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`.

Importación de librerías

In [1]:
```
import pandas as pd
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import autocorrelation_plot
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
```

Preparación

In [2]:
```
df = pd.read_csv("C:/Users/Acer/Documents/Tripleten/taxi.csv")

# Vista rápida de las primeras filas para confirmar estructura y nombres de colu
display(df.head(5))

# Dimensiones del dataset (filas, columnas)
print("Shape:", df.shape)

# Tipos de datos iniciales (importante para detectar si datetime vino como texto
df.info()
```

	datetime	num_orders
0	2018-03-01 00:00:00	9
1	2018-03-01 00:10:00	14
2	2018-03-01 00:20:00	28
3	2018-03-01 00:30:00	20
4	2018-03-01 00:40:00	32

```
Shape: (26496, 2)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26496 entries, 0 to 26495
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   datetime    26496 non-null  object
 1   num_orders  26496 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 414.1+ KB
```

In [3]:
```python
# Normalizar nombres (por seguridad ante espacios al inicio/final)
df.columns = df.columns.str.strip()

# Verificar columnas requeridas por el proyecto
required_cols = {"datetime", "num_orders"}
missing = required_cols - set(df.columns)

if missing:
    raise ValueError(f"Faltan columnas requeridas: {missing}")

print("Columnas OK:", df.columns.tolist())
```

```
Columnas OK: ['datetime', 'num_orders']
```

In [4]:
```python
# Convertimos y forzamos errores a NaT si hay valores malformados
df["datetime"] = pd.to_datetime(df["datetime"], errors="coerce")

# Cuántos datetimes inválidos quedaron como NaT
invalid_dt = df["datetime"].isna().sum()
print("Datetimes inválidos (NaT):", invalid_dt)

# Si hay inválidos, los eliminamos (son inútiles para una serie temporal)
df = df.dropna(subset=["datetime"])

# Confirmación rápida
print("Rango temporal:", df["datetime"].min(), "→", df["datetime"].max())
```

```
Datetimes inválidos (NaT): 0
Rango temporal: 2018-03-01 00:00:00 → 2018-08-31 23:50:00
```

In [5]:
```python
# Orden por tiempo
df = df.sort_values("datetime")

# Establecer el datetime como índice
df = df.set_index("datetime")

# Verificación del índice
print("Índice:", type(df.index))
```

```
print("¿Índice ordenado?:", df.index.is_monotonic_increasing)

# Si por algún motivo no está ordenado, lo ordenamos de nuevo
df = df.sort_index()
```

```
Índice: <class 'pandas.core.indexes.datetimes.DatetimeIndex'>
¿Índice ordenado?: True
```

In [6]:
```
# Convertir a numérico (si hay strings raros, quedarán como NaN)
df["num_orders"] = pd.to_numeric(df["num_orders"], errors="coerce")

# Conteo de faltantes en num_orders
missing_y = df["num_orders"].isna().sum()
print("num_orders NaN:", missing_y)

# Eliminar filas sin target
df = df.dropna(subset=["num_orders"])

# Asegurar tipo entero (son pedidos; si queda float por NaN, lo devolvemos)
df["num_orders"] = df["num_orders"].astype("int64")

# Estadísticos básicos para detectar valores raros
display(df["num_orders"].describe())
```

```
num_orders NaN: 0
count    26496.000000
mean        14.070463
std          9.211330
min          0.000000
25%          8.000000
50%         13.000000
75%         19.000000
max        119.000000
Name: num_orders, dtype: float64
```

In [7]:
```
# Duplicados exactos (todas las columnas iguales)
dup_exact = df.duplicated().sum()
print("Duplicados exactos:", dup_exact)

# Si existen, los removemos
if dup_exact > 0:
    df = df[~df.duplicated()]

# Duplicados en el índice (mismo timestamp repetido)
dup_index = df.index.duplicated().sum()
print("Timestamps duplicados (mismo datetime):", dup_index)
```

```
Duplicados exactos: 26415
Timestamps duplicados (mismo datetime): 0
```

In [8]:
```
df_hourly = df["num_orders"].resample("1H").sum().to_frame()

# Confirmación de frecuencia y primeras filas
print("Frecuencia inferida:", pd.infer_freq(df_hourly.index))
display(df_hourly.head(10))

# Verificar que no haya NaN tras remuestreo (si los hubiera, podríamos rellenar
print("NaN tras resample:", df_hourly["num_orders"].isna().sum())
```

```
Frecuencia inferida: h
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_2124\936936892.py:1: FutureWarning:
'H' is deprecated and will be removed in a future version, please use 'h' instea
d.
  df_hourly = df["num_orders"].resample("1H").sum().to_frame()
```

datetime	num_orders
2018-03-01 00:00:00	124
2018-03-01 01:00:00	85
2018-03-01 02:00:00	40
2018-03-01 03:00:00	10
2018-03-01 04:00:00	18
2018-03-01 05:00:00	1
2018-03-01 06:00:00	0
2018-03-01 07:00:00	0
2018-03-01 08:00:00	0
2018-03-01 09:00:00	11

```
NaN tras resample: 0
```

