Proyecto: Los Camachos

El objetivo principal de la empresa elegida, Los Camachos, es incrementar el número de visitantes al parque acuático, por lo que se seleccionó como KPI el número total de visitantes. Este indicador fue seleccionado ya que muestra directamente el objetivo de la empresa. El número total de visitantes se desglosa en la cantidad de visitantes adultos y la cantidad de visitantes niños. Para analizar este KPI, se recopilaron y utilizaron diversas variables, que ya sea directa o indirectamente, afectan y se relacionan con el indicador elegido, y se describen a continuación:

- Fecha: Fecha específica del día.
- Temperatura promedio: Temperatura promedio registrada por día.
- Cantidad adultos: Número de adultos que visitaron el parque por día.
- Cantidad niños: Número de niños que visitaron el parque por día.
- Precio adulto: Precio individual de los boletos para adultos.
- Precio niño: Precio individual de los boletos para niños.
- Total adultos: Ingresos totales por boletos de adulto por día.
- Total niños: Ingresos totales por boletos de niño por día.
- Cantidad familiar: Número de cupones familiares comprados por día.
- Precio familiar: Precio de un cupón familiar, que incluye un paquete de entradas.
- Total familiar: Ingresos totales por cupones familiares por día.
- Cantidad de cupones especiales: Número de cupones especiales comprados por día.
- Precio especial: Precio de un cupón especial, que incluye un descuento o promoción.
- Total especial: Ingresos totales por cupones especiales por día.
- Asistentes totales: Todas las personas que visitaron el parque por día.
- Alimento: Ingresos totales por venta de alimentos dentro del parque por día.
- Extras: Ingresos totales por venta de objetos y servicios adicionales por día.
- Raya: Egresos correspondientes al pago de salarios y compensaciones a los empleados por día.
- Gastos: Otros gastos operativos incurridos por día.
- Neto: El total de ingresos netos por día, calculado como la diferencia entre los ingresos totales y los egresos.

Librerías y Datos

In [5]:
```
import time
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from sklearn import set_config
from sklearn.metrics import mean_squared_error
import statsmodels.api as sm

from auto_ts import auto_timeseries
from semopy import Model, Optimizer, semplot
import os
os.environ['PATH'] = '/opt/homebrew/bin:' + os.environ['PATH']

plt.rcParams['figure.figsize'] = (12, 6)
set_config(working_memory=1024)
```

```
Imported auto_timeseries version:0.0.92. Call by using:
model = auto_timeseries(score_type='rmse',
        time_interval='M', non_seasonal_pdq=None, seasonality=False,
        seasonal_period=12, model_type=['best'], verbose=2, dask_xgboost_flag=0)
model.fit(traindata, ts_column,target)
model.predict(testdata, model='best')
```

In [6]:
```
start_time = time.perf_counter()
```

In [7]:
```
df = pd.read_excel('Datos.xlsx')
df.head()
```

Out[7]:

	año	mes	fecha	temperatura_promedio	cantidad_adultos	precio_adultos	total_adultos	cantidad_niños	precio_
0	2023	agosto	2023-08-01	22	82	170	13940	39	
1	2023	agosto	2023-08-02	24	153	170	26010	55	
2	2023	agosto	2023-08-03	22	98	170	16660	34	
3	2023	agosto	2023-08-04	21	110	170	18700	55	
4	2023	agosto	2023-08-05	22	151	170	25670	38	

5 rows × 21 columns

In [8]:

```
df['ingreso_total'] = df.total_adultos + df.total_niños + df.total_familia + df.total_especial + df.alimen
df['ingreso_entrada'] = df.total_adultos + df.total_niños + df.total_familia + df.total_especial
df['asistentes_totales'] = df.cantidad_adultos + df.cantidad_niños + df.cantidad_familia * 4 + df.cantidad_

df['fecha'] = pd.to_datetime(df['fecha'])
df['día'] = df['fecha'].dt.day_name()

df.head()
```

Out[8]:

	año	mes	fecha	temperatura_promedio	cantidad_adultos	precio_adultos	total_adultos	cantidad_niños	precio_
0	2023	agosto	2023-08-01	22	82	170	13940	39	
1	2023	agosto	2023-08-02	24	153	170	26010	55	
2	2023	agosto	2023-08-03	22	98	170	16660	34	
3	2023	agosto	2023-08-04	21	110	170	18700	55	
4	2023	agosto	2023-08-05	22	151	170	25670	38	

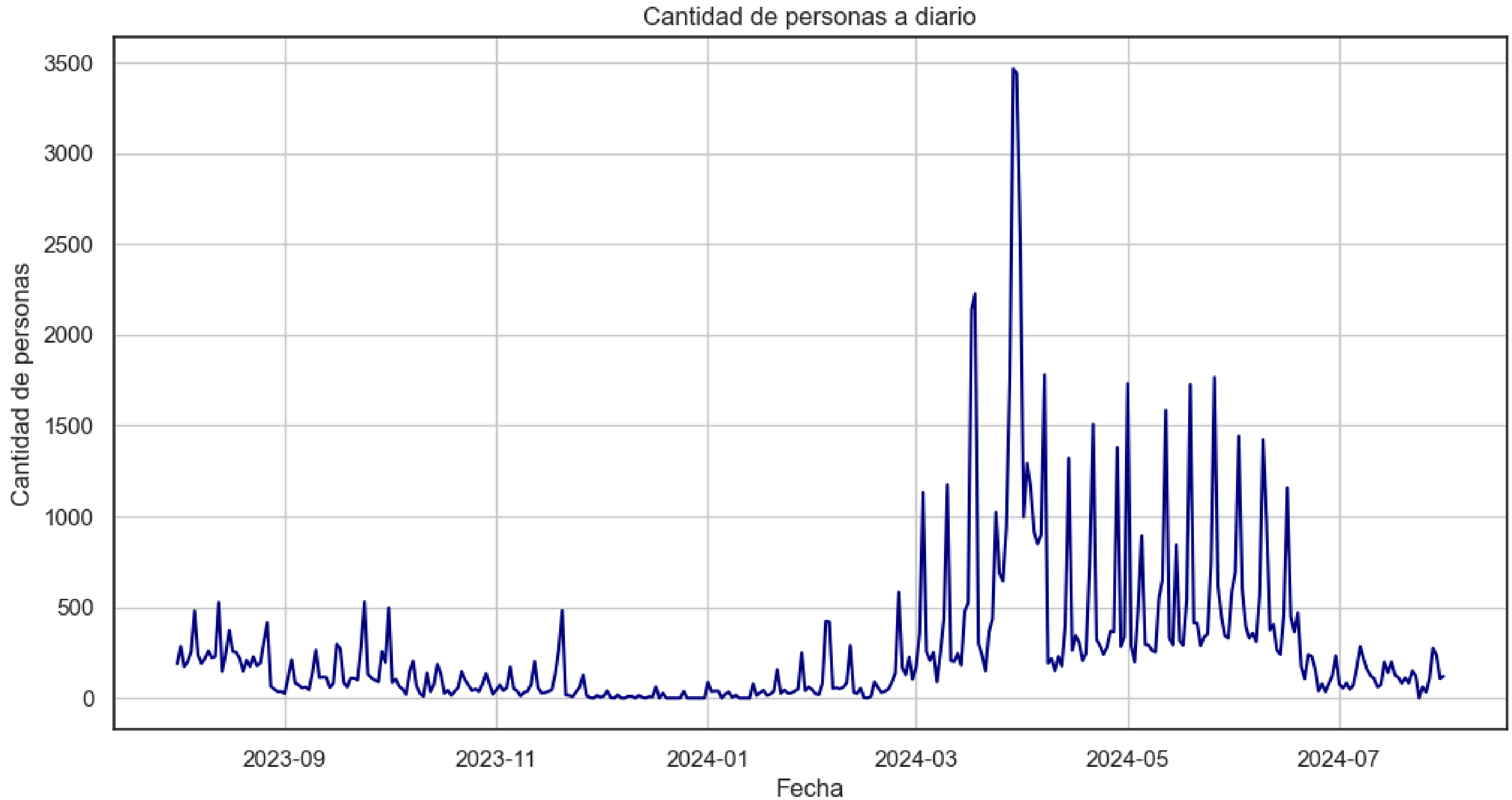
5 rows × 25 columns

Cantidad de asistentes

Asistencia por día

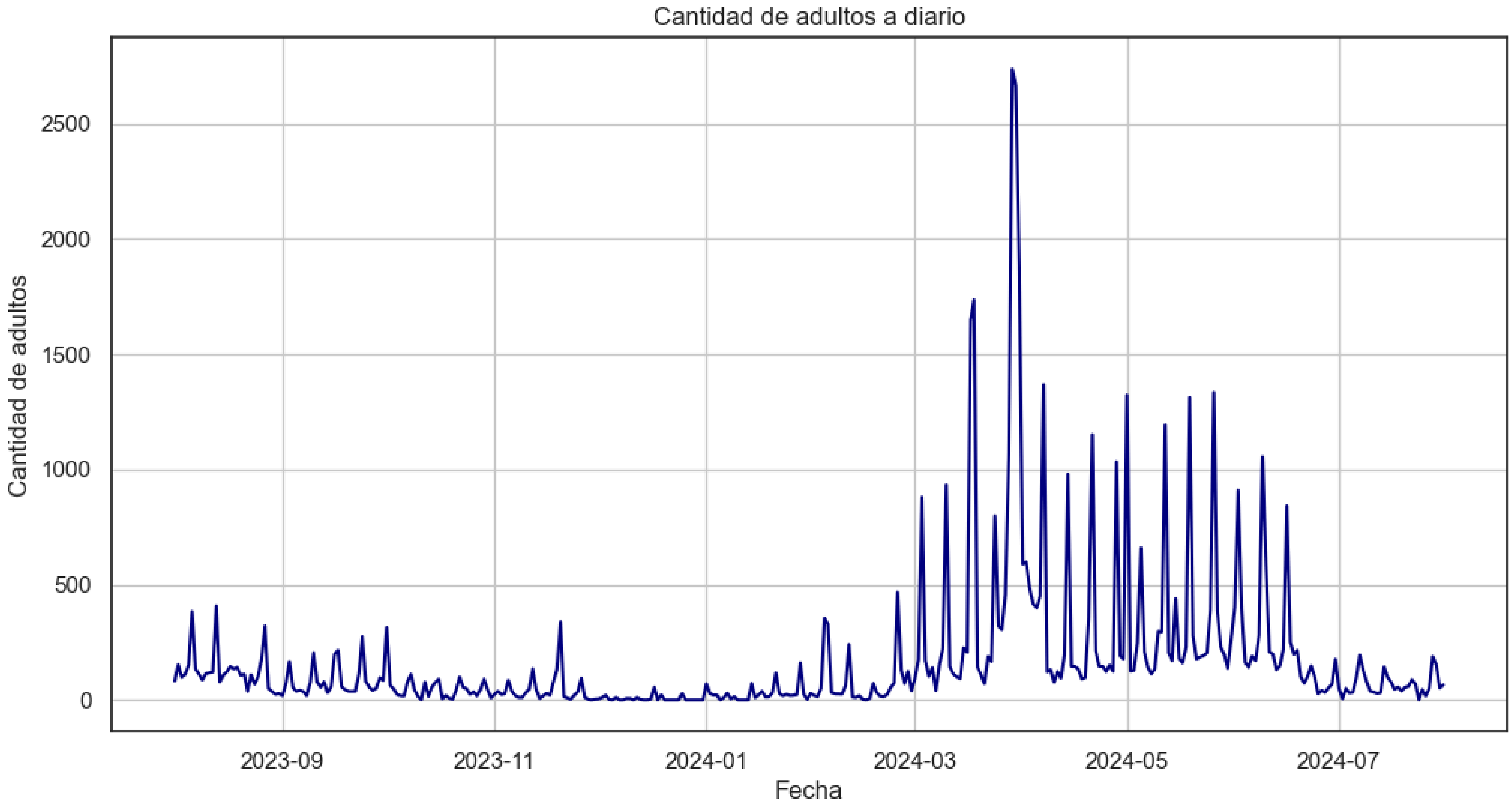
In [11]:

```
plt.plot(df.fecha, df.asistentes_totales, c = 'navy')
plt.title('Cantidad de personas a diario')
plt.xlabel('Fecha')
plt.ylabel('Cantidad de personas')
plt.grid()
```

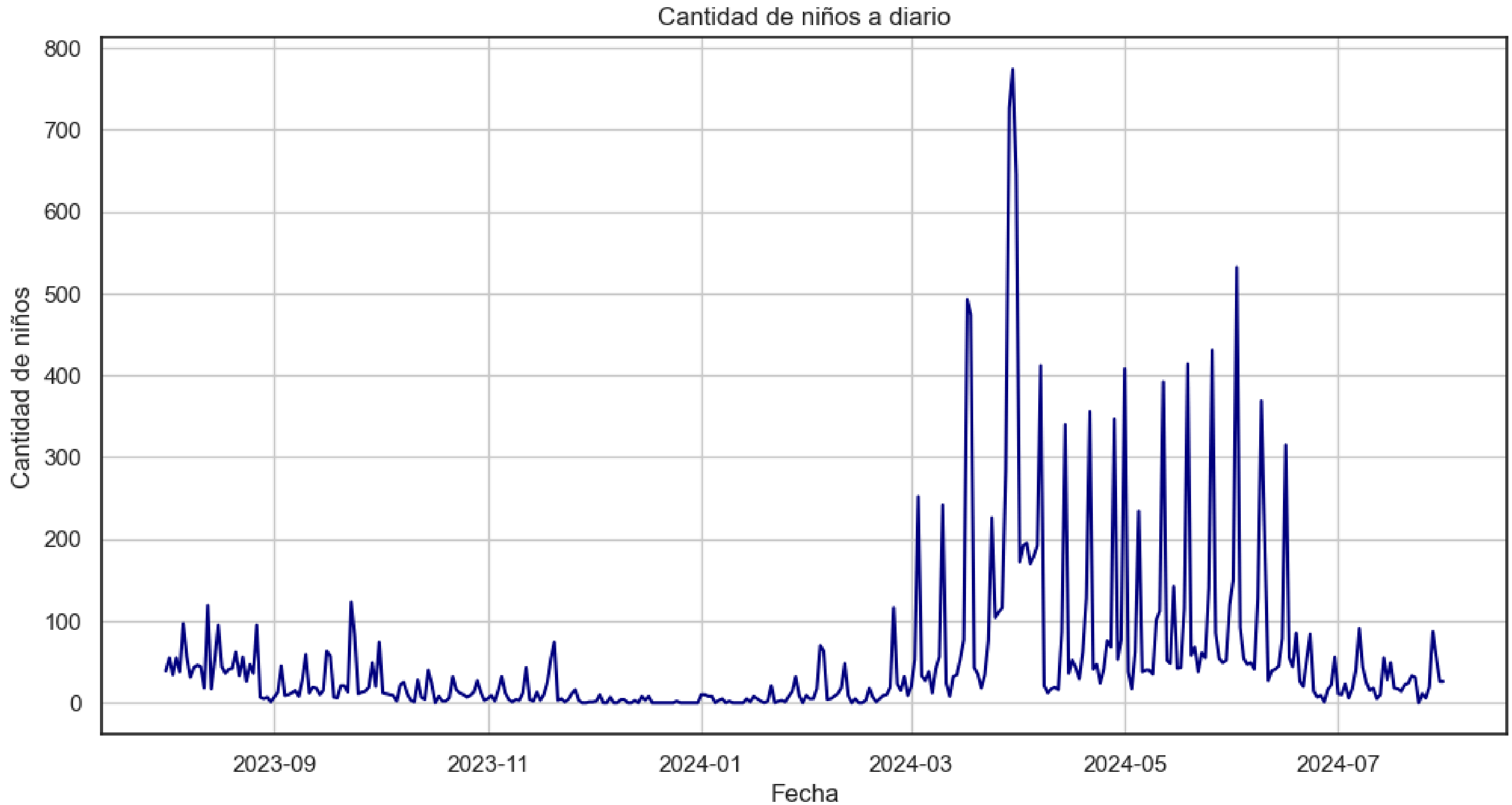


In [12]:

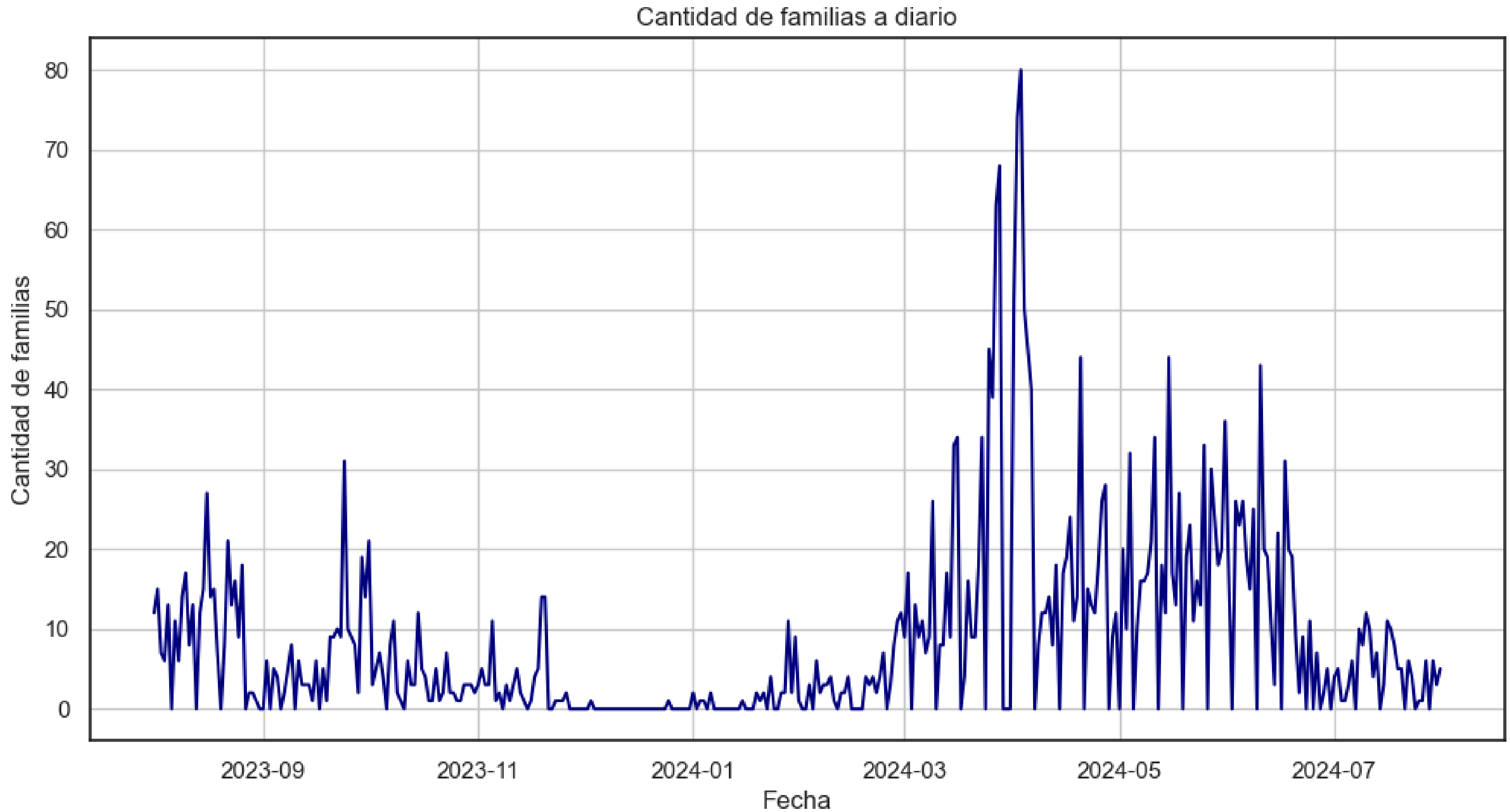
```
plt.plot(df.fecha, df.cantidad_adultos, c = 'navy')
plt.title('Cantidad de adultos a diario')
plt.xlabel('Fecha')
plt.ylabel('Cantidad de adultos')
plt.grid()
```



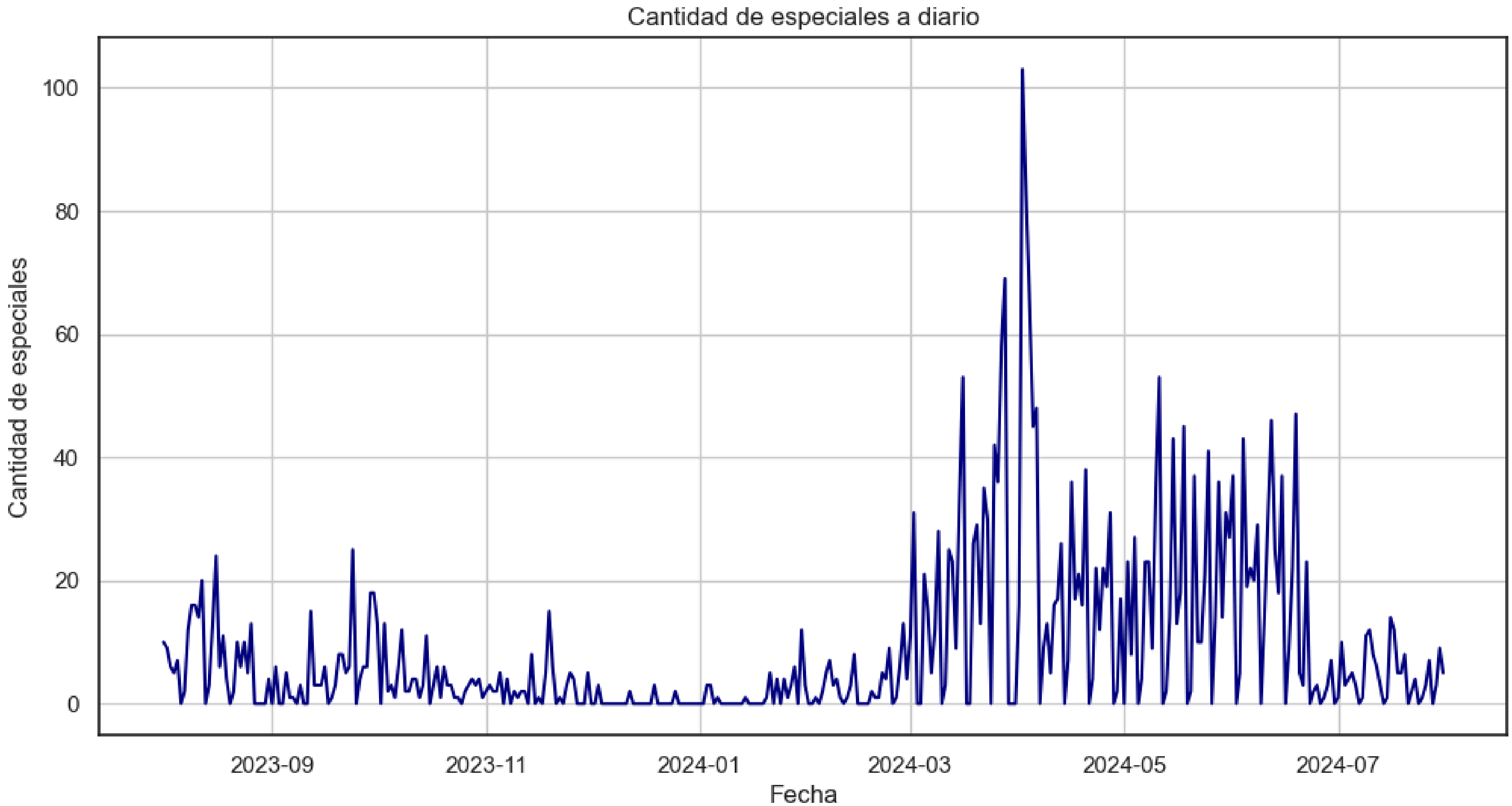
In [13]:
```
plt.plot(df.fecha, df.cantidad_niños, c = 'navy')
plt.title('Cantidad de niños a diario')
plt.xlabel('Fecha')
plt.ylabel('Cantidad de niños')
plt.grid()
```



