Histgradientboosting

March 15, 2024

[ ]:
```
# Libraries
#␣
 ↪============================================================================================
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-darkgrid')
from statsmodels.graphics.tsaplots import plot_acf

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import  HistGradientBoostingRegressor
from lightgbm import LGBMRegressor

from skforecast.ForecasterAutoregMultiSeries import ForecasterAutoregMultiSeries
from skforecast.ForecasterAutoreg import ForecasterAutoreg
from skforecast.model_selection import backtesting_forecaster
from skforecast.model_selection import grid_search_forecaster
from skforecast.model_selection_multiseries import␣
 ↪backtesting_forecaster_multiseries
from skforecast.model_selection_multiseries import␣
 ↪grid_search_forecaster_multiseries
```

1 Se transforman los datos, se crean variables exogenas y se coloca la columna datetime como indice

[ ]:
```
df = pd.read_csv("Final-db.csv")
# Eliminar los espacios adicionales en las fechas
df['date'] = df['date'].str.strip()

# Mapeo de los nombres de los meses en español a los nombres en inglés
meses = {
    'ene': 'Jan',
    'feb': 'Feb',
    'mar': 'Mar',
    'abr': 'Apr',
    'may': 'May',
```

```
    'jun': 'Jun',
    'jul': 'Jul',
    'ago': 'Aug',
    'sep': 'Sep',
    'oct': 'Oct',
    'nov': 'Nov',
    'dic': 'Dec'
}

# Función para convertir los nombres de los meses en español a inglés
def convertir_meses(fecha):
    for mes_es, mes_en in meses.items():
        fecha = fecha.replace(mes_es, mes_en)
    return fecha

# Aplicar la función a la columna de fecha
df['date'] = df['date'].apply(convertir_meses)

# Convertir la columna de fecha a datetime
df['date'] = pd.to_datetime(df['date'], format='%d %b %Y')
```

[ ]:
```
df = df.drop(columns=['Unnamed: 0',"index","Total libre de impuestos","Indefinido total $","Indefinido ctdad"])
df = df.rename(columns={"date":"Fecha","Encoded Products":"Producto"})
df['Fecha'] = pd.to_datetime(df['Fecha'], format="mixed")
df.columns = df.columns.str.replace("total $", "Precio por unidad")
df.columns = df.columns.str.replace("ctdad", "Cantidad")
def div(numerator, denominator):
  return lambda row: 0.0 if row[denominator] == 0 else float(row[numerator]/row[denominator])
for i in range(2, len(df.columns)-1,2):
    df[df.columns[i]] = df.apply(div(df.columns[i], df.columns[i+1]), axis=1)
df = df.set_index('Fecha')
```

2 Se eliminan las columnas de más, de igual manera se eliminan los productos con menos de 50 ventas en 2023

[ ]:
```
data = pd.DataFrame()
for i in df["Producto"].unique():
        x =pd.DataFrame(df[df["Producto"]==i].loc[:].groupby("Fecha").sum()["Ctdad Ordenada"].asfreq("D", fill_value=0)).rename(columns={"Ctdad Ordenada":i})
        data = pd.concat([data,x], axis=1)
data.fillna(0, inplace=True)
exog = pd.DataFrame()
exog["Mes"] = data.index.month
```

```
data["Dia"] = data.index.day
exog["Dia de la semana"] = data.index.dayofweek
exog.index = data.index
exog = pd.get_dummies(exog, columns=["Mes","Dia de la semana"],dtype=int)
exog = pd.concat([exog, data["Dia"]], axis=1)
exog
exog["Dia"].replace(to_replace=[13,14,15,16,17,18,28,29,30,31,1,2], value=1,␣
 ↪inplace=True)
exog["Dia"].
 ↪replace(to_replace=[3,4,5,6,7,8,9,10,11,12,19,20,21,22,23,24,25,26,27,28],␣
 ↪value=0, inplace=True)
for i in data.columns:
    if data.loc["2023-01-01":,i].sum() < 50:
        data = data.drop(columns=i,axis=1)
data.drop(columns="Dia", inplace=True)
data = data[data.sum().sort_values(ascending=False).index[0:20]]
data.head()
```

[ ]:

Fecha	Producto 273	Producto 0	Producto 1	Producto 5	Producto 8
2022-01-02	0.0	10.5	35.0	14.0	21.0
2022-01-03	0.0	38.5	17.5	21.0	0.0
2022-01-04	0.0	56.0	3.5	21.0	10.5
2022-01-05	0.0	49.0	14.0	10.5	10.5
2022-01-06	0.0	17.5	7.0	10.5	3.5

Fecha	Producto 21	Producto 12	Producto 22	Producto 186	Producto 20
2022-01-02	7.0	7.0	0.0	0.0	3.5
2022-01-03	7.0	10.5	3.5	0.0	0.0
2022-01-04	3.5	3.5	7.0	0.0	3.5
2022-01-05	0.0	7.0	3.5	0.0	0.0
2022-01-06	3.5	7.0	0.0	0.0	0.0

Fecha	Producto 33	Producto 245	Producto 16	Producto 17	Producto 38
2022-01-02	0.0	0.0	3.5	0.0	0.0
2022-01-03	0.0	0.0	3.5	0.0	0.0
2022-01-04	7.0	0.0	3.5	0.0	0.0
2022-01-05	0.0	0.0	3.5	3.5	0.0
2022-01-06	0.0	0.0	7.0	3.5	0.0

Fecha	Producto 134	Producto 37	Producto 59	Producto 248	Producto 122
2022-01-02	0.0	7.0	0.0	0.0	0.0
2022-01-03	0.0	0.0	3.5	0.0	0.0
2022-01-04	0.0	3.5	0.0	0.0	0.0

```
2022-01-05           0.0          0.0          0.0          0.0          0.0
2022-01-06           0.0          0.0          3.5          0.0          0.0
```

[ ]:
```
exog.head()
```

[ ]:
```
            Mes_1  Mes_2  Mes_3  Mes_4  Mes_5  Mes_6  Mes_7  Mes_8  Mes_9  \
Fecha
2022-01-02      1      0      0      0      0      0      0      0      0
2022-01-03      1      0      0      0      0      0      0      0      0
2022-01-04      1      0      0      0      0      0      0      0      0
2022-01-05      1      0      0      0      0      0      0      0      0
2022-01-06      1      0      0      0      0      0      0      0      0

            Mes_10  Mes_11  Mes_12  Dia de la semana_0  Dia de la semana_1  \
Fecha
2022-01-02       0       0       0                   0                   0
2022-01-03       0       0       0                   1                   0
2022-01-04       0       0       0                   0                   1
2022-01-05       0       0       0                   0                   0
2022-01-06       0       0       0                   0                   0

            Dia de la semana_2  Dia de la semana_3  Dia de la semana_4  \
Fecha
2022-01-02                   0                   0                   0
2022-01-03                   0                   0                   0
2022-01-04                   0                   0                   0
2022-01-05                   1                   0                   0
2022-01-06                   0                   1                   0

            Dia de la semana_5  Dia de la semana_6  Dia
Fecha
2022-01-02                   0                   1    1
2022-01-03                   0                   0    0
2022-01-04                   0                   0    0
2022-01-05                   0                   0    0
2022-01-06                   0                   0    0
```

3 Se divide el test en validacion, test y entrenamiento

[ ]:
```
# Split data into train-validation-test
#␣
  ↪==============================================================================
end_train = '2023-10-30'
end_val = '2023-11-30'

data_train = data.loc[:end_train, :].copy()
```

```
data_val   = data.loc[end_train:end_val, :].copy()
data_test  = data.loc[end_val:, :].copy()
exog_train = exog.loc[:end_train, :].copy()
exog_val   = exog.loc[end_train:end_val, :].copy()
exog_test  = exog.loc[end_val:, :].copy()

print(f"Train dates      : {data_train.index.min()} --- {data_train.index.
 ↪max()}  (n={len(data_train)})")
print(f"Validation dates : {data_val.index.min()} --- {data_val.index.max()} ␣
 ↪(n={len(data_val)})")
print(f"Test dates       : {data_test.index.min()} --- {data_test.index.max()} ␣
 ↪(n={len(data_test)})")
```

