

Aeropuertos - Operaciones y Pasajeros

April 30, 2024

1 Aeropuertos - Operaciones y Pasajeros

Creado por:

- V. D. Betancourt

1.1 Generar Datos Sintéticos

Se creará un dataset llamado `datos_aeropuertos.csv`, que contendrá “datos sintéticos” (generados aleatoriamente) con los siguientes campos (columnas):

- `'fecha'`: Es la fecha de cada registro (fila) que comprenderá el rango del `'2022-03-31'` al `'2024-03-31'`, y siempre corresponderán a las fechas del último día del mes.
- `'nombre_aeropuerto'`: Se refiere al nombre del aeropuerto, el cual, para simplificar este análisis, corresponderán a 12 nombres de la forma: `'Aeropuerto_1'`, `'Aeropuerto_2'`, ..., `'Aeropuerto_12'`.
- `'tipo_aeropuerto'`: Se considerarán solamente aeropuertos del tipo `'Internacional'`.
- `'numero_aerolineas'`: Será un número aleatorio entre 5 y 20 aerolíneas.
- `'destinos_nacionales'`: Será un número aleatorio entre 5 y 30 destinos nacionales.
- `'destinos_internacionales'`: Será un número aleatorio entre 1 y 10 destinos internacionales.
- `'destinos_total'`: Es la suma de `'destinos_nacionales'` y `'destinos_internacionales'`.
- `'Operaciones_comercial'`: Será un número aleatorio entre 1,000 y 50,000.
- `'Operaciones_general'`: Será un número aleatorio entre 100 y 5,000.
- `'pasajeros_comercial'`: Será un número aleatorio entre 100,000 y 5,000,000.
- `'pasajeros_general'`: Será un número aleatorio entre 1000 y 15,000.

```
[1]: import pandas as pd
import numpy as np
from datetime import datetime

# Establecer una semilla para reproducibilidad
np.random.seed(42)
```

```

# Generar fechas de cierre de mes para un año específico
fechas = pd.date_range(start='2022-03-31', end='2024-03-31', freq='M')

# Crear nombres de aeropuertos
aeropuertos = [f'Aeropuerto_{i}' for i in range(1, 13)]

# Crear DataFrame con Datos Sintéticos
data = []
for fecha in fechas:
    for aeropuerto in aeropuertos:
        destinos_nacionales = np.random.randint(10, 30)
        destinos_internacionales = np.random.randint(1, 10)
        data.append({
            'fecha': fecha,
            'nombre_aeropuerto': aeropuerto,
            'tipo_aeropuerto': 'Internacional',
            'numero_aerolineas': np.random.randint(5, 20),
            'destinos_nacionales': destinos_nacionales,
            'destinos_internacionales': destinos_internacionales,
            'destinos_total': destinos_nacionales + destinos_internacionales,
            'Operaciones_comercial': np.random.randint(1000, 50000),
            'Operaciones_general': np.random.randint(100, 5000),
            'pasajeros_comercial': np.random.randint(100000, 5000000),
            'pasajeros_general': np.random.randint(100, 15000),
        })

df = pd.DataFrame(data)

# Diccionario de Nombres Reales de Aeropuertos
# Modificar diccionario cuando se tengan disponibles los nombres reales
nombres_aeropuertos = {
    'Aeropuerto_1': 'Primer_Aeropuerto',
    'Aeropuerto_2': 'Segundo_Aeropuerto',
    'Aeropuerto_3': 'Tercer_Aeropuerto',
    'Aeropuerto_4': 'Cuarto_Aeropuerto',
    'Aeropuerto_5': 'Quinto_Aeropuerto',
    'Aeropuerto_6': 'Sexto_Aeropuerto',
    'Aeropuerto_7': 'Séptimo_Aeropuerto',
    'Aeropuerto_8': 'Octavo_Aeropuerto',
    'Aeropuerto_9': 'Noveno_Aeropuerto',
    'Aeropuerto_10': 'Décimo_Aeropuerto',
    'Aeropuerto_11': 'Undécimo_Aeropuerto',
    'Aeropuerto_12': 'Duodécimo_Aeropuerto'
}

```

```
# Reemplazar los nombres genéricos por nombres reales en el DataFrame
df['nombre_aeropuerto'] = df['nombre_aeropuerto'].map(nombres_aeropuertos)

# Mostrar el DataFrame actualizado
df.head()
```

```
[1]:
```

	fecha	nombre_aeropuerto	tipo_aeropuerto	numero_aerolineas	\
0	2022-03-31	Primer_Aeropuerto	Internacional	17	
1	2022-03-31	Segundo_Aeropuerto	Internacional	9	
2	2022-03-31	Tercer_Aeropuerto	Internacional	16	
3	2022-03-31	Cuarto_Aeropuerto	Internacional	5	
4	2022-03-31	Quinto_Aeropuerto	Internacional	16	

	destinos_nacionales	destinos_internacionales	destinos_total	\
0	16	4	20	
1	20	8	28	
2	11	8	19	
3	21	9	30	
4	25	3	28	

	Operaciones_comercial	Operaciones_general	pasajeros_comercial	\
0	39158	3872	2334489	
1	45131	3019	4572471	
2	3433	1284	1496025	
3	42434	1182	3630409	
4	2267	1628	2528388	

	pasajeros_general
0	566
1	8422
2	12409
3	2147
4	3990

```
[2]: # Asegurarse de que destinos_total es la suma de nacionales e internacionales
df['destinos_total'] = df['destinos_nacionales'] +
    df['destinos_internacionales']
```

1.1.1 Exportar Datos Sintéticos

```
[3]: # Exportar a CSV
csv_file = 'datos_aeropuertos.csv'
df.to_csv(csv_file, index=False)
```

1.1.2 Carga de Datos

En esta sección se cargan los **datos “reales”** si ya están disponibles. De lo contrario, se cargan los **datos sintéticos** creados anteriormente.

```
[4]: # Datos reales
# Si ya se tiene el CSV con los datos reales, sólo hay que cargarlo

# Cargar el DataFrame desde un archivo CSV
#df_real = pd.read_csv('datos_aeropuertos.csv')
```

```
[5]: # Cargar Datos CSV a un DataFrame
df = pd.read_csv('datos_aeropuertos.csv')
df.head()
```

```
[5]:
```

	fecha	nombre_aeropuerto	tipo_aeropuerto	numero_aerolineas	\
0	2022-03-31	Primer_Aeropuerto	Internacional	17	
1	2022-03-31	Segundo_Aeropuerto	Internacional	9	
2	2022-03-31	Tercer_Aeropuerto	Internacional	16	
3	2022-03-31	Cuarto_Aeropuerto	Internacional	5	
4	2022-03-31	Quinto_Aeropuerto	Internacional	16	

	destinos_nacionales	destinos_internacionales	destinos_total	\
0	16	4	20	
1	20	8	28	
2	11	8	19	
3	21	9	30	
4	25	3	28	

	Operaciones_comercial	Operaciones_general	pasajeros_comercial	\
0	39158	3872	2334489	
1	45131	3019	4572471	
2	3433	1284	1496025	
3	42434	1182	3630409	
4	2267	1628	2528388	

	pasajeros_general
0	566
1	8422
2	12409
3	2147
4	3990

```
[6]: # Asegurarse de que destinos_total es la suma de nacionales e internacionales
#df['destinos_total'] = df['destinos_nacionales'] +
    ↪ df['destinos_internacionales']
```

1.1.3 Información

```
[7]: # Filas y Columnas
print("Cantidad de Filas y Columnas en el DataFrame")
df.shape
```

Cantidad de Filas y Columnas en el DataFrame

```
[7]: (300, 11)
```

```
[8]: # Info General
print("Información de Variables, Cantidad de Registros No Nulos, y Tipos de_
↳Datos")
df.info()
```

Información de Variables, Cantidad de Registros No Nulos, y Tipos de Datos

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 300 entries, 0 to 299

Data columns (total 11 columns):

#	Column	Non-Null Count	Dtype
0	fecha	300 non-null	object
1	nombre_aeropuerto	300 non-null	object
2	tipo_aeropuerto	300 non-null	object
3	numero_aerolineas	300 non-null	int64
4	destinos_nacionales	300 non-null	int64
5	destinos_internacionales	300 non-null	int64
6	destinos_total	300 non-null	int64
7	Operaciones_comercial	300 non-null	int64
8	Operaciones_general	300 non-null	int64
9	pasajeros_comercial	300 non-null	int64
10	pasajeros_general	300 non-null	int64

dtypes: int64(8), object(3)

memory usage: 25.9+ KB

```
[9]: # Variables (Columnas)
print("Nombres de las Variables (Columnas)")
df.columns
```