In [14]:
```
plt.plot(df.fecha, df.cantidad_familia, c = 'navy')
plt.title('Cantidad de familias a diario')
plt.xlabel('Fecha')
plt.ylabel('Cantidad de familias')
plt.grid()
```

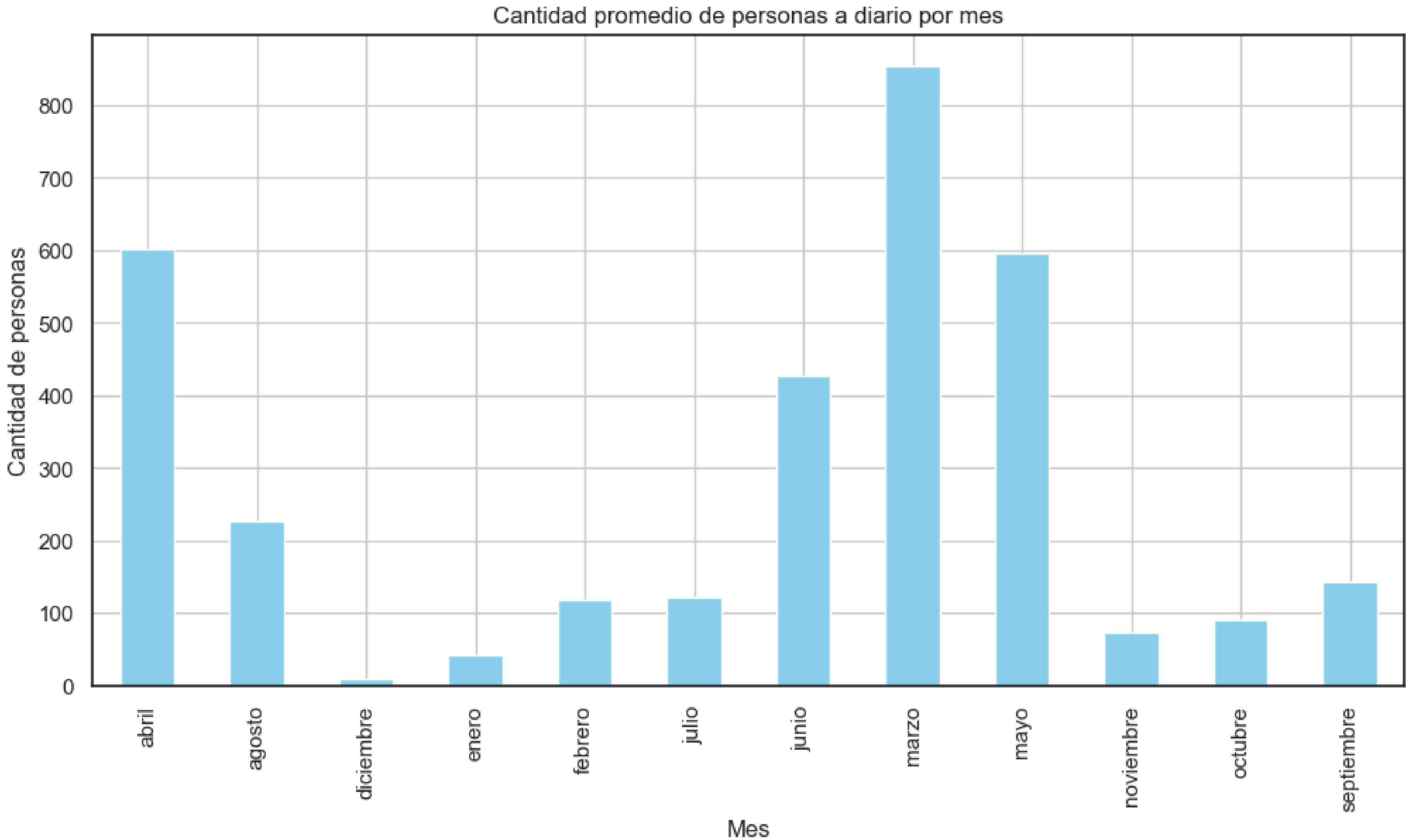


In [15]:
```
plt.plot(df.fecha, df.cantidad_especial, c = 'navy')
plt.title('Cantidad de especiales a diario')
plt.xlabel('Fecha')
plt.ylabel('Cantidad de especiales')
plt.grid()
```

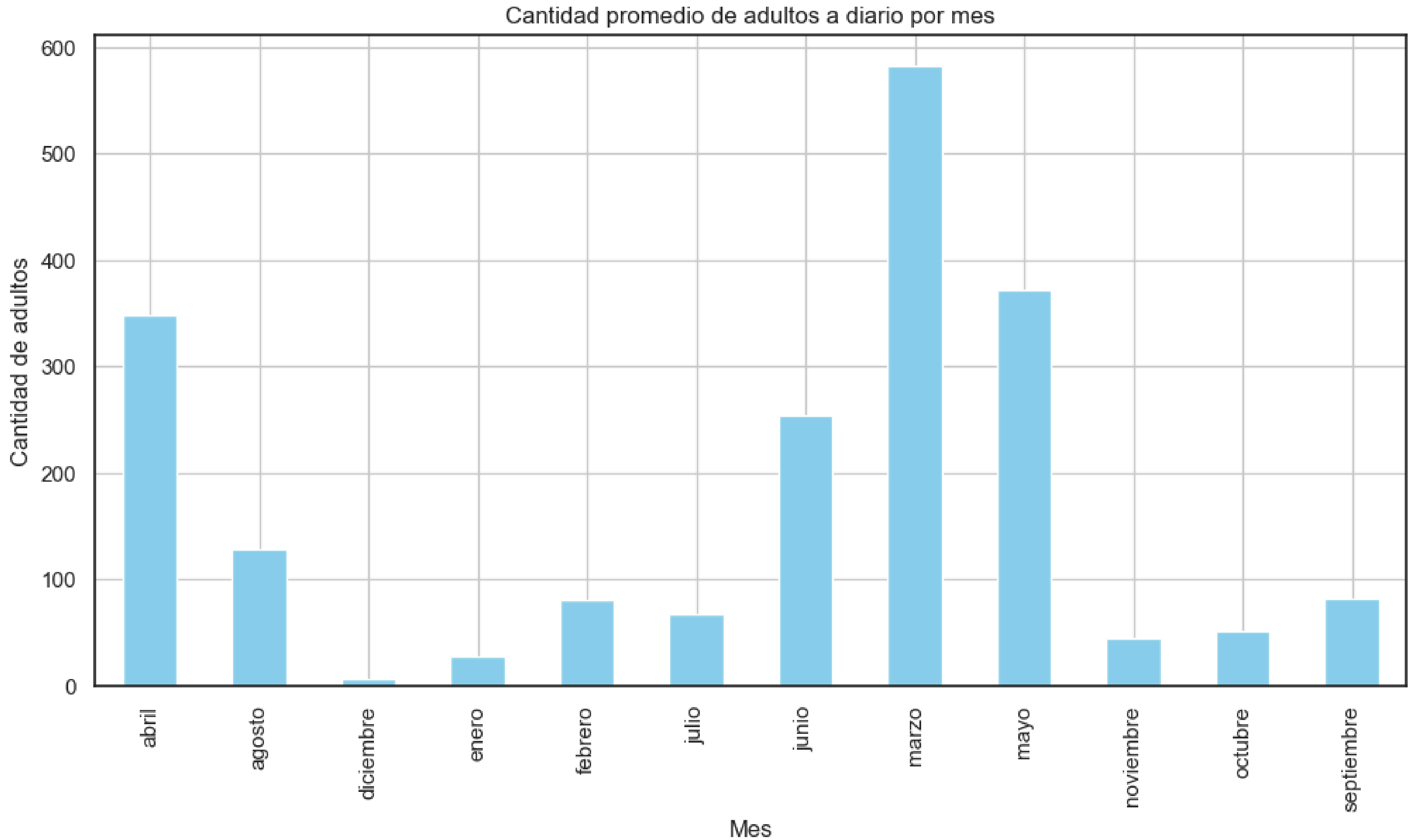


Asistencias promedio diaria por mes

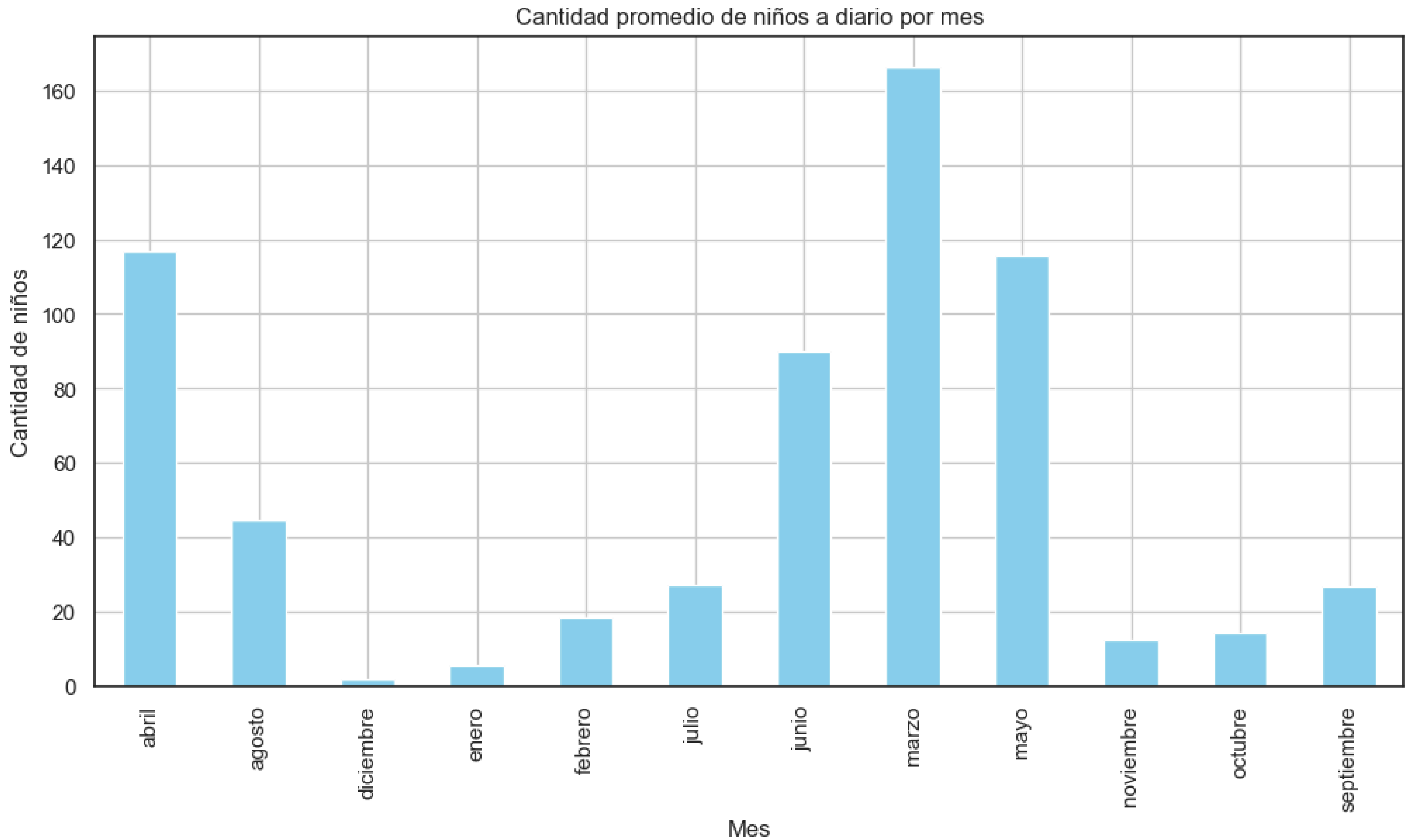
In [17]:
```
df.groupby('mes')['asistentes_totales'].mean().plot(kind='bar', color='skyblue')
plt.title('Cantidad promedio de personas a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Cantidad de personas')
plt.grid()
```



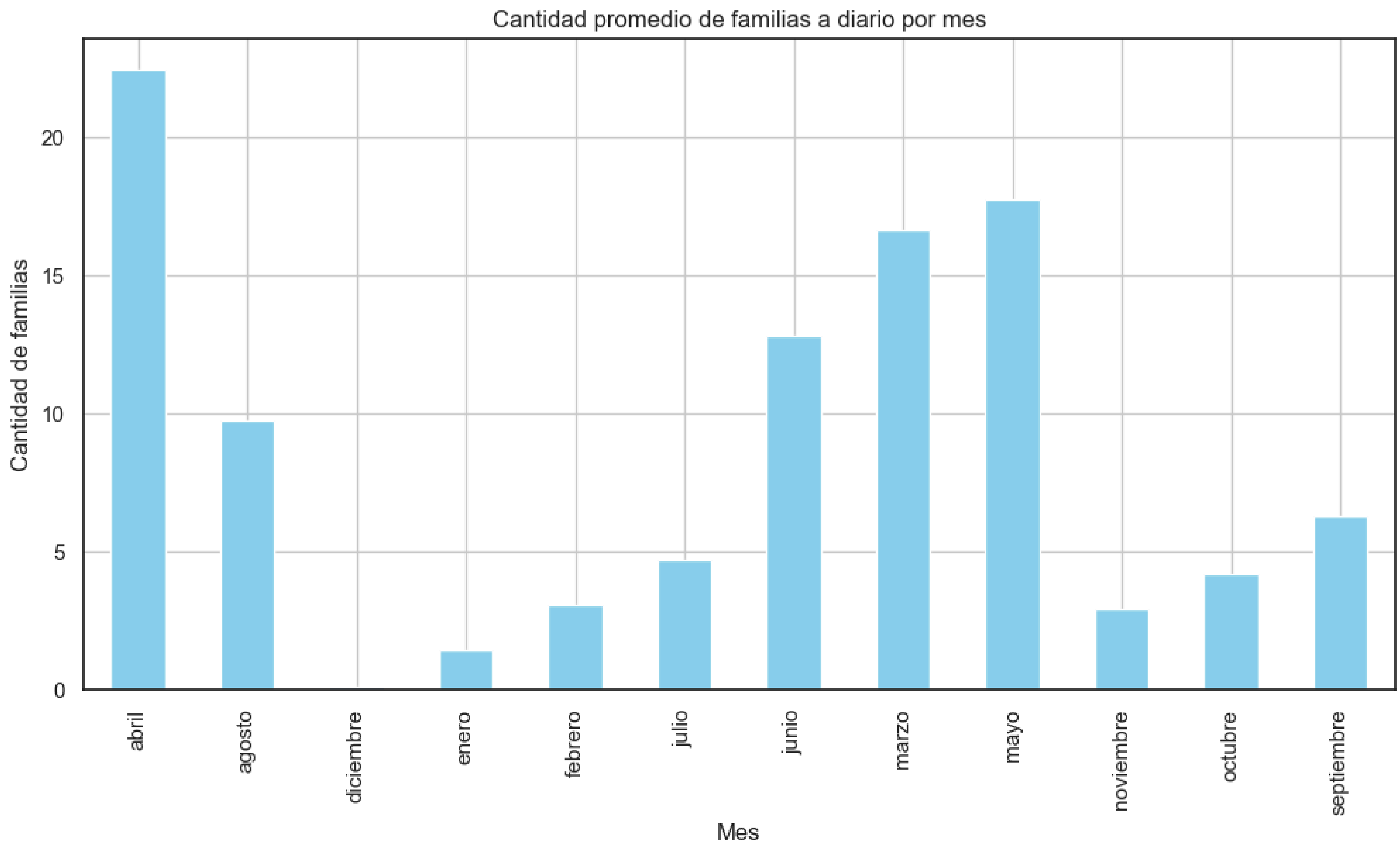
In [18]:
```
df.groupby('mes')['cantidad_adultos'].mean().plot(kind='bar', color='skyblue')
plt.title('Cantidad promedio de adultos a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Cantidad de adultos')
plt.grid()
```



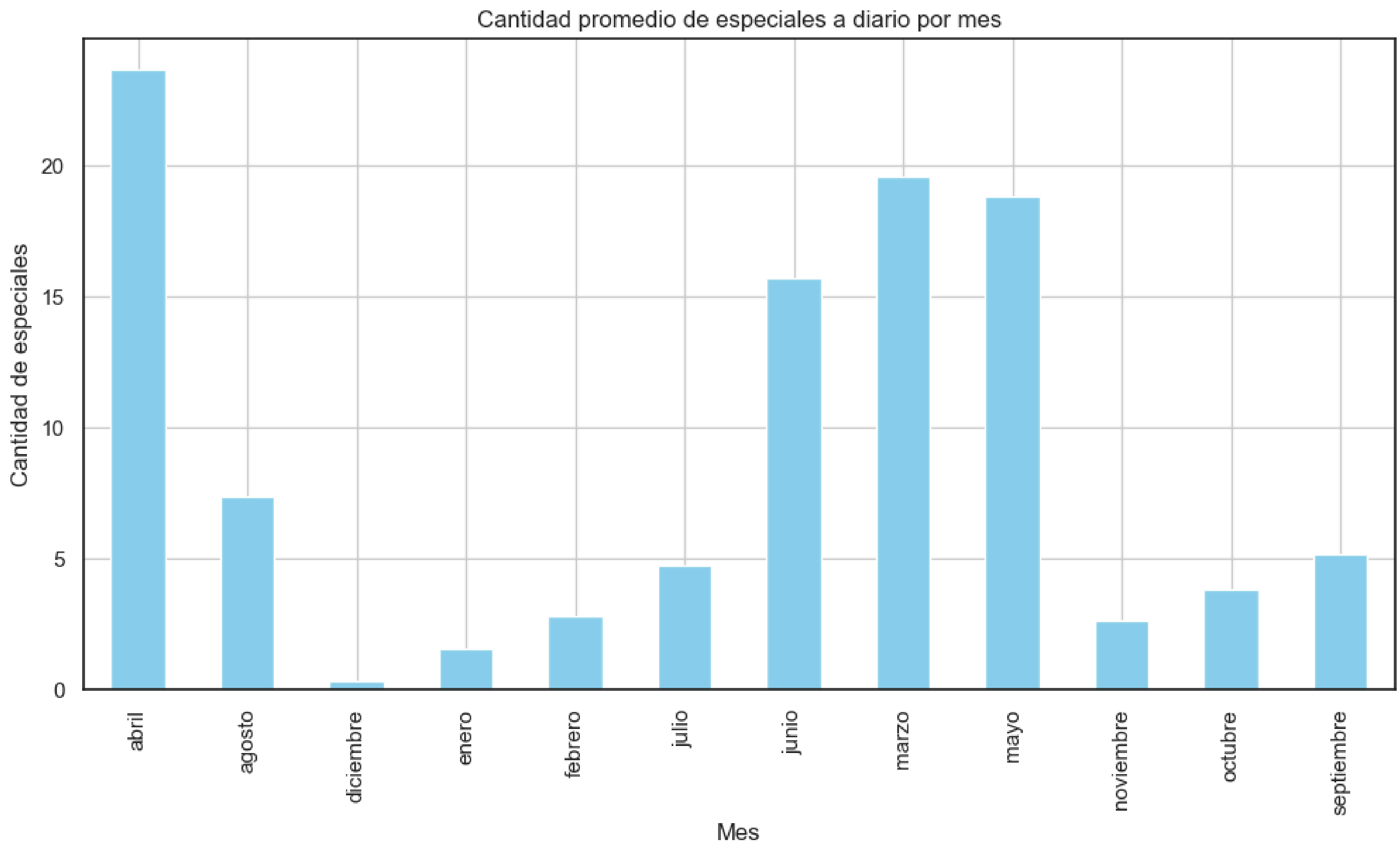
In [19]:
```
df.groupby('mes')['cantidad_niños'].mean().plot(kind='bar', color='skyblue')
plt.title('Cantidad promedio de niños a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Cantidad de niños')
plt.grid()
```



In [20]:
```
df.groupby('mes')['cantidad_familia'].mean().plot(kind='bar', color='skyblue')
plt.title('Cantidad promedio de familias a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Cantidad de familias')
plt.grid()
```



In [21]:
```
df.groupby('mes')['cantidad_especial'].mean().plot(kind='bar', color='skyblue')
plt.title('Cantidad promedio de especiales a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Cantidad de especiales')
plt.grid()
```