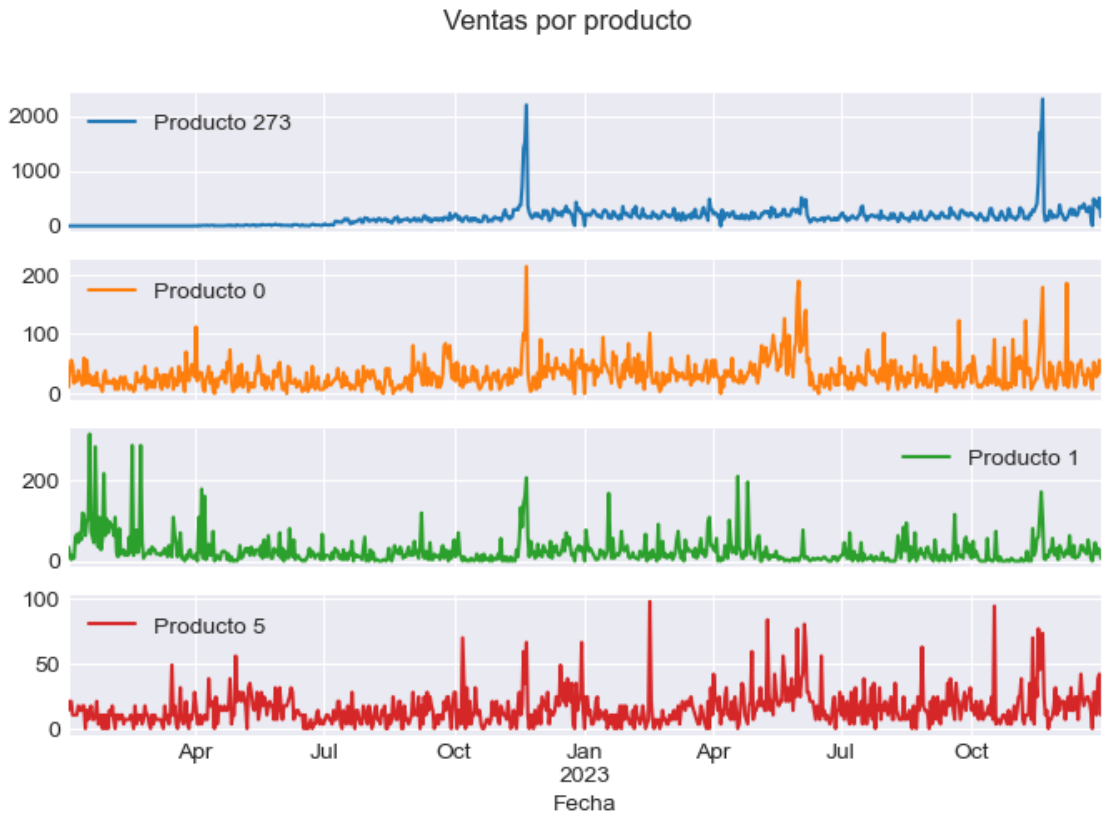
```
Train dates      : 2022-01-02 00:00:00 --- 2023-10-30 00:00:00  (n=667)
Validation dates : 2023-10-30 00:00:00 --- 2023-11-30 00:00:00  (n=32)
Test dates       : 2023-11-30 00:00:00 --- 2023-12-31 00:00:00  (n=32)
```