Nombres de las Variables (Columnas)

```
[9]: Index(['fecha', 'nombre_aeropuerto', 'tipo_aeropuerto', 'numero_aerolineas',
        'destinos_nacionales', 'destinos_internacionales', 'destinos_total',
        'Operaciones_comercial', 'Operaciones_general', 'pasajeros_comercial',
        'pasajeros_general'],
        dtype='object')
```

1.2 Análisis Exploratorio de Datos (EDA) Básico

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

# Asegúrate de que matplotlib muestre los gráficos
%matplotlib inline

# Se sigue usando el mismo df previo

# Visualización de las primeras filas del conjunto de datos
print("Primeras filas del conjunto de datos:")
df.head()
```

Primeras filas del conjunto de datos:

```
[10]:
```

	fecha	nombre_aeropuerto	tipo_aeropuerto	numero_aerolineas	\
0	2022-03-31	Primer_Aeropuerto	Internacional	17	
1	2022-03-31	Segundo_Aeropuerto	Internacional	9	
2	2022-03-31	Tercer_Aeropuerto	Internacional	16	
3	2022-03-31	Cuarto_Aeropuerto	Internacional	5	
4	2022-03-31	Quinto_Aeropuerto	Internacional	16	

	destinos_nacionales	destinos_internacionales	destinos_total	\
0	16	4	20	
1	20	8	28	
2	11	8	19	
3	21	9	30	
4	25	3	28	

	Operaciones_comercial	Operaciones_general	pasajeros_comercial	\
0	39158	3872	2334489	
1	45131	3019	4572471	
2	3433	1284	1496025	
3	42434	1182	3630409	
4	2267	1628	2528388	

	pasajeros_general
0	566
1	8422
2	12409
3	2147
4	3990

1.2.1 Estadísticas

```
[11]: # Resumen estadístico de las columnas numéricas
print("\nResumen estadístico del conjunto de datos:")
df.describe()
```

Resumen estadístico del conjunto de datos:

```
[11]:
```

	numero_aerolineas	destinos_nacionales	destinos_internacionales	\
count	300.000000	300.000000	300.000000	
mean	12.013333	19.816667	4.960000	
std	4.480335	5.834655	2.576924	
min	5.000000	10.000000	1.000000	
25%	8.000000	15.000000	3.000000	
50%	12.000000	20.500000	5.000000	
75%	16.000000	25.000000	7.000000	
max	19.000000	29.000000	9.000000	

	destinos_total	Operaciones_comercial	Operaciones_general	\
count	300.000000	300.000000	300.000000	
mean	24.776667	24804.473333	2487.136667	
std	6.322978	14526.371602	1412.868771	
min	12.000000	1060.000000	102.000000	
25%	19.000000	11989.750000	1132.750000	
50%	25.000000	24666.500000	2598.500000	
75%	30.000000	37977.000000	3655.000000	
max	38.000000	49988.000000	4995.000000	

	pasajeros_comercial	pasajeros_general
count	3.000000e+02	300.000000
mean	2.616075e+06	7347.833333
std	1.418837e+06	4172.858786
min	1.004040e+05	101.000000
25%	1.324220e+06	3722.250000
50%	2.625982e+06	7229.000000
75%	3.858899e+06	10937.750000
max	4.997954e+06	14890.000000

1.2.2 Barplot

Operaciones Comerciales

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

# Asegúrate de que la columna fecha es tipo datetime si no lo es
```

```

df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna
↳ 'year_month' si no existe

# Configuración de Seaborn
sns.set(style="white")

# Configurar una paleta de colores
palette = sns.color_palette("crest", n_colors=len(df['nombre_aeropuerto']).
↳ unique()))

# Preparar los datos para Seaborn FacetGrid
operaciones_comerciales_long = df[['nombre_aeropuerto', 'year_month',
↳ 'Operaciones_comercial']]

# Crear el FacetGrid
g = sns.FacetGrid(operaciones_comerciales_long, col='year_month', col_wrap=4,
↳ height=3, aspect=1.5)
g.map_dataframe(sns.barplot, x='nombre_aeropuerto', y='Operaciones_comercial',
↳ hue='nombre_aeropuerto', palette=palette, order=df['nombre_aeropuerto'].
↳ unique(), legend=False)

# Ajustar propiedades de las barras después de crearlas
for ax in g.axes.flat:
    for bar in ax.patches:
        bar.set_edgecolor('black') # Ajustar el color de borde
        bar.set_alpha(0.7) # Ajustar la transparencia

# Ajustar títulos y etiquetas
g.set_titles('{col_name}')
g.set_axis_labels('Aeropuerto', 'Operaciones Comerciales')
for ax in g.axes.flatten():
    for label in ax.get_xticklabels():
        label.set_rotation(45)

# Ajustar espacio entre gráficos para evitar solapamiento
plt.subplots_adjust(hspace=0.4, wspace=0.4)

# Añadir un título general sobre todos los subgráficos
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Operaciones Comerciales en Aeropuertos Vistos en Subgráficos
↳ Mensuales', fontsize=16)

# Mostrar el gráfico
plt.show()

