Energías Renovables - Potencia y Producción

May 6, 2024

1 Energías Renovables - Potencia y Producción

Creado por:

- **V. D. Betancourt**

1.1 Introducción

1.1.1 Descripción

El presente proyecto pertenece al ámbito de las Energías Renovables, en particular, en datos de **Potencia Instalada y Producción de Energía** de una Empresa para 3 países donde se tiene presencia.

1.1.2 Objetivo

El principal objetivo de este análisis es comprender el desempeño operativo de la Empresa a lo largo del tiempo mediante el estudio de la **Potencia Instalada y Producción de Energía**, del **2016** al **2023**. Al profundizar en estos datos, se busca identificar patrones, tendencias y posibles anomalías que podrían influir en la toma de decisiones estratégicas de la empresa.

Posteriormente, el análisis pretende establecer una base sólida para el desarrollo de **modelos predictivos**. Estos modelos serán diseñados para **pronosticar futuros valores** de potencia instalada y producción basándose en los datos históricos disponibles. La capacidad de generar predicciones precisas permitirá a la empresa planificar con mayor efectividad sus inversiones, mantenimientos y estrategias de expansión en los mercados energéticos correspondientes.

1.2 Carga de Datos

El dataset `'datos_potencia_prod.xlsx'` es una fuente propia que compila la información pública disponible de la Empresa y está conformado por los siguientes **5 campos**:

- `"Fecha"`: Fechas del cierre del año desde el 2016 hasta el 2023.
- `"Empresa"`: Corresponde al nombre de la "Empresa_1".
- `"País"`: Puede ser `"España"`, `"Francia"`, `"Polonia"`.
- `"Potencia instalada (MW)"`: Representa la capacidad máxima de producción de energía eléctrica que la empresa puede alcanzar en condiciones óptimas y sin restricciones mecánicas o de otro tipo. Este dato es crucial para evaluar la capacidad y el crecimiento potencial de la infraestructura energética de la empresa.

- **"Producción (GWh)"**: Indica la cantidad real de energía generada por la empresa en un año. Este valor es fundamental para analizar la eficiencia operativa de la empresa, así como la utilización efectiva de su capacidad instalada.

[ ]:
```
import pandas as pd

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

1.3 Análisis Exploratorio de Datos

1.3.1 Análisis Preliminar

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

[ ]:

	Fecha	Empresa	País	Potencia instalada (MW)	Producción (GWh)
0	2023-12-31	Empresa_1	España	151.0	171.0
1	2023-12-31	Empresa_1	Francia	12.0	29.0
2	2023-12-31	Empresa_1	Polonia	34.0	79.0
3	2023-12-31	Empresa_1	Panamá	66.0	242.0
4	2022-12-31	Empresa_1	España	133.0	160.0

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

[ ]:
```
(28, 5)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28 entries, 0 to 27
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Fecha                   28 non-null     datetime64[ns]
 1   Empresa                 28 non-null     object
 2   País                    28 non-null     object
 3   Potencia instalada (MW)  28 non-null     float64
 4   Producción (GWh)        28 non-null     float64
dtypes: datetime64[ns](1), float64(2), object(2)
memory usage: 1.2+ KB
```

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

[ ]:
```
Index(['Fecha', 'Empresa', 'País', 'Potencia instalada (MW)',
       'Producción (GWh)'],
      dtype='object')
```

1.3.2 Missing Values

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

```
Total de Missing Values por Columna:
Fecha                      0
Empresa                    0
País                       0
Potencia instalada (MW)    0
Producción (GWh)           0
dtype: int64
```

1.3.3 Data Cleaning

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

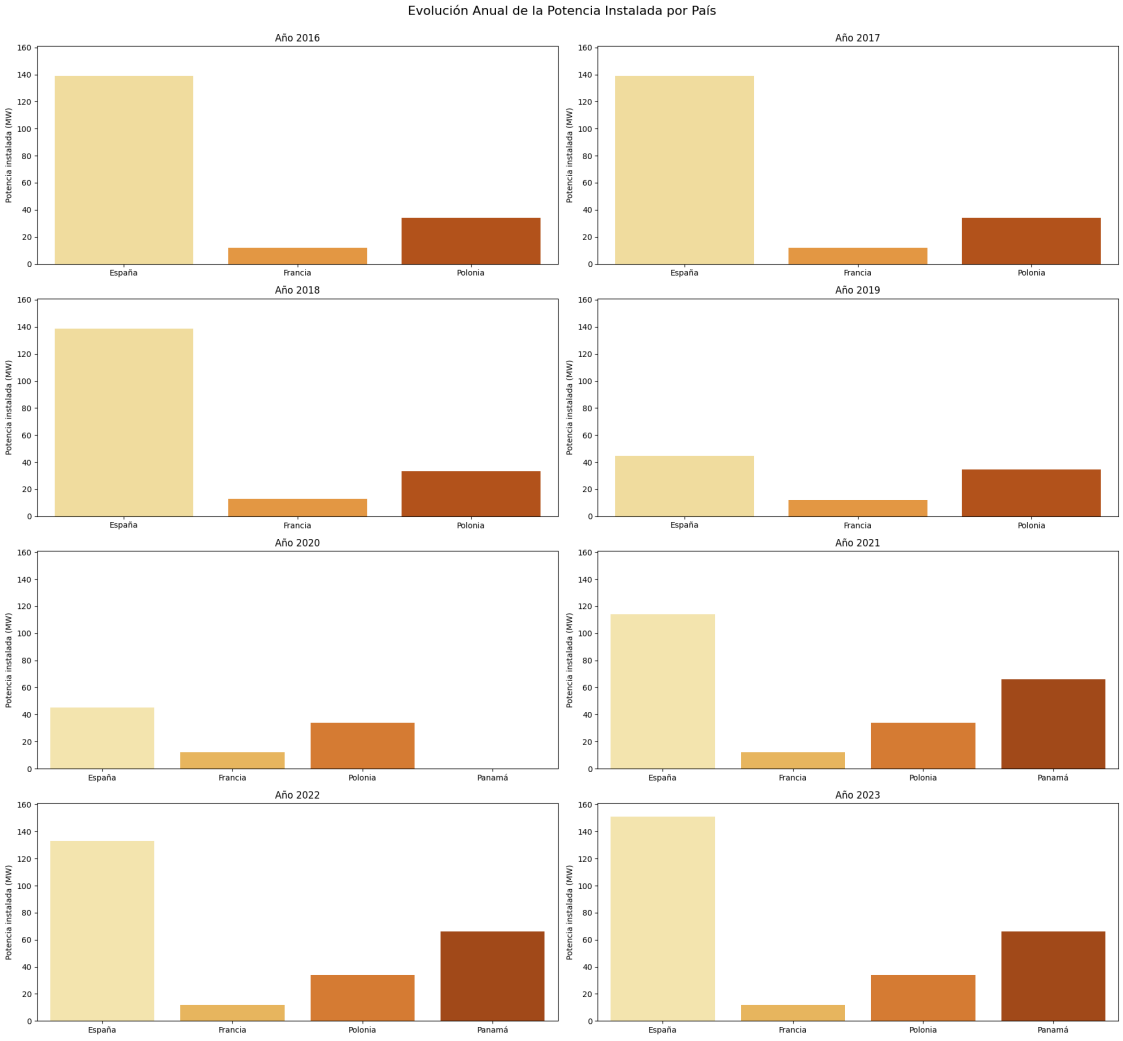
[ ]:
```
# Asegurarse de que 'Fecha' está en formato datetime
df['Fecha'] = pd.to_datetime(df['Fecha'])
```

1.3.4 Barplots

Potencia Instalada (MW)

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

# Cargar datos desde
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Seleccionar solo las columnas necesarias para el gráfico
data = df[['Año', 'País', 'Potencia instalada (MW)']]

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

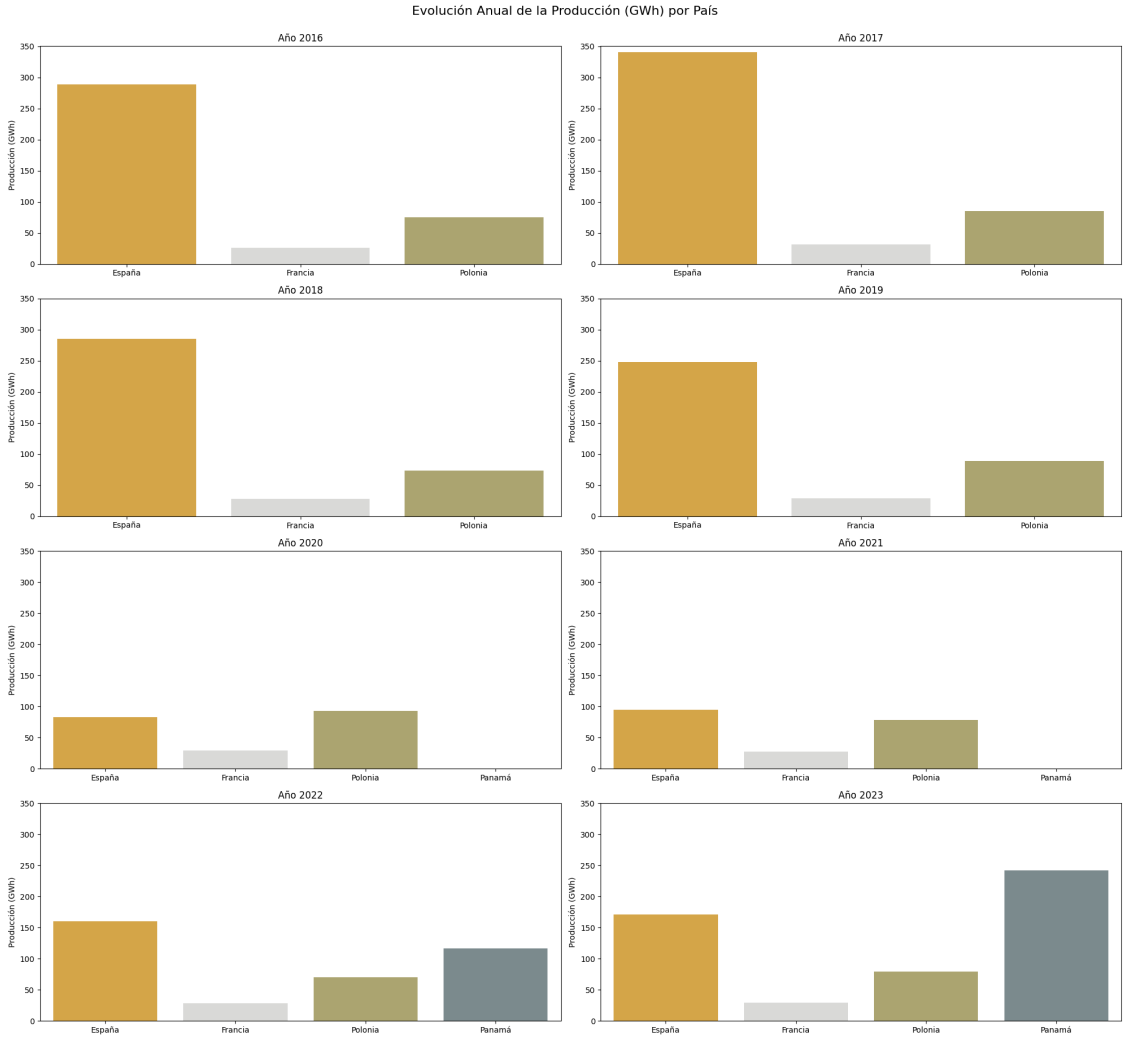
```
# Definir la paleta de colores
palette = 'YlOrBr'
colors = {
    "España": "#ECAD30",
    "Francia": "#D9D9D7",
    "Polonia": "#B5AB66",
    "Panamá": "#788C90",
    "Otro_1": "#73BBA1",
    "Otro_2": "#6CB978"
}