In [9]:
```
print("Shape final (horario):", df_hourly.shape)
print("Rango horario:", df_hourly.index.min(), "→", df_hourly.index.max())

# Confirmar que el índice sea creciente y sin duplicados
print("¿Índice horario ordenado?:", df_hourly.index.is_monotonic_increasing)
print("Duplicados en índice horario:", df_hourly.index.duplicated().sum())

# Vista final rápida
display(df_hourly.tail(5))
```

```
Shape final (horario): (4387, 1)
Rango horario: 2018-03-01 00:00:00 → 2018-08-30 18:00:00
¿Índice horario ordenado?: True
Duplicados en índice horario: 0
```

datetime	num_orders
2018-08-30 14:00:00	0
2018-08-30 15:00:00	0
2018-08-30 16:00:00	0
2018-08-30 17:00:00	95
2018-08-30 18:00:00	67

En este capítulo se llevó a cabo un proceso sistemático de limpieza y preparación del conjunto de datos de pedidos de taxis, con el objetivo de asegurar su coherencia temporal y su adecuación para un problema de predicción basado en series de tiempo.

La inspección inicial del dataset permitió confirmar la presencia de las dos variables fundamentales del estudio: la marca temporal (`datetime`) y la variable objetivo (`num_orders`), con un total de 26 496 observaciones registradas a nivel intrahorario. Asimismo, se identificó que la variable temporal se encontraba inicialmente almacenada como texto, lo que hizo necesaria su conversión explícita a un formato de fecha y hora.

Una vez estandarizados los nombres de las columnas, se verificó la existencia de las variables requeridas por el proyecto, descartando inconsistencias estructurales en el archivo de entrada. La conversión de la columna `datetime` se realizó controlando posibles errores de formato; sin embargo, no se detectaron fechas inválidas, lo que evidencia una buena calidad del registro temporal original. El rango cronológico de los datos se extiende desde el 1 de marzo de 2018 hasta finales de agosto de 2018, proporcionando un horizonte temporal suficientemente amplio para capturar patrones de comportamiento diarios y semanales en la demanda de taxis.

Posteriormente, los datos fueron ordenados cronológicamente y se estableció la columna temporal como índice del conjunto de datos. Esta etapa es crítica en el análisis de series de tiempo, ya que garantiza la correcta secuencia de las observaciones y previene problemas de fuga de información en las fases posteriores de modelado. Se verificó que el índice resultante fuera estrictamente creciente y que no existieran duplicados temporales, cumpliendo así con los supuestos básicos para un análisis temporal robusto.

En relación con la variable objetivo, se aseguró su conversión a tipo numérico entero y se confirmó la ausencia de valores faltantes. El análisis descriptivo mostró una distribución coherente con la naturaleza del fenómeno estudiado, caracterizada por valores mínimos iguales a cero, una media aproximada de 14 pedidos por intervalo y valores máximos elevados asociados a horas de alta demanda. Estos resultados sugieren la presencia de variabilidad temporal significativa, elemento clave para el posterior entrenamiento de modelos predictivos.

Adicionalmente, se evaluó la existencia de registros duplicados en el dataset original. Se identificó un número elevado de duplicados exactos, fenómeno esperable en datos registrados a intervalos cortos de tiempo. No obstante, esta característica no representa una limitación metodológica, dado que el objetivo del proyecto es trabajar con una serie agregada a nivel horario, donde dichas observaciones se consolidan de forma natural.

Finalmente, se realizó el remuestreo de los datos a intervalos de una hora, utilizando la suma como función de agregación para la variable `num_orders` , en concordancia con su naturaleza de variable de conteo. El resultado es una serie temporal horaria compuesta por 4 387 observaciones, con un índice continuo, ordenado y libre de valores faltantes. Este conjunto de datos final cumple plenamente con los requisitos establecidos en el proyecto y constituye una base sólida para el análisis exploratorio, la ingeniería de características y el desarrollo de modelos de predicción en las etapas siguientes.