```


Operaciones Comerciales en Aeropuertos Vistos en Subgráficos Mensuales



Pasajeros Comerciales

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

```

# Asegúrate de que la columna fecha es tipo datetime si no lo es
df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna
↳ 'year_month' si no existe

# Configuración de Seaborn
sns.set(style="white")

# Configurar una paleta de colores
palette = sns.color_palette("magma", n_colors=len(df['nombre_aeropuerto']).
↳ unique()))

# Preparar los datos para Seaborn FacetGrid
operaciones_comerciales_long = df[['nombre_aeropuerto', 'year_month',
↳ 'pasajeros_comercial']]

# Crear el FacetGrid
g = sns.FacetGrid(operaciones_comerciales_long, col='year_month', col_wrap=4,
↳ height=3, aspect=1.5)
g.map_dataframe(sns.barplot, x='nombre_aeropuerto', y='pasajeros_comercial',
↳ hue='nombre_aeropuerto', palette=palette, order=df['nombre_aeropuerto'].
↳ unique(), legend=False)

# Ajustar propiedades de las barras después de crearlas
for ax in g.axes.flat:
    for bar in ax.patches:
        bar.set_edgecolor('black') # Ajustar el color de borde
        bar.set_alpha(0.7) # Ajustar la transparencia

# Ajustar títulos y etiquetas
g.set_titles('{col_name}')
g.set_axis_labels('Aeropuerto', 'Pasajeros Comerciales')
for ax in g.axes.flatten():
    for label in ax.get_xticklabels():
        label.set_rotation(45)

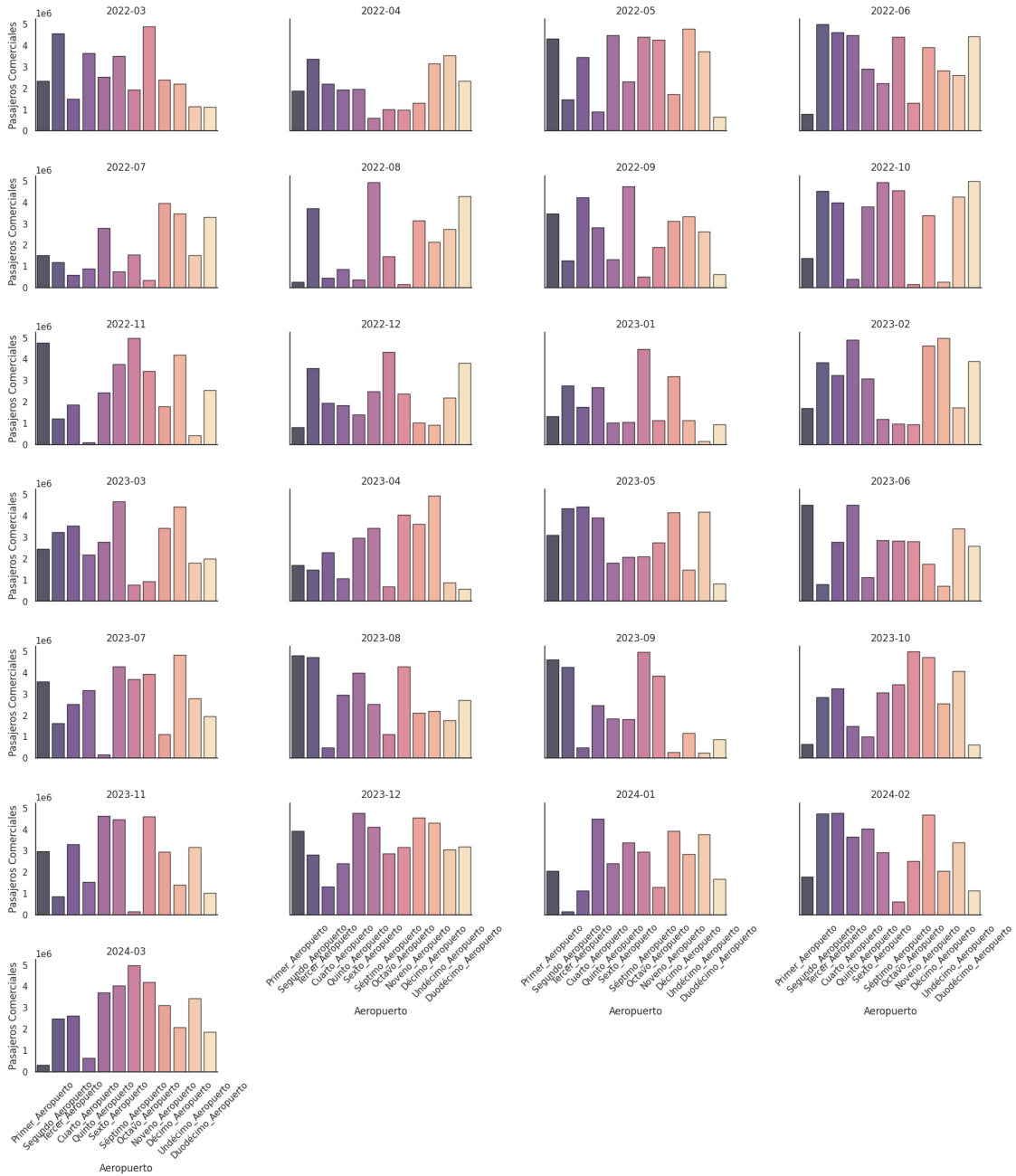
# Ajustar espacio entre gráficos para evitar solapamiento
plt.subplots_adjust(hspace=0.4, wspace=0.4)

# Añadir un título general sobre todos los subgráficos
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Pasajeros Comerciales en Aeropuertos Vistos en Subgráficos
↳ Mensuales', fontsize=16)

# Mostrar el gráfico
plt.show()

```

Pasajeros Comerciales en Aeropuertos Vistos en Subgráficos Mensuales



1.2.3 Lineplot

Operaciones Comerciales

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

```

import pandas as pd

# Asegurarse de que la columna fecha es tipo datetime si no lo es
df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna
↳ 'year_month' si no existe

# Configuración de Seaborn
sns.set(style="white") # Cambiado a "whitegrid" para mejor visualización de
↳ las líneas

# Configurar una paleta de colores
palette = sns.color_palette("crest", n_colors=len(df['nombre_aeropuerto']).
↳ unique())

# Crear el FacetGrid, cada columna es un aeropuerto
g = sns.FacetGrid(df, col='nombre_aeropuerto', col_wrap=4, height=3, aspect=1.5)

# Mapear un lineplot para cada subgráfico del grid
g.map_dataframe(sns.lineplot, x='year_month', y='Operaciones_comercial',
↳ color='#f7a541')

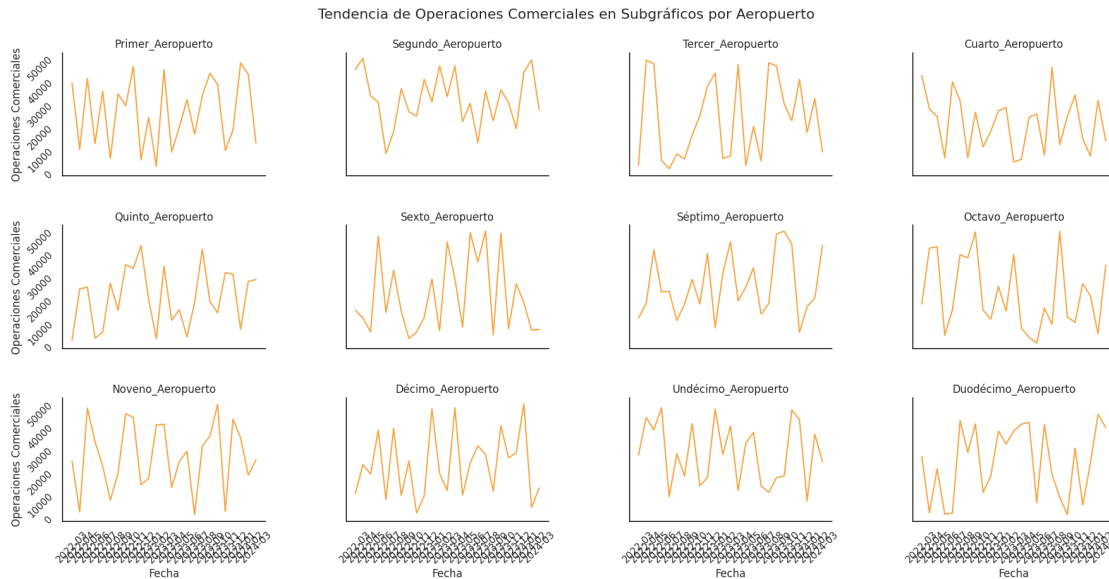
# Ajustar títulos y etiquetas
g.set_titles('{col_name}')
g.set_axis_labels('Fecha', 'Operaciones Comerciales')
for ax in g.axes.flatten():
    ax.tick_params(labelrotation=45) # Rotar etiquetas del eje X para mejor
↳ visualización

# Ajustar espacio entre gráficos para evitar solapamiento
plt.subplots_adjust(hspace=0.4, wspace=0.4)

# Añadir un título general sobre todos los subgráficos
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Tendencia de Operaciones Comerciales en Subgráficos por
↳ Aeropuerto', fontsize=16)

# Mostrar el gráfico
plt.show()

```



Pasajeros Comercial

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

# Asegurarse de que la columna fecha es tipo datetime si no lo es
df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna_
↳ 'year_month' si no existe

# Configuración de Seaborn
sns.set(style="white")

# Configurar una paleta de colores
palette = sns.color_palette("crest", n_colors=len(df['nombre_aeropuerto']).
↳ unique()))

# Crear el FacetGrid, cada columna es un aeropuerto
g = sns.FacetGrid(df, col='nombre_aeropuerto', col_wrap=4, height=3, aspect=1.5)

# Mapear un lineplot para cada subgráfico del grid
g.map_dataframe(sns.lineplot, x='year_month', y='pasajeros_comercial',
↳ color='#31b2f2')