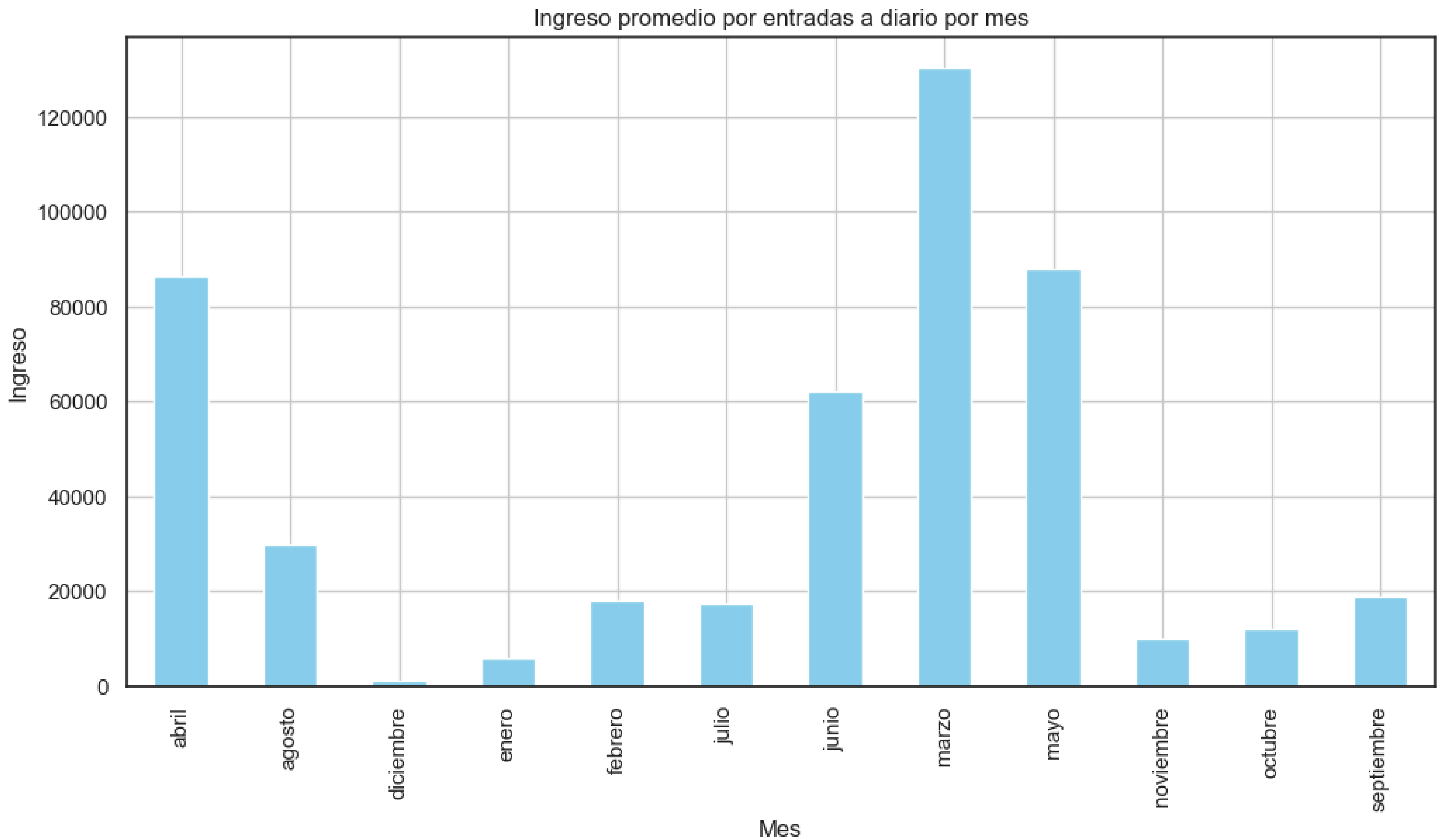
Ingresos

Ingresos por entradas

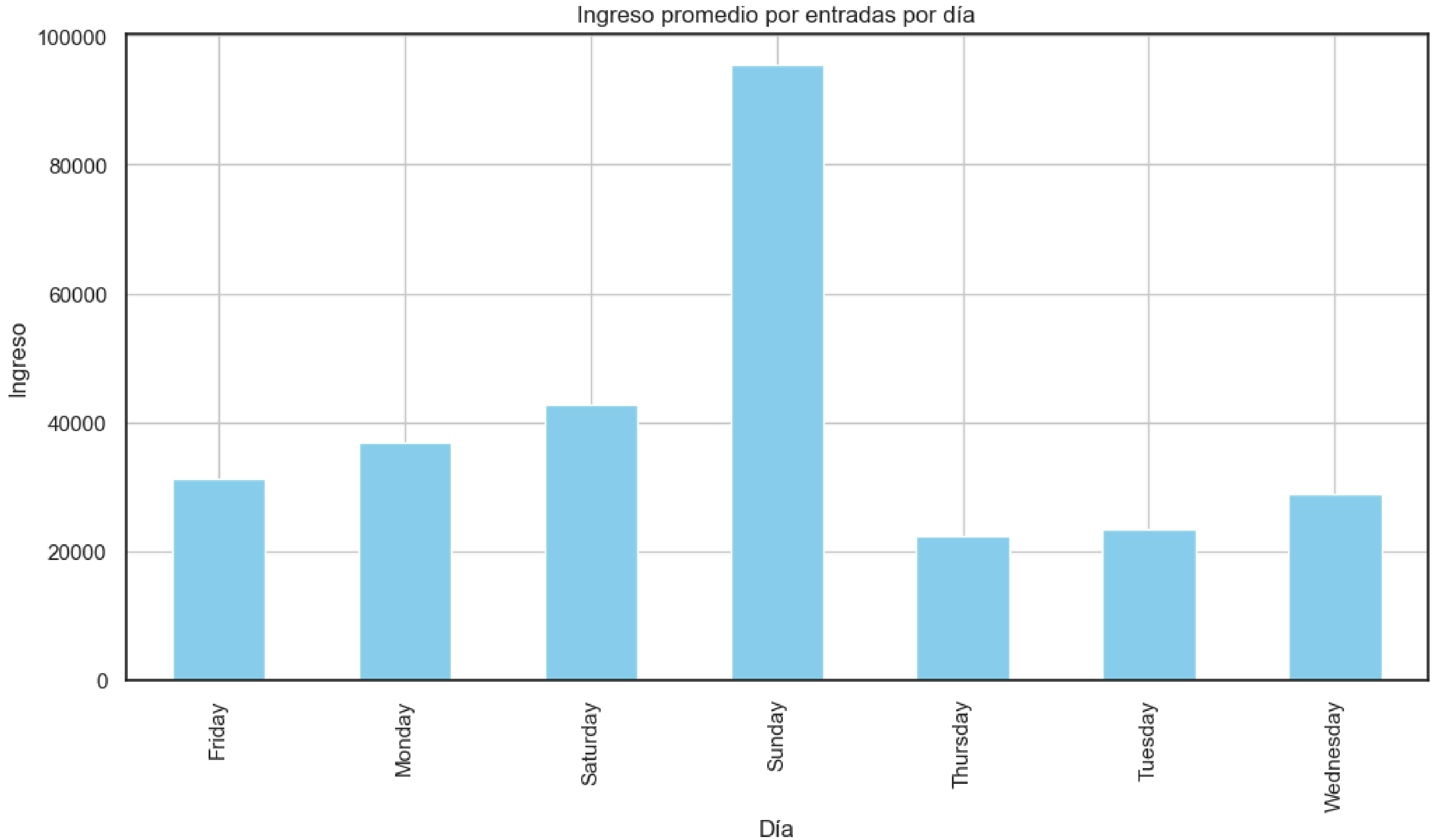
In [24]:
```
fig = px.line(df, x='fecha', y='ingreso_entrada', title='Ingreso por entradas a diario')
fig.update_layout(
    width=1200,
    height=600
```

```
)
fig.update_traces(line=dict(color='navy'))
fig.show()
```

In [25]:
```
df.groupby('mes')['ingreso_entrada'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso promedio por entradas a diario por mes')
plt.xlabel('Mes')
plt.ylabel('Ingreso')
plt.grid()
```



In [26]:
```
df.groupby('día')['ingreso_entrada'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso promedio por entradas por día')
plt.xlabel('Día')
plt.ylabel('Ingreso')
plt.grid()
```

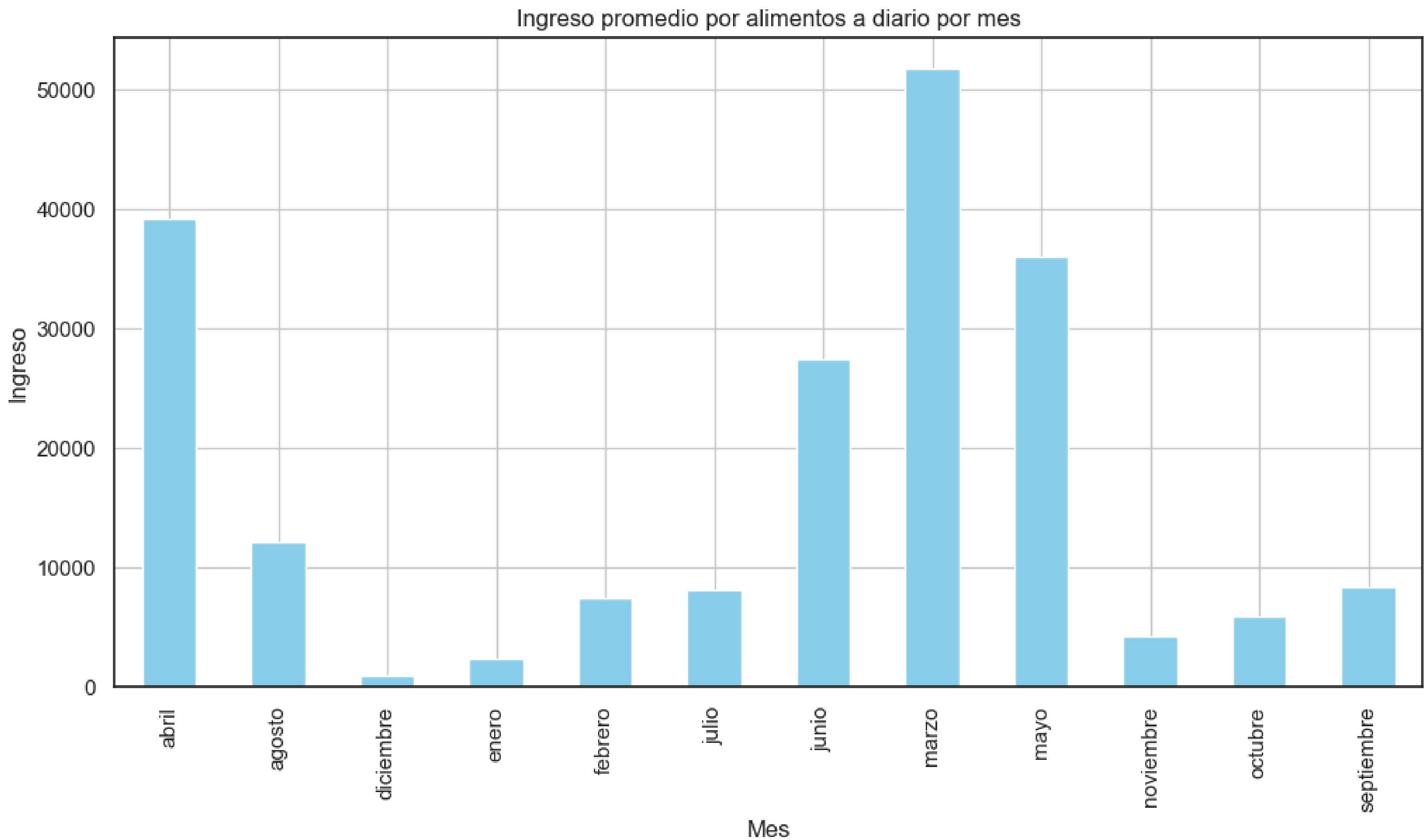


Ingreso por alimentos

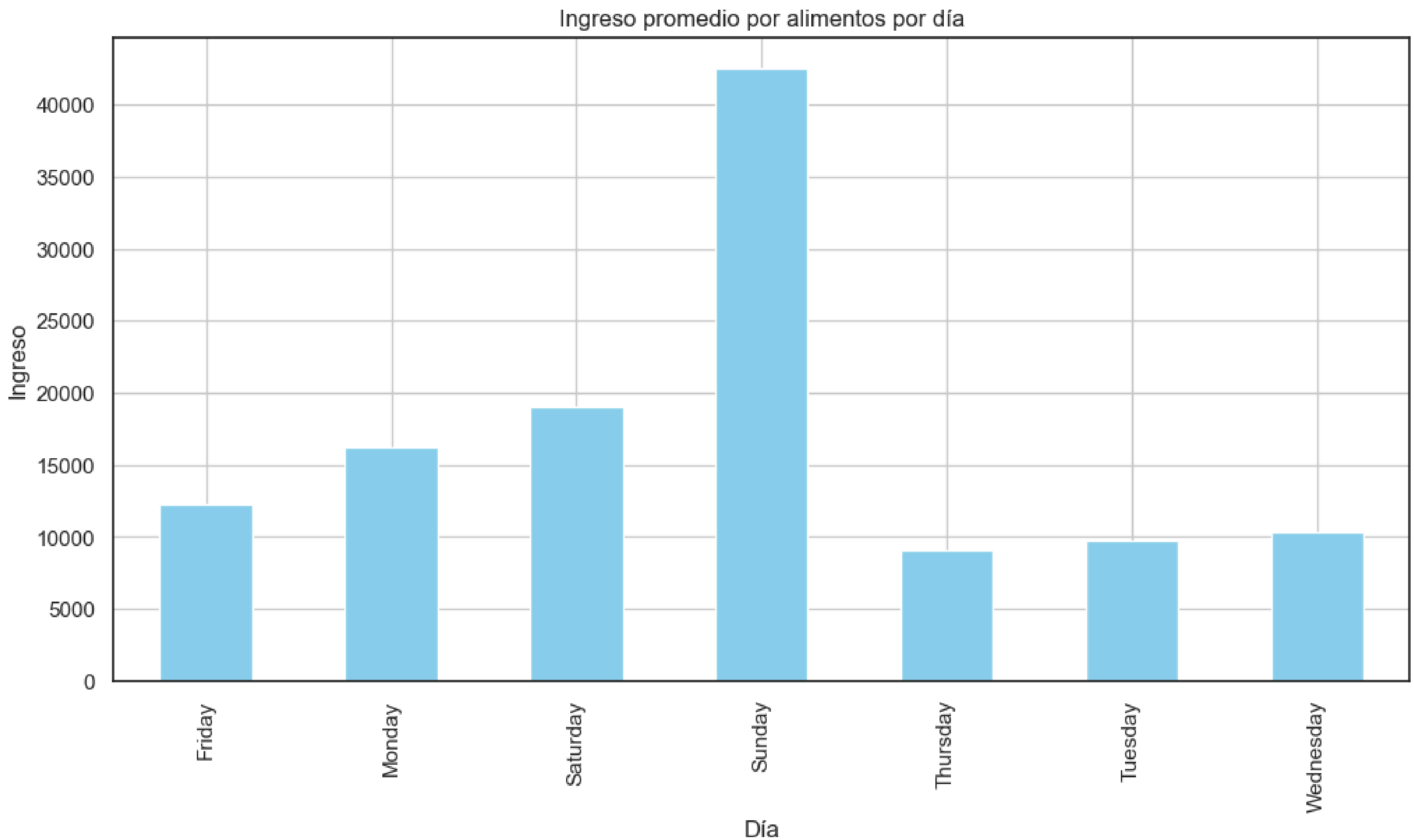
In [28]:
```
fig = px.line(df, x='fecha', y='alimento', title='Ingreso por comida a diario')
fig.update_layout(
    width=1200,
    height=600
)
fig.update_traces(line=dict(color='navy'))
fig.show()
```

In [29]:
```
df.groupby('mes')['alimento'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso promedio por alimentos a diario por mes')
plt.xlabel('Mes')
```

```
plt.ylabel('Ingreso')
plt.grid()
```



In [30]:
```
df.groupby('día')['alimento'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso promedio por alimentos por día')
plt.xlabel('Día')
plt.ylabel('Ingreso')
plt.grid()
```

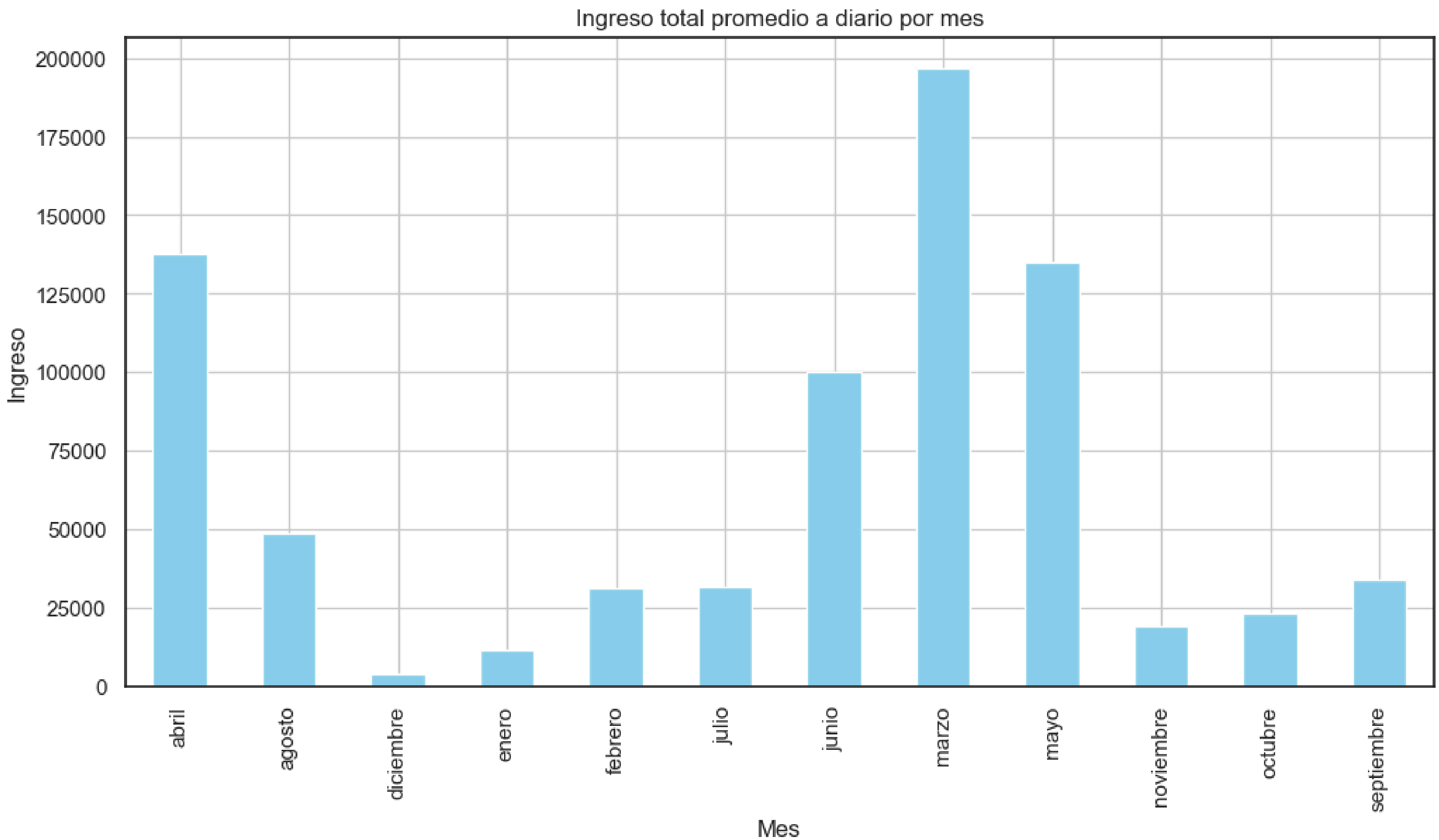


Ingresos totales

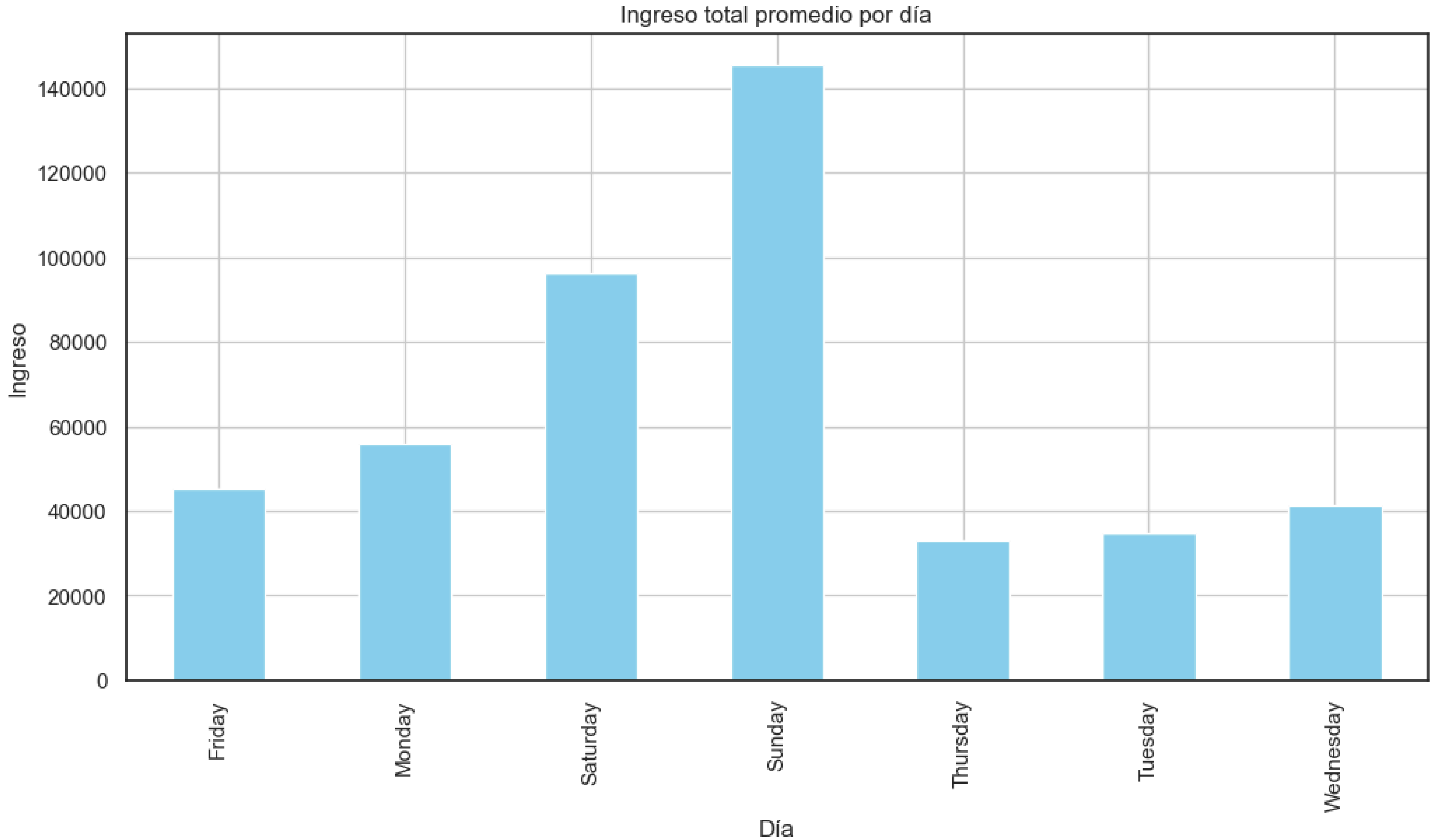
In [32]:
```
fig = px.line(df, x='fecha', y='ingreso_total', title='Ingreso total a diario')
fig.update_layout(
    width=1200,
    height=600
)
fig.update_traces(line=dict(color='navy'))
fig.show()
```