# Crear un subplot para cada año
for i, year in enumerate(sorted(data['Año'].unique()), 1):
    plt.subplot(4, 2, i)  # Ajusta las dimensiones de la cuadrícula según el␣
 ↪número de años/subplots
    sns.barplot(
        data=data[data['Año'] == year],
        x='País',
        y='Potencia instalada (MW)',
        hue='País',
        palette=palette
    )
    plt.title(f'Año {year}')
    plt.xlabel('')  # Ocultar la etiqueta x si es necesario
    plt.ylabel('Potencia instalada (MW)')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data['Potencia instalada (MW)'].max() + 10)

# Ajustar el espaciado entre subplots
plt.tight_layout()

# Añadir un título general para toda la figura
plt.suptitle('Evolución Anual de la Potencia Instalada por País', fontsize=16,␣
 ↪y=1.02)

plt.show()
```



Producción (GWh)

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

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Seleccionar solo las columnas necesarias para el gráfico
data = df[['Año', 'País', 'Producción (GWh)']]
```

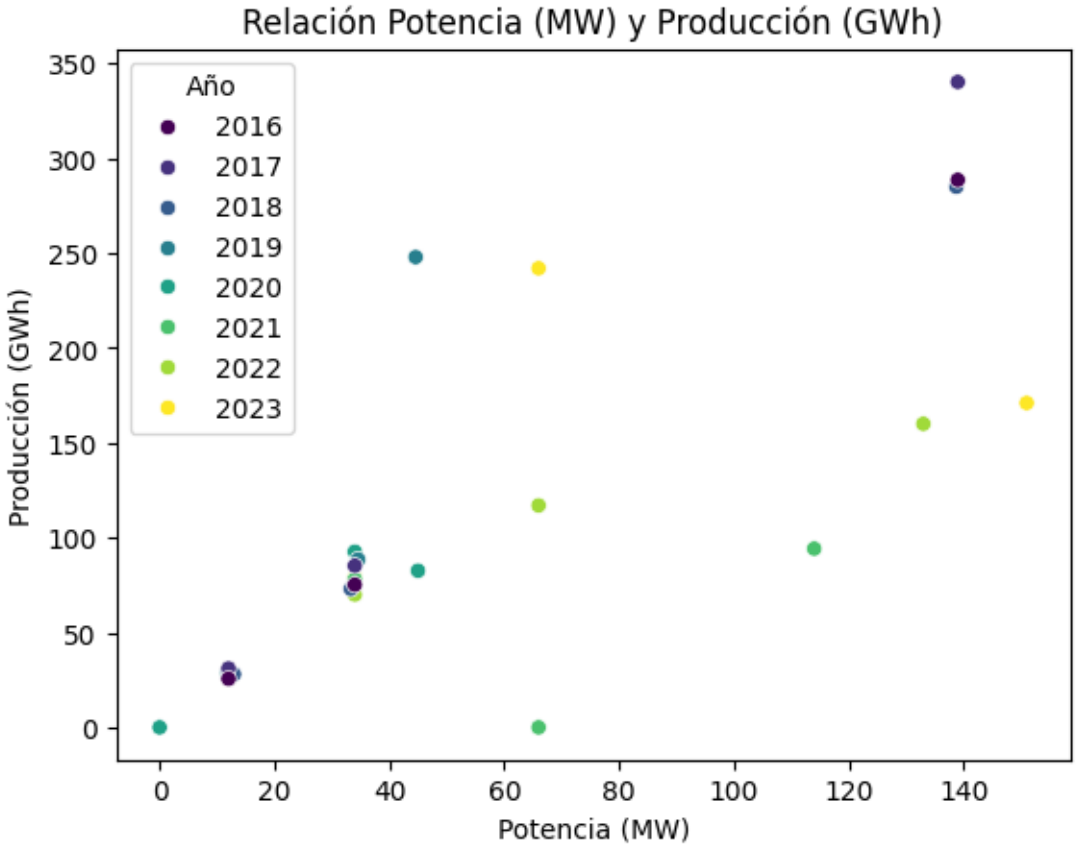
```python
# Preparar la figura para los subplots
plt.figure(figsize=(20, 18))  # Ajusta el tamaño de la figura total según tus necesidades

# Definir la paleta de colores
palette = 'mako'
colors = {
    "España": "#ECAD30",
    "Francia": "#D9D9D7",
    "Polonia": "#B5AB66",
    "Panamá": "#788C90",
    "Otro_1": "#73BBA1",
    "Otro_2": "#6CB978"
}

# Crear un subplot para cada año
for i, year in enumerate(sorted(data['Año'].unique()), 1):
    plt.subplot(4, 2, i)  # Ajusta las dimensiones de la cuadrícula según el número de años/subplots
    sns.barplot(
        data=data[data['Año'] == year],
        x='País',
        y='Producción (GWh)',
        hue='País',
        palette=colors
    )
    plt.title(f'Año {year}')
    plt.xlabel('')  # Ocultar la etiqueta x si es necesario
    plt.ylabel('Producción (GWh)')

    # Ajustar los límites si es necesario para mejor visualización
    plt.ylim(0, data['Producción (GWh)'].max() + 10)

# Ajustar el espaciado entre subplots
plt.tight_layout()

# Añadir un título general para toda la figura
plt.suptitle('Evolución Anual de la Producción (GWh) por País', fontsize=16, y=1.02)

plt.show()
```



1.3.5 Scatterplot

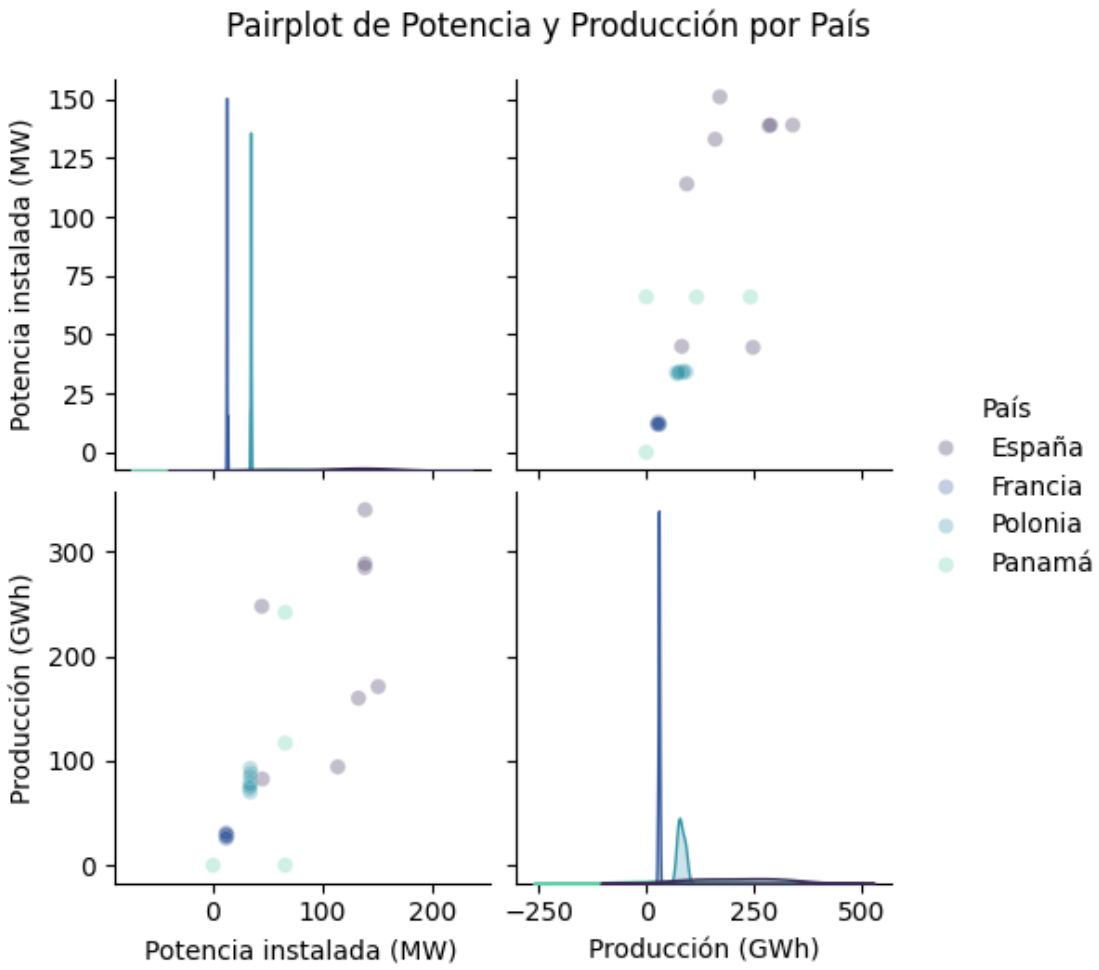
[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Crear el Scatter plot
scatter_plot = sns.scatterplot(
```

```
    x='Potencia instalada (MW)',
    y='Producción (GWh)',
    hue='Año',  # Colorear por año
    data=df,
    palette='viridis',
    legend='full'
)

# Configurar el título y las etiquetas del gráfico
scatter_plot.set_title('Relación Potencia (MW) y Producción (GWh)')
scatter_plot.set_xlabel('Potencia (MW)')
scatter_plot.set_ylabel('Producción (GWh)')

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

# Mostrar el gráfico
plt.show()
```



1.3.6 Pairplot

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

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Seleccionar columnas
columns_to_plot = [
    "Potencia instalada (MW)", "Producción (GWh)"
]

# Crear el pair plot
pair_plot = sns.pairplot(
    df[columns_to_plot + ['País']],
    hue='País',
    palette='mako',
    diag_kind='kde',  # 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('Pairplot de Potencia y Producción por País', y=1.05)

# Mostrar el gráfico
plt.show()
```