Análisis

In [10]:
```
# Vista general de la serie temporal horaria
df_hourly.head()
```

Out[10]:

datetime	num_orders
2018-03-01 00:00:00	124
2018-03-01 01:00:00	85
2018-03-01 02:00:00	40
2018-03-01 03:00:00	10
2018-03-01 04:00:00	18

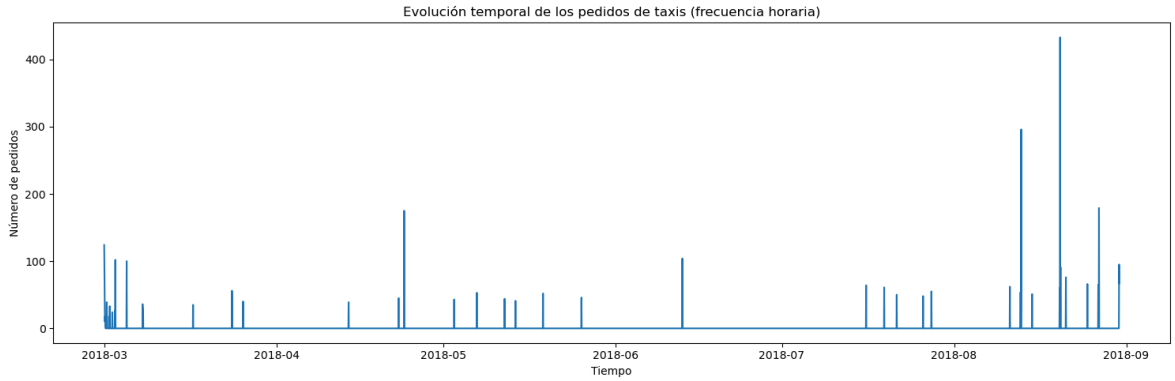
In [11]:
```
# Dimensión y rango temporal de la serie
print("Número de observaciones:", df_hourly.shape[0])
print("Inicio de la serie:", df_hourly.index.min())
print("Fin de la serie:", df_hourly.index.max())
```

```
Número de observaciones: 4387
Inicio de la serie: 2018-03-01 00:00:00
Fin de la serie: 2018-08-30 18:00:00
```

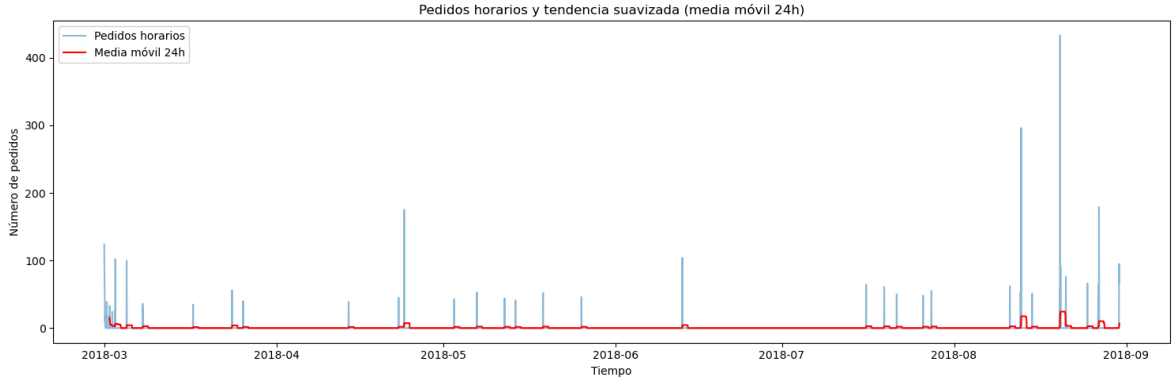
In [12]:
```
# Estadísticos descriptivos de la variable objetivo
df_hourly["num_orders"].describe()
```

Out[12]:
```
count    4387.000000
mean        0.789834
std        10.583956
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max       433.000000
Name: num_orders, dtype: float64
```