In [33]:
```
df.groupby('mes')['ingreso_total'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso total promedio a diario por mes')
plt.xlabel('Mes')
```

```
plt.ylabel('Ingreso')
plt.grid()
```



In [34]:
```
df.groupby('día')['ingreso_total'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso total promedio por día')
plt.xlabel('Día')
plt.ylabel('Ingreso')
plt.grid()
```

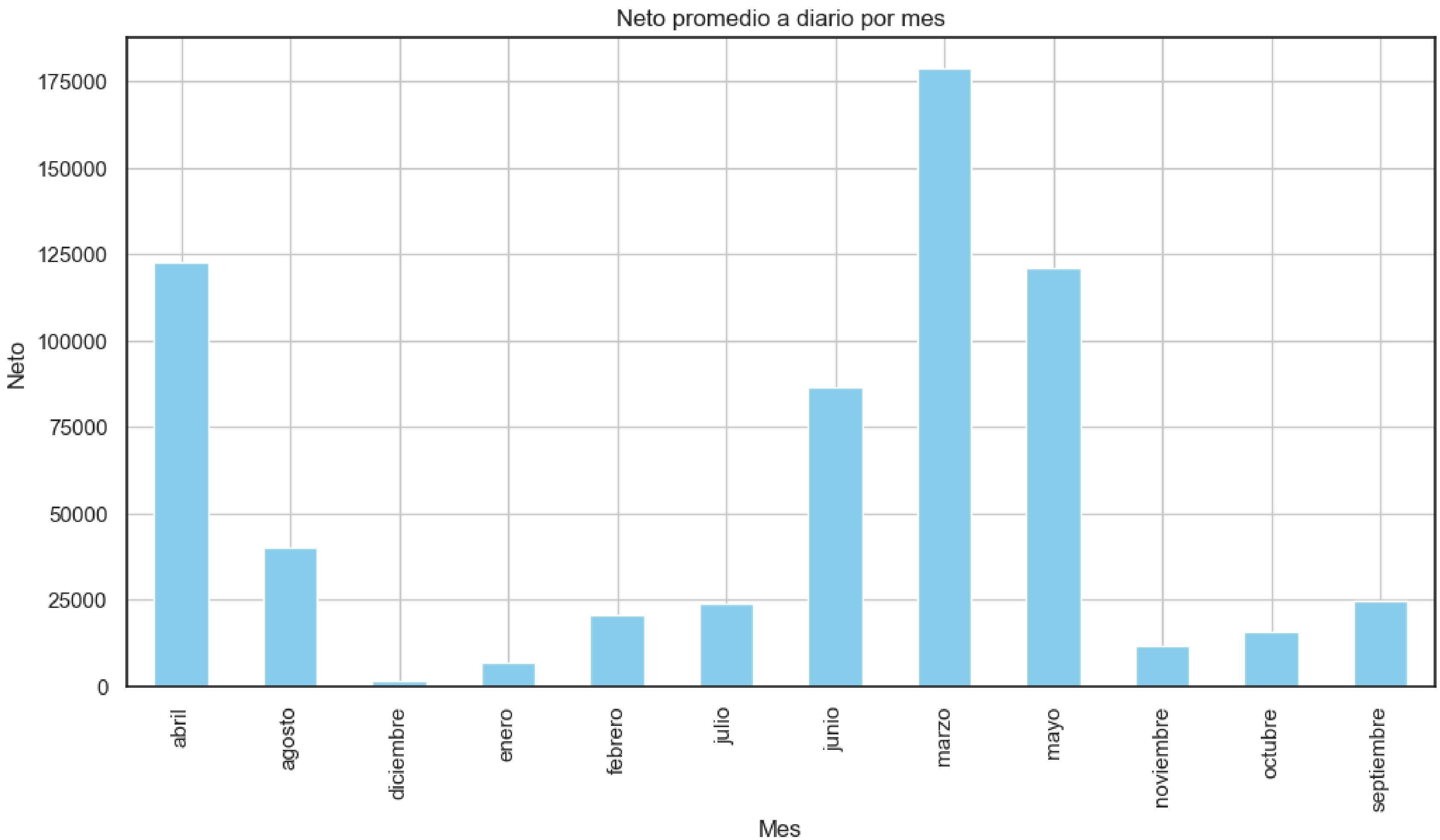


Neto

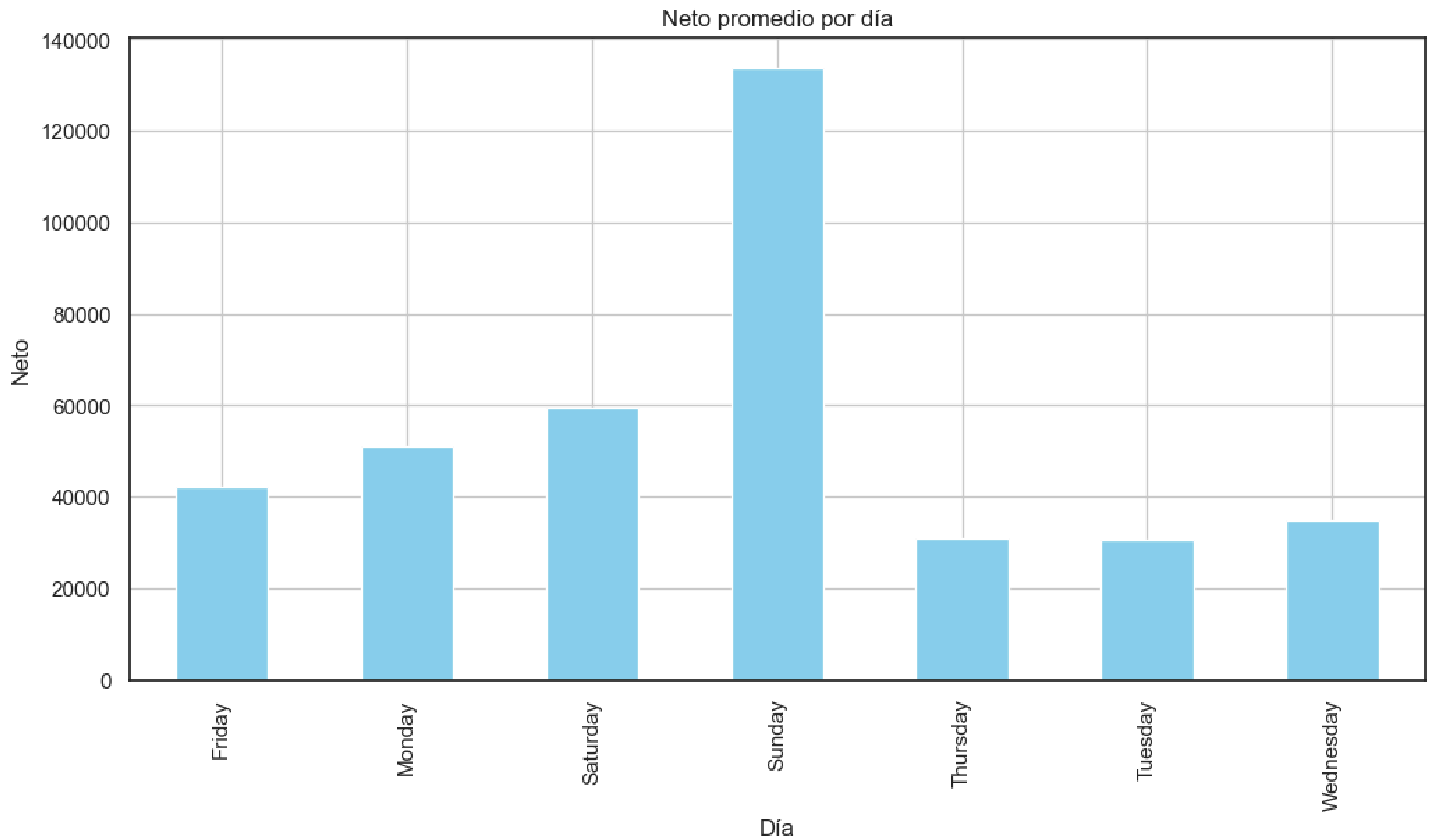
In [36]:
```
fig = px.line(df, x='fecha', y='neto', title='Neto diario')
fig.update_layout(
    width=1200,
    height=600
)
fig.update_traces(line=dict(color='navy'))
fig.show()
```

In [37]:
```
df.groupby('mes')['neto'].mean().plot(kind='bar', color='skyblue')
plt.title('Neto promedio a diario por mes')
plt.xlabel('Mes')
```

```python
plt.ylabel('Neto')
plt.grid()
```



In [38]:
```python
df.groupby('día')['neto'].mean().plot(kind='bar', color='skyblue')
plt.title('Neto promedio por día')
plt.xlabel('Día')
plt.ylabel('Neto')
plt.grid()
```

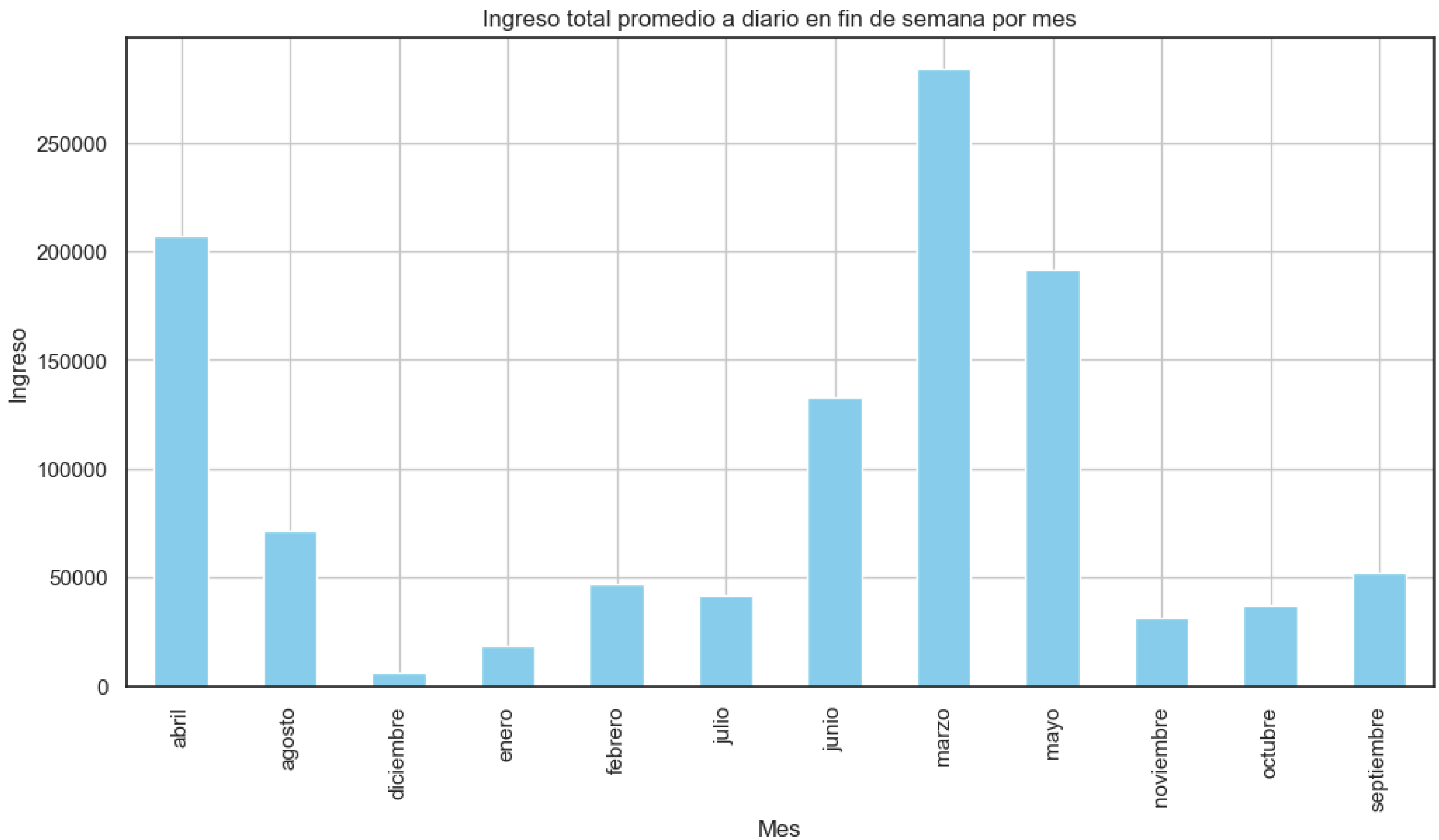


Separación

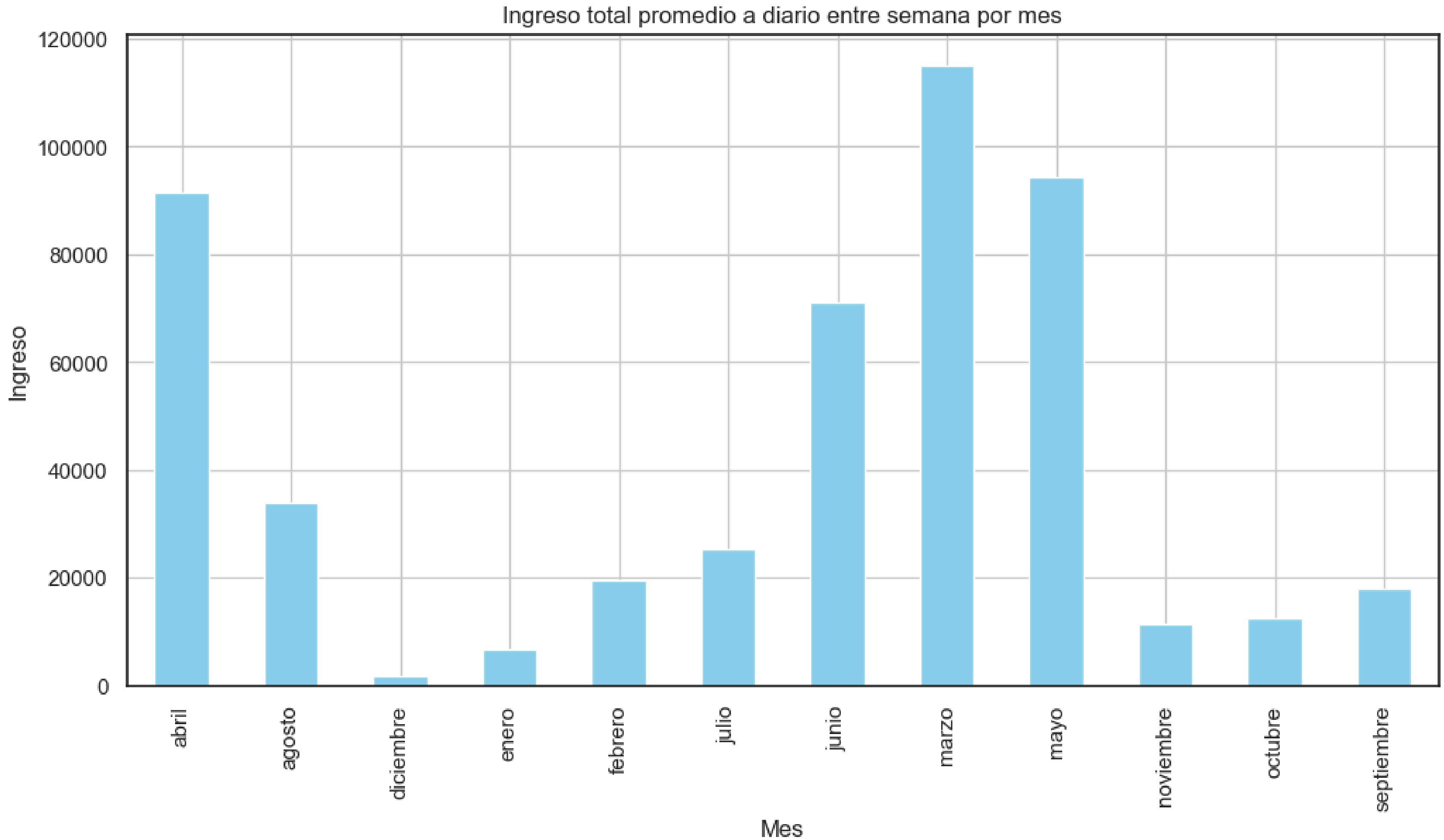
Ingreso total

In [41]:
```
wknd = df.query("día == 'Friday' or día == 'Saturday' or día == 'Sunday'")
wd = df.query("día == 'Monday' or día == 'Tuesday' or día == 'Wednesday' or día == 'Thursday'")

wknd.groupby('mes')['ingreso_total'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso total promedio a diario en fin de semana por mes')
```

```
plt.xlabel('Mes')
plt.ylabel('Ingreso')
plt.grid()
```

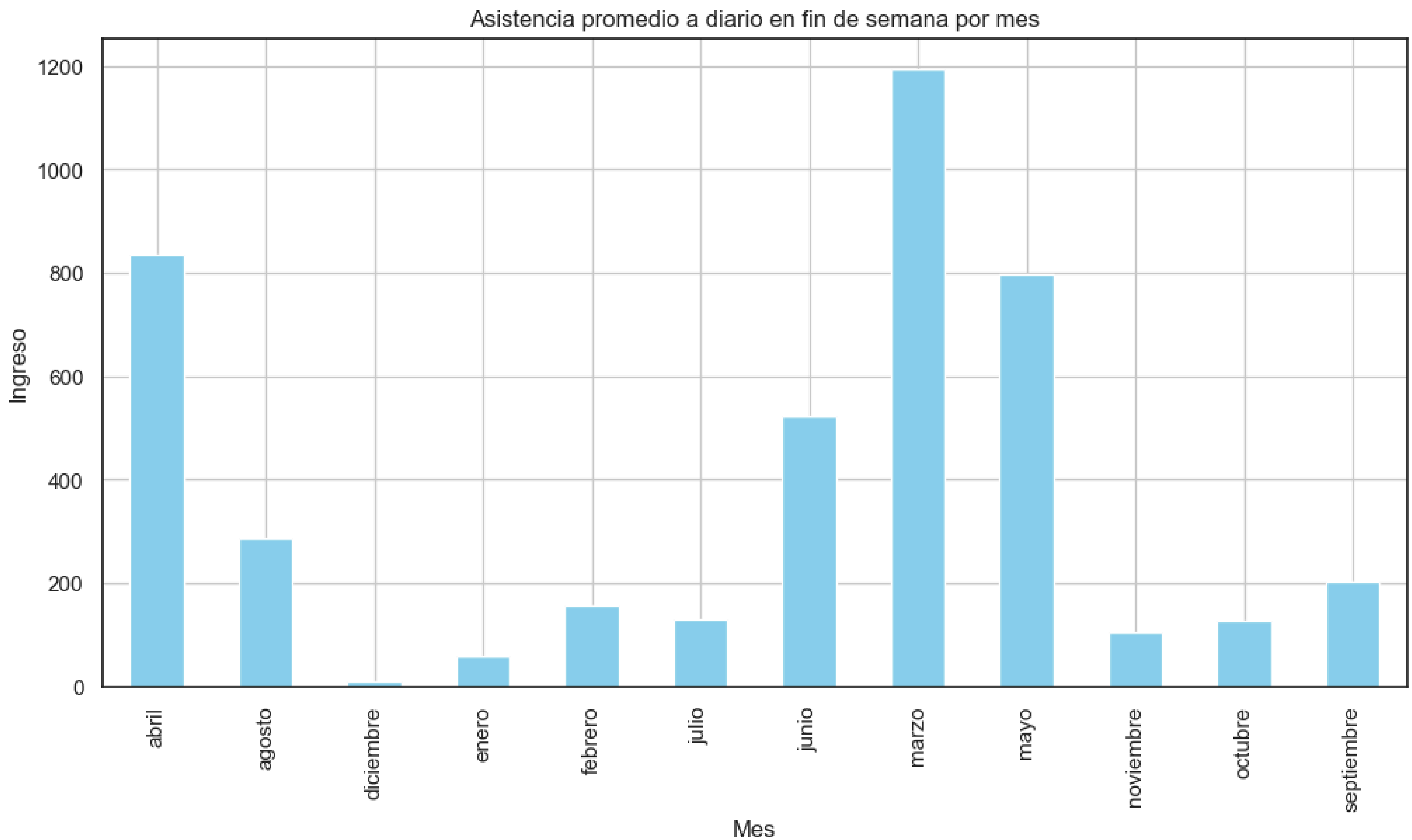


In [42]:
```
wd.groupby('mes')['ingreso_total'].mean().plot(kind='bar', color='skyblue')
plt.title('Ingreso total promedio a diario entre semana por mes')
plt.xlabel('Mes')
plt.ylabel('Ingreso')
plt.grid()
```