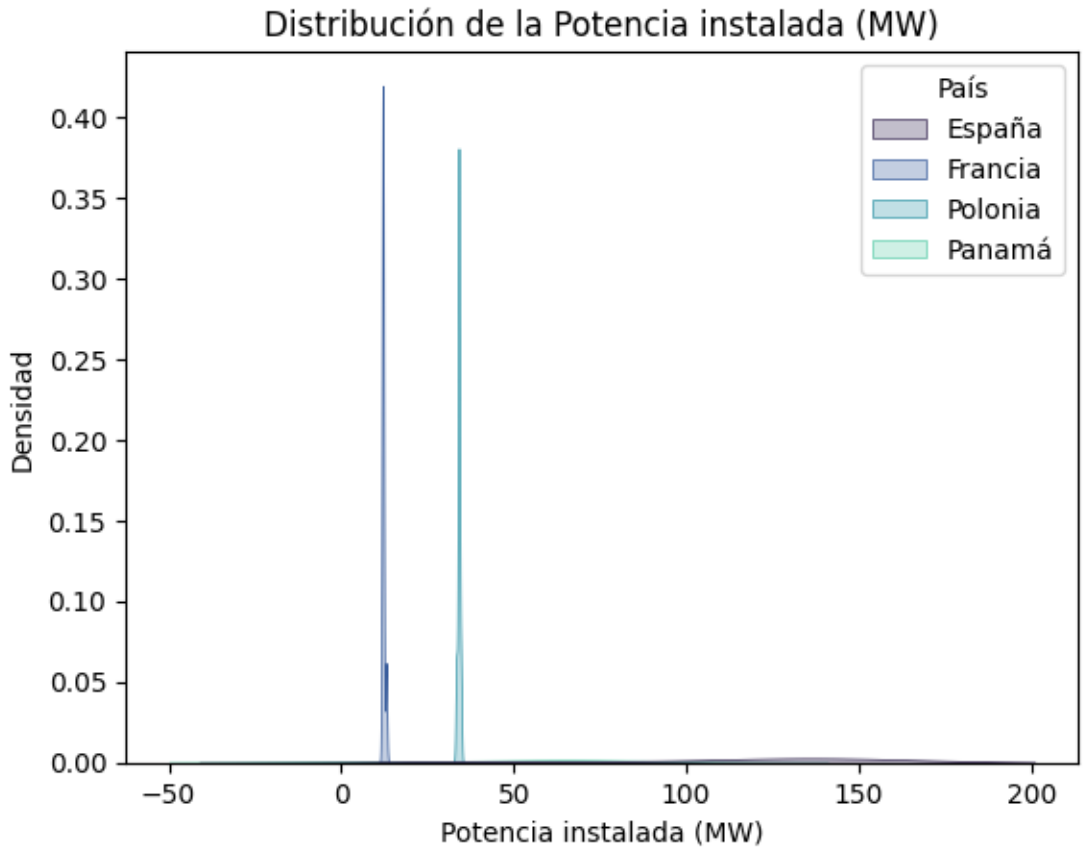


1.3.7 Gráfico KDE

Potencia instalada (MW)

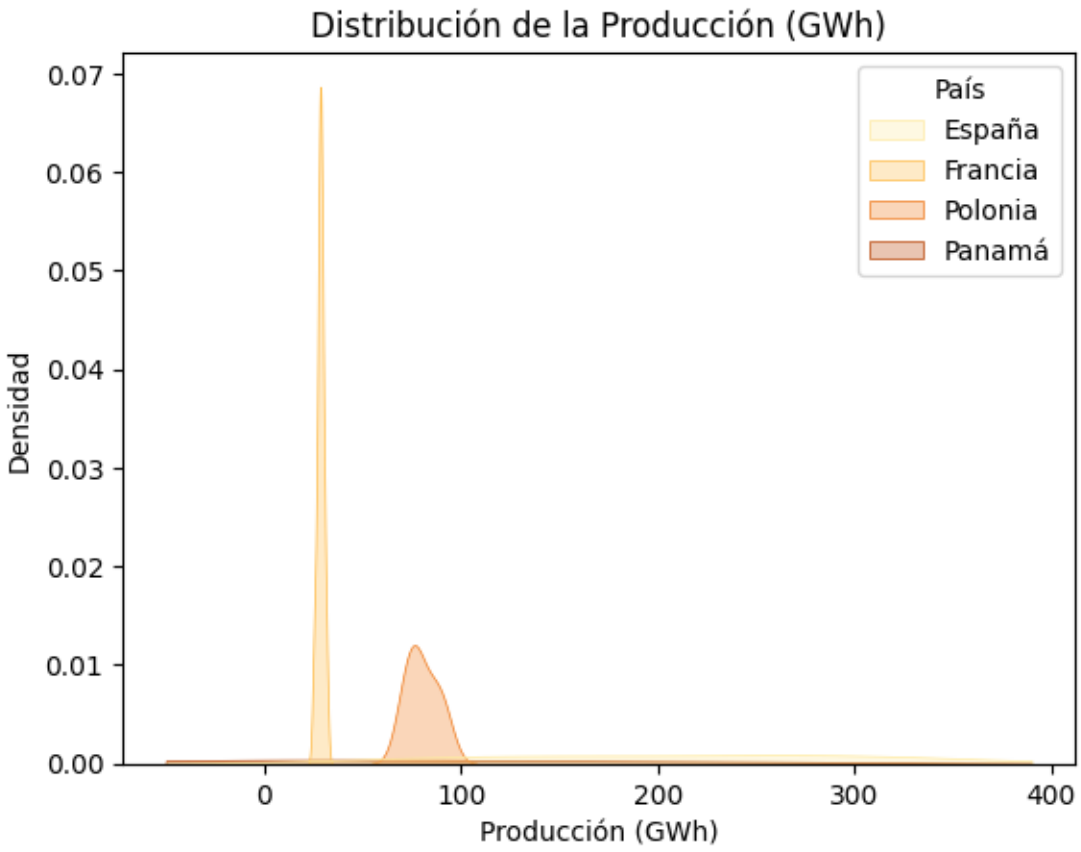
[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Definir los límites basados en los valores mínimos y máximos observados en␣
 ↪tus datos
```

```
min_potencia, max_potencia = df['Potencia instalada (MW)'].min(), df['Potencia␣
 ↪instalada (MW)'].max()

# Gráfico KDE
sns.kdeplot(
    data=df,
    x="Potencia instalada (MW)",
    fill=True,
    clip=(min_potencia - 50, max_potencia + 50),
    hue="País",
    palette="mako",
    alpha=0.3,
    linewidth=0.5
)
plt.title('Distribución de la Potencia instalada (MW)')
plt.xlabel('Potencia instalada (MW)')
plt.ylabel('Densidad')
plt.show()
```



Producción (GWh)

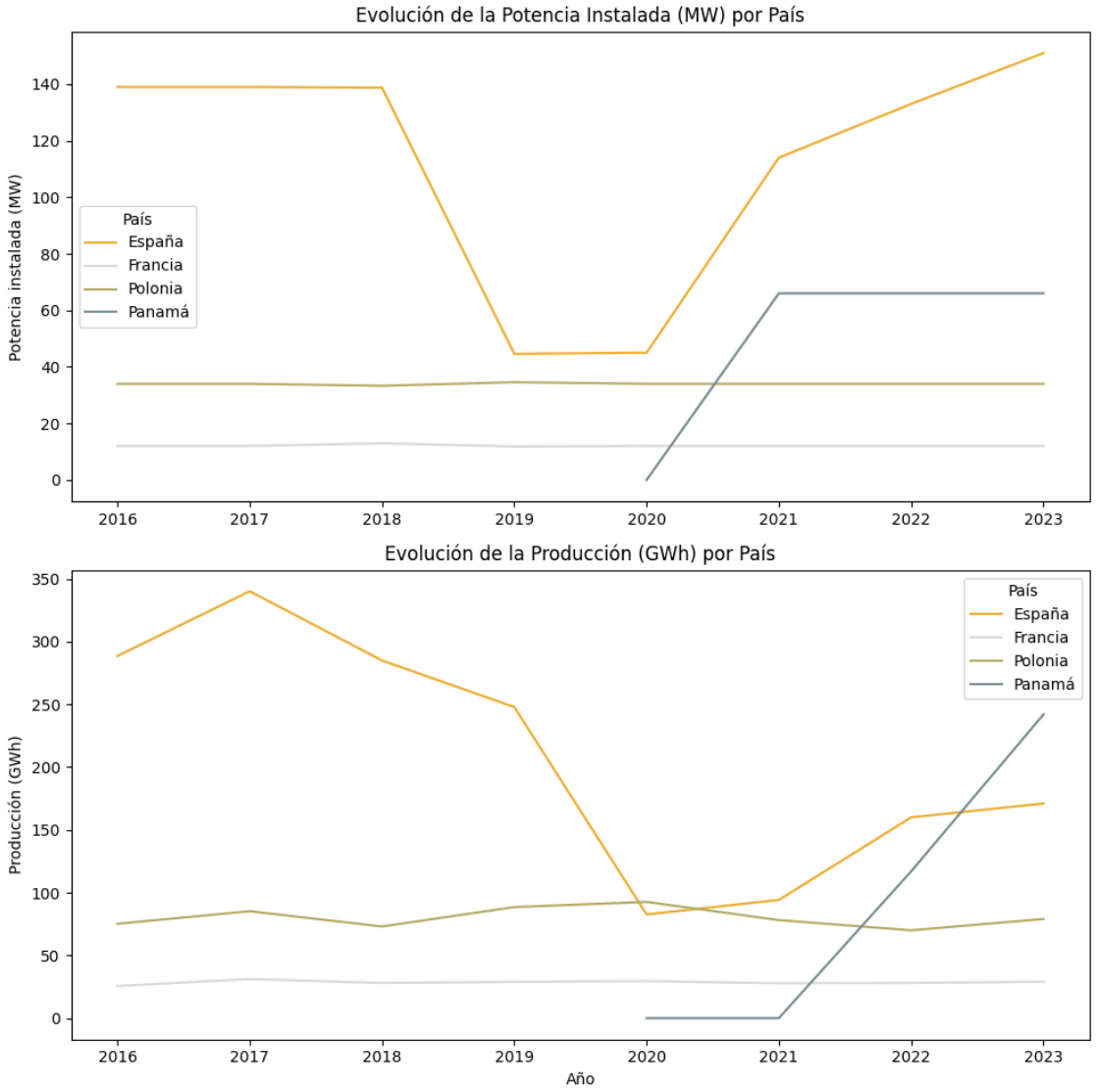
[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Definir los límites basados en los valores mínimos y máximos observados en␣
 ↪tus datos
min_potencia, max_potencia = df['Producción (GWh)'].min(), df['Producción␣
 ↪(GWh)'].max()

# Gráfico KDE
sns.kdeplot(
    data=df,
    x="Producción (GWh)",
    fill=True,
    clip=(min_potencia - 50, max_potencia + 50),
    hue="País",
    palette="YlOrBr",
    alpha=0.3,
    linewidth=0.5
)
plt.title('Distribución de la Producción (GWh)')
plt.xlabel('Producción (GWh)')
plt.ylabel('Densidad')
plt.show()
```



