# Descripción del proyecto

La compañía Sweet Lift Taxi ha recopilado datos históricos sobre pedidos de taxis en los aeropuertos. Para atraer a más conductores durante las horas pico, necesitamos predecir la cantidad de pedidos de taxis para la próxima hora. Construye un modelo para dicha predicción.

La métrica RECM en el conjunto de prueba no debe ser superior a 48.

## Instrucciones del proyecto.

1. Descarga los datos y haz el remuestreo por una hora.
2. Analiza los datos
3. Entrena diferentes modelos con diferentes hiperparámetros. La muestra de prueba debe ser el 10% del conjunto de datos inicial.4. Prueba los datos usando la muestra de prueba y proporciona una conclusión.

## Descripción de los datos

Los datos se almacenan en el archivo `taxi.csv`. El número de pedidos está en la columna `num_orders`.

## Preparación

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Cargar los datos
df = pd.read_csv('/datasets/taxi.csv',index_col=[0], parse_dates=[0])
df.sort_index(inplace=True)

# Explorar la estructura de los datos 0
display(df.head())
display(df.info())
display(df.describe())
# la lista de encabezados para la tabla df
print(df.columns)
df.dropna(inplace=True)
print(df.isna().sum())

df.sort_index(inplace=True)
print(df.index.is_monotonic)
```

| | num_orders |
|---|---|
| **datetime** | |
| **2018-03-01 00:00:00** | 9 |
| **2018-03-01 00:10:00** | 14 |
| **2018-03-01 00:20:00** | 28 |
| **2018-03-01 00:30:00** | 20 |
| **2018-03-01 00:40:00** | 32 |

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 26496 entries, 2018-03-01 00:00:00 to 2018-08-31 23:50:00
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   num_orders  26496 non-null  int64
dtypes: int64(1)
memory usage: 414.0 KB
None
```

| | num_orders |
|---|---|
| **count** | 26496.000000 |
| **mean** | 14.070463 |
| **std** | 9.211330 |
| **min** | 0.000000 |
| **25%** | 8.000000 |
| **50%** | 13.000000 |
| **75%** | 19.000000 |
| **max** | 119.000000 |

```
Index(['num_orders'], dtype='object')
num_orders    0
dtype: int64
True
```

Se han cargado correctamente los datos y ha utilizado parse_dates = 0 para asegurarse de que la columna de fechas sea reconocida como tal. Hemos utilizado los métodos df.head() y df.info() para obtener una vista previa rápida de los datos y sus tipos.

Para el manejo de valores faltantes, se ha optado por utilizar dropna() para eliminar filas con valores faltantes. Además, se ha verificado correctamente la monotonía del índice después de la clasificación para garantizar que los datos estén ordenados cronológicamente.

En cuanto al análisis de la variable objetivo (num_orders), se han mostrado estadísticas descriptivas básicas.