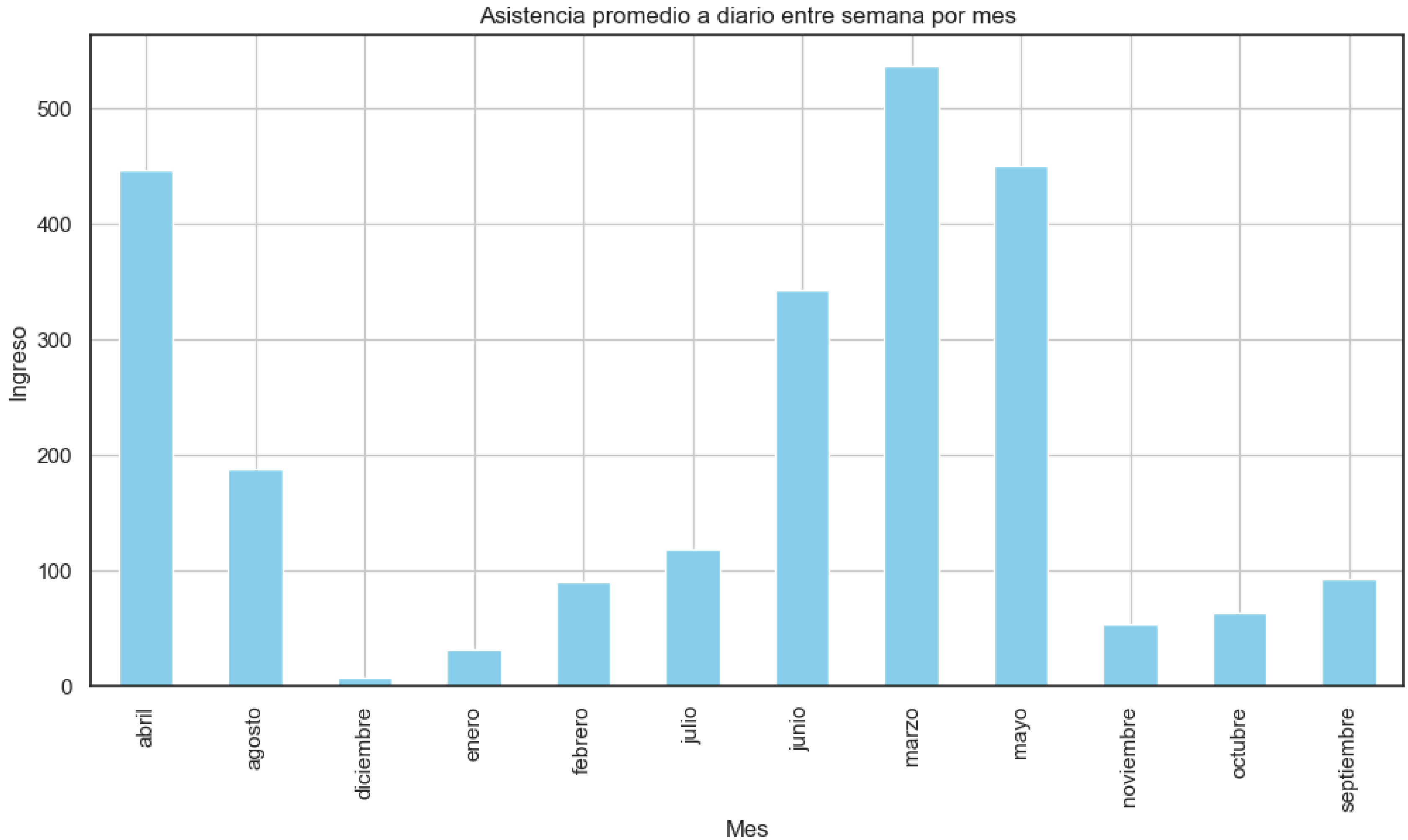


Asistencia total

In [44]:
```
wknd.groupby('mes')['asistentes_totales'].mean().plot(kind='bar', color='skyblue')
plt.title('Asistencia promedio a diario en fin de semana por mes')
plt.xlabel('Mes')
plt.ylabel('Ingreso')
plt.grid()
```

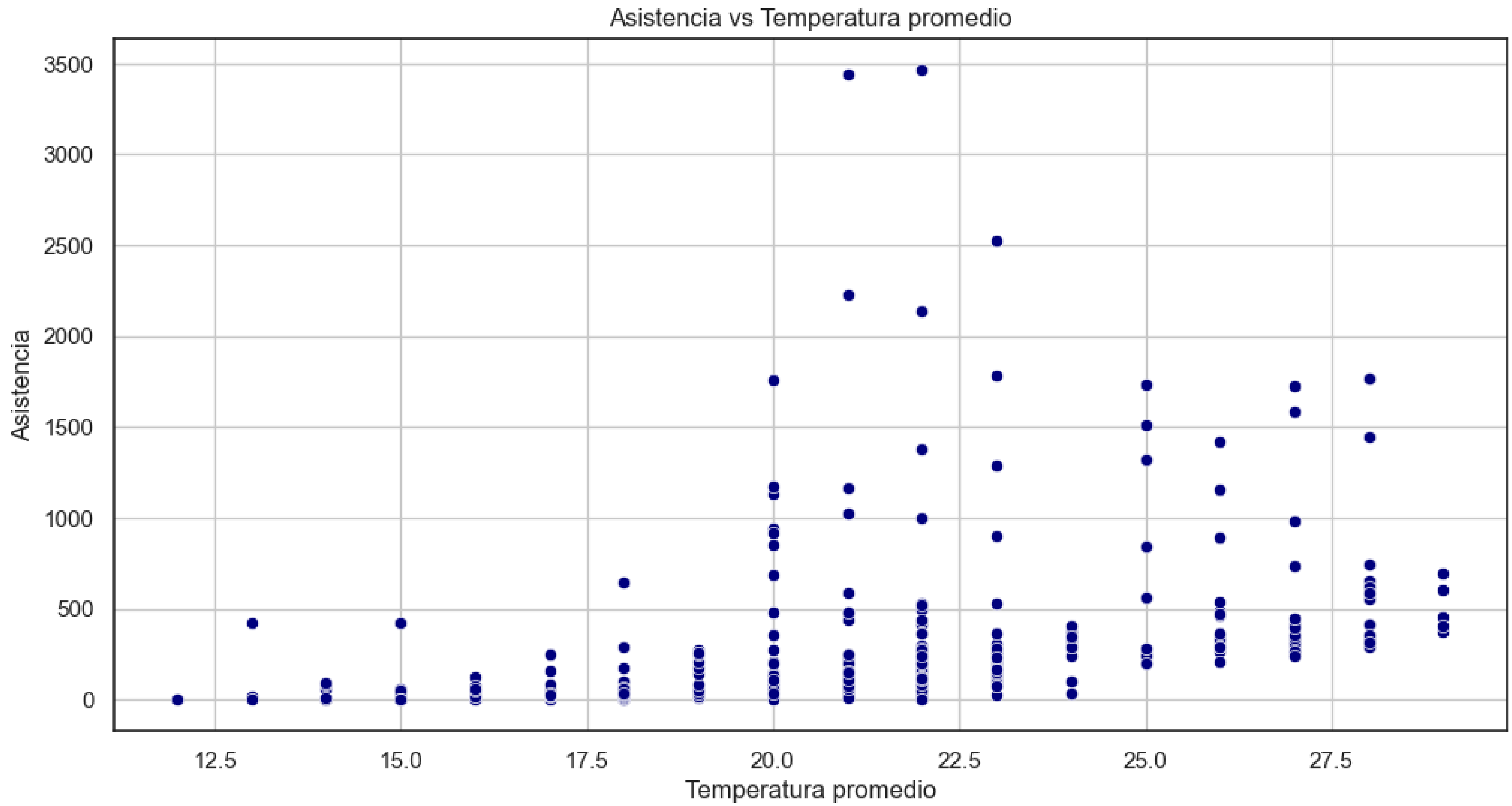


In [45]:
```
wd.groupby('mes')['asistentes_totales'].mean().plot(kind='bar', color='skyblue')
plt.title('Asistencia promedio a diario entre semana por mes')
plt.xlabel('Mes')
plt.ylabel('Ingreso')
plt.grid()
```



Correlación

In [47]:
```
sns.scatterplot(x = df.temperatura_promedio, y = df.asistentes_totales, c = 'navy')
plt.title('Asistencia vs Temperatura promedio')
plt.xlabel('Temperatura promedio')
plt.ylabel('Asistencia')
plt.grid()
plt.show()
```



In [48]:
```
df[['temperatura_promedio', 'asistentes_totales']].corr()
```

Out[48]:

	temperatura_promedio	asistentes_totales
temperatura_promedio	1.000000	0.375123
asistentes_totales	0.375123	1.000000

Regresión lineal

Dummies arbitrarias

In [51]:
```
df['is_wknd'] = df['día'].isin(['Friday', 'Saturday', 'Sunday'])
df['is_hs'] = df['mes'].isin(['marzo', 'abril', 'mayo'])

columns_from_education = df.loc[:, 'is_wknd':].select_dtypes(include='bool').columns
df[columns_from_education] = df[columns_from_education].astype(int)
```

In [52]:
```
X = df[['temperatura_promedio','is_wknd','is_hs']]
y = df['asistentes_totales']
```

In [53]:
```
# Agregar la constante (intercepto) a las variables independientes
X = sm.add_constant(X)

# Ajustar el modelo usando OLS
model = sm.OLS(y, X)
results = model.fit()

# Mostrar el resumen del modelo
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     asistentes_totales   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.343
Method:                 Least Squares   F-statistic:                     64.39
Date:                Mon, 21 Oct 2024   Prob (F-statistic):           2.19e-33
Time:                        18:37:55   Log-Likelihood:                -2672.9
No. Observations:                 366   AIC:                             5354.
Df Residuals:                     362   BIC:                             5369.
Df Model:                           3
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
const                 -372.1470    119.040     -3.126      0.002    -606.243    -138.051
temperatura_promedio    21.7387      5.777      3.763      0.000      10.378      33.099
is_wknd                178.5335     38.196      4.674      0.000     103.420     253.647
is_hs                  464.1459     48.562      9.558      0.000     368.647     559.645
==============================================================================
Omnibus:                      314.549   Durbin-Watson:                   0.925
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             7005.504
Skew:                           3.554   Prob(JB):                         0.00
Kurtosis:                      23.220   Cond. No.                         136.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Hacemos una regresión lineal para predecir la cantidad de personas que asistirán al parque segun ciertas características y se oobtiene los siguiente:

- Temperatura promedio: Por cada unidad que se mueva la temperatura promedio del día afecta 21.7387 a la cantidad de personas.
- is_wknd: Esto significa que si es fin de semana se esperan 178.5335 personas.
- is_hs: Si es temporada alta (marzo, abril o mayo) se esperan 464.1459 personas.

El modelo tiene una R^2 de 0.348, por lo que explica muy poca variación de los datos pero también tiene un F-estadístico muy bajo lo que significa que al menos una de las variables impacta significativamente a la cantidad de asistentes.

In [55]:
```
new_day_data = {
    'const': [1],
    'temperatura_promedio': [27],
    'is_wknd': [1],
    'is_hs': [1]
}

new_day_data = pd.DataFrame(new_day_data)

asistentes_predecidos = results.predict(new_day_data)

print(f"Asistentes predecidos del nuevo día: {asistentes_predecidos[0]:,.2f}")
```

```
Asistentes predecidos del nuevo día: 857.48
```

Dummies de código

In [57]:
```
categorical_features = ['mes','día']
numerical_features = ['temperatura_promedio']

data_encoded = pd.get_dummies(df[['asistentes_totales'] + categorical_features], columns=categorical_featu

data_encoded['temperatura_promedio'] = df['temperatura_promedio']

df_to_model = data_encoded
```

In [58]:
```
X1 = df_to_model[['temperatura_promedio','mes_marzo','mes_abril','mes_mayo',
                'día_Sunday']]
y1 = df_to_model['asistentes_totales']
```

In [59]:
```
# Agregar la constante (intercepto) a las variables independientes
X1 = sm.add_constant(X1)

# Ajustar el modelo usando OLS
model1 = sm.OLS(y1, X1)
results1 = model1.fit()

# Mostrar el resumen del modelo
print(results1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     asistentes_totales   R-squared:                       0.447
Model:                            OLS   Adj. R-squared:                  0.440
Method:                 Least Squares   F-statistic:                     58.28
Date:                Mon, 21 Oct 2024   Prob (F-statistic):           2.44e-44
Time:                        18:37:56   Log-Likelihood:                -2642.6
No. Observations:                 366   AIC:                             5297.
Df Residuals:                     360   BIC:                             5321.
Df Model:                           5
Covariance Type:            nonrobust
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
const                 -559.4356    116.855     -4.787      0.000    -789.241    -329.631
temperatura_promedio    32.0352      5.740      5.581      0.000      20.747      43.323
mes_marzo              681.2852     63.403     10.745      0.000     556.598     805.972
mes_abril              354.0969     67.275      5.263      0.000     221.795     486.398
mes_mayo               246.9681     74.120      3.332      0.001     101.207     392.730
día_Sunday             400.5772     49.948      8.020      0.000     302.350     498.805
==============================================================================
Omnibus:                      314.794   Durbin-Watson:                   0.904
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             8737.355
Skew:                           3.454   Prob(JB):                         0.00
Kurtosis:                      25.918   Cond. No.                         153.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Hacemos una regresión lineal para predecir la cantidad de personas que asistirán al parque segun ciertas características y se oobtiene los siguiente:

- Temperatura promedio: Por cada unidad que se mueva la temperatura promedio del día afecta 32.0352 a la cantidad de personas.
- mes_marzo: Esto significa que si es marzo se esperan 681 personas.
- mes_abril: Esto significa que si es abril se esperan 354 personas.
- mes_mayo: Esto significa que si es mayo se esperan 246 personas.
- día_Sunday: Esto significa que si es domingo se esperan 400 personas.

El modelo tiene una R^2 de 0.447, por lo que explica casi la mitad de la variación de los datos, también tiene un F-estadístico muy bajo lo que significa que al menos una de las variables impacta significativamente a la cantidad de asistentes. Cabe resaltar que está segunda regresión nos da un mejor modelo.

In [61]:
```
new_day_data1 = {
    'const': [1],
    'temperatura_promedio': [27],
    'mes_marzo': [1],
    'mes_abril': [0],
    'mes_mayo' : [0],
    'día_Sunday' : [1]
}

new_day_data1 = pd.DataFrame(new_day_data1)

asistentes_predecidos1 = results1.predict(new_day_data1)

print(f"Asistentes predecidos del nuevo día: {asistentes_predecidos1[0]:,.2f}")
```

```
Asistentes predecidos del nuevo día: 1,387.38
```

In [62]:
```
### Regresión lineal sin outliers
#Q1 = df['asistentes_totales'].quantile(0.25)  # Primer Cuartil (25%)
#Q3 = df['asistentes_totales'].quantile(0.75)  # Tercer Cuartil (75%)
#IQR = Q3 - Q1  # Rango Intercuartílico
#limite_inferior = Q1 - 1.5 * IQR
#limite_superior = Q3 + 1.5 * IQR
#limite_inferior, limite_superior
#df_3 = df.query('-333.25 < asistentes_totales < 664.75')
```

Predicción con series de tiempo

Datos

In [65]:
```
df_ts = df[['fecha','asistentes_totales']]
df_ts.set_index('fecha', inplace=True)
df_ts.head()
```

Out[65]:

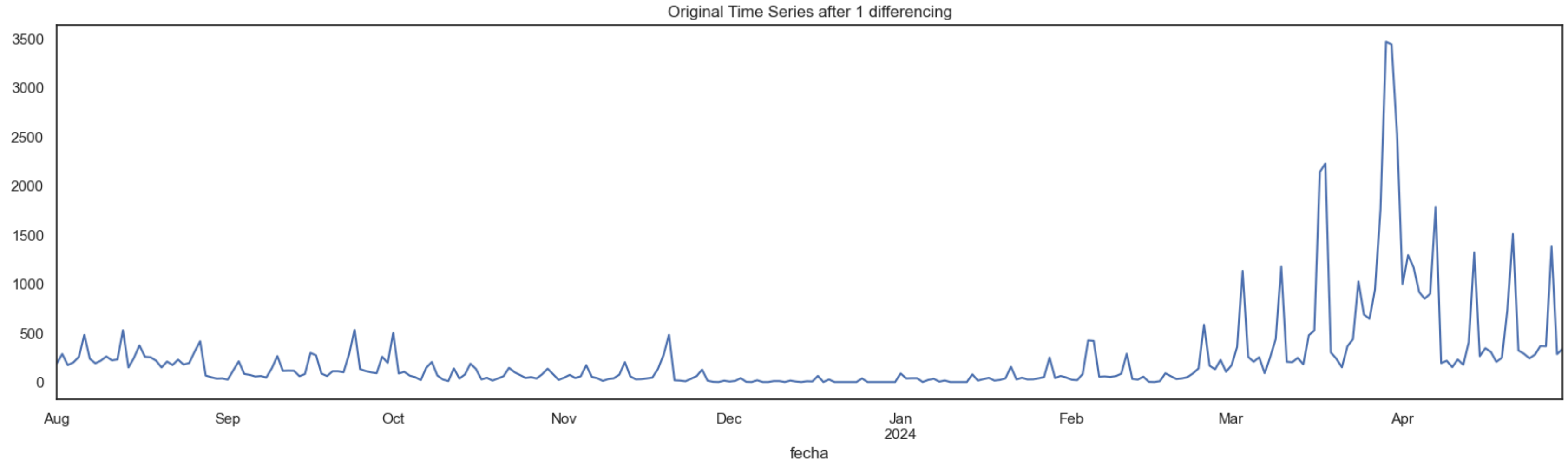
	asistentes_totales
fecha	
2023-08-01	189
2023-08-02	286
2023-08-03	172
2023-08-04	199
2023-08-05	255

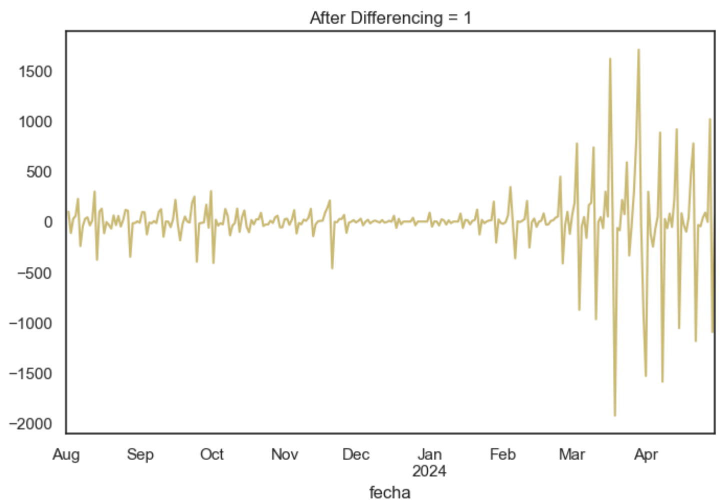
Definición del Modelo

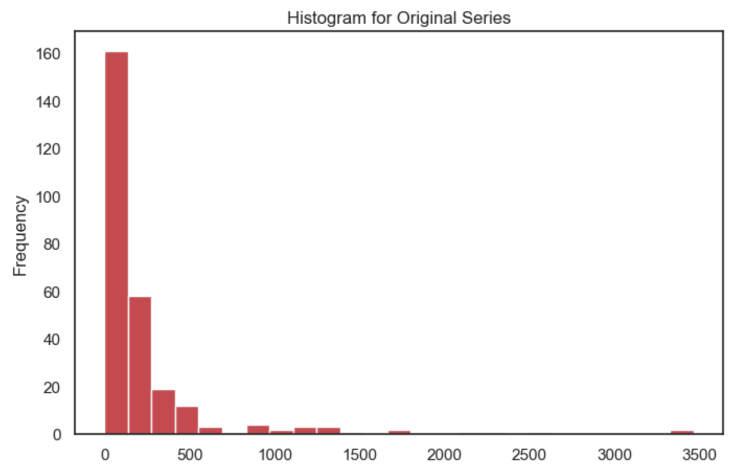
In [67]:
```
model_ts = auto_timeseries(
    score_type='rmse',        # Métrica de evaluación
    time_interval='D',        # Intervalo diario
    non_seasonal_pdq=None,    # Para modelos SARIMAX
    seasonality=False,        # Deshabilitar búsqueda de estacionalidad (se puede cambiar)
    model_type='best',        # Seleccionar el mejor modelo
    verbose=2                 # Nivel de verbosidad
)
# ['best', 'prophet', 'stats', 'ml', 'arima', 'ARIMA', 'Prophet', 'SARIMAX', 'VAR', 'ML']
```

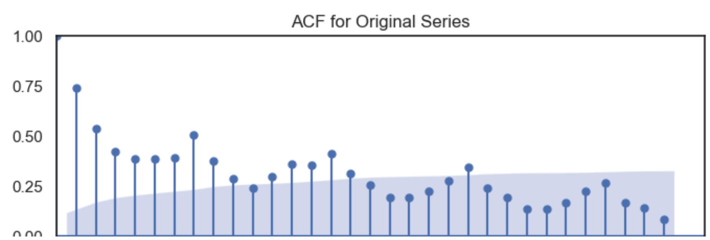
In [68]:
```
train_size = int(0.75 * len(df_ts))
train_df = df_ts[:train_size]
test_df = df_ts[train_size:]

model_ts.fit(
    traindata=train_df,
    ts_column=train_df.index.name,
    target='asistentes_totales'
)
```

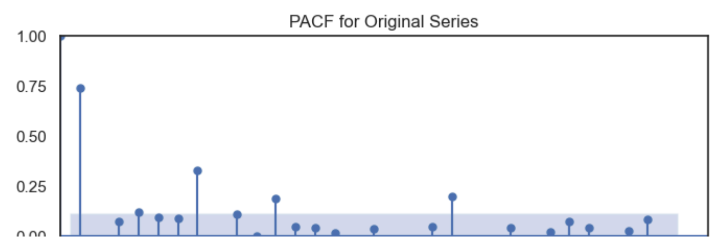
```
Start of Fit.....
    Target variable given as = asistentes_totales
Start of loading of data.....
    Inputs: ts_column = fecha, sep = ,, target = ['asistentes_totales']
    Using given input: pandas dataframe...
    train time series fecha column is the index on test data...
    train data shape = (274, 1)
Alert: Could not detect strf_time_format of fecha. Provide strf_time format during "setup" for better resul
ts.

Running Augmented Dickey-Fuller test with paramters:
    maxlag: 31 regression: c autolag: BIC
Data is stationary after one differencing
There is 1 differencing needed in this datasets for VAR model
```


Original Time Series after 1 differencing
3500
3000
2500
2000
1500
1000
500
0
Aug
Sep
Oct
Nov
Dec
Jan
2024
Feb
Mar
Apr
fecha