1.3.8 Lineplots

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

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Definir la paleta de colores
palette = 'mako'  # Se usará para otra parte del gráfico si lo necesitas
colors = {
    "España": "#ECAD30",
    "Francia": "#D9D9D7",
    "Polonia": "#B5AB66",
```

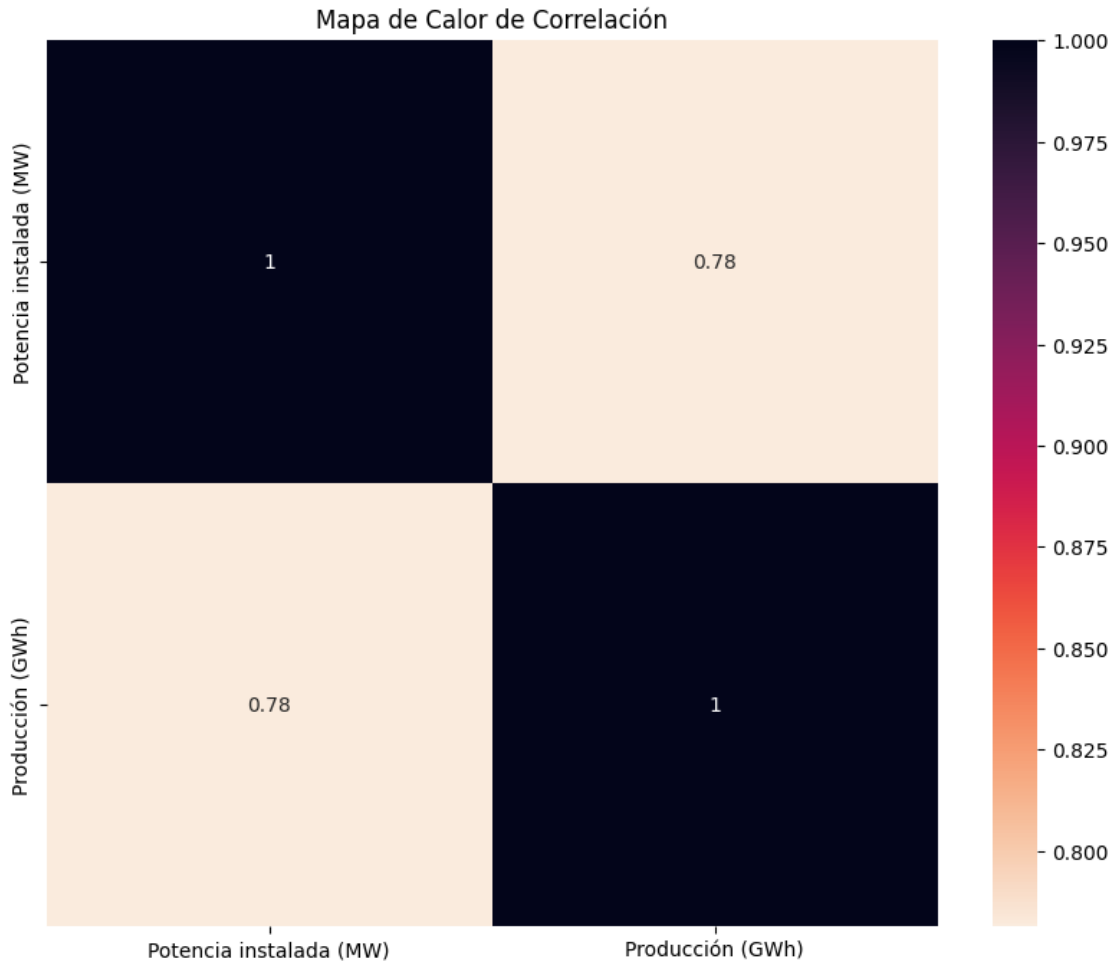
```
    "Panamá": "#788C90",
    "Otro_1": "#73BBA1",
    "Otro_2": "#6CB978"
}

# Crear figura y subplots
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(10, 10))  # Ajusta el tamaño␣
 ↪como necesario

# Subplot para "Potencia instalada (MW)"
sns.lineplot(
    ax=axs[0],
    x='Año',
    y='Potencia instalada (MW)',
    hue='País',
    data=df,
    palette=colors  # Usar la paleta de colores definida para los países
)
axs[0].set_title('Evolución de la Potencia Instalada (MW) por País')
axs[0].set_ylabel('Potencia instalada (MW)')
axs[0].set_xlabel('')

# Subplot para "Producción (GWh)"
sns.lineplot(
    ax=axs[1],
    x='Año',
    y='Producción (GWh)',
    hue='País',
    data=df,
    palette=colors  # Usar la paleta de colores definida para los países
)
axs[1].set_title('Evolución de la Producción (GWh) por País')
axs[1].set_ylabel('Producción (GWh)')
axs[1].set_xlabel('Año')

# Ajustar el layout y mostrar la leyenda
plt.tight_layout()
plt.legend(title='País', loc='upper right')

plt.show()
```



1.3.9 Heatmap

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

# Cargar datos
df = pd.read_excel('datos_potencia_prod.xlsx')

# Convertir la columna 'Fecha' a tipo datetime y extraer el año
df['Fecha'] = pd.to_datetime(df['Fecha'])
df['Año'] = pd.to_datetime(df['Fecha']).dt.year
```

```python
# Correlación y mapa de calor
corr_matrix = df[['Potencia instalada (MW)', 'Producción (GWh)']].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='rocket_r')
plt.title('Mapa de Calor de Correlación')
plt.show()
```



Una correlación de `0.78` entre dos variables, como en este caso entre **`"Potencia instalada (MW)"`** y **`"Producción (GWh)"`** sugiere que la relación es fuerte. Esto implica que los cambios en la potencia instalada pueden predecir de manera razonablemente fiable los cambios en la producción de energía.

Aunque una correlación de `0.78` es alta, *no es perfecta*. Esto significa que mientras que gran parte de la variabilidad en la producción de energía puede explicarse por cambios en la potencia instalada, hay *otros factores* que también podrían influir en la producción y que no están capturados solo por la potencia instalada.

1.4 Proyecciones

1.4.1 Modelo de Redes Neuronales

Preparación

[ ]:
```
# Preparación del Modelo
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
import numpy as np
from itertools import product

# Cargar y preparar datos
df = pd.read_excel('datos_potencia_prod.xlsx')
df['Año'] = pd.to_datetime(df['Fecha']).dt.year

# Preparar características y salida
features = df[['Año', 'País']]
output = df[['Potencia instalada (MW)', 'Producción (GWh)']]

# Preprocesamiento
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['Año']),
        ('cat', OneHotEncoder(), ['País'])
    ]
)

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

# Aplicar preprocesamiento
preprocessor.fit(X_train)
X_train_transformed = preprocessor.transform(X_train)
X_test_transformed = preprocessor.transform(X_test)
```

Definición

[ ]:
```
# Definición del Modelo
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import pandas as pd
import numpy as np
from itertools import product

# Definir y compilar el modelo
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_transformed.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(2)
])

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               768

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 2)                 130

=================================================================
Total params: 9154 (35.76 KB)
Trainable params: 9154 (35.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compilar el Modelo

[ ]:
```
# Compilar el modelo
model.compile(optimizer='adam', loss='mean_squared_error')
```

Entrenamiento con EarlyStopping

[ ]:
```
# Entrenamiento con EarlyStopping
from tensorflow.keras.callbacks import EarlyStopping

# Callback de EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Entrenar el modelo
history = model.fit(X_train_transformed, y_train, epochs=200,␣
 ↪validation_split=0.2, callbacks=[early_stopping], batch_size=32)
```

```
Epoch 1/200
1/1 [==============================] - 1s 1s/step - loss: 10211.3945 - val_loss:
14146.9814
Epoch 2/200
1/1 [==============================] - 0s 43ms/step - loss: 10206.5859 -
val_loss: 14139.3389
Epoch 3/200
1/1 [==============================] - 0s 38ms/step - loss: 10207.0391 -
val_loss: 14131.7207
Epoch 4/200
1/1 [==============================] - 0s 37ms/step - loss: 10200.0986 -
val_loss: 14124.2344
Epoch 5/200
1/1 [==============================] - 0s 35ms/step - loss: 10195.3555 -
val_loss: 14116.7783
Epoch 6/200
1/1 [==============================] - 0s 37ms/step - loss: 10184.6123 -
val_loss: 14109.5469
Epoch 7/200
1/1 [==============================] - 0s 47ms/step - loss: 10183.8184 -
val_loss: 14102.4355
Epoch 8/200
1/1 [==============================] - 0s 39ms/step - loss: 10172.8145 -
val_loss: 14095.4355
Epoch 9/200
1/1 [==============================] - 0s 38ms/step - loss: 10167.6445 -
val_loss: 14088.3252
Epoch 10/200
1/1 [==============================] - 0s 38ms/step - loss: 10161.5625 -
val_loss: 14080.8965
Epoch 11/200
1/1 [==============================] - 0s 38ms/step - loss: 10165.8359 -
val_loss: 14073.2109
Epoch 12/200
1/1 [==============================] - 0s 42ms/step - loss: 10147.4834 -
val_loss: 14065.2832
```

```
Epoch 13/200
1/1 [==============================] - 0s 42ms/step - loss: 10143.4531 -
val_loss: 14056.9004
Epoch 14/200
1/1 [==============================] - 0s 40ms/step - loss: 10140.9268 -
val_loss: 14048.3203
Epoch 15/200
1/1 [==============================] - 0s 43ms/step - loss: 10141.2930 -
val_loss: 14039.5176
Epoch 16/200
1/1 [==============================] - 0s 38ms/step - loss: 10129.6504 -
val_loss: 14030.4639
Epoch 17/200
1/1 [==============================] - 0s 40ms/step - loss: 10127.8584 -
val_loss: 14021.1855
Epoch 18/200
1/1 [==============================] - 0s 44ms/step - loss: 10123.5918 -
val_loss: 14011.6504
Epoch 19/200
1/1 [==============================] - 0s 39ms/step - loss: 10110.9590 -
val_loss: 14001.7559
Epoch 20/200
1/1 [==============================] - 0s 40ms/step - loss: 10091.6484 -
val_loss: 13991.4326
Epoch 21/200
1/1 [==============================] - 0s 42ms/step - loss: 10108.3350 -
val_loss: 13980.6777
Epoch 22/200
1/1 [==============================] - 0s 38ms/step - loss: 10100.9092 -
val_loss: 13969.5596
Epoch 23/200
1/1 [==============================] - 0s 37ms/step - loss: 10064.1133 -
val_loss: 13957.8857
Epoch 24/200
1/1 [==============================] - 0s 57ms/step - loss: 10067.8311 -
val_loss: 13945.7295
Epoch 25/200
1/1 [==============================] - 0s 41ms/step - loss: 10052.6523 -
val_loss: 13933.0674
Epoch 26/200
1/1 [==============================] - 0s 38ms/step - loss: 10041.5215 -
val_loss: 13919.8379
Epoch 27/200
1/1 [==============================] - 0s 36ms/step - loss: 10067.2705 -
val_loss: 13906.2793
Epoch 28/200
1/1 [==============================] - 0s 37ms/step - loss: 10049.1016 -
val_loss: 13892.3105
```