# Ajustar títulos y etiquetas
g.set_titles('{col_name}')
g.set_axis_labels('Fecha', 'Pasajeros Comerciales')
```

```

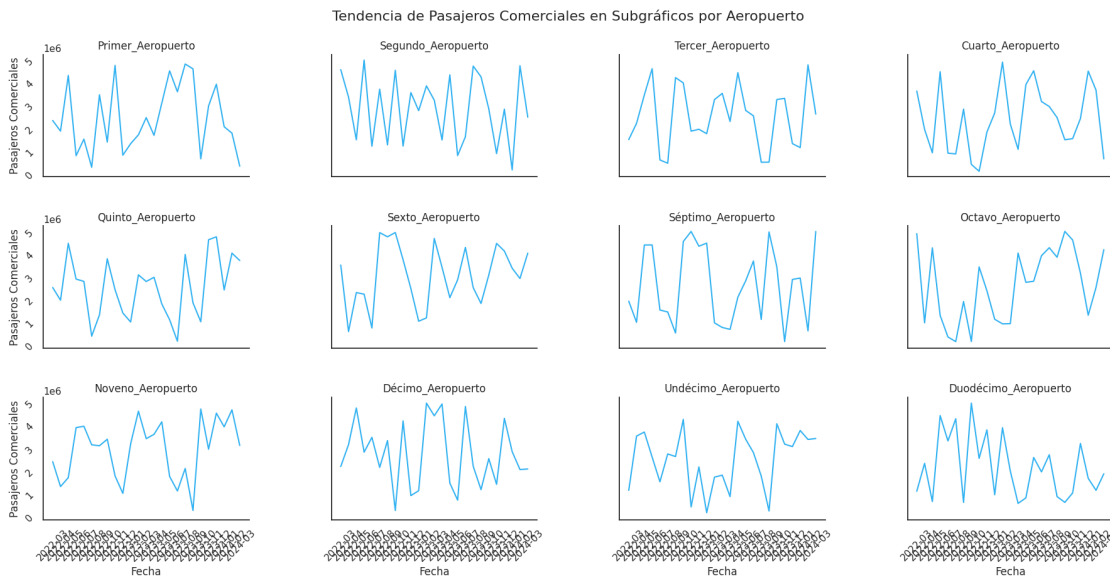
for ax in g.axes.flatten():
    ax.tick_params(labelrotation=45) # Rotar etiquetas del eje X para mejor
    ↪ visualización

# Ajustar espacio entre gráficos para evitar solapamiento
plt.subplots_adjust(hspace=0.4, wspace=0.4)

# Añadir un título general sobre todos los subgráficos
plt.subplots_adjust(top=0.9)
g.fig.suptitle('Tendencia de Pasajeros Comerciales en Subgráficos por
    ↪ Aeropuerto', fontsize=16)

# Mostrar el gráfico
plt.show()

```



1.2.4 Boxplot

Operaciones Comerciales

```

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

# Asegúrate de que la columna 'fecha' es tipo datetime si no lo es
df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna
    ↪ 'year_month'

```

```

# Configuración de Seaborn
sns.set(style="white")

# Seleccionar una paleta de colores
palette = sns.color_palette("mako", n_colors=len(df['nombre_aeropuerto']).
    ↳unique()))

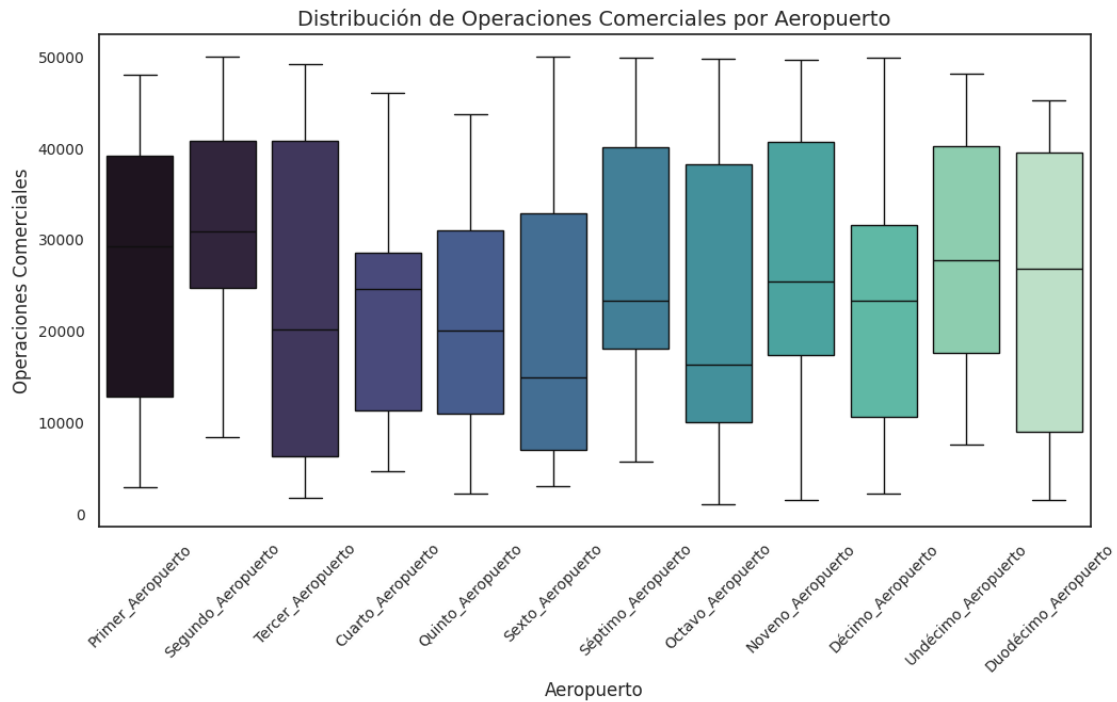
# Crear un boxplot de Operaciones Comerciales para cada aeropuerto
plt.figure(figsize=(12, 6)) # Configurar el tamaño de la figura

# Crear el boxplot usando la paleta de colores seleccionada
sns.boxplot(x='nombre_aeropuerto', y='Operaciones_comercial', data=df,
    ↳hue='nombre_aeropuerto', palette=palette)

# Ajustar detalles del gráfico
plt.title('Distribución de Operaciones Comerciales por Aeropuerto', fontsize=14)
plt.xlabel('Aeropuerto', fontsize=12) # Ajustar el tamaño de la etiqueta del
    ↳eje X
plt.ylabel('Operaciones Comerciales', fontsize=12) # Ajustar el tamaño de la
    ↳etiqueta del eje Y
plt.xticks(rotation=45) # Rotar las etiquetas del eje X para mejor legibilidad
plt.tick_params(axis='x', labelsz=10) # Ajustar el tamaño de las etiquetas
    ↳de los ticks del eje X
plt.tick_params(axis='y', labelsz=10) # Ajustar el tamaño de las etiquetas
    ↳de los ticks del eje Y

# Mostrar el gráfico
plt.show()

```



1.2.5 Outliers

Operaciones Comerciales

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

# Asegúrate de que la columna 'fecha' es tipo datetime si no lo es
df['fecha'] = pd.to_datetime(df['fecha'])
df['year_month'] = df['fecha'].dt.strftime('%Y-%m') # Crear columna
↳ 'year_month'

# Configuración de Seaborn
sns.set(style="white")

# Seleccionar una paleta de colores
palette = sns.color_palette("magma", n_colors=len(df['nombre_aeropuerto']).
↳ unique())

# Crear un boxplot de Operaciones Comerciales para cada aeropuerto
plt.figure(figsize=(12, 6)) # Configurar el tamaño de la figura

# Crear el boxplot usando la paleta de colores seleccionada
```



```

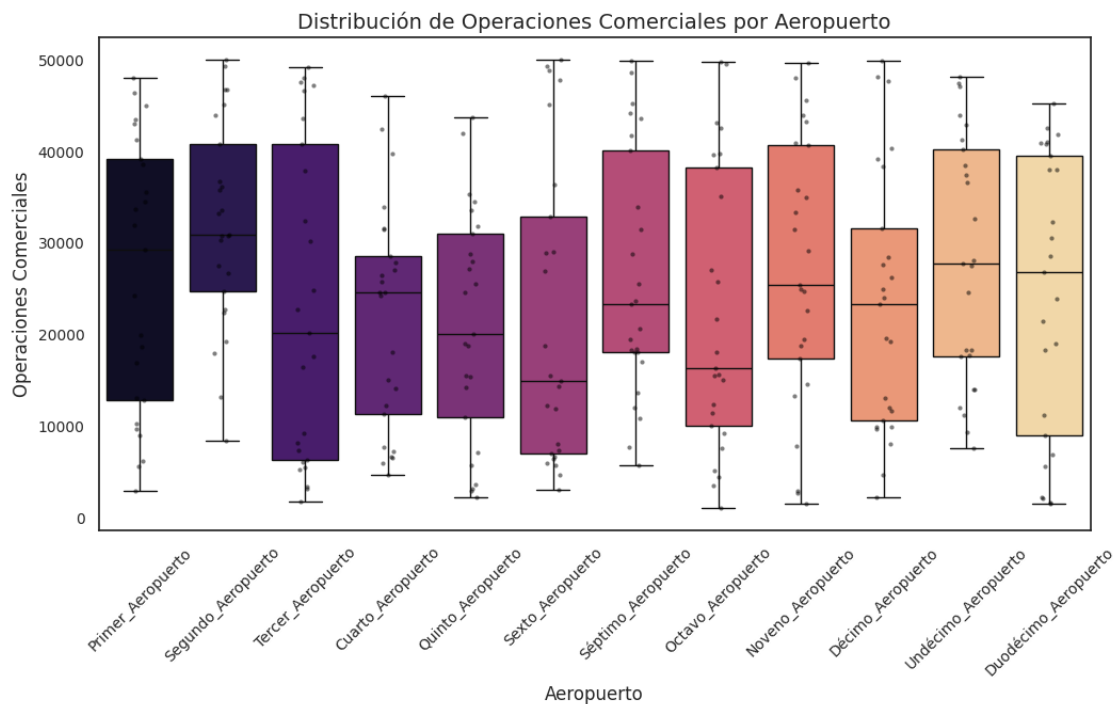
ax = sns.boxplot(x='nombre_aeropuerto', y='Operaciones_comercial', data=df,
    hue='nombre_aeropuerto', palette=palette, showfliers=True)

# Añadir stripplot para mostrar todos los puntos de datos
sns.stripplot(x='nombre_aeropuerto', y='Operaciones_comercial', data=df,
    color='black', size=3, jitter=True, alpha=0.5)

# Ajustar detalles del gráfico
plt.title('Distribución de Operaciones Comerciales por Aeropuerto', fontsize=14)
plt.xlabel('Aeropuerto', fontsize=12) # Ajustar el tamaño de la etiqueta del
    eje X
plt.ylabel('Operaciones Comerciales', fontsize=12) # Ajustar el tamaño de la
    etiqueta del eje Y
plt.xticks(rotation=45) # Rotar las etiquetas del eje X para mejor legibilidad
plt.tick_params(axis='x', labelsiz=10) # Ajustar el tamaño de las etiquetas
    de los ticks del eje X
plt.tick_params(axis='y', labelsiz=10) # Ajustar el tamaño de las etiquetas
    de los ticks del eje Y

# Mostrar el gráfico
plt.show()

