
# Visualizar Predicciones de Cuentas de Resultados
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Asegurarse de que 'Fecha' está en formato datetime y extraer el año
combined_df['Fecha'] = pd.to_datetime(combined_df['Fecha'])
combined_df['Año'] = combined_df['Fecha'].dt.year.astype('int32')

# Filtrar los datos para los años 2024, 2025, 2026
years_of_interest = [2024, 2025, 2026]
combined_df = combined_df[combined_df['Año'].isin(years_of_interest)]

# Variables de interés para el gráfico
variables_of_interest = ['Ingresos de las operaciones', 'EBITDA', 'Resultado␣
 ↪antes de impuestos']

# Crear un DataFrame con los datos para graficar, transformando a formato largo
data_to_plot = combined_df.melt(id_vars=['Año'],␣
 ↪value_vars=variables_of_interest)

# Preparar la figura para los subplots
```

```
plt.figure(figsize=(12, 8))  # Ajusta el tamaño de la figura total aquí según␣
 ↪tus necesidades

# Definir paleta de colores personalizada o usar una predefinida
palette = 'viridis'
colors = {
    2024: "#ECAD30",
    2025: "#D9D9D7",
    2026: "#B5AB66"
}

# Crear un subplot para cada variable de interés
for i, variable in enumerate(variables_of_interest, 1):
    plt.subplot(1, 3, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de variables/subplots
    sns.barplot(
        data=data_to_plot[data_to_plot['variable'] == variable],
        x='Año',
        y='value',
        hue='Año',
        palette=colors,
        dodge=False
    )
    plt.title(f'{variable} por Año')
    plt.xlabel('Año')
    plt.ylabel('Millones de euros')
    plt.legend(title='Año')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data_to_plot['value'].max() * 1.1)

# Ajustar el espaciado entre subplots y añadir un título general
plt.tight_layout()
plt.suptitle('Proyecciones Financieras con RM de Cuentas de Resultados␣
 ↪(2024-2026)', fontsize=16, y=1.05)

# Mostrar el gráfico
plt.show()
```


Proyecciones Financieras con RM de Cuentas de Resultados (2024-2026)
Ingresos de las operaciones por Año
EBITDA por Año
Resultado antes de impuestos por Año
Año
2024
2025
2026
Millones de euros
0
500
1000
1500
2000
2500
Año

1.6 Fin