After Differencing = 1
1500
1000
500
0
−500
−1000
−1500
−2000
Aug
Sep
Oct
Nov
Dec
Jan
2024
Feb
Mar
Apr
fecha


Histogram for Original Series
Frequency
160
140
120
100
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500


ACF for Original Series
1.00
0.75
0.50
0.25


PACF for Original Series
1.00
0.75
0.50
0.25


```
18:37:56 - cmdstanpy - INFO - Chain [1] start processing
18:37:56 - cmdstanpy - INFO - Chain [1] done processing
18:37:56 - cmdstanpy - INFO - Chain [1] start processing
18:37:56 - cmdstanpy - INFO - Chain [1] done processing
18:37:56 - cmdstanpy - INFO - Chain [1] start processing
18:37:56 - cmdstanpy - INFO - Chain [1] done processing
```

```
Time Interval is given as D
    Correct Time interval given as a valid Pandas date-range frequency...
WARNING: Running best models will take time... Be Patient...


==================================================
Building Prophet Model
==================================================




Running Facebook Prophet Model...
kwargs for Prophet model: {'iter': 100}
    Fit-Predict data (shape=(274, 2)) with Confidence Interval = 0.95...
  Starting Prophet Fit
      No seasonality assumed since seasonality flag is set to False
  Starting Prophet Cross Validation
Max. iterations using expanding window cross validation = 5

Fold Number: 1 --> Train Shape: 249 Test Shape: 5
    RMSE = 740.77
    Std Deviation of actuals = 630.02
    Normalized RMSE (as pct of std dev) = 118%
Cross Validation window: 1 completed

Fold Number: 2 --> Train Shape: 254 Test Shape: 5
    RMSE = 708.64
    Std Deviation of actuals = 427.37
    Normalized RMSE (as pct of std dev) = 166%
Cross Validation window: 2 completed

Fold Number: 3 --> Train Shape: 259 Test Shape: 5
```

```
18:37:56 - cmdstanpy - INFO - Chain [1] start processing
18:37:56 - cmdstanpy - INFO - Chain [1] done processing
18:37:56 - cmdstanpy - INFO - Chain [1] start processing
18:37:56 - cmdstanpy - INFO - Chain [1] done processing
18:37:57 - cmdstanpy - INFO - Chain [1] start processing
```

```
    RMSE = 578.77
    Std Deviation of actuals = 189.00
    Normalized RMSE (as pct of std dev) = 306%
Cross Validation window: 3 completed

Fold Number: 4 --> Train Shape: 264 Test Shape: 5
    RMSE = 525.90
    Std Deviation of actuals = 490.48
    Normalized RMSE (as pct of std dev) = 107%
Cross Validation window: 4 completed

Fold Number: 5 --> Train Shape: 269 Test Shape: 5
    RMSE = 463.17
    Std Deviation of actuals = 417.79
    Normalized RMSE (as pct of std dev) = 111%
Cross Validation window: 5 completed

-------------------------------------------
Model Cross Validation Results:
-------------------------------------------
    MAE (Mean Absolute Error = 568.64
    MSE (Mean Squared Error = 375395.31
    MAPE (Mean Absolute Percent Error) = 213%
    RMSE (Root Mean Squared Error) = 612.6951
    Normalized RMSE (MinMax) = 38%
    Normalized RMSE (as Std Dev of Actuals)= 130%
Time Taken = 0 seconds
```

```
18:37:57 - cmdstanpy - INFO - Chain [1] done processing
```

```
  End of Prophet Fit


==================================================
Building Auto SARIMAX Model
==================================================
```

Running Auto SARIMAX Model...

```
Best Parameters:
p: None, d: None, q: None
P: None, D: None, Q: None
Seasonality: False
Seasonal Period: 12

Fold Number: 1 --> Train Shape: 249 Test Shape: 5
```

Finding the best parameters using AutoArima:

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3505.036, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3506.696, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3506.432, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3503.058, Time=0.00 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3477.543, Time=0.08 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3460.511, Time=0.13 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3495.165, Time=0.02 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3461.992, Time=0.09 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.12 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3464.354, Time=0.06 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=3487.182, Time=0.02 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.13 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3459.860, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3476.670, Time=0.02 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=3493.204, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3461.366, Time=0.05 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3461.123, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=3504.716, Time=0.00 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3463.611, Time=0.03 sec
 ARIMA(3,1,0)(0,0,0)[0]             : AIC=3485.250, Time=0.01 sec
```

```
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=inf, Time=0.09 sec

Best model:  ARIMA(2,1,1)(0,0,0)[0]
Total fit time: 1.019 seconds

Best model is a Seasonal SARIMAX(2,1,1)*(0,0,0,12), aic = 3459.860
Static Forecasts:
    RMSE = 900.42
    Std Deviation of Actuals = 630.02
    Normalized RMSE (as pct of std dev) = 142.9%

Fold Number: 2 --> Train Shape: 254 Test Shape: 5
```

Finding the best parameters using AutoArima:

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3609.980, Time=0.00 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3611.767, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3611.613, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3607.981, Time=0.00 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3580.161, Time=0.05 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3572.914, Time=0.08 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3603.706, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3574.407, Time=0.09 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=3570.447, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3574.640, Time=0.06 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=3576.257, Time=0.09 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=3571.330, Time=0.16 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=3571.839, Time=0.11 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.17 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3569.203, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3573.421, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3571.664, Time=0.06 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=3574.977, Time=0.06 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=3570.073, Time=0.12 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3578.951, Time=0.04 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=3570.630, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3573.125, Time=0.06 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=0.18 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 1.618 seconds
```

```
Best model is a Seasonal SARIMAX(2,1,2)*(0,0,0,12), aic = 3569.203
Static Forecasts:
    RMSE = 384.77
    Std Deviation of Actuals = 427.37
    Normalized RMSE (as pct of std dev) = 90.0%

Fold Number: 3 --> Train Shape: 259 Test Shape: 5
```

Finding the best parameters using AutoArima:

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3697.928, Time=0.00 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3699.020, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3698.303, Time=0.03 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3695.929, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3664.843, Time=0.05 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3661.168, Time=0.07 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3690.404, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3663.144, Time=0.10 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=3652.388, Time=0.08 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3661.707, Time=0.06 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=3653.783, Time=0.13 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=3662.861, Time=0.12 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=3666.814, Time=0.08 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3650.971, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3660.197, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3659.562, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=3652.305, Time=0.05 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=3661.031, Time=0.07 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3663.484, Time=0.03 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=3664.849, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3661.508, Time=0.05 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=0.16 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 1.467 seconds

Best model is a Seasonal SARIMAX(2,1,2)*(0,0,0,12), aic = 3650.971
Static Forecasts:
    RMSE = 307.90
    Std Deviation of Actuals = 189.00
    Normalized RMSE (as pct of std dev) = 162.9%
```

```
Fold Number: 4 --> Train Shape: 264 Test Shape: 5

    Finding the best parameters using AutoArima:
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3767.152, Time=0.00 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3768.190, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3767.485, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3765.164, Time=0.00 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3733.533, Time=0.06 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3729.997, Time=0.08 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3760.269, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3731.917, Time=0.13 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=3721.438, Time=0.08 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3730.574, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=3722.724, Time=0.13 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=3733.006, Time=0.13 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=3731.008, Time=0.11 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.16 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3719.943, Time=0.05 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3728.981, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3728.299, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=3721.126, Time=0.09 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=3731.178, Time=0.07 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3732.105, Time=0.03 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=3729.198, Time=0.05 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3730.163, Time=0.05 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=0.17 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 1.573 seconds

Best model is a Seasonal SARIMAX(2,1,2)*(0,0,0,12), aic = 3719.943
Static Forecasts:
    RMSE = 498.63
    Std Deviation of Actuals = 490.48
    Normalized RMSE (as pct of std dev) = 101.7%

Fold Number: 5 --> Train Shape: 269 Test Shape: 5

    Finding the best parameters using AutoArima:
Performing stepwise search to minimize aic
```

```
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3854.196, Time=0.00 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3854.723, Time=0.09 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3853.564, Time=0.01 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3852.197, Time=0.00 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3818.056, Time=0.05 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3814.489, Time=0.08 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3845.912, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3816.258, Time=0.10 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=3805.459, Time=0.10 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3815.209, Time=0.07 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=3807.081, Time=0.09 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=3815.387, Time=0.05 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=3818.595, Time=0.07 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.18 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3803.741, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3813.401, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3812.608, Time=0.04 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=3805.290, Time=0.05 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=3813.417, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3816.417, Time=0.03 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=3816.616, Time=0.05 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3814.341, Time=0.05 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=0.14 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 1.402 seconds

Best model is a Seasonal SARIMAX(2,1,2)*(0,0,0,12), aic = 3803.741
Static Forecasts:
    RMSE = 398.72
    Std Deviation of Actuals = 417.79
    Normalized RMSE (as pct of std dev) = 95.4%

SARIMAX RMSE (all folds): 498.0898
SARIMAX Norm RMSE (all folds): 109%


-------------------------------------------
Model Cross Validation Results:
-------------------------------------------
    MAE (Mean Absolute Error = 440.72
    MSE (Mean Squared Error = 292245.27
```

```
    MAPE (Mean Absolute Percent Error) = 156%
    RMSE (Root Mean Squared Error) = 540.5971
    Normalized RMSE (MinMax) = 33%
    Normalized RMSE (as Std Dev of Actuals)= 114%
```

Finding the best parameters using AutoArima:

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3942.589, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3941.383, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3938.930, Time=0.01 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3940.590, Time=0.00 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3902.167, Time=0.06 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3900.330, Time=0.08 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3933.300, Time=0.01 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=3901.899, Time=0.11 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=3888.545, Time=0.11 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3900.885, Time=0.06 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=3890.528, Time=0.11 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=3897.459, Time=0.08 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=3900.519, Time=0.07 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.20 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=3886.765, Time=0.07 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=3899.054, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=3898.451, Time=0.05 sec
 ARIMA(3,1,2)(0,0,0)[0]             : AIC=3888.731, Time=0.06 sec
 ARIMA(2,1,3)(0,0,0)[0]             : AIC=3895.545, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3900.452, Time=0.04 sec
 ARIMA(1,1,3)(0,0,0)[0]             : AIC=3898.562, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=3899.976, Time=0.05 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=inf, Time=0.16 sec

Best model:  ARIMA(2,1,2)(0,0,0)[0]
Total fit time: 1.469 seconds

Best model is a Seasonal SARIMAX(2,1,2)*(0,0,0,12), aic = 3886.765
```

Refitting data with previously found best parameters

```
    Best aic metric = 3848.1
                               SARIMAX Results
==============================================================================
Dep. Variable:     asistentes_totales   No. Observations:                  274
Model:               SARIMAX(2, 1, 2)   Log Likelihood               -1917.073
Date:                Mon, 21 Oct 2024   AIC                           3848.147
```

```
Time:                        18:38:05   BIC                           3873.335
Sample:                    08-01-2023   HQIC                          3858.261
                         - 04-30-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -4.5989      4.430     -1.038      0.299     -13.281       4.083
drift          0.0442      0.027      1.635      0.102      -0.009       0.097
ar.L1         -0.1695      0.059     -2.886      0.004      -0.285      -0.054
ar.L2          0.4547      0.050      9.042      0.000       0.356       0.553
ma.L1         -0.0609      0.129     -0.470      0.638      -0.314       0.193
ma.L2         -0.9377      0.116     -8.053      0.000      -1.166      -0.710
sigma2      8.805e+04   1.22e+04      7.211      0.000    6.41e+04    1.12e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):              1796.48
Prob(Q):                              0.87   Prob(JB):                         0.00
Heteroskedasticity (H):              19.06   Skew:                             1.25
Prob(H) (two-sided):                  0.00   Kurtosis:                        15.39
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).


==================================================
Building VAR Model - best suited for small datasets < 1000 rows and < 10 columns
==================================================
```

No VAR model created since no explanatory variables given in data set

```
==================================================
Building ML Model
==================================================
```

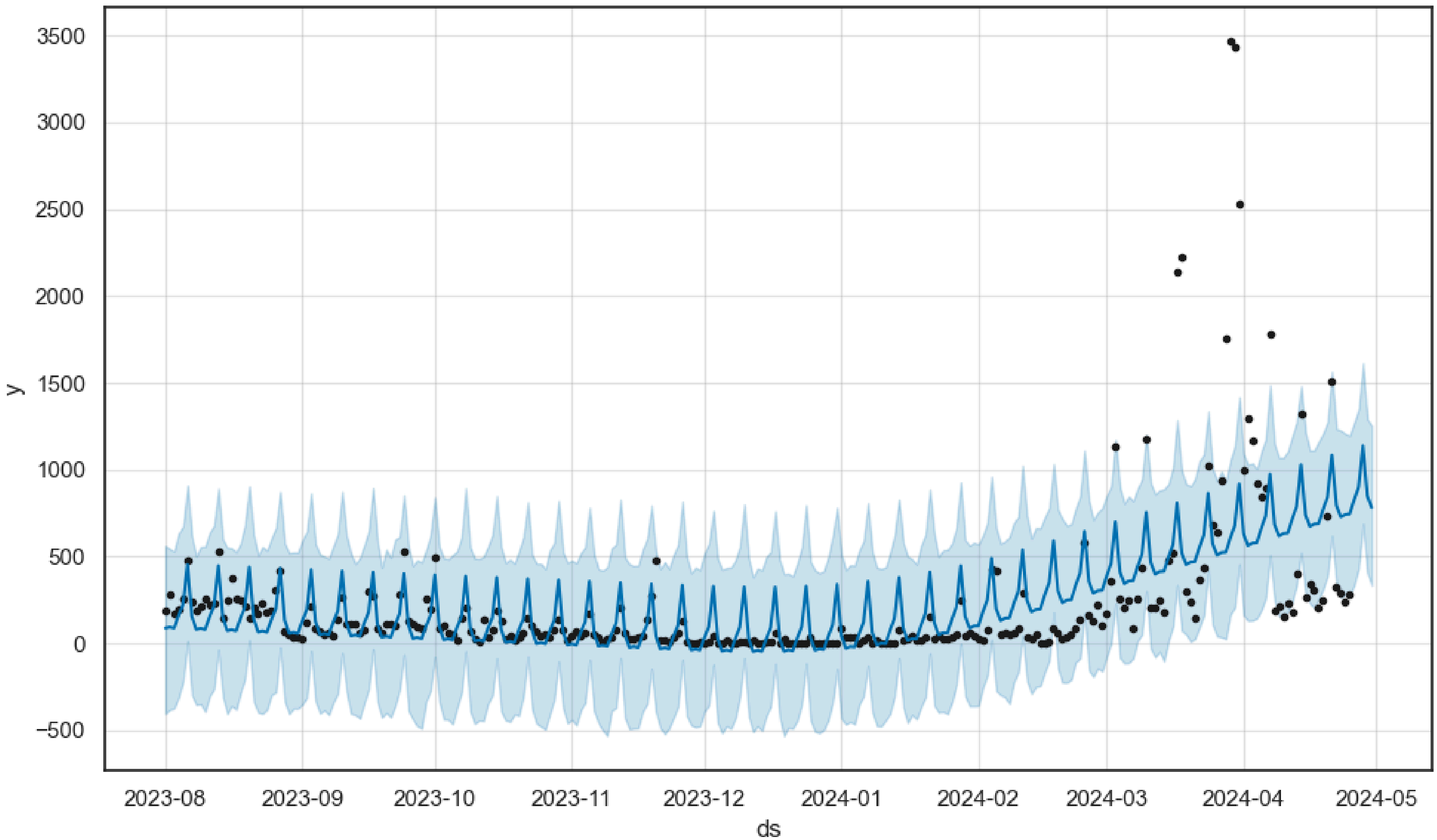
Creating 2 lagged variables for Machine Learning model...

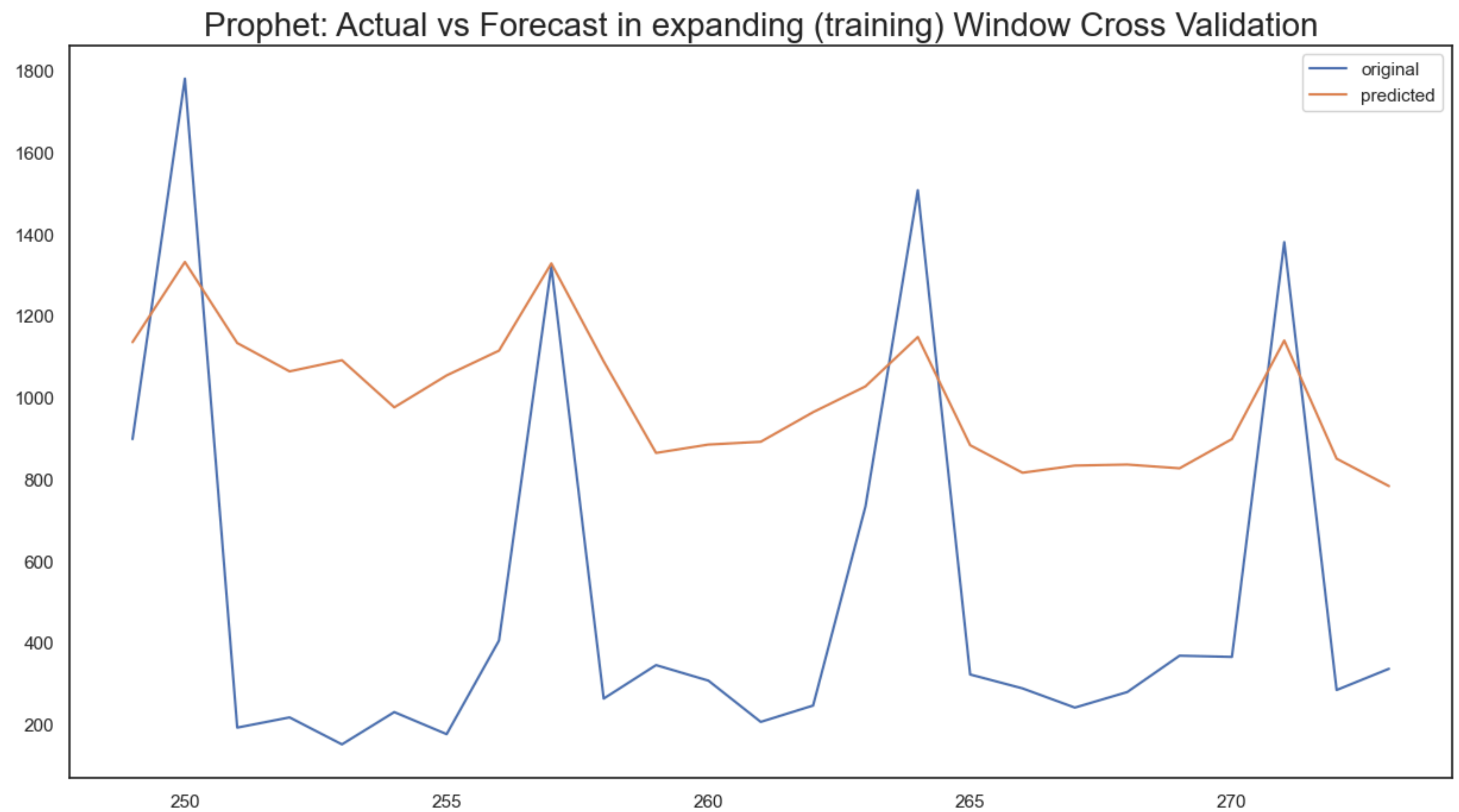
```
    You have set lag = 3 in auto_timeseries setup to feed prior targets. You cannot set lags > 10 ...
```