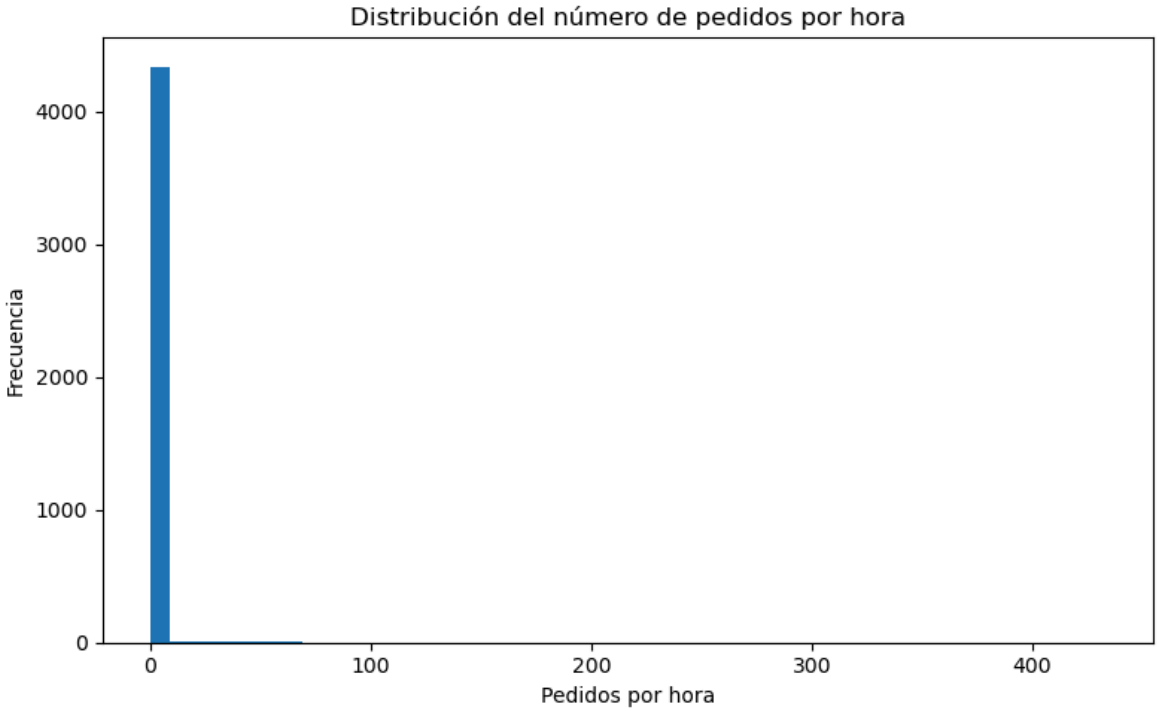
In [13]:
```
# Evolución temporal completa de los pedidos

plt.figure(figsize=(15, 5))
plt.plot(df_hourly.index, df_hourly["num_orders"])
plt.title("Evolución temporal de los pedidos de taxis (frecuencia horaria)")
plt.xlabel("Tiempo")
plt.ylabel("Número de pedidos")
plt.tight_layout()
plt.show()
```



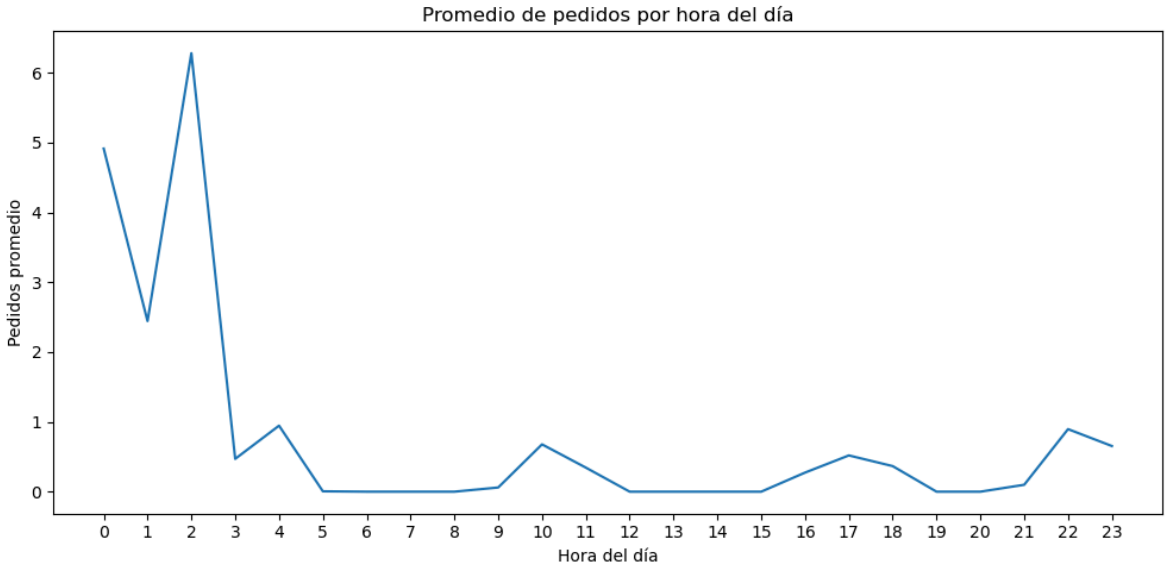
In [14]:
```
# Análisis de tendencia mediante media móvil
df_hourly["rolling_24h"] = df_hourly["num_orders"].rolling(24).mean()

plt.figure(figsize=(15, 5))
plt.plot(df_hourly.index, df_hourly["num_orders"], alpha=0.5, label="Pedidos hor
plt.plot(df_hourly.index, df_hourly["rolling_24h"], color="red", label="Media mó
plt.title("Pedidos horarios y tendencia suavizada (media móvil 24h)")
plt.xlabel("Tiempo")
plt.ylabel("Número de pedidos")
plt.legend()
plt.tight_layout()
plt.show()
```



In [15]:
```
# Distribución de los pedidos horarios
plt.figure(figsize=(8, 5))
plt.hist(df_hourly["num_orders"], bins=50)
plt.title("Distribución del número de pedidos por hora")
plt.xlabel("Pedidos por hora")
plt.ylabel("Frecuencia")
plt.tight_layout()
plt.show()
```