```
Epoch 29/200
1/1 [==============================] - 0s 37ms/step - loss: 10045.5908 -
val_loss: 13877.9316
Epoch 30/200
1/1 [==============================] - 0s 39ms/step - loss: 10004.7627 -
val_loss: 13862.9785
Epoch 31/200
1/1 [==============================] - 0s 39ms/step - loss: 9994.7422 -
val_loss: 13847.3906
Epoch 32/200
1/1 [==============================] - 0s 41ms/step - loss: 9973.6084 -
val_loss: 13831.1201
Epoch 33/200
1/1 [==============================] - 0s 43ms/step - loss: 9984.0469 -
val_loss: 13814.2529
Epoch 34/200
1/1 [==============================] - 0s 45ms/step - loss: 9957.4268 -
val_loss: 13796.7246
Epoch 35/200
1/1 [==============================] - 0s 49ms/step - loss: 9955.7451 -
val_loss: 13778.4424
Epoch 36/200
1/1 [==============================] - 0s 42ms/step - loss: 9935.1748 -
val_loss: 13759.4717
Epoch 37/200
1/1 [==============================] - 0s 41ms/step - loss: 9931.3145 -
val_loss: 13739.8691
Epoch 38/200
1/1 [==============================] - 0s 40ms/step - loss: 9898.7520 -
val_loss: 13719.4951
Epoch 39/200
1/1 [==============================] - 0s 42ms/step - loss: 9860.0303 -
val_loss: 13698.2549
Epoch 40/200
1/1 [==============================] - 0s 60ms/step - loss: 9885.0850 -
val_loss: 13676.2686
Epoch 41/200
1/1 [==============================] - 0s 38ms/step - loss: 9846.9873 -
val_loss: 13653.4111
Epoch 42/200
1/1 [==============================] - 0s 38ms/step - loss: 9851.9062 -
val_loss: 13629.6738
Epoch 43/200
1/1 [==============================] - 0s 39ms/step - loss: 9838.9531 -
val_loss: 13605.0518
Epoch 44/200
1/1 [==============================] - 0s 38ms/step - loss: 9800.5352 -
val_loss: 13579.5049
```

```
Epoch 45/200
1/1 [==============================] - 0s 38ms/step - loss: 9770.8271 -
val_loss: 13552.9004
Epoch 46/200
1/1 [==============================] - 0s 38ms/step - loss: 9797.6133 -
val_loss: 13525.5449
Epoch 47/200
1/1 [==============================] - 0s 39ms/step - loss: 9741.2471 -
val_loss: 13497.1377
Epoch 48/200
1/1 [==============================] - 0s 40ms/step - loss: 9670.8223 -
val_loss: 13467.4893
Epoch 49/200
1/1 [==============================] - 0s 38ms/step - loss: 9681.4121 -
val_loss: 13436.7578
Epoch 50/200
1/1 [==============================] - 0s 41ms/step - loss: 9663.1729 -
val_loss: 13404.8486
Epoch 51/200
1/1 [==============================] - 0s 37ms/step - loss: 9675.8330 -
val_loss: 13371.7051
Epoch 52/200
1/1 [==============================] - 0s 38ms/step - loss: 9655.3232 -
val_loss: 13337.4033
Epoch 53/200
1/1 [==============================] - 0s 48ms/step - loss: 9622.0762 -
val_loss: 13301.9014
Epoch 54/200
1/1 [==============================] - 0s 38ms/step - loss: 9617.8643 -
val_loss: 13265.2324
Epoch 55/200
1/1 [==============================] - 0s 40ms/step - loss: 9622.2891 -
val_loss: 13227.4316
Epoch 56/200
1/1 [==============================] - 0s 37ms/step - loss: 9569.4072 -
val_loss: 13188.4580
Epoch 57/200
1/1 [==============================] - 0s 36ms/step - loss: 9502.1084 -
val_loss: 13148.0684
Epoch 58/200
1/1 [==============================] - 0s 39ms/step - loss: 9496.4229 -
val_loss: 13106.2773
Epoch 59/200
1/1 [==============================] - 0s 39ms/step - loss: 9482.4785 -
val_loss: 13063.1016
Epoch 60/200
1/1 [==============================] - 0s 57ms/step - loss: 9411.5518 -
val_loss: 13018.3105
```

```
Epoch 61/200
1/1 [==============================] - 0s 40ms/step - loss: 9455.4297 -
val_loss: 12972.1895
Epoch 62/200
1/1 [==============================] - 0s 39ms/step - loss: 9396.0889 -
val_loss: 12924.6338
Epoch 63/200
1/1 [==============================] - 0s 39ms/step - loss: 9269.1641 -
val_loss: 12875.2676
Epoch 64/200
1/1 [==============================] - 0s 54ms/step - loss: 9319.3047 -
val_loss: 12824.3691
Epoch 65/200
1/1 [==============================] - 0s 37ms/step - loss: 9265.7021 -
val_loss: 12771.8721
Epoch 66/200
1/1 [==============================] - 0s 44ms/step - loss: 9151.4453 -
val_loss: 12717.5137
Epoch 67/200
1/1 [==============================] - 0s 40ms/step - loss: 9183.9365 -
val_loss: 12661.4902
Epoch 68/200
1/1 [==============================] - 0s 44ms/step - loss: 9275.0684 -
val_loss: 12604.3906
Epoch 69/200
1/1 [==============================] - 0s 42ms/step - loss: 9027.2979 -
val_loss: 12545.3711
Epoch 70/200
1/1 [==============================] - 0s 41ms/step - loss: 9104.8955 -
val_loss: 12484.6484
Epoch 71/200
1/1 [==============================] - 0s 41ms/step - loss: 8860.7373 -
val_loss: 12421.5693
Epoch 72/200
1/1 [==============================] - 0s 37ms/step - loss: 8956.1045 -
val_loss: 12356.7412
Epoch 73/200
1/1 [==============================] - 0s 39ms/step - loss: 8980.8066 -
val_loss: 12290.3691
Epoch 74/200
1/1 [==============================] - 0s 41ms/step - loss: 9112.2334 -
val_loss: 12223.1562
Epoch 75/200
1/1 [==============================] - 0s 45ms/step - loss: 8910.9678 -
val_loss: 12154.2295
Epoch 76/200
1/1 [==============================] - 0s 40ms/step - loss: 8718.4229 -
val_loss: 12083.2139
```

```
Epoch 77/200
1/1 [==============================] - 0s 39ms/step - loss: 8692.1719 -
val_loss: 12010.0703
Epoch 78/200
1/1 [==============================] - 0s 42ms/step - loss: 8742.6787 -
val_loss: 11935.3340
Epoch 79/200
1/1 [==============================] - 0s 38ms/step - loss: 8672.0518 -
val_loss: 11858.8965
Epoch 80/200
1/1 [==============================] - 0s 44ms/step - loss: 8698.3779 -
val_loss: 11781.0996
Epoch 81/200
1/1 [==============================] - 0s 42ms/step - loss: 8573.2559 -
val_loss: 11701.4980
Epoch 82/200
1/1 [==============================] - 0s 37ms/step - loss: 8557.9102 -
val_loss: 11620.0039
Epoch 83/200
1/1 [==============================] - 0s 38ms/step - loss: 8408.2168 -
val_loss: 11536.2256
Epoch 84/200
1/1 [==============================] - 0s 60ms/step - loss: 8282.0566 -
val_loss: 11450.2939
Epoch 85/200
1/1 [==============================] - 0s 40ms/step - loss: 8183.2803 -
val_loss: 11362.2100
Epoch 86/200
1/1 [==============================] - 0s 42ms/step - loss: 8131.3320 -
val_loss: 11271.8369
Epoch 87/200
1/1 [==============================] - 0s 49ms/step - loss: 8090.6606 -
val_loss: 11179.3906
Epoch 88/200
1/1 [==============================] - 0s 36ms/step - loss: 8264.4043 -
val_loss: 11085.7363
Epoch 89/200
1/1 [==============================] - 0s 38ms/step - loss: 7844.9146 -
val_loss: 10989.6025
Epoch 90/200
1/1 [==============================] - 0s 39ms/step - loss: 8184.1387 -
val_loss: 10892.6504
Epoch 91/200
1/1 [==============================] - 0s 38ms/step - loss: 8227.4707 -
val_loss: 10795.1338
Epoch 92/200
1/1 [==============================] - 0s 38ms/step - loss: 8015.3145 -
val_loss: 10695.7910
```

```
Epoch 93/200
1/1 [==============================] - 0s 36ms/step - loss: 7842.8574 -
val_loss: 10594.6562
Epoch 94/200
1/1 [==============================] - 0s 54ms/step - loss: 7931.1533 -
val_loss: 10492.4727
Epoch 95/200
1/1 [==============================] - 0s 42ms/step - loss: 7830.8867 -
val_loss: 10389.1904
Epoch 96/200
1/1 [==============================] - 0s 40ms/step - loss: 7490.2852 -
val_loss: 10283.8701
Epoch 97/200
1/1 [==============================] - 0s 39ms/step - loss: 7826.2280 -
val_loss: 10177.7803
Epoch 98/200
1/1 [==============================] - 0s 41ms/step - loss: 7696.2026 -
val_loss: 10071.1211
Epoch 99/200
1/1 [==============================] - 0s 42ms/step - loss: 7318.5488 -
val_loss: 9962.6201
Epoch 100/200
1/1 [==============================] - 0s 43ms/step - loss: 7674.8706 -
val_loss: 9853.7529
Epoch 101/200
1/1 [==============================] - 0s 40ms/step - loss: 7191.4810 -
val_loss: 9743.1826
Epoch 102/200
1/1 [==============================] - 0s 43ms/step - loss: 6872.1123 -
val_loss: 9630.0801
Epoch 103/200
1/1 [==============================] - 0s 42ms/step - loss: 7343.4619 -
val_loss: 9516.5381
Epoch 104/200
1/1 [==============================] - 0s 38ms/step - loss: 7120.1895 -
val_loss: 9401.8955
Epoch 105/200
1/1 [==============================] - 0s 37ms/step - loss: 6834.0269 -
val_loss: 9285.8887
Epoch 106/200
1/1 [==============================] - 0s 40ms/step - loss: 6983.1108 -
val_loss: 9169.5098
Epoch 107/200
1/1 [==============================] - 0s 38ms/step - loss: 6575.4805 -
val_loss: 9052.0840
Epoch 108/200
1/1 [==============================] - 0s 39ms/step - loss: 6374.2534 -
val_loss: 8932.9502
```

```
Epoch 109/200
1/1 [==============================] - 0s 46ms/step - loss: 6663.1147 -
val_loss: 8813.2979
Epoch 110/200
1/1 [==============================] - 0s 53ms/step - loss: 6962.7241 -
val_loss: 8694.3105
Epoch 111/200
1/1 [==============================] - 0s 44ms/step - loss: 6476.4922 -
val_loss: 8574.6309
Epoch 112/200
1/1 [==============================] - 0s 40ms/step - loss: 6058.1064 -
val_loss: 8453.2188
Epoch 113/200
1/1 [==============================] - 0s 37ms/step - loss: 6588.2612 -
val_loss: 8332.5918
Epoch 114/200
1/1 [==============================] - 0s 39ms/step - loss: 6516.0698 -
val_loss: 8213.0059
Epoch 115/200
1/1 [==============================] - 0s 43ms/step - loss: 6201.7695 -
val_loss: 8093.1616
Epoch 116/200
1/1 [==============================] - 0s 42ms/step - loss: 5927.4844 -
val_loss: 7973.0415
Epoch 117/200
1/1 [==============================] - 0s 41ms/step - loss: 6344.9746 -
val_loss: 7853.7251
Epoch 118/200
1/1 [==============================] - 0s 41ms/step - loss: 6359.1445 -
val_loss: 7736.1025
Epoch 119/200
1/1 [==============================] - 0s 43ms/step - loss: 6187.7817 -
val_loss: 7619.2515
Epoch 120/200
1/1 [==============================] - 0s 40ms/step - loss: 6040.5728 -
val_loss: 7503.6162
Epoch 121/200
1/1 [==============================] - 0s 38ms/step - loss: 6093.6694 -
val_loss: 7388.9282
Epoch 122/200
1/1 [==============================] - 0s 41ms/step - loss: 5763.9497 -
val_loss: 7275.1826
Epoch 123/200
1/1 [==============================] - 0s 46ms/step - loss: 5823.1582 -
val_loss: 7162.5327
Epoch 124/200
1/1 [==============================] - 0s 39ms/step - loss: 5975.8594 -
val_loss: 7051.3867
```