```



1.3 Modelo de Redes Neuronales

```
[18]: import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Asegurarse de que la columna fecha es tipo datetime
df['fecha'] = pd.to_datetime(df['fecha'])

# Convertir 'fecha' en componentes numéricos
df['year'] = df['fecha'].dt.year
df['month'] = df['fecha'].dt.month

# Codificar variables categóricas
le = LabelEncoder()
df['nombre_aeropuerto_encoded'] = le.fit_transform(df['nombre_aeropuerto'])

# Normalizar los datos - Crear dos scalers
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

features = ['year', 'month', 'nombre_aeropuerto_encoded']
df[features] = scaler_X.fit_transform(df[features])

# Preparar características y etiquetas
X = df[features]
y = df[['numero_aerolineas', 'destinos_nacionales', 'destinos_internacionales',
        'Operaciones_comercial', 'Operaciones_general', 'pasajeros_comercial',
        'pasajeros_general']]
y = scaler_y.fit_transform(y) # Normalizar salidas usando un scaler diferente

# Dividir los datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)

# Construcción del modelo
inputs = Input(shape=(X_train.shape[1],))
x = Dense(64, activation='relu')(inputs)
x = Dropout(0.5)(x)
x = Dense(64, activation='relu')(x)
outputs = Dense(y_train.shape[1], activation='linear')(x) # Usar 'linear'
        porque las salidas están normalizadas
```

```

model = Model(inputs, outputs)
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse')

# Entrenamiento del modelo
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.1)

# Evaluación del modelo
loss = model.evaluate(X_test, y_test)
print(f"Loss en el conjunto de prueba: {loss}")

```

```

Epoch 1/100
7/7 [=====] - 1s 42ms/step - loss: 0.1848 - val_loss: 0.0950
Epoch 2/100
7/7 [=====] - 0s 8ms/step - loss: 0.1248 - val_loss: 0.0959
Epoch 3/100
7/7 [=====] - 0s 7ms/step - loss: 0.1117 - val_loss: 0.0912
Epoch 4/100
7/7 [=====] - 0s 7ms/step - loss: 0.0994 - val_loss: 0.0910
Epoch 5/100
7/7 [=====] - 0s 9ms/step - loss: 0.0982 - val_loss: 0.0920
Epoch 6/100
7/7 [=====] - 0s 8ms/step - loss: 0.0936 - val_loss: 0.0892
Epoch 7/100
7/7 [=====] - 0s 9ms/step - loss: 0.0920 - val_loss: 0.0898
Epoch 8/100
7/7 [=====] - 0s 8ms/step - loss: 0.0910 - val_loss: 0.0900
Epoch 9/100
7/7 [=====] - 0s 8ms/step - loss: 0.0919 - val_loss: 0.0896
Epoch 10/100
7/7 [=====] - 0s 8ms/step - loss: 0.0917 - val_loss: 0.0901
Epoch 11/100
7/7 [=====] - 0s 8ms/step - loss: 0.0896 - val_loss: 0.0905
Epoch 12/100
7/7 [=====] - 0s 8ms/step - loss: 0.0895 - val_loss: 0.0899

```

Epoch 13/100
7/7 [=====] - 0s 11ms/step - loss: 0.0908 - val_loss: 0.0901
Epoch 14/100
7/7 [=====] - 0s 8ms/step - loss: 0.0897 - val_loss: 0.0895
Epoch 15/100
7/7 [=====] - 0s 8ms/step - loss: 0.0908 - val_loss: 0.0906
Epoch 16/100
7/7 [=====] - 0s 8ms/step - loss: 0.0895 - val_loss: 0.0903
Epoch 17/100
7/7 [=====] - 0s 8ms/step - loss: 0.0898 - val_loss: 0.0908
Epoch 18/100
7/7 [=====] - 0s 8ms/step - loss: 0.0896 - val_loss: 0.0896
Epoch 19/100
7/7 [=====] - 0s 7ms/step - loss: 0.0898 - val_loss: 0.0896
Epoch 20/100
7/7 [=====] - 0s 8ms/step - loss: 0.0903 - val_loss: 0.0891
Epoch 21/100
7/7 [=====] - 0s 12ms/step - loss: 0.0891 - val_loss: 0.0921
Epoch 22/100
7/7 [=====] - 0s 8ms/step - loss: 0.0897 - val_loss: 0.0906
Epoch 23/100
7/7 [=====] - 0s 8ms/step - loss: 0.0903 - val_loss: 0.0911
Epoch 24/100
7/7 [=====] - 0s 8ms/step - loss: 0.0894 - val_loss: 0.0912
Epoch 25/100
7/7 [=====] - 0s 8ms/step - loss: 0.0903 - val_loss: 0.0908
Epoch 26/100
7/7 [=====] - 0s 8ms/step - loss: 0.0894 - val_loss: 0.0895
Epoch 27/100
7/7 [=====] - 0s 8ms/step - loss: 0.0890 - val_loss: 0.0903
Epoch 28/100
7/7 [=====] - 0s 8ms/step - loss: 0.0896 - val_loss: 0.0894

Epoch 29/100
7/7 [=====] - 0s 8ms/step - loss: 0.0896 - val_loss: 0.0907
Epoch 30/100
7/7 [=====] - 0s 7ms/step - loss: 0.0897 - val_loss: 0.0904
Epoch 31/100
7/7 [=====] - 0s 8ms/step - loss: 0.0894 - val_loss: 0.0905
Epoch 32/100
7/7 [=====] - 0s 8ms/step - loss: 0.0895 - val_loss: 0.0901
Epoch 33/100
7/7 [=====] - 0s 9ms/step - loss: 0.0890 - val_loss: 0.0918
Epoch 34/100
7/7 [=====] - 0s 8ms/step - loss: 0.0893 - val_loss: 0.0911
Epoch 35/100
7/7 [=====] - 0s 8ms/step - loss: 0.0894 - val_loss: 0.0918
Epoch 36/100
7/7 [=====] - 0s 8ms/step - loss: 0.0902 - val_loss: 0.0927
Epoch 37/100
7/7 [=====] - 0s 8ms/step - loss: 0.0896 - val_loss: 0.0931
Epoch 38/100
7/7 [=====] - 0s 10ms/step - loss: 0.0895 - val_loss: 0.0886
Epoch 39/100
7/7 [=====] - 0s 8ms/step - loss: 0.0891 - val_loss: 0.0938
Epoch 40/100
7/7 [=====] - 0s 9ms/step - loss: 0.0889 - val_loss: 0.0911
Epoch 41/100
7/7 [=====] - 0s 8ms/step - loss: 0.0904 - val_loss: 0.0922
Epoch 42/100
7/7 [=====] - 0s 8ms/step - loss: 0.0891 - val_loss: 0.0919
Epoch 43/100
7/7 [=====] - 0s 7ms/step - loss: 0.0892 - val_loss: 0.0912
Epoch 44/100
7/7 [=====] - 0s 7ms/step - loss: 0.0892 - val_loss: 0.0929