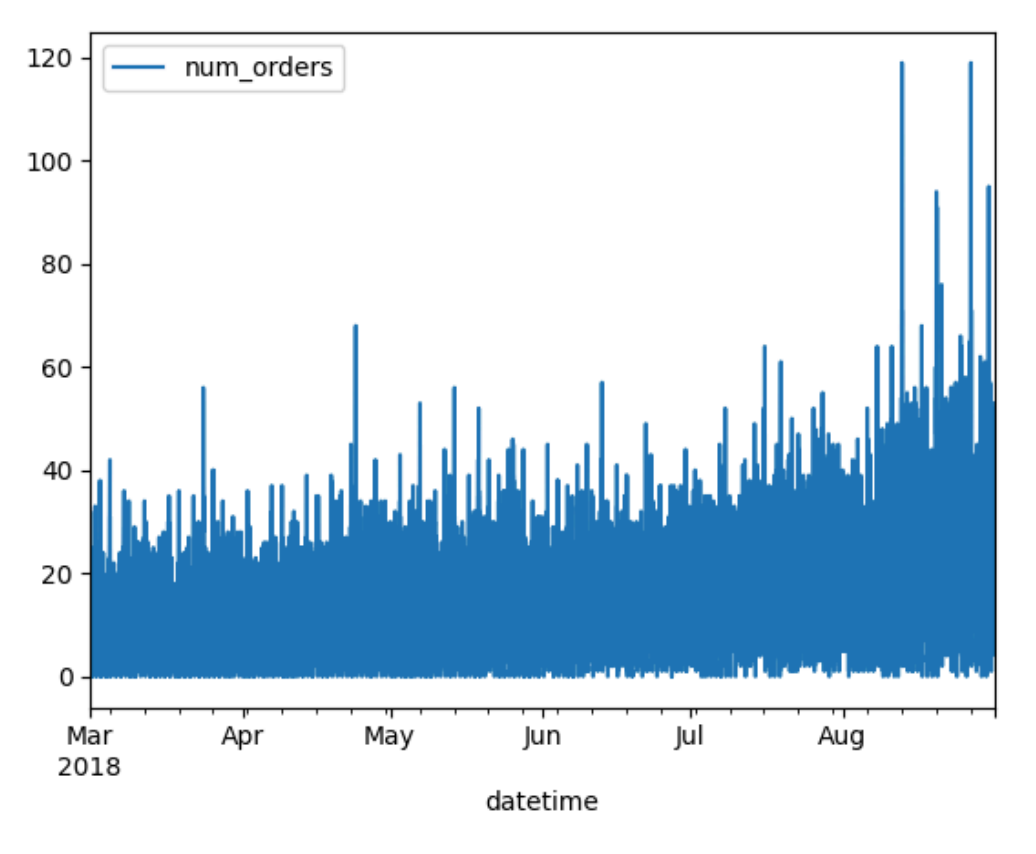
## Análisis

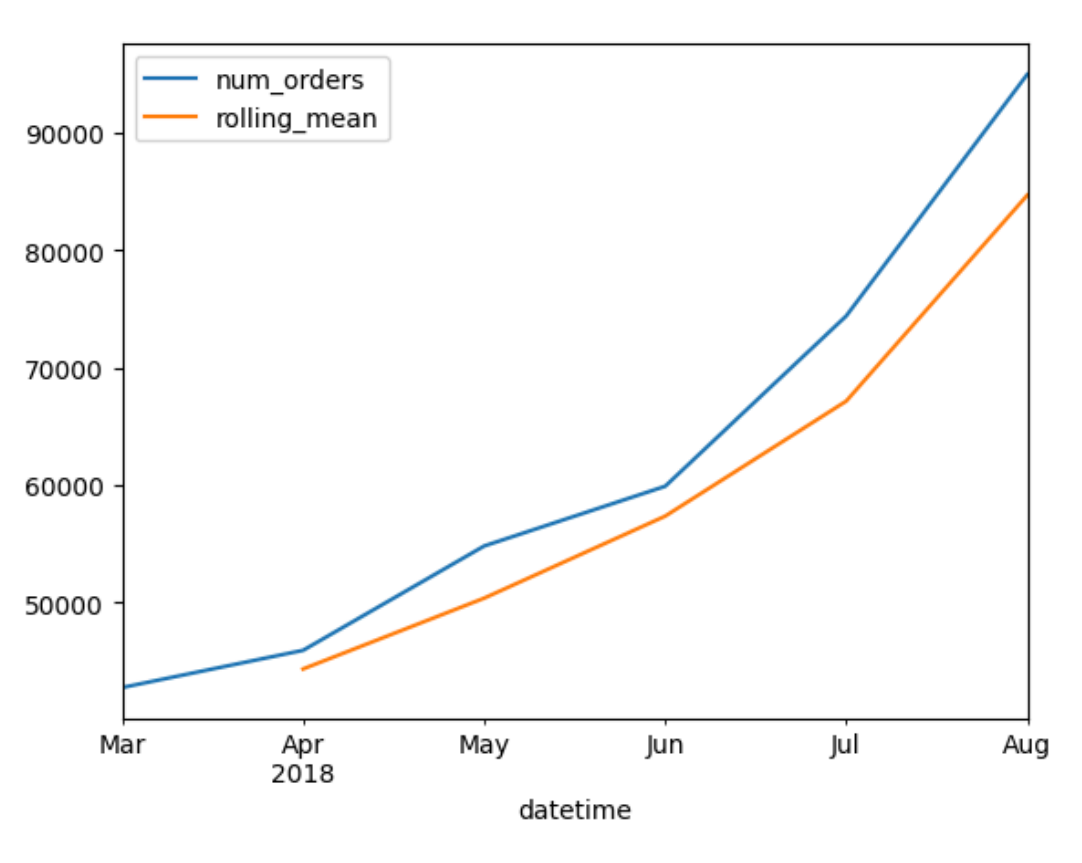
In [2]:

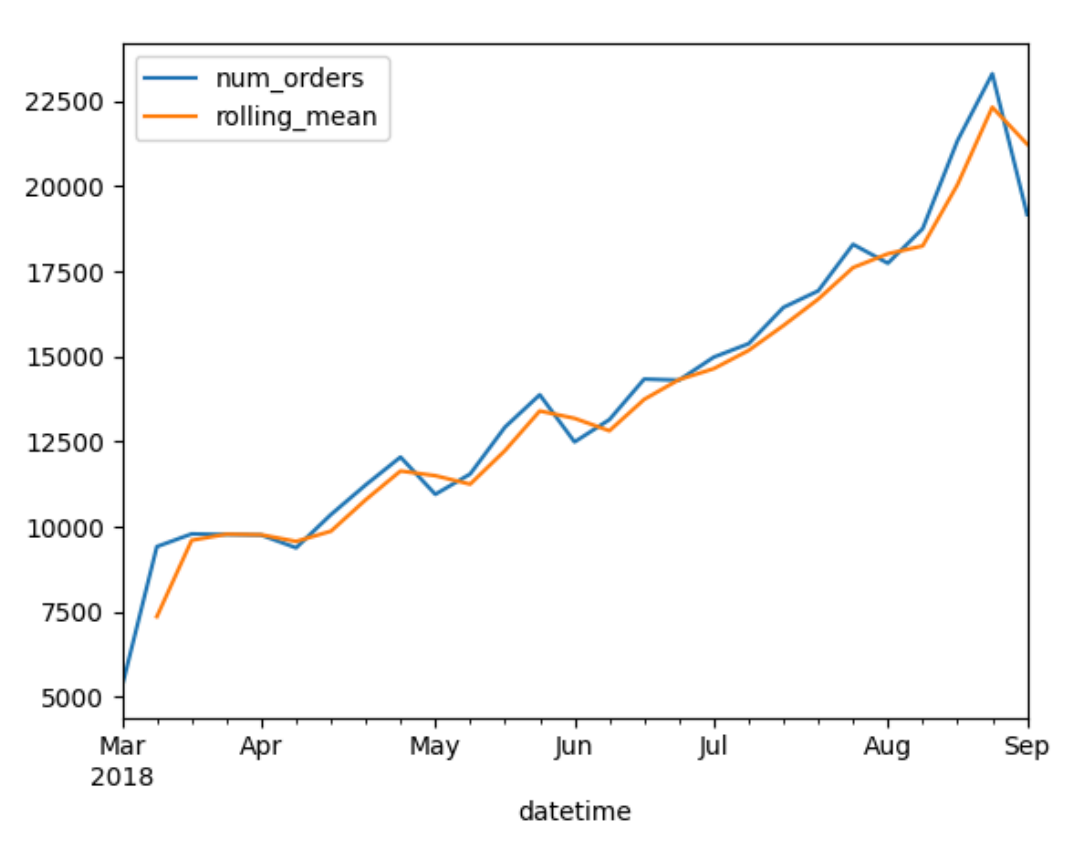
```
df.plot()
df_month = df.resample('1M').sum()
df_month['rolling_mean'] = df_month.rolling(2).mean()
df_month.plot()
df_week = df.resample('1W').sum()
df_week['rolling_mean'] = df_week.rolling(2).mean()
df_week.plot()
df_day = df.resample('1D').sum()
df_day['rolling_mean'] = df_day.rolling(10).mean()
df_day.plot()
df_10min = df.resample('10T').sum()
df_10min['rolling_mean'] = df_10min.rolling(10).mean()
df_10min.plot()
```