```
Epoch 125/200
1/1 [==============================] - 0s 40ms/step - loss: 5478.3623 -
val_loss: 6940.5439
Epoch 126/200
1/1 [==============================] - 0s 38ms/step - loss: 5646.8784 -
val_loss: 6831.3740
Epoch 127/200
1/1 [==============================] - 0s 38ms/step - loss: 5665.4116 -
val_loss: 6723.7998
Epoch 128/200
1/1 [==============================] - 0s 40ms/step - loss: 5459.9883 -
val_loss: 6617.0562
Epoch 129/200
1/1 [==============================] - 0s 42ms/step - loss: 5147.1509 -
val_loss: 6511.4893
Epoch 130/200
1/1 [==============================] - 0s 42ms/step - loss: 5721.5068 -
val_loss: 6408.9668
Epoch 131/200
1/1 [==============================] - 0s 44ms/step - loss: 5101.8813 -
val_loss: 6307.4092
Epoch 132/200
1/1 [==============================] - 0s 41ms/step - loss: 5419.1147 -
val_loss: 6207.5845
Epoch 133/200
1/1 [==============================] - 0s 50ms/step - loss: 5041.9175 -
val_loss: 6110.1440
Epoch 134/200
1/1 [==============================] - 0s 38ms/step - loss: 5215.6899 -
val_loss: 6013.7002
Epoch 135/200
1/1 [==============================] - 0s 40ms/step - loss: 4921.2397 -
val_loss: 5919.6431
Epoch 136/200
1/1 [==============================] - 0s 37ms/step - loss: 4949.8652 -
val_loss: 5827.6870
Epoch 137/200
1/1 [==============================] - 0s 37ms/step - loss: 4718.3530 -
val_loss: 5737.1304
Epoch 138/200
1/1 [==============================] - 0s 39ms/step - loss: 5391.0254 -
val_loss: 5650.0498
Epoch 139/200
1/1 [==============================] - 0s 36ms/step - loss: 4533.3740 -
val_loss: 5563.6611
Epoch 140/200
1/1 [==============================] - 0s 37ms/step - loss: 4952.4849 -
val_loss: 5478.2490
```

```
Epoch 141/200
1/1 [==============================] - 0s 40ms/step - loss: 4686.8594 -
val_loss: 5394.1558
Epoch 142/200
1/1 [==============================] - 0s 46ms/step - loss: 4909.0024 -
val_loss: 5311.5347
Epoch 143/200
1/1 [==============================] - 0s 41ms/step - loss: 4604.9336 -
val_loss: 5230.9209
Epoch 144/200
1/1 [==============================] - 0s 48ms/step - loss: 5319.6235 -
val_loss: 5152.8115
Epoch 145/200
1/1 [==============================] - 0s 40ms/step - loss: 4913.5088 -
val_loss: 5077.0430
Epoch 146/200
1/1 [==============================] - 0s 39ms/step - loss: 4204.7378 -
val_loss: 5002.9229
Epoch 147/200
1/1 [==============================] - 0s 43ms/step - loss: 4462.4800 -
val_loss: 4931.0332
Epoch 148/200
1/1 [==============================] - 0s 41ms/step - loss: 4763.1118 -
val_loss: 4861.3237
Epoch 149/200
1/1 [==============================] - 0s 39ms/step - loss: 4516.7778 -
val_loss: 4794.0547
Epoch 150/200
1/1 [==============================] - 0s 40ms/step - loss: 4849.5850 -
val_loss: 4729.3960
Epoch 151/200
1/1 [==============================] - 0s 41ms/step - loss: 4887.5132 -
val_loss: 4665.8584
Epoch 152/200
1/1 [==============================] - 0s 43ms/step - loss: 4503.1211 -
val_loss: 4602.8037
Epoch 153/200
1/1 [==============================] - 0s 42ms/step - loss: 4086.4751 -
val_loss: 4540.3438
Epoch 154/200
1/1 [==============================] - 0s 41ms/step - loss: 4044.3521 -
val_loss: 4478.3770
Epoch 155/200
1/1 [==============================] - 0s 40ms/step - loss: 4365.8511 -
val_loss: 4417.6509
Epoch 156/200
1/1 [==============================] - 0s 51ms/step - loss: 4164.2314 -
val_loss: 4358.3096
```

```
Epoch 157/200
1/1 [==============================] - 0s 40ms/step - loss: 4046.6396 -
val_loss: 4299.6958
Epoch 158/200
1/1 [==============================] - 0s 40ms/step - loss: 4408.9990 -
val_loss: 4242.6187
Epoch 159/200
1/1 [==============================] - 0s 41ms/step - loss: 4204.3389 -
val_loss: 4186.7861
Epoch 160/200
1/1 [==============================] - 0s 41ms/step - loss: 4160.7788 -
val_loss: 4131.0322
Epoch 161/200
1/1 [==============================] - 0s 38ms/step - loss: 4027.5569 -
val_loss: 4075.0203
Epoch 162/200
1/1 [==============================] - 0s 41ms/step - loss: 4161.8076 -
val_loss: 4020.7625
Epoch 163/200
1/1 [==============================] - 0s 40ms/step - loss: 3966.7385 -
val_loss: 3968.2180
Epoch 164/200
1/1 [==============================] - 0s 43ms/step - loss: 3116.8455 -
val_loss: 3915.8796
Epoch 165/200
1/1 [==============================] - 0s 39ms/step - loss: 4115.3628 -
val_loss: 3865.0613
Epoch 166/200
1/1 [==============================] - 0s 39ms/step - loss: 4538.0015 -
val_loss: 3815.3215
Epoch 167/200
1/1 [==============================] - 0s 40ms/step - loss: 3966.4937 -
val_loss: 3767.3040
Epoch 168/200
1/1 [==============================] - 0s 38ms/step - loss: 3579.1584 -
val_loss: 3719.7605
Epoch 169/200
1/1 [==============================] - 0s 41ms/step - loss: 3678.8579 -
val_loss: 3671.5071
Epoch 170/200
1/1 [==============================] - 0s 40ms/step - loss: 3889.2922 -
val_loss: 3624.0718
Epoch 171/200
1/1 [==============================] - 0s 40ms/step - loss: 3736.7715 -
val_loss: 3576.6340
Epoch 172/200
1/1 [==============================] - 0s 37ms/step - loss: 3675.3987 -
val_loss: 3530.3003
```

```
Epoch 173/200
1/1 [==============================] - 0s 42ms/step - loss: 3821.3867 -
val_loss: 3483.1851
Epoch 174/200
1/1 [==============================] - 0s 41ms/step - loss: 3363.7012 -
val_loss: 3437.0554
Epoch 175/200
1/1 [==============================] - 0s 38ms/step - loss: 3668.9265 -
val_loss: 3391.6489
Epoch 176/200
1/1 [==============================] - 0s 39ms/step - loss: 3617.5156 -
val_loss: 3347.5808
Epoch 177/200
1/1 [==============================] - 0s 38ms/step - loss: 3637.8821 -
val_loss: 3304.2017
Epoch 178/200
1/1 [==============================] - 0s 44ms/step - loss: 3208.9695 -
val_loss: 3260.9275
Epoch 179/200
1/1 [==============================] - 0s 51ms/step - loss: 3562.9377 -
val_loss: 3218.1313
Epoch 180/200
1/1 [==============================] - 0s 38ms/step - loss: 3530.6299 -
val_loss: 3176.6638
Epoch 181/200
1/1 [==============================] - 0s 37ms/step - loss: 3323.0254 -
val_loss: 3135.8511
Epoch 182/200
1/1 [==============================] - 0s 41ms/step - loss: 2855.3057 -
val_loss: 3095.5625
Epoch 183/200
1/1 [==============================] - 0s 38ms/step - loss: 3639.2754 -
val_loss: 3057.4951
Epoch 184/200
1/1 [==============================] - 0s 37ms/step - loss: 3609.6379 -
val_loss: 3018.5735
Epoch 185/200
1/1 [==============================] - 0s 39ms/step - loss: 3529.1731 -
val_loss: 2978.3308
Epoch 186/200
1/1 [==============================] - 0s 38ms/step - loss: 2907.9436 -
val_loss: 2937.6860
Epoch 187/200
1/1 [==============================] - 0s 41ms/step - loss: 3319.4592 -
val_loss: 2897.9082
Epoch 188/200
1/1 [==============================] - 0s 41ms/step - loss: 2950.2283 -
val_loss: 2856.7566
```