Epoch 45/100
7/7 [=====] - 0s 7ms/step - loss: 0.0894 - val_loss: 0.0905
Epoch 46/100
7/7 [=====] - 0s 7ms/step - loss: 0.0893 - val_loss: 0.0910
Epoch 47/100
7/7 [=====] - 0s 7ms/step - loss: 0.0896 - val_loss: 0.0915
Epoch 48/100
7/7 [=====] - 0s 7ms/step - loss: 0.0883 - val_loss: 0.0901
Epoch 49/100
7/7 [=====] - 0s 7ms/step - loss: 0.0903 - val_loss: 0.0909
Epoch 50/100
7/7 [=====] - 0s 7ms/step - loss: 0.0894 - val_loss: 0.0921
Epoch 51/100
7/7 [=====] - 0s 6ms/step - loss: 0.0898 - val_loss: 0.0909
Epoch 52/100
7/7 [=====] - 0s 7ms/step - loss: 0.0892 - val_loss: 0.0922
Epoch 53/100
7/7 [=====] - 0s 7ms/step - loss: 0.0876 - val_loss: 0.0904
Epoch 54/100
7/7 [=====] - 0s 7ms/step - loss: 0.0886 - val_loss: 0.0913
Epoch 55/100
7/7 [=====] - 0s 7ms/step - loss: 0.0881 - val_loss: 0.0924
Epoch 56/100
7/7 [=====] - 0s 7ms/step - loss: 0.0888 - val_loss: 0.0915
Epoch 57/100
7/7 [=====] - 0s 9ms/step - loss: 0.0898 - val_loss: 0.0923
Epoch 58/100
7/7 [=====] - 0s 7ms/step - loss: 0.0888 - val_loss: 0.0909
Epoch 59/100
7/7 [=====] - 0s 8ms/step - loss: 0.0886 - val_loss: 0.0922
Epoch 60/100
7/7 [=====] - 0s 7ms/step - loss: 0.0889 - val_loss: 0.0940

Epoch 61/100
7/7 [=====] - 0s 8ms/step - loss: 0.0883 - val_loss: 0.0914
Epoch 62/100
7/7 [=====] - 0s 7ms/step - loss: 0.0890 - val_loss: 0.0931
Epoch 63/100
7/7 [=====] - 0s 8ms/step - loss: 0.0888 - val_loss: 0.0921
Epoch 64/100
7/7 [=====] - 0s 8ms/step - loss: 0.0883 - val_loss: 0.0934
Epoch 65/100
7/7 [=====] - 0s 7ms/step - loss: 0.0894 - val_loss: 0.0932
Epoch 66/100
7/7 [=====] - 0s 7ms/step - loss: 0.0894 - val_loss: 0.0902
Epoch 67/100
7/7 [=====] - 0s 7ms/step - loss: 0.0896 - val_loss: 0.0920
Epoch 68/100
7/7 [=====] - 0s 7ms/step - loss: 0.0888 - val_loss: 0.0928
Epoch 69/100
7/7 [=====] - 0s 7ms/step - loss: 0.0892 - val_loss: 0.0915
Epoch 70/100
7/7 [=====] - 0s 10ms/step - loss: 0.0890 - val_loss: 0.0941
Epoch 71/100
7/7 [=====] - 0s 11ms/step - loss: 0.0894 - val_loss: 0.0932
Epoch 72/100
7/7 [=====] - 0s 10ms/step - loss: 0.0892 - val_loss: 0.0927
Epoch 73/100
7/7 [=====] - 0s 13ms/step - loss: 0.0891 - val_loss: 0.0923
Epoch 74/100
7/7 [=====] - 0s 19ms/step - loss: 0.0893 - val_loss: 0.0920
Epoch 75/100
7/7 [=====] - 0s 13ms/step - loss: 0.0892 - val_loss: 0.0907
Epoch 76/100
7/7 [=====] - 0s 13ms/step - loss: 0.0888 - val_loss: 0.0934

Epoch 77/100
7/7 [=====] - 0s 11ms/step - loss: 0.0889 - val_loss: 0.0915
Epoch 78/100
7/7 [=====] - 0s 9ms/step - loss: 0.0885 - val_loss: 0.0918
Epoch 79/100
7/7 [=====] - 0s 10ms/step - loss: 0.0883 - val_loss: 0.0926
Epoch 80/100
7/7 [=====] - 0s 10ms/step - loss: 0.0878 - val_loss: 0.0921
Epoch 81/100
7/7 [=====] - 0s 9ms/step - loss: 0.0878 - val_loss: 0.0925
Epoch 82/100
7/7 [=====] - 0s 12ms/step - loss: 0.0882 - val_loss: 0.0926
Epoch 83/100
7/7 [=====] - 0s 10ms/step - loss: 0.0879 - val_loss: 0.0931
Epoch 84/100
7/7 [=====] - 0s 9ms/step - loss: 0.0883 - val_loss: 0.0943
Epoch 85/100
7/7 [=====] - 0s 11ms/step - loss: 0.0889 - val_loss: 0.0922
Epoch 86/100
7/7 [=====] - 0s 9ms/step - loss: 0.0890 - val_loss: 0.0930
Epoch 87/100
7/7 [=====] - 0s 9ms/step - loss: 0.0880 - val_loss: 0.0938
Epoch 88/100
7/7 [=====] - 0s 16ms/step - loss: 0.0875 - val_loss: 0.0913
Epoch 89/100
7/7 [=====] - 0s 13ms/step - loss: 0.0895 - val_loss: 0.0921
Epoch 90/100
7/7 [=====] - 0s 9ms/step - loss: 0.0884 - val_loss: 0.0908
Epoch 91/100
7/7 [=====] - 0s 10ms/step - loss: 0.0897 - val_loss: 0.0907
Epoch 92/100
7/7 [=====] - 0s 10ms/step - loss: 0.0897 - val_loss: 0.0926


```

Epoch 93/100
7/7 [=====] - 0s 15ms/step - loss: 0.0888 - val_loss: 0.0897
Epoch 94/100
7/7 [=====] - 0s 14ms/step - loss: 0.0890 - val_loss: 0.0920
Epoch 95/100
7/7 [=====] - 0s 13ms/step - loss: 0.0878 - val_loss: 0.0922
Epoch 96/100
7/7 [=====] - 0s 11ms/step - loss: 0.0884 - val_loss: 0.0915
Epoch 97/100
7/7 [=====] - 0s 10ms/step - loss: 0.0874 - val_loss: 0.0943
Epoch 98/100
7/7 [=====] - 0s 11ms/step - loss: 0.0889 - val_loss: 0.0916
Epoch 99/100
7/7 [=====] - 0s 13ms/step - loss: 0.0889 - val_loss: 0.0943
Epoch 100/100
7/7 [=====] - 0s 14ms/step - loss: 0.0877 - val_loss: 0.0911
2/2 [=====] - 0s 8ms/step - loss: 0.0895
Loss en el conjunto de prueba: 0.08953887969255447

```

```

[19]: # Resumen del Modelo
      model.summary()

```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 3)]	0
dense (Dense)	(None, 64)	256
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 7)	455

```

=====
Total params: 4871 (19.03 KB)
Trainable params: 4871 (19.03 KB)
Non-trainable params: 0 (0.00 Byte)