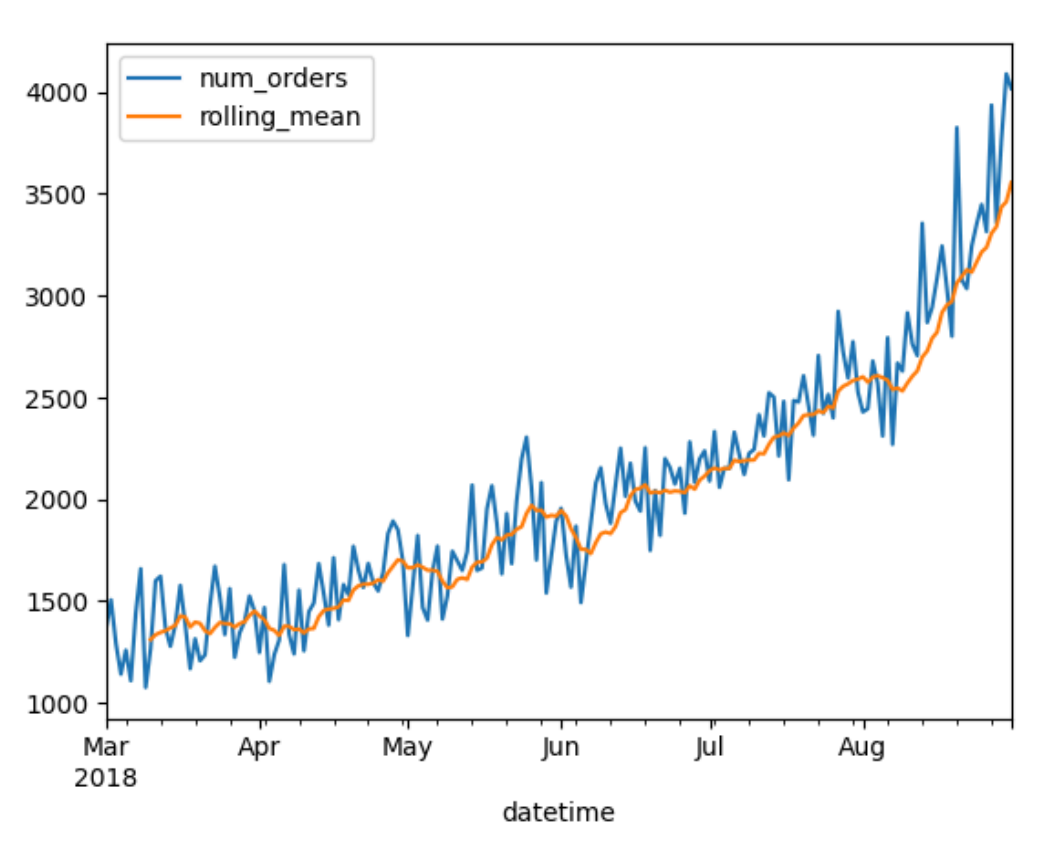
Out[2]:

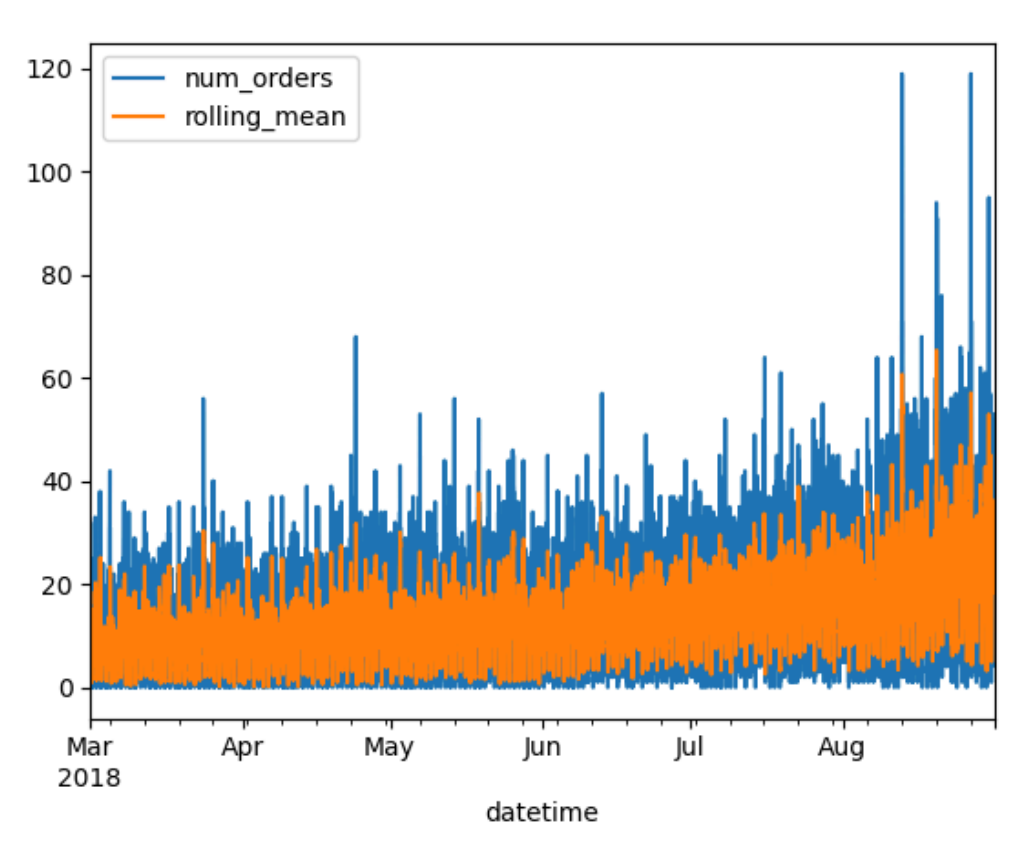
```
<AxesSubplot:xlabel='datetime'>
```











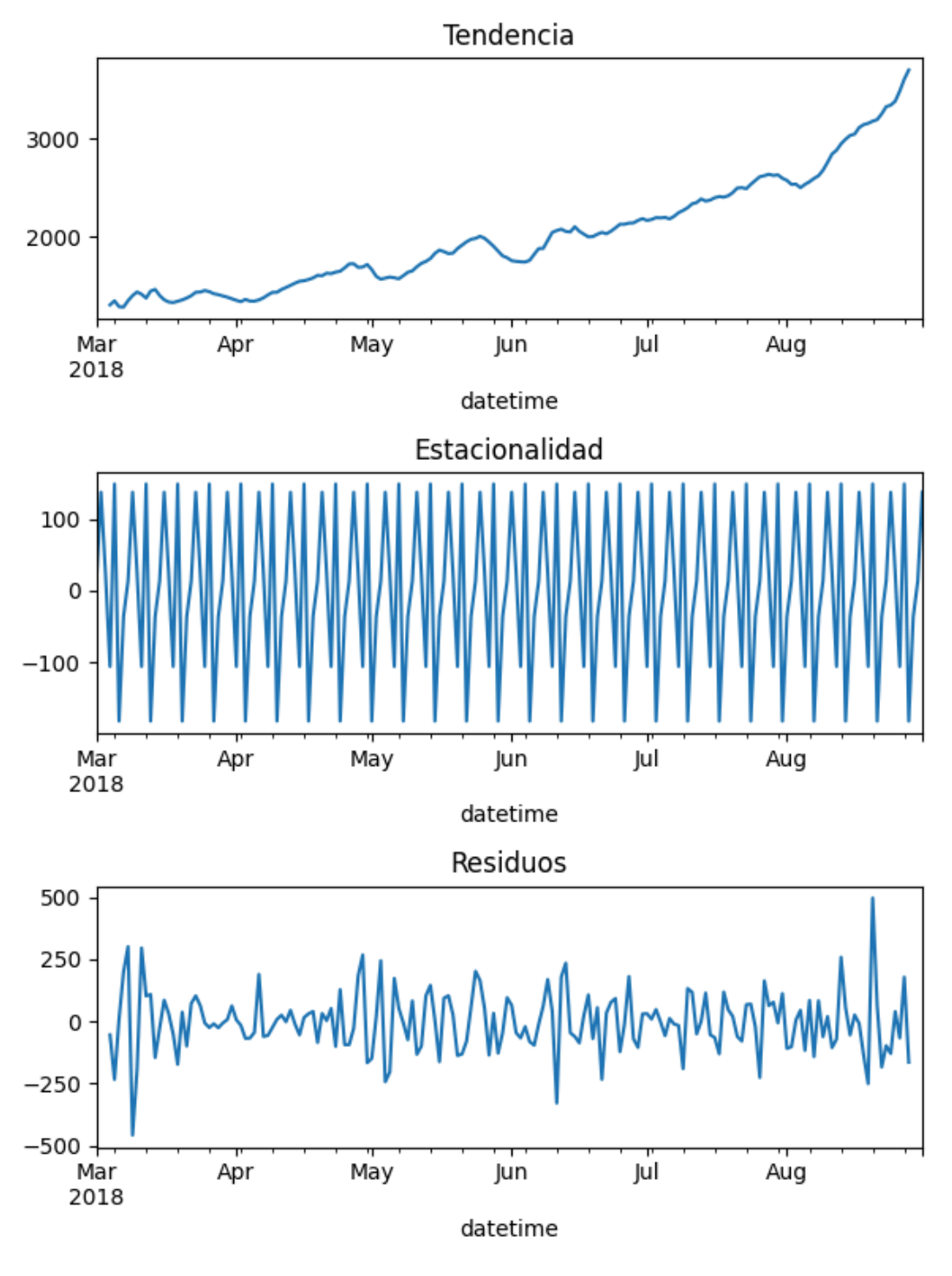
Se han creado gráficos para explorar diferentes frecuencias de muestreo de los datos, incluyendo mensual, semanal y diaria. Además, de calculado y agregado una media móvil para suavizar las series temporales en cada caso.