```
Epoch 189/200
1/1 [==============================] - 0s 40ms/step - loss: 3300.6494 -
val_loss: 2817.4287
Epoch 190/200
1/1 [==============================] - 0s 42ms/step - loss: 3514.4192 -
val_loss: 2779.9905
Epoch 191/200
1/1 [==============================] - 0s 42ms/step - loss: 3009.6907 -
val_loss: 2743.1702
Epoch 192/200
1/1 [==============================] - 0s 39ms/step - loss: 3072.0461 -
val_loss: 2706.9924
Epoch 193/200
1/1 [==============================] - 0s 41ms/step - loss: 2907.4707 -
val_loss: 2671.7148
Epoch 194/200
1/1 [==============================] - 0s 61ms/step - loss: 2842.0684 -
val_loss: 2635.2065
Epoch 195/200
1/1 [==============================] - 0s 78ms/step - loss: 3500.5500 -
val_loss: 2599.3491
Epoch 196/200
1/1 [==============================] - 0s 71ms/step - loss: 3134.4099 -
val_loss: 2563.1499
Epoch 197/200
1/1 [==============================] - 0s 53ms/step - loss: 2753.5825 -
val_loss: 2527.5244
Epoch 198/200
1/1 [==============================] - 0s 87ms/step - loss: 2594.8042 -
val_loss: 2492.4731
Epoch 199/200
1/1 [==============================] - 0s 91ms/step - loss: 2687.5029 -
val_loss: 2457.5410
Epoch 200/200
1/1 [==============================] - 0s 89ms/step - loss: 3164.4902 -
val_loss: 2422.2527
```

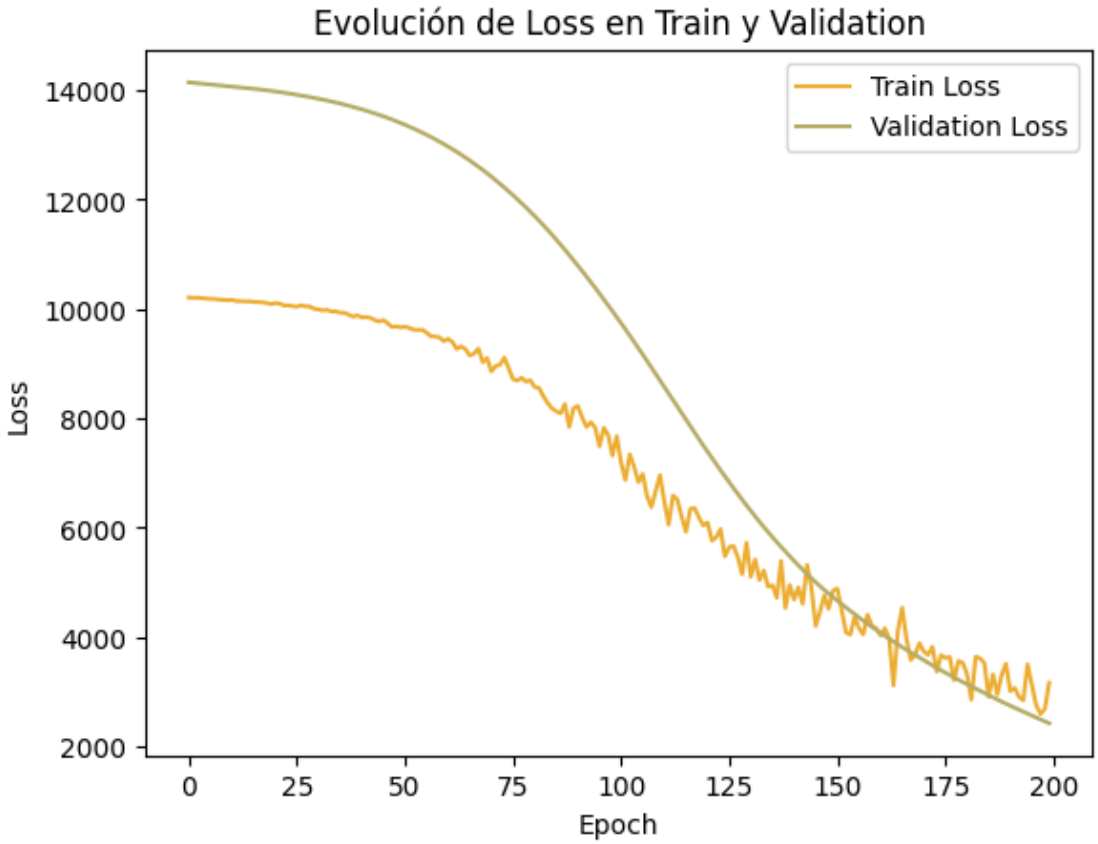
Evaluación del Modelo

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

```
1/1 [==============================] - 0s 25ms/step - loss: 2879.2595
Test Loss: 2879.259521484375
```

Visualizar Historial de Entrenamiento

[ ]:
```
import matplotlib.pyplot as plt

# Gráfico de la evolución del loss
plt.plot(history.history['loss'], label='Train Loss', color='#ECAD30')
plt.plot(history.history['val_loss'], label='Validation Loss', color='#B5AB66')
plt.title('Evolución de Loss en Train y Validation')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



Generar Predicciones

[ ]:
```
# Generar Predicciones

# Preparar datos futuros para predicción
future_years = [2024, 2025, 2026]
countries = ['España', 'Francia', 'Polonia', 'Panamá']
future_data = pd.DataFrame(list(product(future_years, countries)),␣
 ↪columns=['Año', 'País'])
```

```
# Transformar datos futuros
future_data['Año'] = pd.to_datetime(future_data['Año'], format='%Y').dt.year
future_data_transformed = preprocessor.transform(future_data)

# Predecir los valores para los años futuros
predictions = model.predict(future_data_transformed)

# Organizar y exportar las predicciones
df_predictions = pd.DataFrame(predictions, columns=['Potencia instalada (MW)',␣
↪'Producción (GWh)'])
df_predictions['Año'] = np.tile(future_years, len(countries))
df_predictions['País'] = np.repeat(countries, len(future_years))

# Agregar columnas faltantes para mantener la estructura del DataFrame original
df_predictions['Empresa'] = 'Empresa_1'  # O el nombre real de tu empresa

# Crear y asignar la columna 'Fecha' basada en el 'Año'
df_predictions['Fecha'] = pd.to_datetime(df_predictions['Año'].astype(str) +␣
↪'-12-31')

# Exportar
#df_predictions.to_csv('datos_potencia_prod_pred_nn.csv', index=False)

# Mostrar
df_predictions.tail(12)
```

```
1/1 [==============================] - 0s 44ms/step
```

[ ]:

	Potencia instalada (MW)	Producción (GWh)	Año	País	Empresa	\
0	76.822479	152.613297	2024	España	Empresa_1	
1	40.385281	80.868004	2025	España	Empresa_1	
2	56.120064	112.755264	2026	España	Empresa_1	
3	63.777851	126.509079	2024	Francia	Empresa_1	
4	81.333275	161.729858	2025	Francia	Empresa_1	
5	47.470684	94.977722	2026	Francia	Empresa_1	
6	62.600758	125.633904	2024	Polonia	Empresa_1	
7	70.644218	140.185379	2025	Polonia	Empresa_1	
8	87.121948	173.316071	2026	Polonia	Empresa_1	
9	54.570652	109.119484	2024	Panamá	Empresa_1	
10	69.386787	139.141113	2025	Panamá	Empresa_1	
11	77.567238	153.994141	2026	Panamá	Empresa_1	

	Fecha
0	2024-12-31
1	2025-12-31
2	2026-12-31

```
3  2024-12-31
4  2025-12-31
5  2026-12-31
6  2024-12-31
7  2025-12-31
8  2026-12-31
9  2024-12-31
10 2025-12-31
11 2026-12-31
```

Exportar Datos

[ ]:
```
# Exportar
df_predictions.to_csv('datos_potencia_prod_pred_nn.csv', index=False)
```

Combinar Histórico y Predicciones

[ ]:
```
# Combinar con el DataFrame original
df_combinado = pd.concat([df, df_predictions], ignore_index=True, sort=False)
df_combinado.tail(12)
```

[ ]:

	Fecha	Empresa	País	Potencia instalada (MW)	Producción (GWh)	\
28	2024-12-31	Empresa_1	España	76.822479	152.613297	
29	2025-12-31	Empresa_1	España	40.385281	80.868004	
30	2026-12-31	Empresa_1	España	56.120064	112.755264	
31	2024-12-31	Empresa_1	Francia	63.777851	126.509079	
32	2025-12-31	Empresa_1	Francia	81.333275	161.729858	
33	2026-12-31	Empresa_1	Francia	47.470684	94.977722	
34	2024-12-31	Empresa_1	Polonia	62.600758	125.633904	
35	2025-12-31	Empresa_1	Polonia	70.644218	140.185379	
36	2026-12-31	Empresa_1	Polonia	87.121948	173.316071	
37	2024-12-31	Empresa_1	Panamá	54.570652	109.119484	
38	2025-12-31	Empresa_1	Panamá	69.386787	139.141113	
39	2026-12-31	Empresa_1	Panamá	77.567238	153.994141	

	Año
28	2024
29	2025
30	2026
31	2024
32	2025
33	2026
34	2024
35	2025
36	2026
37	2024
38	2025

```
39  2026
```

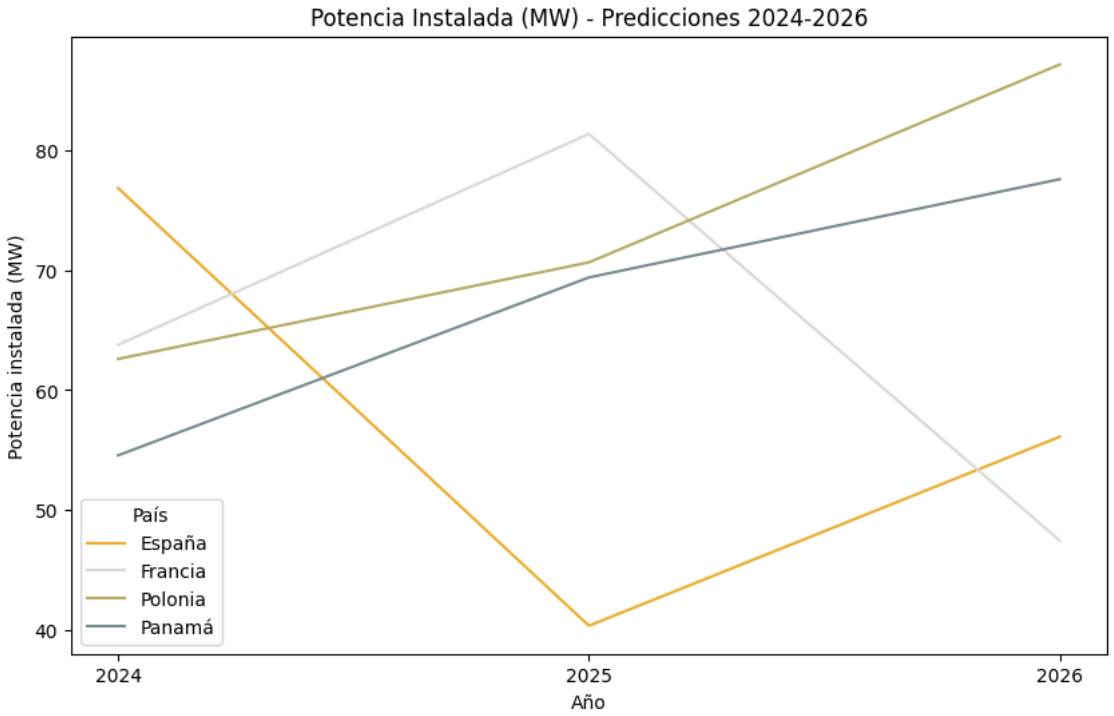
[ ]:
```python
# Exportar a CSV
df_combinado.to_csv('datos_potencia_prod_concat_nn_.csv', index=False)
```