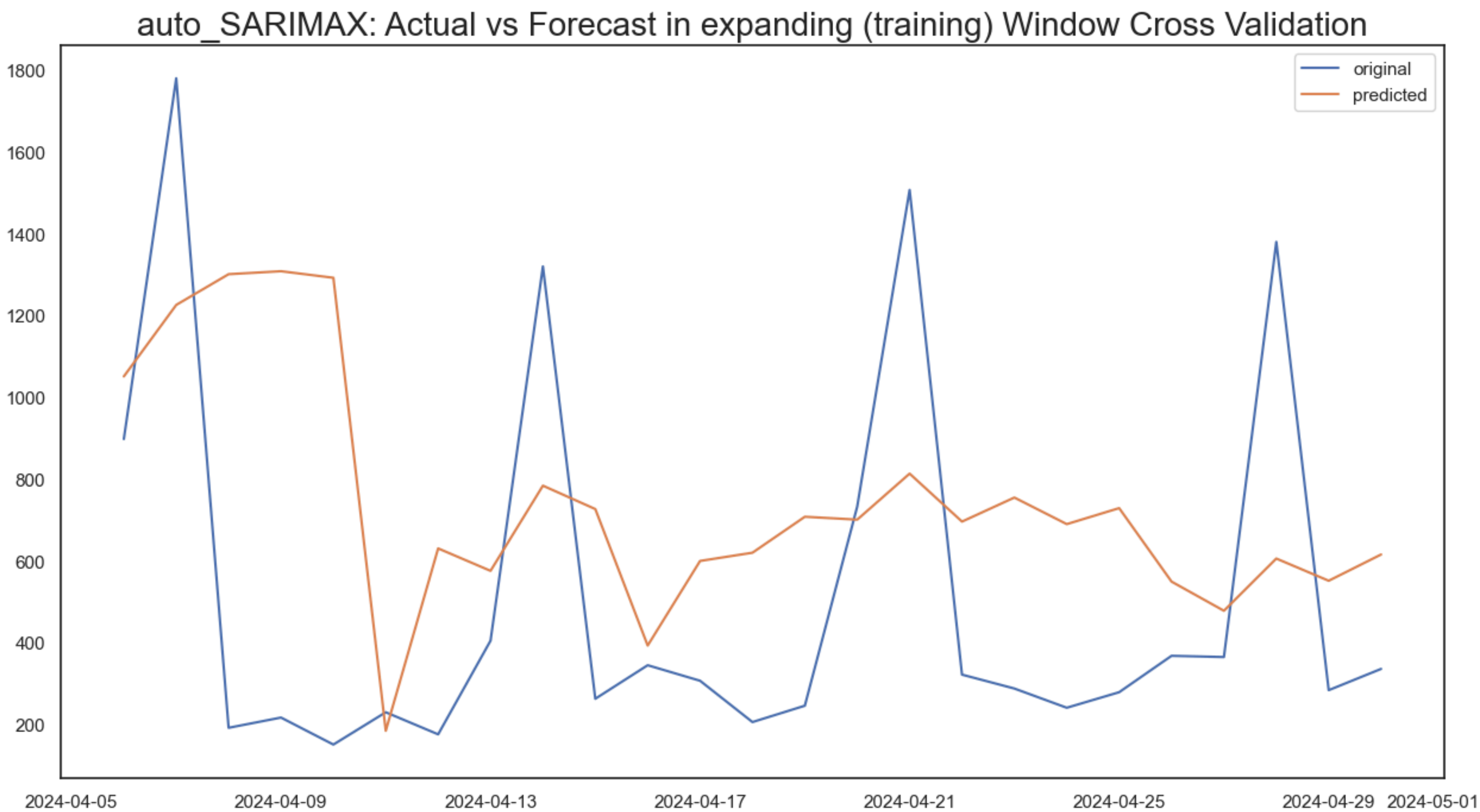
```
### Be careful setting dask_xgboost_flag to True since dask is unstable and doesn't work sometime's ###

########### Single-Label Regression Model Tuning and Training Started ####

Fitting ML model
    11 variables used in training ML model = ['asistentes_totales(t-1)', 'fecha_hour', 'fecha_minute', 'fec
ha_dayofweek', 'fecha_quarter', 'fecha_month', 'fecha_year', 'fecha_dayofyear', 'fecha_dayofmonth', 'fecha_
weekofyear', 'fecha_weekend']

Running Cross Validation using XGBoost model..
    Max. iterations using expanding window cross validation = 2
train fold shape (245, 11), test fold shape = (28, 11)
### Number of booster rounds = 250 for XGBoost which can be set during setup ####
    Hyper Param Tuning XGBoost with CPU parameters. This will take time. Please be patient...
Cross-validated Score = 238.844 in num rounds = 249
Time taken for Hyper Param tuning of XGBoost (in minutes) = 0.0
Top 10 features:
['fecha_year', 'asistentes_totales(t-1)', 'fecha_quarter', 'fecha_month', 'fecha_dayofyear', 'fecha_dayofwe
ek', 'fecha_dayofmonth', 'fecha_weekofyear', 'fecha_weekend']
    Time taken for training XGBoost on entire train data (in minutes) = 0.0
Returning the following:
    Model = <xgboost.core.Booster object at 0x156324a70>
    Scaler = Pipeline(steps=[('columntransformer',
                 ColumnTransformer(transformers=[('simpleimputer',
                                                  SimpleImputer(),
                                                  ['asistentes_totales(t-1)',
                                                   'fecha_hour', 'fecha_minute',
                                                   'fecha_dayofweek',
                                                   'fecha_quarter',
                                                   'fecha_month', 'fecha_year',
                                                   'fecha_dayofyear',
                                                   'fecha_dayofmonth',
                                                   'fecha_weekofyear',
                                                   'fecha_weekend'])])),
                ('maxabsscaler', MaxAbsScaler())])
    (3) sample predictions:[710.3362  747.2335  985.16724]
XGBoost model tuning completed
Target = asistentes_totales...CV results:
    RMSE = 434.64
    Std Deviation of actuals = 462.62
    Normalized RMSE (as pct of std dev) = 94%
```

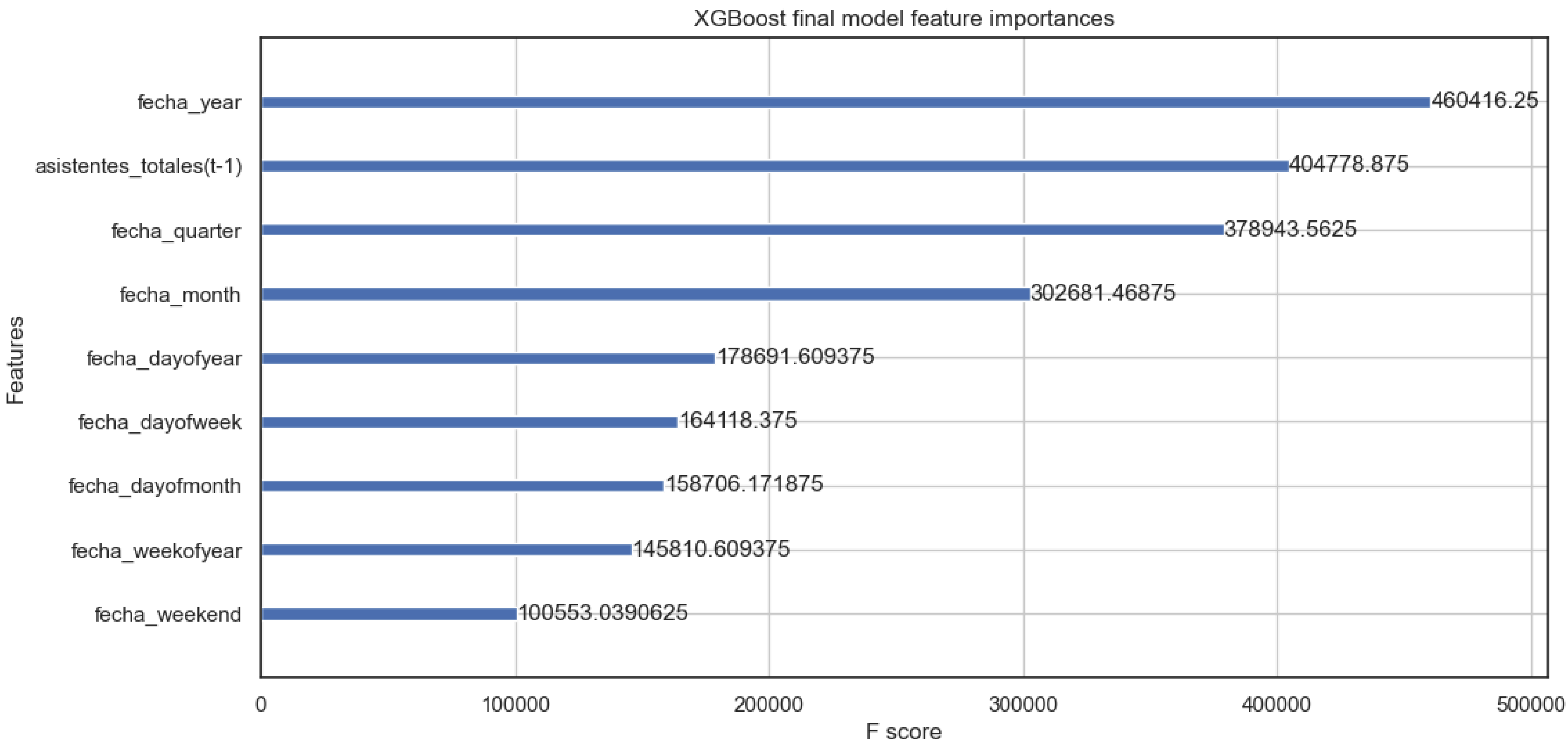
```
Fitting model on entire train set. Please be patient...
    Time taken to train model (in seconds) = 0

Best Model is: ML
    Best Model (Mean CV) Score: 434.64


--------------------------------------------------
Total time taken: 10 seconds.
--------------------------------------------------


Leaderboard with best model on top of list:
            name        rmse
3             ML  434.636833
1   auto_SARIMAX  498.089773
0        Prophet  603.449065
2            VAR         inf
```

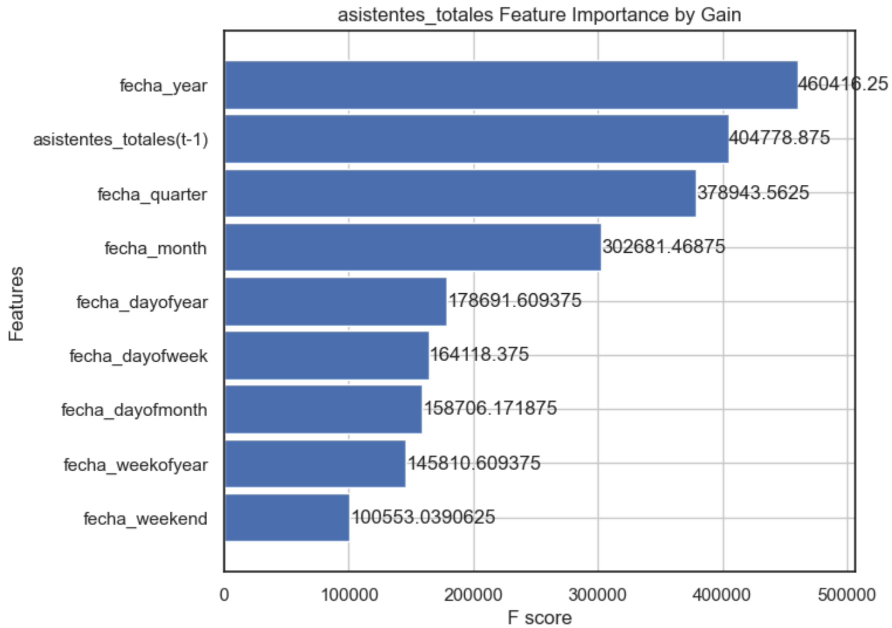
Out[68]:
```
<auto_ts.auto_timeseries at 0x151fb42f0>
```


3500
3000
2500
2000
1500
1000
500
0
−500
y
2023-08
2023-09
2023-10
2023-11
2023-12
2024-01
2024-02
2024-03
2024-04
2024-05
ds


Prophet: Actual vs Forecast in expanding (training) Window Cross Validation
original
predicted
1800
1600
1400
1200
1000
800
600
400
200
250
255
260
265
270


auto_SARIMAX: Actual vs Forecast in expanding (training) Window Cross Validation
original
predicted
1800
1600
1400
1200
1000
800
600
400
200
2024-04-05
2024-04-09
2024-04-13
2024-04-17
2024-04-21
2024-04-25
2024-04-29
2024-05-01


XGBoost final model feature importances
fecha_year
asistentes_totales(t-1)
fecha_quarter
fecha_month
fecha_dayofyear
fecha_dayofweek
fecha_dayofmonth
fecha_weekofyear
fecha_weekend
460416.25
404778.875
378943.5625
302681.46875
178691.609375
164118.375
158706.171875
145810.609375
100553.0390625
Features
F score
0
100000
200000
300000
400000
500000



Test

In [70]:
```
forecast = model_ts.predict(testdata=test_df)
forecast
```

```
Predicting using test dataframe shape = (92, 1) for ML model
For large datasets: ML predictions will take time since it has to predict each row and use that for future predictions...
    Using given input: pandas dataframe...
    Alert: No strf_time_format given for fecha. Provide strf_time format during "setup" for better results.
ML predictions completed
```

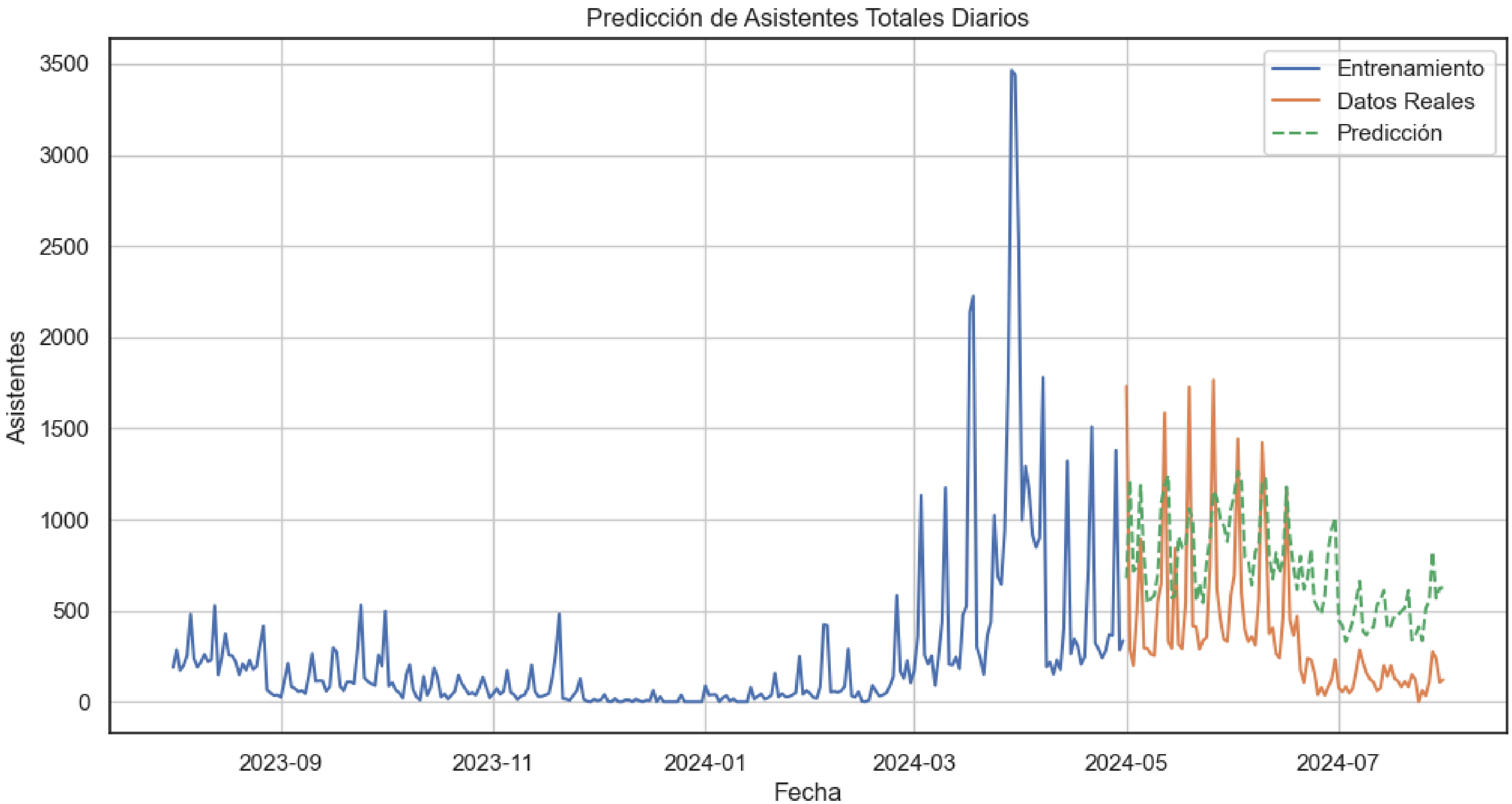
Out[70]:

	yhat	mean_se	mean_ci_lower	mean_ci_upper
fecha				
2024-05-01	675.981689	NaN	NaN	NaN
2024-05-02	1215.401978	NaN	NaN	NaN
2024-05-03	716.235901	NaN	NaN	NaN
2024-05-04	738.473511	NaN	NaN	NaN
2024-05-05	1187.199951	NaN	NaN	NaN
...	...	...	...	...
2024-07-27	552.684204	NaN	NaN	NaN
2024-07-28	826.076599	NaN	NaN	NaN
2024-07-29	568.383362	NaN	NaN	NaN
2024-07-30	621.908508	NaN	NaN	NaN
2024-07-31	626.753418	NaN	NaN	NaN

92 rows × 4 columns

In [71]:

```
plt.figure(figsize=(12,6))
plt.plot(train_df.index, train_df['asistentes_totales'], label='Entrenamiento')
plt.plot(test_df.index, test_df['asistentes_totales'], label='Datos Reales')
plt.plot(test_df.index, forecast['yhat'], label='Predicción', linestyle='--')
plt.title('Predicción de Asistentes Totales Diarios')
plt.xlabel('Fecha')
plt.ylabel('Asistentes')
plt.legend()
plt.grid(True)
plt.show()

rmse = mean_squared_error(test_df['asistentes_totales'], forecast['yhat'], squared=False)
print(f'RMSE en el conjunto de prueba: {rmse:.2f}')
```



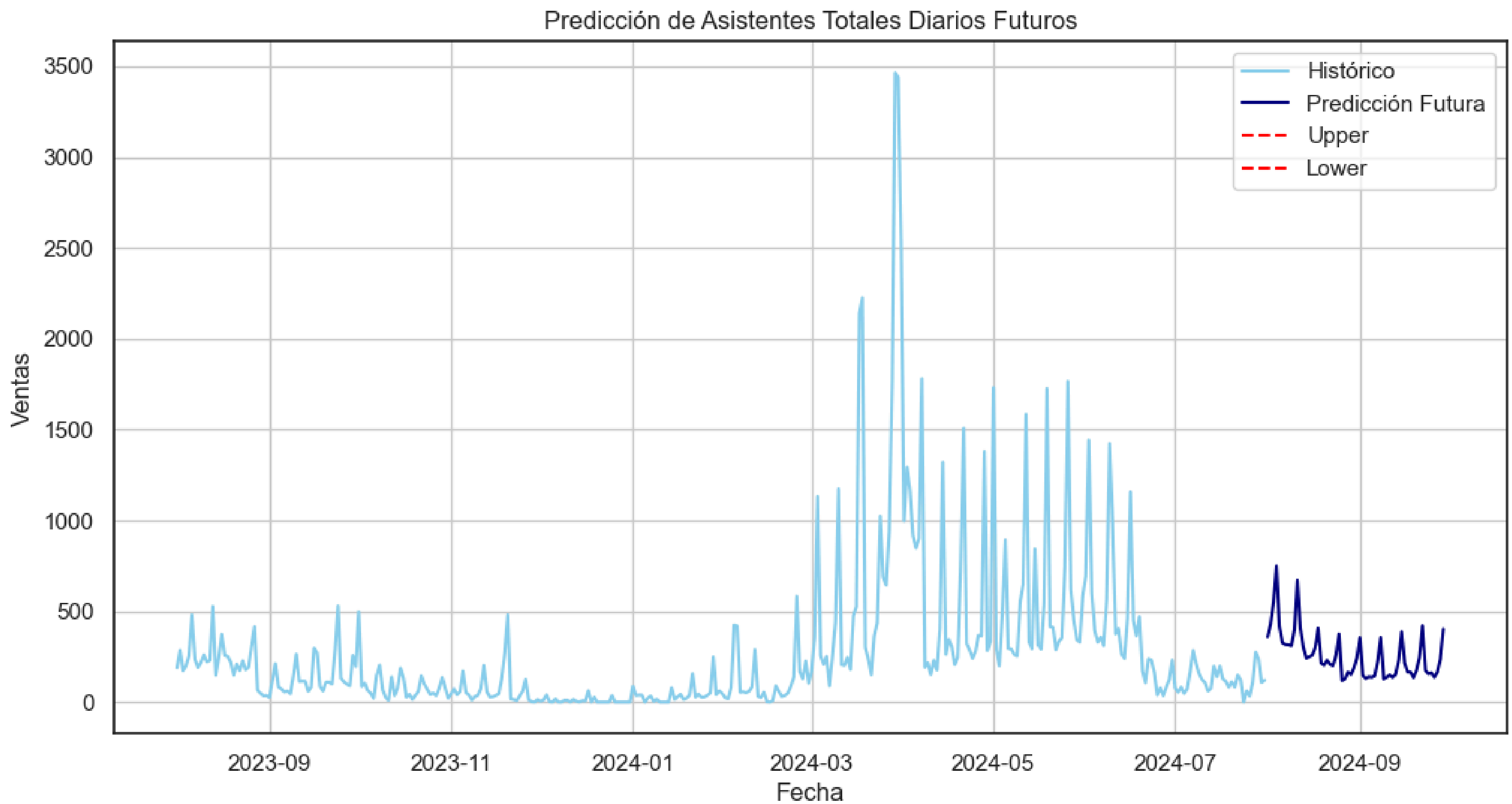

```
RMSE en el conjunto de prueba: 455.28
```

Predicciones Futuras

In [73]:
```
future_periods = 60
last_date = df_ts.index[-1]
future_dates = pd.date_range(start=last_date + pd.DateOffset(days=1), periods=future_periods, freq='D')

future_df = pd.DataFrame(index=future_dates)
future_forecast = model_ts.predict(testdata=future_df)

plt.figure(figsize=(12,6))
plt.plot(df_ts.index, df_ts['asistentes_totales'], label='Histórico', c = 'skyblue')
plt.plot(future_dates, future_forecast['yhat'], label='Predicción Futura', c = 'navy')
```

```python
if len(future_forecast.columns) == 16:
    plt.plot(future_dates, future_forecast['yhat_upper'], label='Upper', linestyle='--', c = 'red')
    plt.plot(future_dates, future_forecast['yhat_lower'], label='Lower', linestyle='--', c = 'red')
    plt.fill_between(future_dates, future_forecast['yhat'], future_forecast['yhat_upper'], color='lightblue
    plt.fill_between(future_dates, future_forecast['yhat'], future_forecast['yhat_lower'], color='lightblue

if len(future_forecast.columns) == 4:
    plt.plot(future_dates, future_forecast['mean_ci_upper'], label='Upper', linestyle='--', c = 'red')
    plt.plot(future_dates, future_forecast['mean_ci_lower'], label='Lower', linestyle='--', c = 'red')

    plt.fill_between(future_dates, future_forecast['yhat'], future_forecast['mean_ci_upper'], color='lightb
    plt.fill_between(future_dates, future_forecast['yhat'], future_forecast['mean_ci_lower'], color='lightb

plt.title('Predicción de Asistentes Totales Diarios Futuros')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicting using test dataframe shape = (60, 0) for ML model
For large datasets: ML predictions will take time since it has to predict each row and use that for future
predictions...
    Using given input: pandas dataframe...
    Alert: No strf_time_format given for fecha. Provide strf_time format during "setup" for better results.
    converting testdata to datetime index erroring. Please check input and try again.
ML predictions completed
```