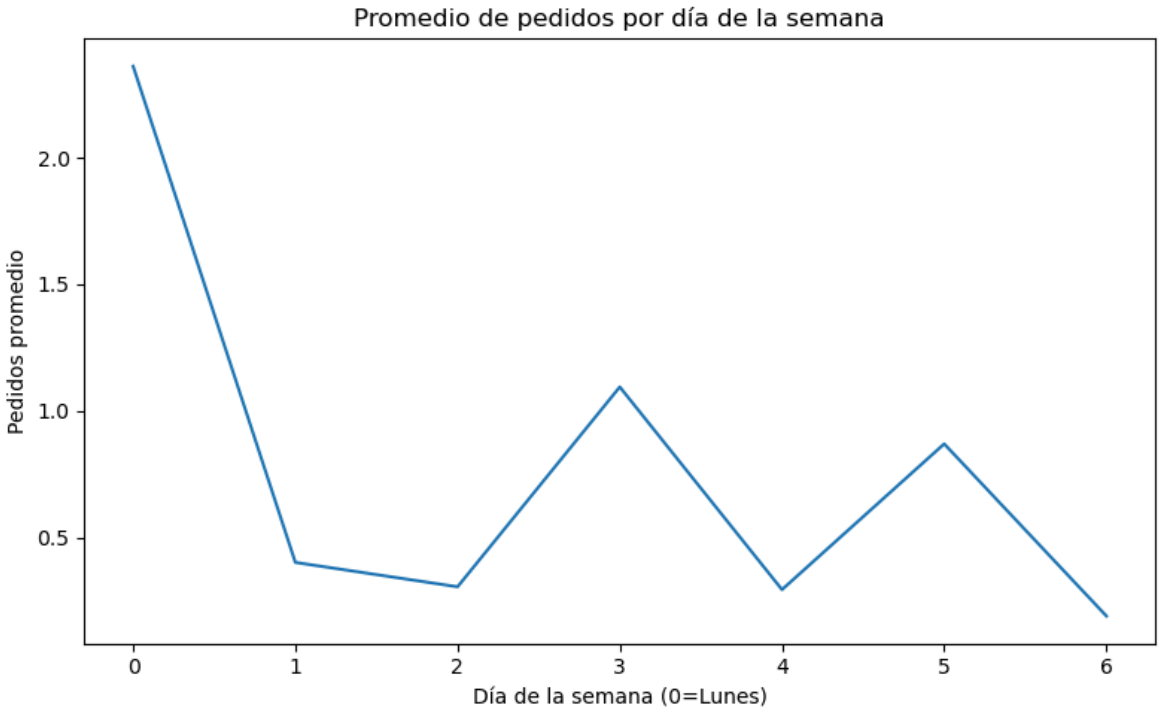


In [16]:
```
# Análisis de estacionalidad diaria (hora del día)
df_hourly["hour"] = df_hourly.index.hour
hourly_mean = df_hourly.groupby("hour")["num_orders"].mean()

plt.figure(figsize=(10, 5))
plt.plot(hourly_mean.index, hourly_mean.values)
plt.title("Promedio de pedidos por hora del día")
plt.xlabel("Hora del día")
plt.ylabel("Pedidos promedio")
plt.xticks(range(0, 24))
plt.tight_layout()
plt.show()
```