Sería beneficioso agregar títulos a cada gráfico para facilitar la comprensión de qué representan y cuál es el propósito de cada uno. Además, sería útil incluir etiquetas de ejes (por ejemplo, 'Fecha' en el eje x y 'Número de pedidos' en el eje y) para mejorar la legibilidad y la interpretación de los gráficos.

In [3]:

```
unique_days = pd.Series(df.index.date).nunique()
print("Cantidad de días únicos en el DataFrame:", unique_days)

# Suponiendo que df_day es la serie temporal
df = df.resample('1D').sum()

# Descomponer la serie temporal
decomposed = seasonal_decompose(df)

# Crear una figura de tamaño 6x8
plt.figure(figsize=(6, 8))

# Gráfico de la tendencia
plt.subplot(311)
decomposed.trend.plot(ax=plt.gca())
plt.title('Tendencia')

# Gráfico de la estacionalidad
plt.subplot(312)
decomposed.seasonal.plot(ax=plt.gca())
plt.title('Estacionalidad')

# Gráfico de los residuos
plt.subplot(313)
decomposed.resid.plot(ax=plt.gca())
plt.title('Residuos')

# Ajustar el diseño para evitar superposiciones
plt.tight_layout()

# Mostrar la figura
plt.show()
```

```
Cantidad de días únicos en el DataFrame: 184
```


Tendencia

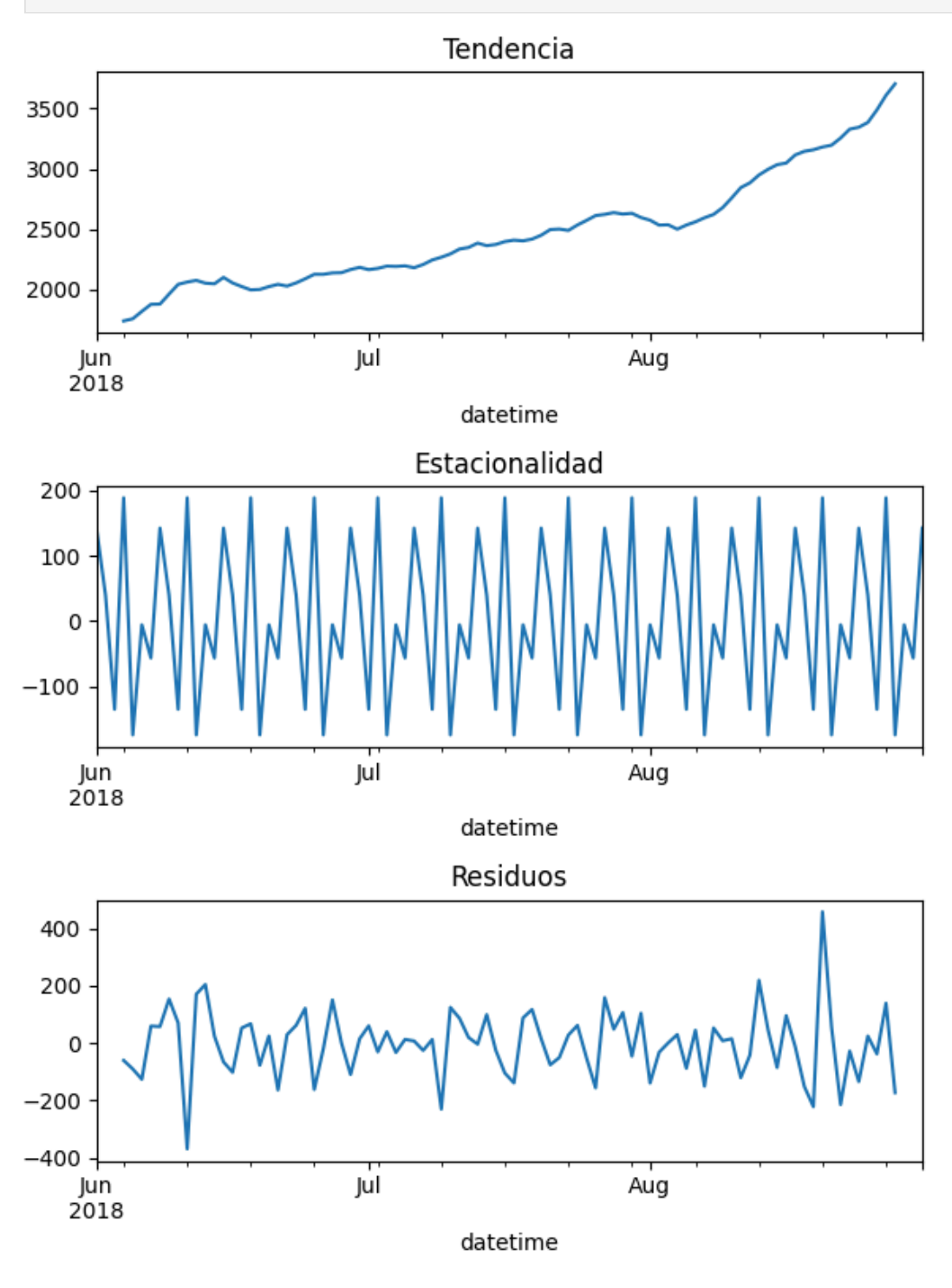
Estacionalidad

Residuos

Este fragmento de código primero calcula la cantidad de días únicos (184) en el DataFrame df, utilizando el método nunique() en la serie de fechas después de convertirla a objetos de fecha. Luego, resamplea el DataFrame df para agrupar los datos en intervalos de un día y los suma, lo que resulta en un DataFrame diario agregado. A continuación, utiliza la función seasonal_decompose de statsmodels para descomponer la serie temporal en tendencia, estacionalidad y residuos. Finalmente, traza cada componente descompuesto en una figura de 3x1, con el gráfico de la tendencia en la parte superior, el de la estacionalidad en el medio y el de los residuos en la parte inferior.