```

1.4 Generar Predicciones

```
[20]: import pandas as pd
import numpy as np

nuevos_nombres_aeropuertos = [
    'Primer_Aeropuerto', 'Segundo_Aeropuerto', 'Tercer_Aeropuerto',
    'Cuarto_Aeropuerto', 'Quinto_Aeropuerto', 'Sexto_Aeropuerto',
    'Séptimo_Aeropuerto', 'Octavo_Aeropuerto', 'Noveno_Aeropuerto',
    'Décimo_Aeropuerto', 'Undécimo_Aeropuerto', 'Duodécimo_Aeropuerto'
]

# Año y mes para la predicción
anio = 2024
mes = 4

# Asegurarse de que los nombres de las columnas coincidan con los utilizados
# durante el entrenamiento del scaler
columnas_para_prediccion = ['year', 'month', 'nombre_aeropuerto_encoded']

# Inicializar predicted_df justo antes de generar predicciones
predicted_df = pd.DataFrame()

# Generar y almacenar las predicciones para cada aeropuerto
for aeropuerto in nuevos_nombres_aeropuertos:
    # Obtener el código del aeropuerto con LabelEncoder
    codigo_aeropuerto = le.transform([aeropuerto])[0]

    # Preparar los datos de entrada como DataFrame para mantener los nombres de
    # las columnas
    new_data_example = pd.DataFrame([[anio, mes, codigo_aeropuerto]],
    columns=columnas_para_prediccion)

    # Normalizar los datos usando scaler_X
    new_data_normalized = scaler_X.transform(new_data_example)

    # Hacer la predicción
    prediction = model.predict(new_data_normalized)
    prediction = scaler_y.inverse_transform(prediction) # Des-normalizar las
    predicciones
    prediction = np.round(prediction).astype(int) # Redondear y convertir a
    enteros

    # Formatear la fecha para la nueva fila
```

```

    fecha_prediccion = f"{anio}-{mes:02d}-30" # Asume que el día de cierre es
    ↪ el último del mes

    # Agregar la predicción al DataFrame
    new_row = pd.DataFrame({
        'nombre_aeropuerto': [aeropuerto],
        'fecha': [fecha_prediccion],
        'numero_aerolineas': [prediction[0, 0]],
        'destinos_nacionales': [prediction[0, 1]],
        'destinos_internacionales': [prediction[0, 2]],
        'destinos_totales': [prediction[0, 1] + prediction[0, 2]],
        'Operaciones_comercial': [prediction[0, 3]],
        'Operaciones_general': [prediction[0, 4]],
        'pasajeros_comercial': [prediction[0, 5]],
        'pasajeros_general': [prediction[0, 6]]
    }, columns=['nombre_aeropuerto', 'fecha', 'numero_aerolineas',
    ↪ 'destinos_nacionales', 'destinos_internacionales', 'destinos_totales',
    ↪ 'Operaciones_comercial', 'Operaciones_general', 'pasajeros_comercial',
    ↪ 'pasajeros_general'])

    predicted_df = pd.concat([predicted_df, new_row], ignore_index=True)

    # Mostrar el nuevo DataFrame (primeras filas)
    predicted_df.head(25)

```

```

1/1 [=====] - 0s 95ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 33ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 20ms/step

```

```

[20]:
      nombre_aeropuerto  fecha  numero_aerolineas  destinos_nacionales \
0      Primer_Aeropuerto  2024-04-30             12                20
1      Segundo_Aeropuerto  2024-04-30             13                20
2      Tercer_Aeropuerto  2024-04-30             13                21
3      Cuarto_Aeropuerto  2024-04-30             11                19
4      Quinto_Aeropuerto  2024-04-30             12                20
5      Sexto_Aeropuerto  2024-04-30             13                21
6      Séptimo_Aeropuerto  2024-04-30             13                21
7      Octavo_Aeropuerto  2024-04-30             11                20

```

8	Noveno_Aeropuerto	2024-04-30	11	20
9	Décimo_Aeropuerto	2024-04-30	11	19
10	Undécimo_Aeropuerto	2024-04-30	13	21
11	Duodécimo_Aeropuerto	2024-04-30	11	19

	destinos_internacionales	destinos_totales	Operaciones_comercial \
0	5	25	25066
1	5	25	25083
2	5	26	24848
3	5	24	23186
4	5	25	25068
5	5	26	24999
6	5	26	24923
7	5	25	24865
8	5	25	24282
9	5	24	23877
10	5	26	24785
11	5	24	23421

	Operaciones_general	pasajeros_comercial	pasajeros_general
0	2169	2694190	6773
1	2077	2787317	6941
2	2030	2860266	6705
3	2392	2519562	6534
4	2125	2734832	6926
5	2057	2811626	6869
6	2044	2835946	6787
7	2243	2636510	6533
8	2352	2586356	6292
9	2411	2525129	6201
10	2018	2885447	6632
11	2407	2508222	6328

1.4.1 Combinar/Concatenar DataFrames

```
[21]: import pandas as pd

# Concatenar el DataFrame inicial con el de predicciones
combined_df = pd.concat([df, predicted_df], ignore_index=True)

# Convertir la columna 'fecha' a datetime si aún no lo está
combined_df['fecha'] = pd.to_datetime(combined_df['fecha'])

# Mostrar el DataFrame combinado
combined_df.head()
```

```
[21]:
```

	fecha	nombre_aeropuerto	tipo_aeropuerto	numero_aerolineas	\
0	2022-03-31	Primer_Aeropuerto	Internacional	17	
1	2022-03-31	Segundo_Aeropuerto	Internacional	9	
2	2022-03-31	Tercer_Aeropuerto	Internacional	16	
3	2022-03-31	Cuarto_Aeropuerto	Internacional	5	
4	2022-03-31	Quinto_Aeropuerto	Internacional	16	

	destinos_nacionales	destinos_internacionales	destinos_total	\
0	16	4	20.0	
1	20	8	28.0	
2	11	8	19.0	
3	21	9	30.0	
4	25	3	28.0	

	Operaciones_comercial	Operaciones_general	pasajeros_comercial	\
0	39158	3872	2334489	
1	45131	3019	4572471	
2	3433	1284	1496025	
3	42434	1182	3630409	
4	2267	1628	2528388	

	pasajeros_general	year_month	year	month	nombre_aeropuerto_encoded	\
0	566	2022-03	0.0	0.181818	0.454545	
1	8422	2022-03	0.0	0.181818	0.636364	
2	12409	2022-03	0.0	0.181818	0.909091	
3	2147	2022-03	0.0	0.181818	0.000000	
4	3990	2022-03	0.0	0.181818	0.545455	

	destinos_totales
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN

```
[22]: # Configurar la fecha de interés
fecha_interes = pd.to_datetime('2024-04-30')

# Verificar si la fecha de interés está en el DataFrame
fecha_presente = fecha_interes in combined_df['fecha'].values
print("¿Está la fecha '2024-04-30' presente en el DataFrame combinado?:",
      fecha_presente)
```

¿Está la fecha '2024-04-30' presente en el DataFrame combinado?: True

```
[23]: # Filtrar el DataFrame para mostrar todas las entradas en la fecha de interés
entradas_fecha_interes = combined_df[combined_df['fecha'] == fecha_interes]
```

```
entradas_fecha_interes.head()
```

```
[23]:
```

	fecha	nombre_aeropuerto	tipo_aeropuerto	numero_aerolineas	\
300	2024-04-30	Primer_Aeropuerto	NaN	12	
301	2024-04-30	Segundo_Aeropuerto	NaN	13	
302	2024-04-30	Tercer_Aeropuerto	NaN	13	
303	2024-04-30	Cuarto_Aeropuerto	NaN	11	
304	2024-04-30	Quinto_Aeropuerto	NaN	12	

	destinos_nacionales	destinos_internacionales	destinos_total	\
300	20	5	NaN	
301	20	5	NaN	
302	21	5	NaN	
303	19	5	NaN	
304	20	5	NaN	