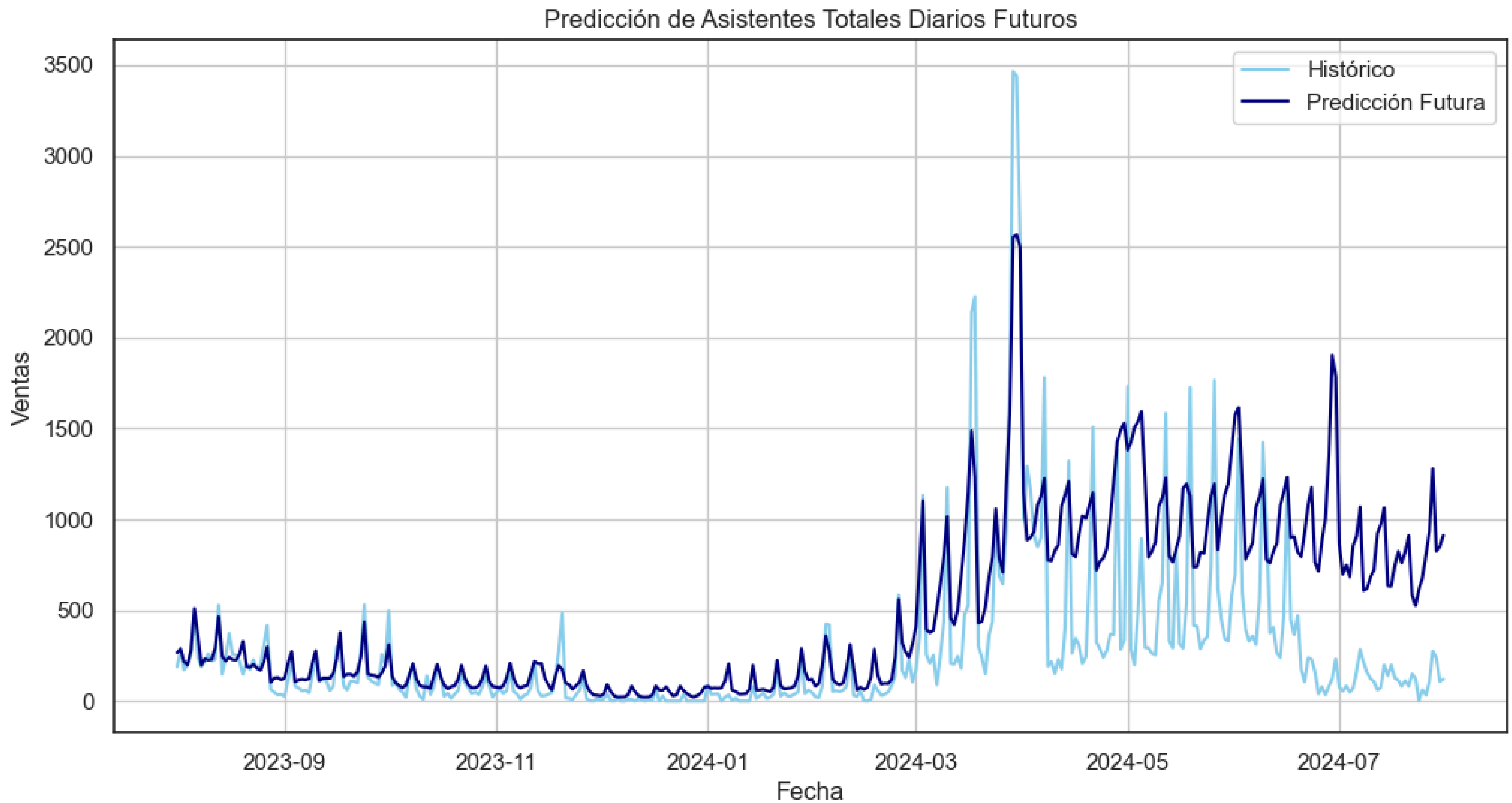


Forecast del pasado

In [75]:
```
future_periods = 60
last_date = df_ts.index[-1]
future_dates = df_ts.index

future_df = pd.DataFrame(index=future_dates)
past_forecast = model_ts.predict(testdata=future_df)

plt.figure(figsize=(12,6))
plt.plot(df_ts.index, df_ts['asistentes_totales'], label='Histórico', c = 'skyblue')
plt.plot(future_dates, past_forecast['yhat'], label='Predicción Futura', c = 'navy')
```

```
plt.title('Predicción de Asistentes Totales Diarios Futuros')
plt.xlabel('Fecha')
plt.ylabel('Ventas')
plt.legend()
plt.grid(True)
plt.show()
```

```
Predicting using test dataframe shape = (366, 0) for ML model
For large datasets: ML predictions will take time since it has to predict each row and use that for future
predictions...
    Using given input: pandas dataframe...
    Alert: No strf_time_format given for fecha. Provide strf_time format during "setup" for better results.
ML predictions completed
```



Transformación de datos a proporciones

Se hace con real/prediccion

In [77]:
```
df_proporciones = pd.DataFrame()
df_proporciones['proporcion_niños'] = df['cantidad_niños'] / df['asistentes_totales']
df_proporciones['proporcion_adultos'] = df['cantidad_adultos'] / df['asistentes_totales']
df_proporciones['proporcion_alimento'] = df['alimento'] / df['ingreso_total']
df_proporciones['proporcion_extras'] = df['extras'] / df['ingreso_total']
df_proporciones['proporcion_total'] = (df_ts['asistentes_totales'] / past_forecast['yhat']).reset_index().
df_proporciones.head()
```

Out[77]:

	proporcion_niños	proporcion_adultos	proporcion_alimento	proporcion_extras	proporcion_total
0	0.206349	0.433862	0.230317	0.060398	0.711224
1	0.192308	0.534965	0.248808	0.060527	0.992992
2	0.197674	0.569767	0.289409	0.054595	0.791973
3	0.276382	0.552764	0.233494	0.062997	1.002248
4	0.149020	0.592157	0.140576	0.053917	0.915151

Método SEM

Variable latente: Satisfacción general por día.

Varoables observables:

- temperatura_promedio
- proporcion_adultos
- proporcion_niños
- proporcion_extras
- proporcion_total
- is_wknd
- is_hs

Si el p-value de alguna variable excede el 5%, esta se remueve del modelo y se repite el proceso hasta no tener variables con p-value mayor al 5%.

Definición del modelo

In [81]:
```
# Definir el modelo SEM usando la notación estándar de SEM
model_desc = """
# Latent Variables
Satisfaccion =~ temperatura_promedio + proporcion_adultos + proporcion_alimento + proporcion_extras + propo
"""
```

In [82]:
```
df_SEM = df_proporciones
df_SEM['temperatura_promedio'] = df['temperatura_promedio']
df_SEM.head()
```

Out[82]:

	proporcion_niños	proporcion_adultos	proporcion_alimento	proporcion_extras	proporcion_total	temperatura_promedi
0	0.206349	0.433862	0.230317	0.060398	0.711224	2
1	0.192308	0.534965	0.248808	0.060527	0.992992	2
2	0.197674	0.569767	0.289409	0.054595	0.791973	2
3	0.276382	0.552764	0.233494	0.062997	1.002248	2
4	0.149020	0.592157	0.140576	0.053917	0.915151	2

Ajuste del modelo

In [84]:
```
numerical_features_SEM = ['temperatura_promedio','proporcion_adultos','proporcion_alimento','proporcion_ex

data_to_model_standarized_SEM = StandardScaler().fit_transform(df_SEM[numerical_features_SEM])

data_to_model_SEM = pd.DataFrame(data_to_model_standarized_SEM,
                                 columns=numerical_features_SEM)

data_to_model_SEM['is_hs'] = df['is_hs']
```

In [85]:
```
mod = Model(model_desc)
res_opt = mod.fit(data_to_model_SEM)
estimates = mod.inspect()

# Imprimir los resultados del ajuste del modelo
estimates
```

Out[85]:

	lval	op	rval	Estimate	Std. Err	z-value	p-value
0	temperatura_promedio	~	Satisfaccion	1.000000	-	-	-
1	proporcion_adultos	~	Satisfaccion	-0.204914	0.07671	-2.671273	0.007556
2	proporcion_alimento	~	Satisfaccion	-0.341163	0.089019	-3.832485	0.000127
3	proporcion_extras	~	Satisfaccion	-0.146549	0.07312	-2.004244	0.045044
4	proporcion_total	~	Satisfaccion	0.301936	0.084942	3.554615	0.000379
5	is_hs	~	Satisfaccion	0.256041	0.054621	4.687599	0.000003
6	Satisfaccion	~~	Satisfaccion	0.738407	0.161676	4.5672	0.000005
7	is_hs	~~	is_hs	0.139787	0.014178	9.859128	0.0
8	proporcion_adultos	~~	proporcion_adultos	0.968885	0.072434	13.37616	0.0
9	proporcion_alimento	~~	proporcion_alimento	0.913817	0.070577	12.947835	0.0
10	proporcion_extras	~~	proporcion_extras	0.983976	0.073118	13.457323	0.0
11	proporcion_total	~~	proporcion_total	0.932448	0.071078	13.118662	0.0
12	temperatura_promedio	~~	temperatura_promedio	0.261667	0.146363	1.78779	0.07381

Resultados del modelo

In [87]:
```
# Cargar factores estimados (cargas factoriales)
confianza_factors = [1.0, -0.204914, -0.341163, -0.146549, 0.301936, 0.256041]  #temperatura, proporcion_a
#proporcion_total, is_hs

# Función para calcular el valor de la variable latente
def calculate_latent_values(df):
```

```python
    # Calcular Satisfaccion
    df['satisfaccion'] = (df_SEM['temperatura_promedio'] * confianza_factors[0] +
                          df_proporciones['proporcion_adultos'] * confianza_factors[1] +
                          df_proporciones['proporcion_alimento'] * confianza_factors[2] +
                          df_proporciones['proporcion_extras'] * confianza_factors[3] +
                          df_proporciones['proporcion_total'] * confianza_factors[4] +
                          df['is_hs'] * confianza_factors[5])

    return df[['satisfaccion']]

# Calcular valores latentes en el dataset
latent_values = calculate_latent_values(df)

# Mostrar los primeros valores calculados
latent_values['fecha'] = df['fecha']
latent_values.head()
```

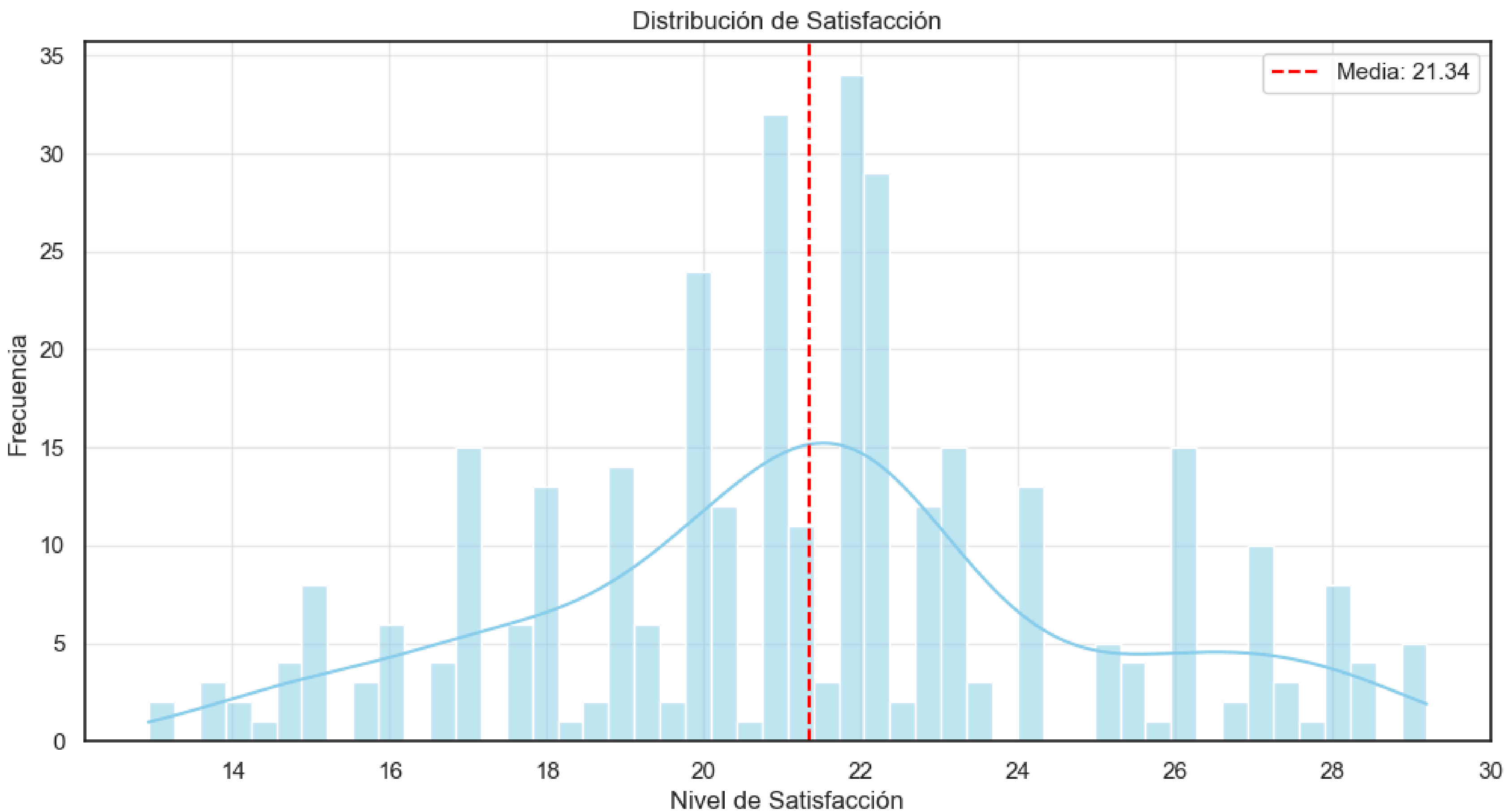
Out[87]:

	satisfaccion	fecha
0	22.038413	2023-08-01
1	24.096444	2023-08-02
2	22.015635	2023-08-03
3	21.100454	2023-08-04
4	22.099115	2023-08-05

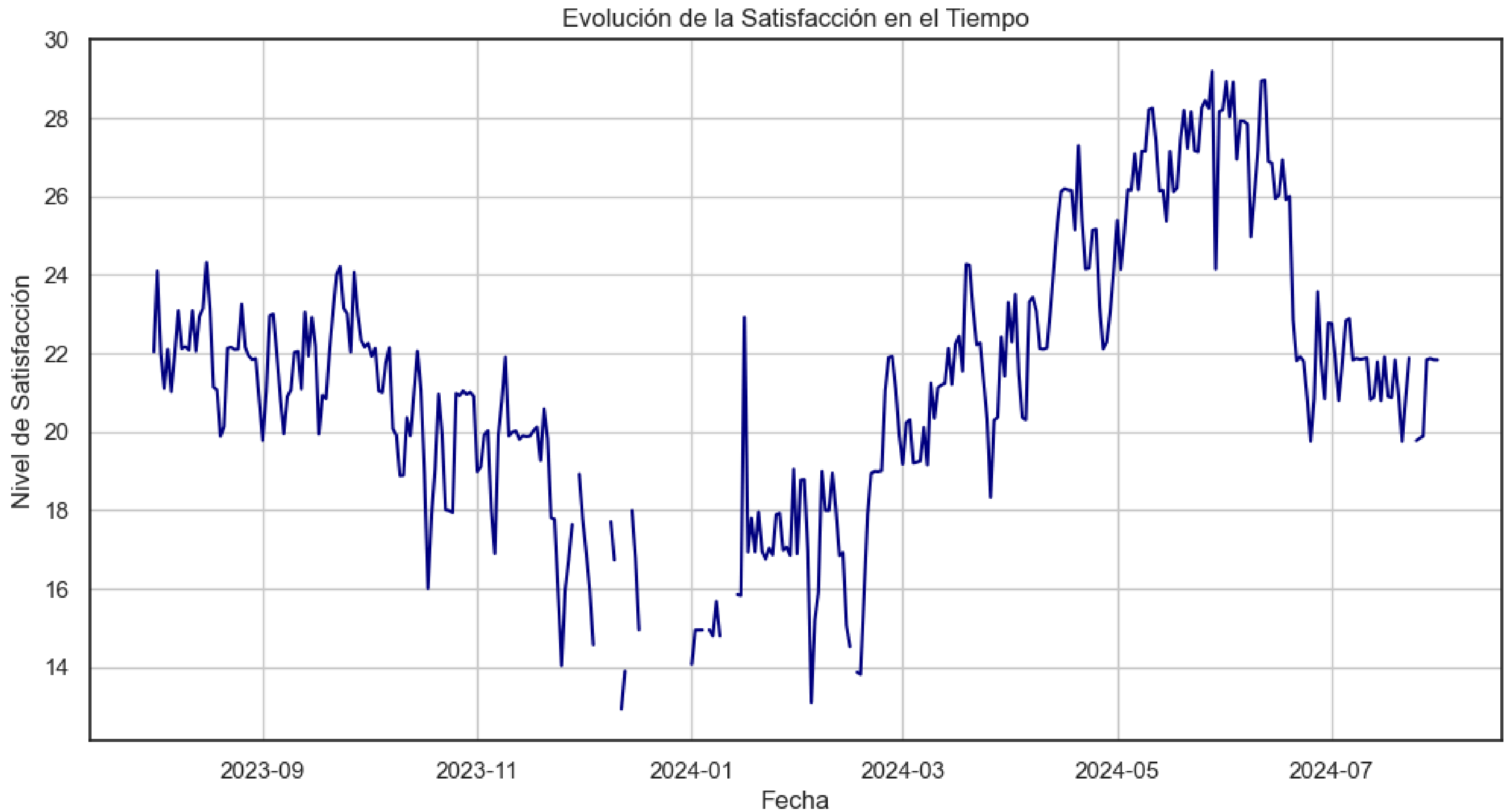
In [88]:

```python
sns.histplot(
    data=latent_values['satisfaccion'],
    bins=50,
    kde=True,  # Añade la curva de densidad
    color='skyblue'
)

sat_mean = latent_values['satisfaccion'].mean()

plt.title('Distribución de Satisfacción')
plt.xlabel('Nivel de Satisfacción')
plt.ylabel('Frecuencia')
```

```
plt.axvline(sat_mean, color='red', linestyle='--', label=f'Media: {sat_mean:.2f}')
plt.legend()

plt.grid(True, alpha=0.4)
plt.show()
```



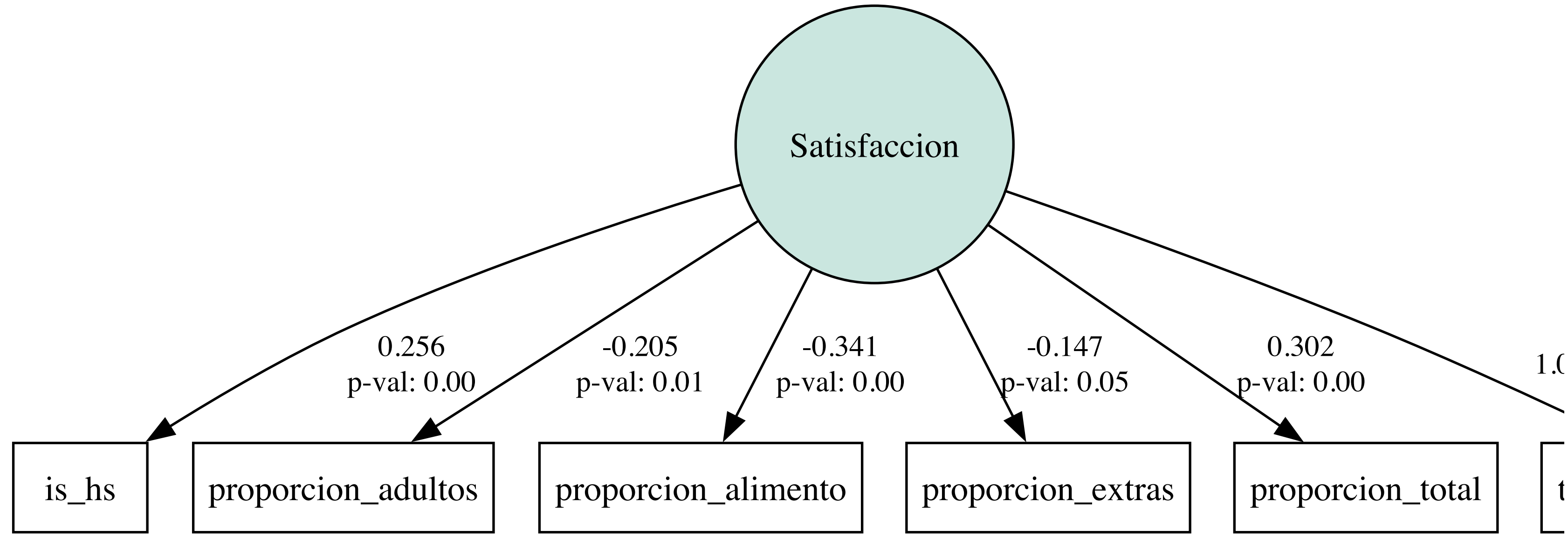
In [89]:
```
plt.plot(latent_values['fecha'],latent_values['satisfaccion'], c = 'navy')

plt.title('Evolución de la Satisfacción en el Tiempo')
plt.xlabel('Fecha')
plt.ylabel('Nivel de Satisfacción')
plt.grid()
```



Visualización del modelo

In [91]:
```
sm = semplot(mod, 'modelo_sem.png', engine = 'dot')
sm
```

Out[91]:



Análisis del modelo

Las variables seleccionadas, tras eliminar aquellas cuyo valor de p-value superaba 0.05, fueron: la temperatura promedio del día, la proporción de adultos respecto al total de visitantes, la proporción de compras adicionales en relación con los ingresos totales, la proporción de visitantes reales en comparación con los esperados y si era temporada alta o no. De estas, se tomó la temperatura promedio como la variable de referencia, asignándole un valor de 1, a partir del cual se compararon las demás.

Las tres variables que más influyeron en la satisfacción general diaria fueron: la temperatura promedio (valor de 1), seguida por la proporción total de asistentes reales contra los esperados(valor de 0.30) y, por último, si era temporada alta (valor de 0.26). También podemos notar que la proporción de adultos, de alimentos y de consumos extra tienen coeficiente negativo, por lo que si esta proporción crece, la satisfacción del día disminuye.

El impacto de estas variables en la satisfacción general se interpreta en función de cuánto afecta la variación de cada una en la ecuación final del modelo. Es decir, un cambio en alguna de estas variables tiene un efecto proporcional en la satisfacción general, según los valores estimados.

Además en la gráfica de la satisfacción en el tiempo se pueden apreciar huecos, estos correponden a los días que el parque recibió 0 visitantes, por lo que el modelo no puede calcular la satisfacción los días que no va nadie, lo cual hace sentido pues si no van personas al parque no hay una satisfacción que medir.

In [94]:
```
end_time = time.perf_counter()
execution_time = end_time - start_time
execution_time
```

Out[94]:
```
17.895308042003307
```

Created with Jupyter by Luis Márquez, Ana Sofía Hinojosa, and Ivanna Herrera