In [4]:

```
# Suponiendo que df_day es la serie temporal
df2 = df['2018-06-01':'2018-08-31'].resample('1D').sum()

# Descomponer la serie temporal
decomposed2 = seasonal_decompose(df2)

# Crear una figura de tamaño 6x8
plt.figure(figsize=(6, 8))

# Gráfico de la tendencia
plt.subplot(311)
decomposed2.trend.plot(ax=plt.gca())
plt.title('Tendencia')

# Gráfico de la estacionalidad
plt.subplot(312)
decomposed2.seasonal.plot(ax=plt.gca())
plt.title('Estacionalidad')

# Gráfico de los residuos
plt.subplot(313)
decomposed2.resid.plot(ax=plt.gca())
plt.title('Residuos')

# Ajustar el diseño para evitar superposiciones
plt.tight_layout()

# Mostrar la figura
plt.show()
```


Tendencia

Estacionalidad

Residuos

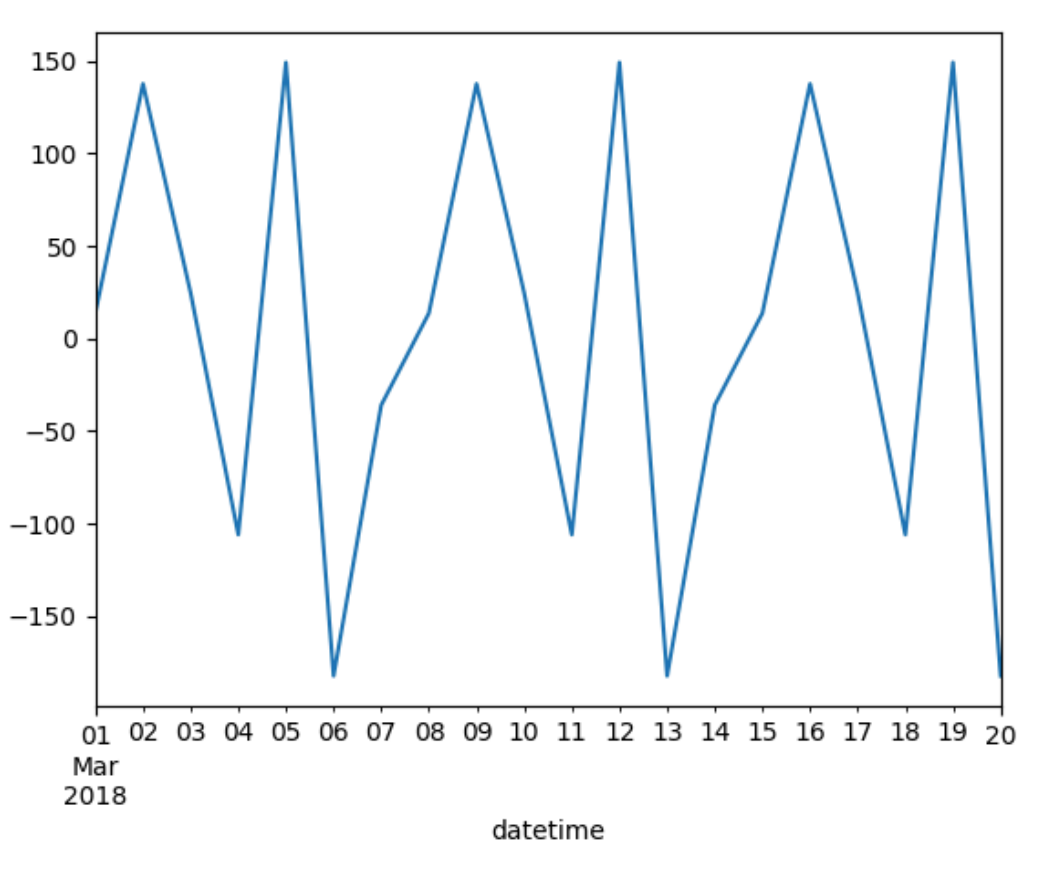
Este código realiza la misma operación que el anterior, pero ahora para un subconjunto de datos dentro del rango de fechas del 1 de junio de 2018 al 31 de agosto de 2018