	Operaciones_comercial	Operaciones_general	pasajeros_comercial	\
300	25066	2169	2694190	
301	25083	2077	2787317	
302	24848	2030	2860266	
303	23186	2392	2519562	
304	25068	2125	2734832	

	pasajeros_general	year_month	year	month	nombre_aeropuerto_encoded	\
300	6773	NaN	NaN	NaN	NaN	
301	6941	NaN	NaN	NaN	NaN	
302	6705	NaN	NaN	NaN	NaN	
303	6534	NaN	NaN	NaN	NaN	
304	6926	NaN	NaN	NaN	NaN	

	destinos_totales
300	25.0
301	25.0
302	26.0
303	24.0
304	25.0

```
[24]: # Filas y Columnas
combined_df.shape
```

```
[24]: (312, 16)
```

```
[25]: # Variables (Columnas)
combined_df.columns
```

```
[25]: Index(['fecha', 'nombre_aeropuerto', 'tipo_aeropuerto', 'numero_aerolineas',
        'destinos_nacionales', 'destinos_internacionales', 'destinos_total',
```

```
'Operaciones_comercial', 'Operaciones_general', 'pasajeros_comercial',
'pasajeros_general', 'year_month', 'year', 'month',
'nombre_aeropuerto_encoded', 'destinos_totales'],
dtype='object')
```

```
[26]: # Info General
combined_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 312 entries, 0 to 311
Data columns (total 16 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   fecha                                312 non-null    datetime64[ns]
1   nombre_aeropuerto                   312 non-null    object
2   tipo_aeropuerto                     300 non-null    object
3   numero_aerolineas                   312 non-null    int64
4   destinos_nacionales                  312 non-null    int64
5   destinos_internacionales             312 non-null    int64
6   destinos_total                       300 non-null    float64
7   Operaciones_comercial                312 non-null    int64
8   Operaciones_general                 312 non-null    int64
9   pasajeros_comercial                 312 non-null    int64
10  pasajeros_general                   312 non-null    int64
11  year_month                           300 non-null    object
12  year                                 300 non-null    float64
13  month                               300 non-null    float64
14  nombre_aeropuerto_encoded            300 non-null    float64
15  destinos_totales                     12 non-null     float64
dtypes: datetime64[ns](1), float64(5), int64(7), object(3)
memory usage: 39.1+ KB
```

1.5 Graficar Proyecciones

Preparación

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

# Fechas
combined_df['fecha'] = pd.to_datetime(combined_df['fecha'])
combined_df['year_month'] = combined_df['fecha'].dt.strftime('%Y-%m')

# Agregar una columna para distinguir entre datos históricos y predicciones
combined_df['Tipo'] = 'Histórico'
combined_df.loc[combined_df['fecha'] >= '2024-04-30', 'Tipo'] = 'Predicción'
```

1.5.1 Duplicados

```
[33]: # Filtrar por la fecha de predicción y verificar duplicados
fecha_prediccion = '2024-04'
predicciones_abril_2024 = combined_df[combined_df['year_month'] ==
    ↪ fecha_prediccion]
print("Predicciones:")
print(predicciones_abril_2024[['nombre_aeropuerto', 'year_month',
    ↪ 'Operaciones_comercial', 'Tipo']])

# Verificar si hay duplicados en la fecha de predicción
duplicados = predicciones_abril_2024.duplicated(subset=['nombre_aeropuerto',
    ↪ 'year_month'], keep=False)
print("\nPredicciones Duplicadas:\n")
print(predicciones_abril_2024[duplicados])
```

Predicciones:

	nombre_aeropuerto	year_month	Operaciones_comercial	Tipo
300	Primer_Aeropuerto	2024-04	25066	Predicción
301	Segundo_Aeropuerto	2024-04	25083	Predicción
302	Tercer_Aeropuerto	2024-04	24848	Predicción
303	Cuarto_Aeropuerto	2024-04	23186	Predicción
304	Quinto_Aeropuerto	2024-04	25068	Predicción
305	Sexto_Aeropuerto	2024-04	24999	Predicción
306	Séptimo_Aeropuerto	2024-04	24923	Predicción
307	Octavo_Aeropuerto	2024-04	24865	Predicción
308	Noveno_Aeropuerto	2024-04	24282	Predicción
309	Décimo_Aeropuerto	2024-04	23877	Predicción
310	Undécimo_Aeropuerto	2024-04	24785	Predicción
311	Duodécimo_Aeropuerto	2024-04	23421	Predicción

Predicciones Duplicadas:

Empty DataFrame

Columns: [fecha, nombre_aeropuerto, tipo_aeropuerto, numero_aerolineas, destinos_nacionales, destinos_internacionales, destinos_total, Operaciones_comercial, Operaciones_general, pasajeros_comercial, pasajeros_general, year_month, year, month, nombre_aeropuerto_encoded, destinos_totales, Tipo]
Index: []

Operaciones Comerciales

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

# Configuración de Seaborn
sns.set(style="white")
```



```

# Crear el FacetGrid
g = sns.FacetGrid(combined_df, col='nombre_aeropuerto', col_wrap=4, height=3,
    ↪aspect=1.5)

# Definir paleta de colores
palette = {'Histórico': '#4287f5', 'Predicción': '#f54242'}

# Mapear gráficos para cada subgráfico del grid
for ax, (name, group) in zip(g.axes.flat, combined_df.
    ↪groupby('nombre_aeropuerto')):
    sns.lineplot(data=group, x='year_month', y='Operaciones_comercial',
    ↪hue='Tipo', palette=palette, ax=ax, legend=False)
    sns.scatterplot(data=group[group['Tipo'] == 'Predicción'], x='year_month',
    ↪y='Operaciones_comercial', hue='Tipo', palette=palette, ax=ax, s=50,
    ↪legend=False)

# Ajustar detalles de gráficos
g.set_titles('{col_name}')
g.set_axis_labels('Fecha', 'Operaciones Comerciales')
for ax in g.axes.flatten():
    ax.tick_params(labelrotation=45)

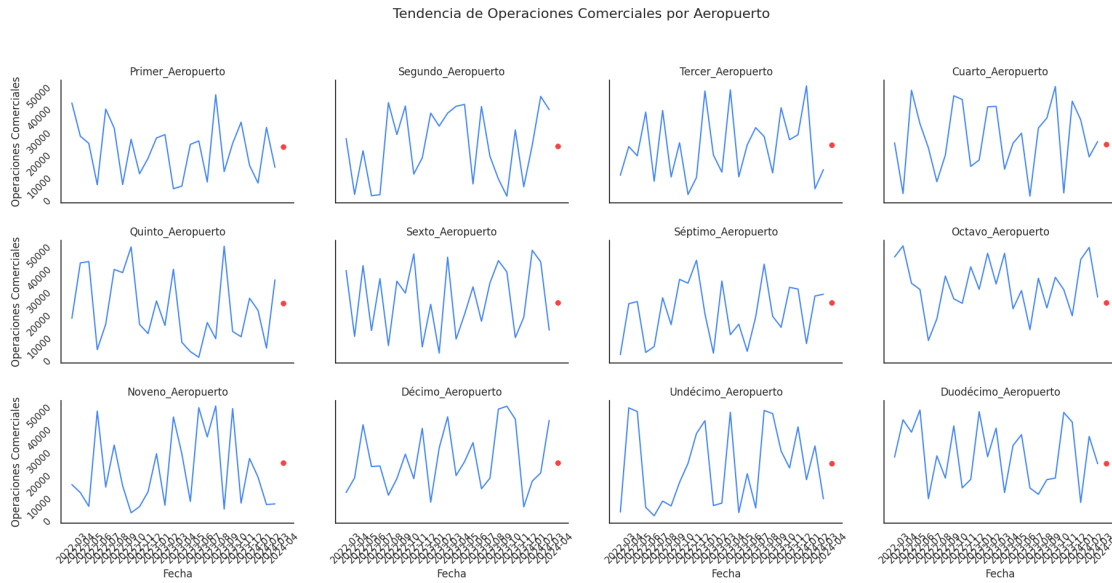
# Ajustar espacio y título general
plt.subplots_adjust(top=0.85)
g.fig.suptitle('Tendencia de Operaciones Comerciales por Aeropuerto',
    ↪fontsize=16)

# Configurar leyenda global
# Obtiene handles y labels del último subgráfico, que seguramente contiene
    ↪ambas categorías
handles, labels = g.axes[-1].get_legend_handles_labels()
# Crear la leyenda con los handles y labels recogidos
if handles:
    g.fig.legend(handles=handles, labels=labels, loc='upper center', ncol=3,
    ↪title='Tipo', bbox_to_anchor=(0.5, 0.95))
else:
    print("Histórico: Líneas")
    print("Predicción: Puntos")

# Mostrar el gráfico
plt.show()

```

Histórico: Líneas
 Predicción: Puntos



1.6 Exportar CSV con Histórico y Proyecciones

```
[35]: # Exportar el DataFrame actualizado a CSV
combined_df.to_csv('datos_aeropuertos_predicciones.csv', index=False)
```

1.7 Fin