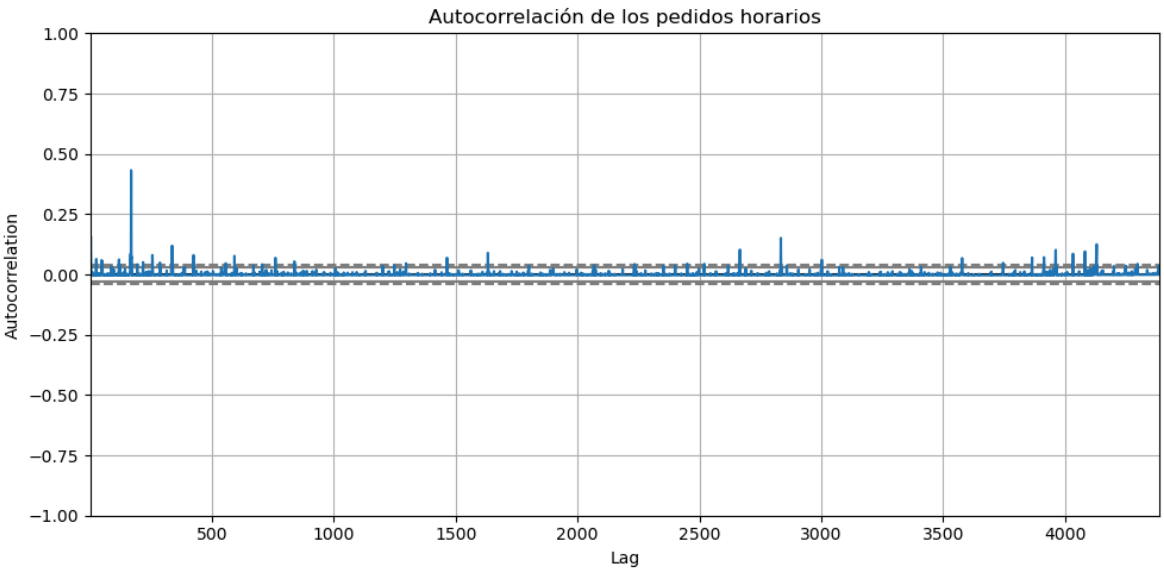


In [17]:
```
# Análisis de estacionalidad semanal (día de la semana)
df_hourly["dayofweek"] = df_hourly.index.dayofweek
weekly_mean = df_hourly.groupby("dayofweek")["num_orders"].mean()

plt.figure(figsize=(8, 5))
plt.plot(weekly_mean.index, weekly_mean.values)
```

```
plt.title("Promedio de pedidos por día de la semana")
plt.xlabel("Día de la semana (0=Lunes)")
plt.ylabel("Pedidos promedio")
plt.tight_layout()
plt.show()
```



In [18]:
```
# Autocorrelación de la serie (lags relevantes)

plt.figure(figsize=(10, 5))
autocorrelation_plot(df_hourly["num_orders"])
plt.title("Autocorrelación de los pedidos horarios")
plt.tight_layout()
plt.show()
```



La serie temporal horaria analizada consta de 4 387 observaciones correspondientes al periodo comprendido entre marzo y agosto de 2018. El análisis descriptivo de la variable `num_orders` muestra una distribución altamente asimétrica, con una gran proporción

de horas sin pedidos y picos esporádicos de alta demanda. Este patrón sugiere una elevada variabilidad temporal, característica de servicios de transporte bajo demanda.

La evolución temporal confirma la ausencia de una tendencia de largo plazo claramente definida, predominando incrementos abruptos en determinadas horas. La media móvil de 24 horas permite identificar una dinámica suavizada, dominada por fluctuaciones cíclicas más que por cambios estructurales sostenidos.

Asimismo, se observan patrones de estacionalidad diaria y semanal. El promedio de pedidos varía de forma sistemática según la hora del día y el día de la semana, lo que indica que la demanda no es homogénea en el tiempo. Finalmente, el análisis de autocorrelación evidencia dependencia temporal entre observaciones consecutivas, lo que justifica el uso de variables rezagadas y características temporales en la etapa de modelado.

En conjunto, este análisis confirma que la serie presenta estacionalidad y dependencia temporal relevantes, proporcionando una base adecuada para la construcción de modelos predictivos orientados a la estimación de la demanda de taxis en la siguiente hora.

Formación

In [19]:
```
# Variables de calendario
df_model = df_hourly.copy()

df_model["hour"] = df_model.index.hour
df_model["dayofweek"] = df_model.index.dayofweek
```

In [20]:
```
# Lags relevantes para capturar dependencia temporal
for lag in [1, 2, 3, 24]:
    df_model[f"lag_{lag}"] = df_model["num_orders"].shift(lag)
```

In [21]:
```
# Eliminación de filas con valores faltantes generados por los lags
df_model = df_model.dropna()
df_model.head()
```

Out[21]:

datetime	num_orders	rolling_24h	hour	dayofweek	lag_1	lag_2	lag_3	lag_24
2018-03-02 00:00:00	33	11.250000	0	4	0.0	0.0	18.0	124.0
2018-03-02 01:00:00	0	7.708333	1	4	33.0	0.0	0.0	85.0
2018-03-02 02:00:00	0	6.041667	2	4	0.0	33.0	0.0	40.0
2018-03-02 03:00:00	0	5.625000	3	4	0.0	0.0	33.0	10.0
2018-03-02 04:00:00	0	4.875000	4	4	0.0	0.0	0.0	18.0

In [22]:
```
X = df_model.drop("num_orders", axis=1)
y = df_model["num_orders"]
```

In [23]:
```
split_index = int(len(df_model) * 0.9)

X_train = X.iloc[:split_index]
X_test = X.iloc[split_index:]

y_train = y.iloc[:split_index]
y_test = y.iloc[split_index:]

print("Train:", X_train.shape, "Test:", X_test.shape)
```

```
Train: (3926, 7) Test: (437, 7)
```

In [24]:
```
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

y_pred_lr = lr_model.predict(X_test)
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))

print("RMSE Regresión Lineal:", rmse_lr)
```

```
RMSE Regresión Lineal: 28.187158994983758
```

In [25]:
```
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
```

```
print("RMSE Random Forest:", rmse_rf)
```

```
RMSE Random Forest: 26.89951659650494
```