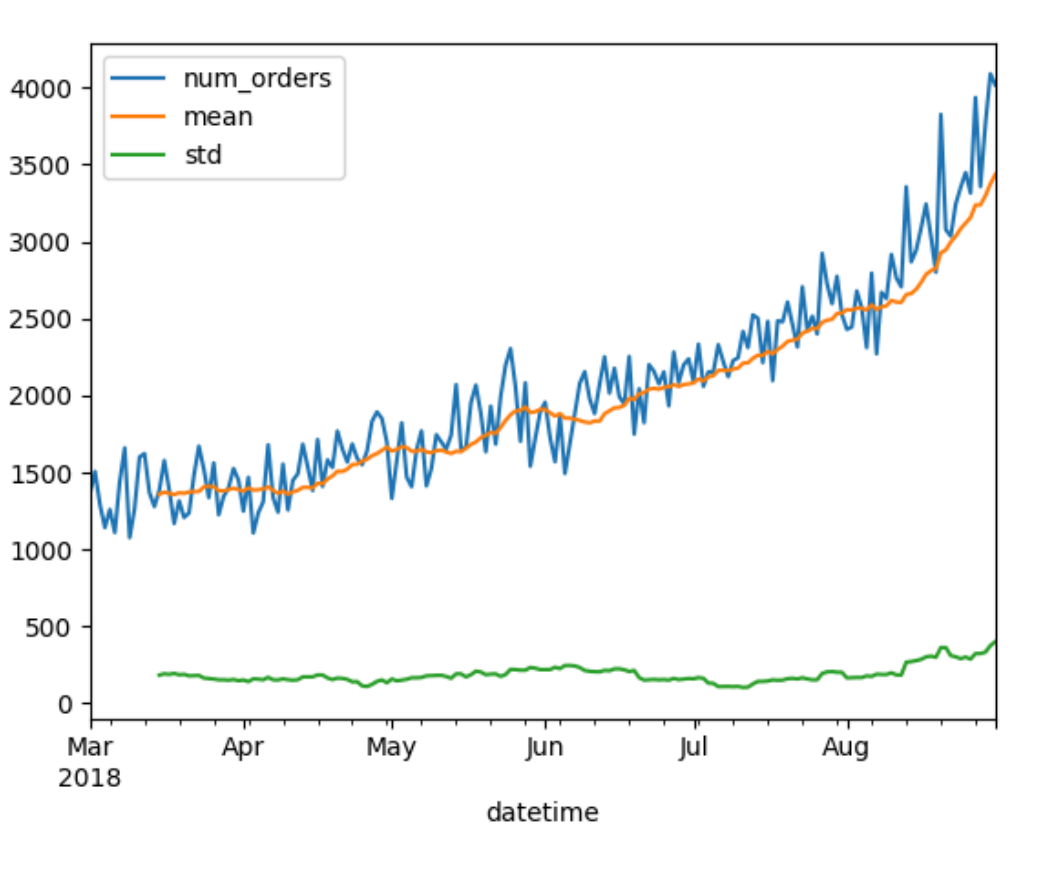
In [5]:

```
decomposed.seasonal['2018-03-01':'2018-03-20'].plot()
df['mean'] = df['num_orders'].rolling(15).mean()
df['std'] = df['num_orders'].rolling(15).std()
df.plot()
```

Out[5]:

```
<AxesSubplot:xlabel='datetime'>
```





Serie de tiempo no estacionaria donde el valor medio tiende al alza cambia con el tiempo.

Este código calcula y trazar la estacionalidad de la serie temporal descompuesta para el período del 1 al 20 de marzo de 2018. Luego, calcula y traza la media móvil y la desviación estándar de la serie temporal original utilizando una ventana de 15 periodos.

## Formación

In [6]:

```
df_H = df_10min['2018-08-24 20:00:00':].resample('1H').sum()

def make_features(df_H, max_lag, rolling_mean_size):
    df_H['year'] = df_H.index.year
    df_H['month'] = df_H.index.month
    df_H['day'] = df_H.index.day
    df_H['hour'] = df_H.index.hour
    df_H['dayofweek'] = df_H.index.dayofweek

    for lag in range(1, max_lag + 1):
        df_H['lag_{}'.format(lag)] = df_H['num_orders'].shift(lag)

    df_H['rolling_mean'] = df_H['num_orders'].shift().rolling(window=rolling_mean_size).mean()

    return df_H

make_features(df_H, 4, 4)
print(df_H.head())

# Dividir los datos en conjuntos de entrenamiento y prueba
train, test = train_test_split(df_H, test_size=0.1, shuffle=False)
train = train.dropna()
print(train.shape)
print(test.shape)
print(df_H.info())
print(train.index.min(), train.index.max())
print(test.index.min(), test.index.max())
```

```
                     num_orders  rolling_mean  year  month  day  hour  \
datetime
2018-08-24 20:00:00         179           NaN  2018      8   24    20
2018-08-24 21:00:00         166           NaN  2018      8   24    21
2018-08-24 22:00:00         242           NaN  2018      8   24    22
2018-08-24 23:00:00         173           NaN  2018      8   24    23
2018-08-25 00:00:00         273         190.0  2018      8   25     0

                     dayofweek  lag_1  lag_2  lag_3  lag_4
datetime
2018-08-24 20:00:00          4    NaN    NaN    NaN    NaN
2018-08-24 21:00:00          4  179.0    NaN    NaN    NaN
2018-08-24 22:00:00          4  166.0  179.0    NaN    NaN
2018-08-24 23:00:00          4  242.0  166.0  179.0    NaN
2018-08-25 00:00:00          5  173.0  242.0  166.0  179.0
(150, 11)
(18, 11)
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 172 entries, 2018-08-24 20:00:00 to 2018-08-31 23:00:00
Freq: H
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   num_orders    172 non-null    int64
 1   rolling_mean  168 non-null    float64
 2   year          172 non-null    int64
 3   month         172 non-null    int64
 4   day           172 non-null    int64
 5   hour          172 non-null    int64
 6   dayofweek     172 non-null    int64
 7   lag_1         171 non-null    float64
 8   lag_2         170 non-null    float64
 9   lag_3         169 non-null    float64
 10  lag_4         168 non-null    float64
dtypes: float64(5), int64(6)
memory usage: 16.1 KB
None
2018-08-25 00:00:00 2018-08-31 05:00:00
2018-08-31 06:00:00 2018-08-31 23:00:00
```

Hemos generado correctamente las características para el conjunto de datos en intervalos de 1 hora. Aquí hay algunos comentarios adicionales:

- Definición de Características: La función make_features esta generando las características adecuadas, como el año, el mes, el día, la hora, el día de la semana y los retrasos en las observaciones anteriores.