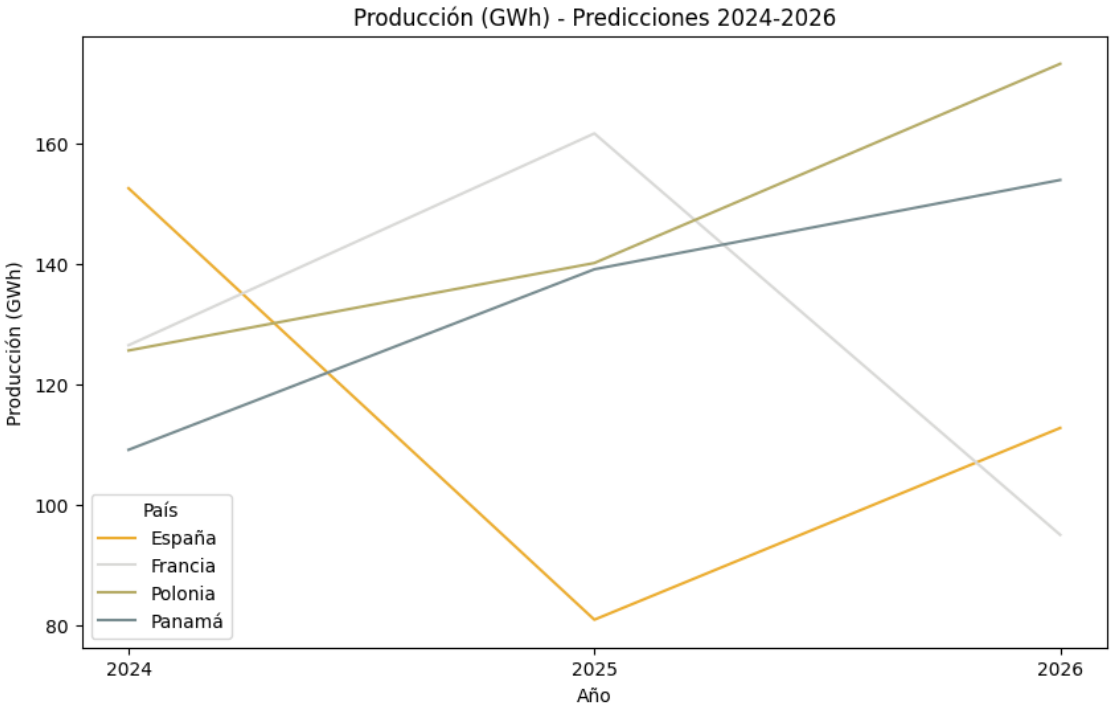
Visualizar Predicciones

[ ]:
```python
# Asegurarse de que 'Año' es de tipo entero y contiene solo el año
df_predictions['Año'] = pd.to_datetime(df_predictions['Fecha']).dt.year
df_combinado['Año'] = pd.to_datetime(df_combinado['Fecha']).dt.year

# Verificar los tipos de datos
print(df_predictions['Año'].dtype, df_combinado['Año'].dtype)
```

```
int32 int32
```

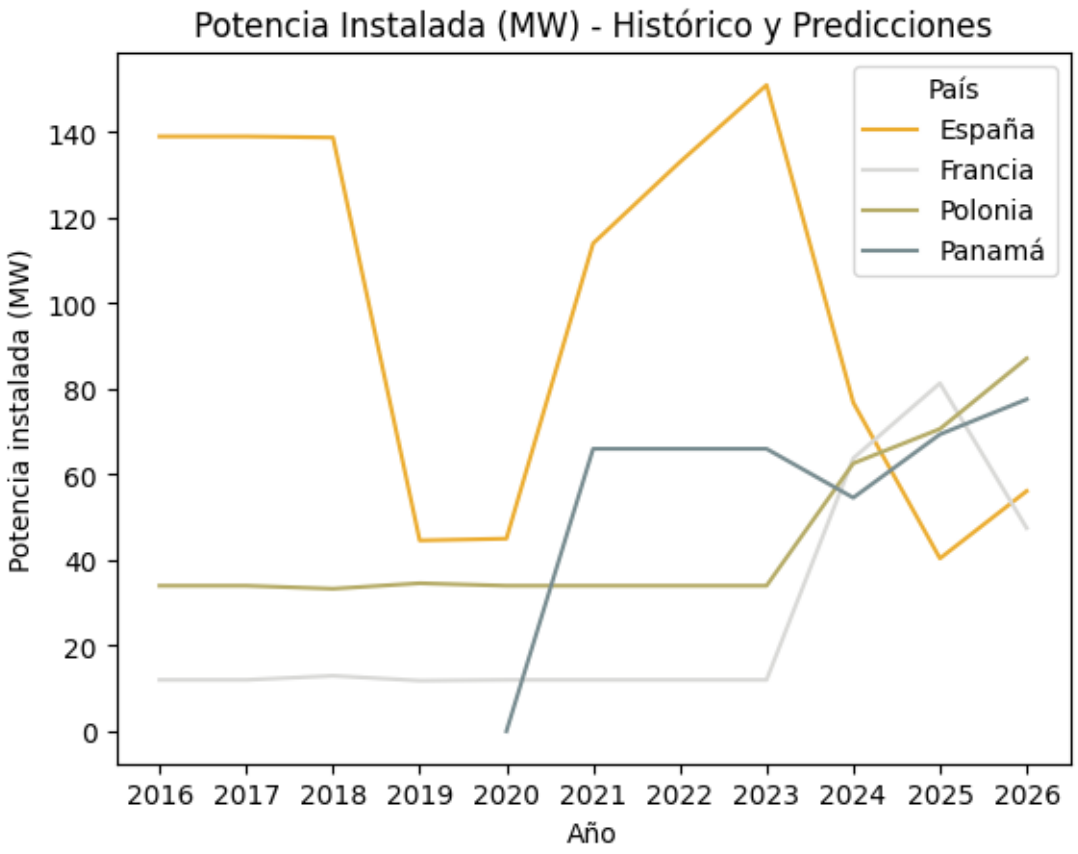
[30]:
```python
# Visualizar Predicciones
import seaborn as sns
import matplotlib.pyplot as plt

# Definir paleta de colores
colors = {
    "España": "#ECAD30",
    "Francia": "#D9D9D7",
    "Polonia": "#B5AB66",
    "Panamá": "#788C90"
}

# Gráfico para Potencia Instalada (MW) usando df_predictions
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_predictions, x='Año', y='Potencia instalada (MW)',␣
 ↪hue='País', palette=colors)
plt.title('Potencia Instalada (MW) - Predicciones 2024-2026')
plt.xticks(sorted(df_predictions['Año'].unique()))  # Forzar marcas en años␣
 ↪enteros
plt.show()

# Gráfico para Producción (GWh) usando df_predictions
plt.figure(figsize=(10, 6))
sns.lineplot(data=df_predictions, x='Año', y='Producción (GWh)', hue='País',␣
 ↪palette=colors)
plt.title('Producción (GWh) - Predicciones 2024-2026')
plt.xticks(sorted(df_predictions['Año'].unique()))  # Forzar marcas en años␣
 ↪enteros
plt.show()
```

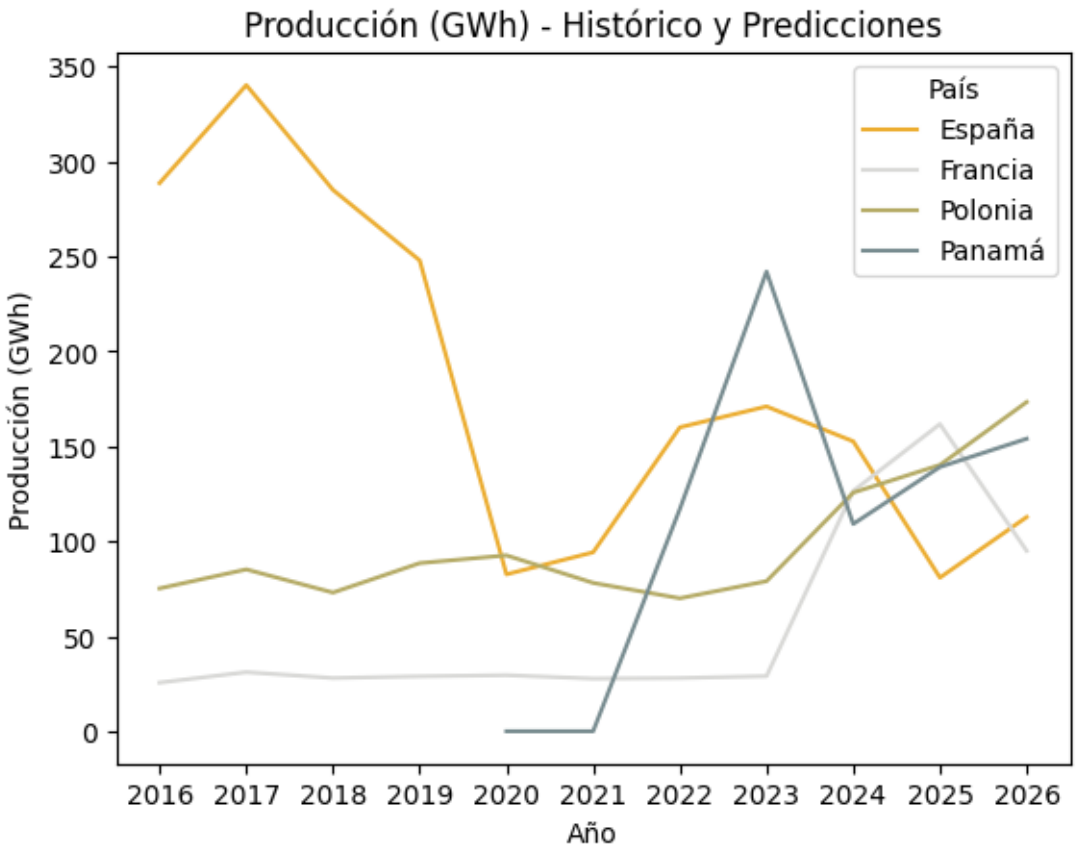

Potencia Instalada (MW) - Predicciones 2024-2026
Potencia instalada (MW)
80
70
60
50
40
2024
2025
2026
Año
País
España
Francia
Polonia
Panamá


Producción (GWh) - Predicciones 2024-2026
Producción (GWh)
160
140
120
100
80
2024
2025
2026
Año
País
España
Francia
Polonia
Panamá

Visualizar Histórico y Predicciones

[29]:
```
# Visualizar Histórico y Predicciones
import matplotlib.pyplot as plt
import seaborn as sns

# Definir paleta de colores
colors = {
    "España": "#ECAD30",
    "Francia": "#D9D9D7",
    "Polonia": "#B5AB66",
    "Panamá": "#788C90",
    "Otro_1": "#73BBA1",
    "Otro_2": "#6CB978"
}

# Gráfico para Potencia Instalada (MW)
sns.lineplot(data=df_combinado, x='Año', y='Potencia instalada (MW)',␣
 ↪hue='País', palette=colors)
plt.title('Potencia Instalada (MW) - Histórico y Predicciones')
plt.xticks(sorted(df_combinado['Año'].unique()))  # Forzar marcas en años␣
 ↪enteros
plt.show()

# Gráfico para Producción (GWh)
sns.lineplot(data=df_combinado, x='Año', y='Producción (GWh)', hue='País',␣
 ↪palette=colors)
plt.title('Producción (GWh) - Histórico y Predicciones')
plt.xticks(sorted(df_combinado['Año'].unique()))  # Forzar marcas en años␣
 ↪enteros
plt.show()
```


Potencia Instalada (MW) - Histórico y Predicciones
Potencia instalada (MW)
Año
País
España
Francia
Polonia
Panamá
0
20
40
60
80
100
120
140
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026


Producción (GWh) - Histórico y Predicciones
Producción (GWh)
Año
País
España
Francia
Polonia
Panamá
0
50
100
150
200
250
300
350
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026

1.5 Fin