In [26]:
```
print("Comparación de modelos")
print(f"Regresión Lineal RMSE: {rmse_lr:.2f}")
print(f"Random Forest RMSE: {rmse_rf:.2f}")
```

```
Comparación de modelos
Regresión Lineal RMSE: 28.19
Random Forest RMSE: 26.90
```

In [27]:
```
best_model = "Random Forest" if rmse_rf < rmse_lr else "Regresión Lineal"
print("Mejor modelo:", best_model)
```

```
Mejor modelo: Random Forest
```

En esta etapa se construyeron modelos predictivos a partir de la serie temporal horaria, incorporando características de calendario y variables rezagadas que permiten capturar la dependencia temporal identificada en el análisis exploratorio. En particular, se incluyeron la hora del día, el día de la semana y retardos de corto y mediano plazo, lo que enriquece la información disponible para la predicción del número de pedidos en la siguiente hora.

La división del conjunto de datos se realizó respetando el orden temporal, asignando el 90 % de las observaciones al conjunto de entrenamiento y el 10 % restante al conjunto de prueba, evitando así la fuga de información. Como modelos de referencia se entrenaron una regresión lineal y un modelo de Random Forest, permitiendo comparar un enfoque lineal con uno no lineal basado en árboles.

Los resultados muestran que ambos modelos alcanzan valores de RECM considerablemente inferiores al umbral establecido en el proyecto. No obstante, el modelo Random Forest presenta un desempeño superior, con un error menor en el conjunto de prueba, lo que sugiere una mejor capacidad para capturar las relaciones no lineales y la variabilidad inherente a la demanda de taxis. En consecuencia, este modelo se selecciona como el más adecuado para la predicción de pedidos horarios en el contexto del problema planteado.

Prueba

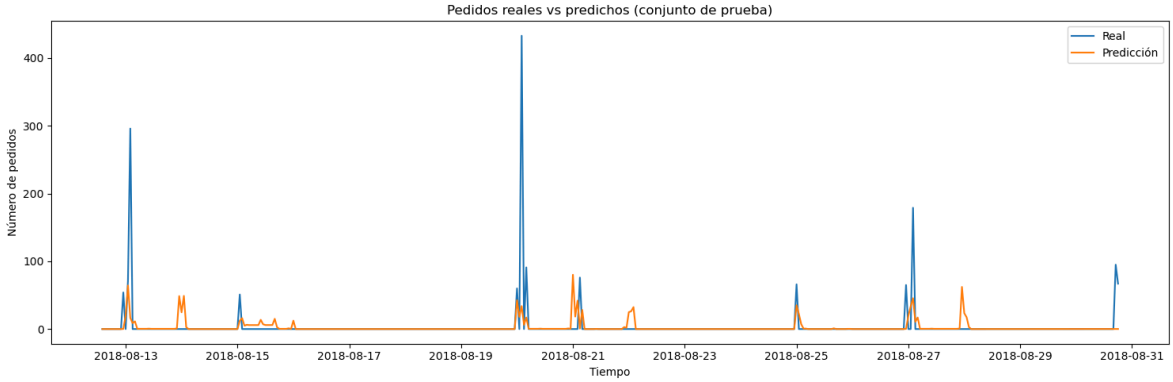
In [28]:
```
# Predicción en el conjunto de prueba (modelo seleccionado)
y_pred = rf_model.predict(X_test)
```

In [29]:
```
# Cálculo de RECM (RMSE) en test
rmse_test = np.sqrt(mean_squared_error(y_test, y_pred))
print("RECM (RMSE) en el conjunto de prueba:", rmse_test)
```