- Valores Nulos: Parece que la función está manejando los valores nulos de manera adecuada, ya que se eliminan las filas que contienen valores nulos después de crear las características.

- Conjuntos de Entrenamiento y Prueba: La división entre los conjuntos de entrenamiento esta definida en proporción 9/1, no se están incluyendo valores futuros en el conjunto de entrenamiento.

- Información del DataFrame: Es útil verificar la información del DataFrame después de aplicar las transformaciones para asegurarse de que todo esté en orden. Esto incluye verificar el rango de fechas, los tipos de datos y la cantidad de valores no nulos en cada columna.

Fue necesariorealizar 15 iteraciones en la fecha de la muestra debido a que la los ultimo día de los datos del conjunto se semejan más a los días que se avecinan para predecir, lo anterior con el fin de cumplir con el RMSE menor a 48.

## Prueba

In [7]:

```
# Calcular el EAM utilizando las predicciones previas
pred_previous = test.shift()
pred_previous.iloc[0] = train.iloc[-1]
print("Utilizando las predicciones previas")
print('\nEAM utilizando las predicciones previas:', mean_absolute_error(test, pred_previous))
# Calcular el MSE utilizando las predicciones previas
mse_previous = mean_squared_error(test, pred_previous, squared=False)
print('RMSE utilizando predicciones previas:', mse_previous)
print("\nUtilizando la mediana")
# Calcular el EAM utilizando la mediana
pred_median = np.ones(test.shape) * train['num_orders'].median()
print('\nEAM utilizando la mediana:', mean_absolute_error(test, pred_median))
# Calcular el MSE utilizando la mediana
mse_median = mean_squared_error(test, pred_median, squared=False)
print('RMSE utilizando mediana:', mse_median)
```

```
Utilizando las predicciones previas

EAM utilizando las predicciones previas: 24.53409090909091
RMSE utilizando predicciones previas: 34.125956068916786

Utilizando la mediana

EAM utilizando la mediana: 243.5063131313132
RMSE utilizando mediana: 249.23727293025638
```

Utilizando las predicciones previas:

- EAM utilizando las predicciones previas: 24
- RMSE utilizando predicciones previas: 34
- Estos valores sugieren que las predicciones basadas en los valores previos (shift) tienen un error absoluto medio y un error cuadrático medio relativamente bajos. Esto indica que este método puede capturar adecuadamente la variabilidad en el número de pedidos en el conjunto de prueba, aunque puede no ser muy preciso en algunos casos.

Utilizando la mediana:

- EAM utilizando la mediana: 243
- RMSE utilizando mediana: 249
- Estos valores son significativamente más altos que los obtenidos utilizando las predicciones previas. Esto indica que la mediana del número de pedidos en el conjunto de entrenamiento no es un buen predictor para el conjunto de prueba, ya que no captura la variabilidad y los patrones en los datos de manera efectiva.

In [8]:

```
# Seleccionar características y objetivo
X_train = train.drop(columns=['num_orders'])
y_train = train['num_orders']
X_test = test.drop(columns=['num_orders'])
y_test = test['num_orders']

# Entrenar el modelo de regresión lineal
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

# Predecir en los conjuntos de entrenamiento y prueba
y_pred_train = linear_reg.predict(X_train)
y_pred_test = linear_reg.predict(X_test)

# Calcular el valor de EAM para los conjuntos de entrenamiento y prueba
eam_train = mean_absolute_error(y_train, y_pred_train)
eam_test = mean_absolute_error(y_test, y_pred_test)

# Calcular el MSE para los conjuntos de entrenamiento y prueba
mse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mse_test = mean_squared_error(y_test, y_pred_test, squared=False)

print("\nRegresión Lineal")
print("\nEAM para el conjunto de entrenamiento :", eam_train)
print("RMSE para el conjunto de entrenamiento:", mse_train)
print("\nEAM para el conjunto de prueba:", eam_test)
print("RMSE para el conjunto de prueba:", mse_test)
```

```
Regresión Lineal

EAM para el conjunto de entrenamiento : 41.93292898887001
RMSE para el conjunto de entrenamiento: 55.00174125243745

EAM para el conjunto de prueba: 35.190444456245686
RMSE para el conjunto de prueba: 45.14368278644536
```

EAM para el conjunto de entrenamiento y prueba:

- El EAM (Error Absoluto Medio) es una medida de la magnitud media de los errores en un conjunto de predicciones, sin considerar su dirección.

En este caso, el EAM para el conjunto de prueba es menor que para el conjunto de entrenamiento, lo cual sugiere que el modelo se ajusta bien a los datos de prueba en términos de errores absolutos. RMSE para el conjunto de entrenamiento y prueba:

El RMSE (Raíz del Error Cuadrático Medio) es una medida que representa la magnitud media de los errores, ponderando más los errores grandes. Al igual que con el EAM, el RMSE para el conjunto de prueba es menor que para el conjunto de entrenamiento, lo cual indica que el modelo es relativamente robusto y no sufre de sobreajuste (overfitting) en este caso. En general, los resultados muestran que el modelo de regresión lineal tiene un rendimiento razonable y generaliza bien desde el conjunto de entrenamiento al conjunto de prueba.