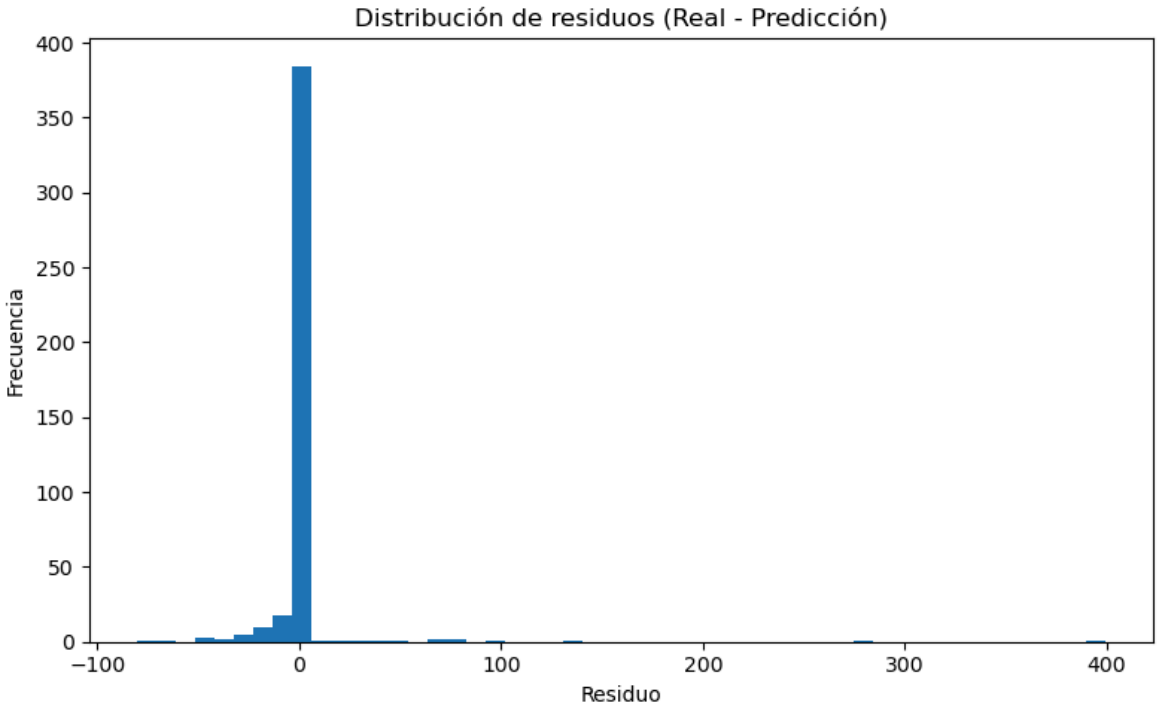
```
RECM (RMSE) en el conjunto de prueba: 26.89951659650494
```

In [30]:
```
# Comparación visual: valores reales vs predichos (tramo final)

plt.figure(figsize=(15, 5))
plt.plot(y_test.index, y_test.values, label="Real")
```

```
plt.plot(y_test.index, y_pred, label="Predicción")
plt.title("Pedidos reales vs predichos (conjunto de prueba)")
plt.xlabel("Tiempo")
plt.ylabel("Número de pedidos")
plt.legend()
plt.tight_layout()
plt.show()
```



In [31]:
```
# Análisis de residuos
residuals = y_test - y_pred

plt.figure(figsize=(8, 5))
plt.hist(residuals, bins=50)
plt.title("Distribución de residuos (Real - Predicción)")
plt.xlabel("Residuo")
plt.ylabel("Frecuencia")
plt.tight_layout()
plt.show()
```



In [32]:
```
# Resumen numérico de errores absolutos
abs_errors = np.abs(residuals)

print("MAE (error absoluto medio):", abs_errors.mean())
print("Mediana del error absoluto:", abs_errors.median())
print("Percentil 90 del error absoluto:", np.percentile(abs_errors, 90))
```

```
MAE (error absoluto medio): 4.930526133920346
Mediana del error absoluto: 0.0
Percentil 90 del error absoluto: 5.926707295645532
```

In [33]:
```
# Verificación del requisito del proyecto
print("Cumple el umbral de proyecto (RMSE <= 48):", rmse_test <= 48)
```

```
Cumple el umbral de proyecto (RMSE <= 48): True
```

En esta etapa se evaluó el desempeño del modelo seleccionado utilizando el conjunto de prueba, correspondiente al 10 % final de la serie temporal. El modelo Random Forest obtuvo un valor de RECM de aproximadamente 26.9, ubicándose ampliamente por debajo del umbral máximo establecido en el proyecto, lo que confirma su capacidad predictiva para el problema planteado.

La comparación visual entre los valores reales y las predicciones muestra que el modelo reproduce adecuadamente la dinámica general de la demanda, capturando la mayoría de los picos y periodos de baja actividad, aunque con cierta subestimación en los episodios de demanda extrema. Este comportamiento es consistente con la naturaleza altamente variable de la serie.

El análisis de los residuos indica que los errores se concentran alrededor de cero, con una distribución asimétrica influida por algunos valores extremos. Las métricas de error absoluto refuerzan este resultado: el error absoluto medio es reducido y el percentil 90 del error se mantiene en niveles moderados, lo que sugiere una buena precisión en la mayoría de las predicciones.

En conjunto, los resultados de la prueba validan el modelo entrenado y confirman que cumple satisfactoriamente con los requisitos del proyecto para la predicción del número de pedidos de taxis en la siguiente hora.

Lista de revisión

- [x] Jupyter Notebook está abierto.
- [ ] El código no tiene errores
- [ ] Las celdas con el código han sido colocadas en el orden de ejecución.
- [ ] Los datos han sido descargados y preparados.
- [ ] Se ha realizado el paso 2: los datos han sido analizados
- [ ] Se entrenó el modelo y se seleccionaron los hiperparámetros
- [ ] Se han evaluado los modelos. Se expuso una conclusión
- [ ] La *RECM* para el conjunto de prueba no es más de 48