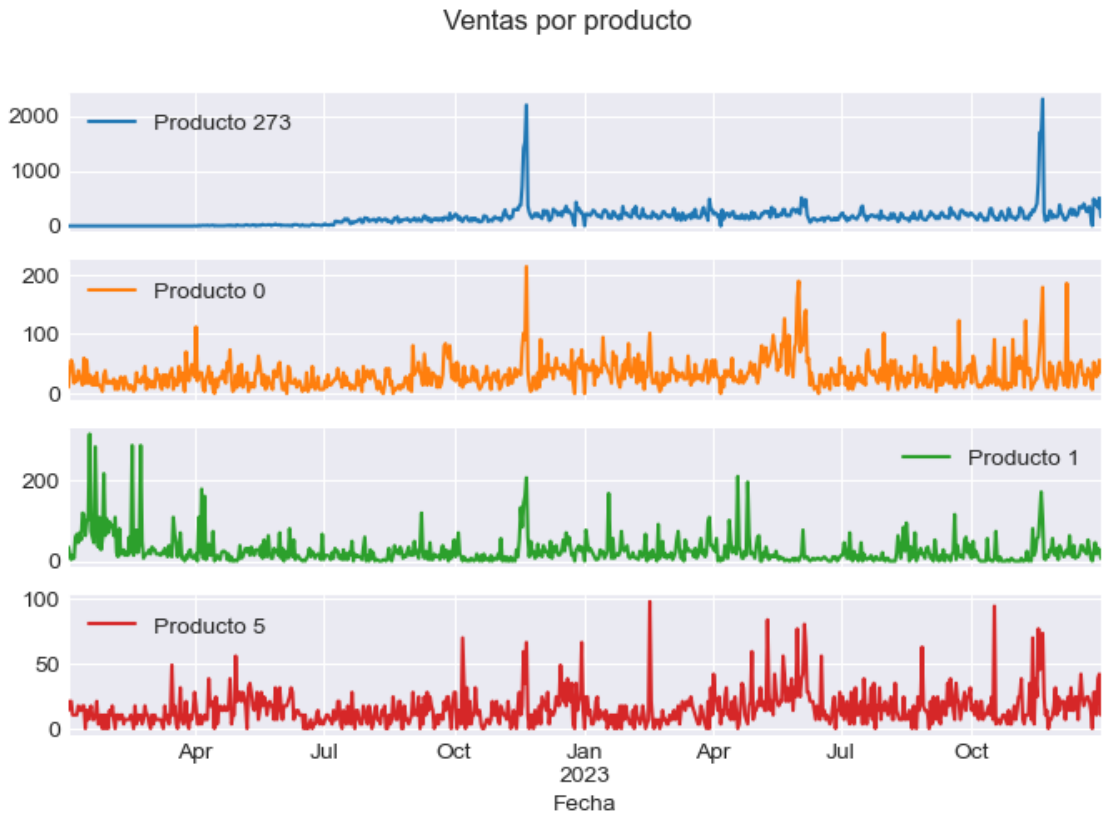
Algunas de los trends que hay

[ ]:
```
# Plot time series
#␣
 ↪==============================================================================
fig, ax = plt.subplots(figsize=(8, 5))
data.iloc[:, :4].plot(
    legend   = True,
    subplots = True,
    sharex   = True,
    title    = 'Ventas por producto',
    ax       = ax,
)
fig
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_5440\694730855.py:4: UserWarning: To output
multiple subplots, the figure containing the passed axes is being cleared.
  data.iloc[:, :4].plot(
```

[ ]:


Ventas por producto
Producto 273
Producto 0
Producto 1
Producto 5
2000
1000
0
200
100
0
200
0
100
50
0
Apr
Jul
Oct
Jan
2023
Apr
Jul
Oct
Fecha



[ ]:
```
from tqdm import tqdm
from functools import partialmethod
```

3.1 Se buscan los mejores hiperparametros para el modelo por grid-search y se guardan en un dataframe

[ ]:
```
# Hyperparameter search and backtesting of each item's model
#␣
↪==========================================================================================
items = []
mae_values  = []
dictes = {}

lags_grid = [7, 14, 21]
param_grid = {
    'max_iter': [100, 500],
    'max_depth': [3, 5, 10],
    'learning_rate': [0.01, 0.1]
}
```

```
for i, item in enumerate(data.columns):

    forecaster = ForecasterAutoreg(
                     regressor     =␣
 ↪HistGradientBoostingRegressor(random_state=123),
                     lags          = 14,
                     transformer_y = StandardScaler()
                 )

    results_grid = grid_search_forecaster(
                       forecaster         = forecaster,
                       y                  = data.loc[:end_val, item],
                       lags_grid          = lags_grid,
                       param_grid         = param_grid,
                       steps              = 7,
                       exog=exog.loc[:end_val, :],
                       metric             = "mean_absolute_error",
                       initial_train_size = len(data_train),
                       refit              = False,
                       fixed_train_size   = False,
                       return_best        = True,
                       verbose            = False,
                       show_progress      = False
                   )

    metric, preds = backtesting_forecaster(
                        forecaster         = forecaster,
                        y                  = data[item],
                        exog=exog,
                        initial_train_size = len(data_train) + len(data_val),
                        steps              = 7,
                        metric             = "mean_absolute_error",
                        refit              = False,
                        fixed_train_size   = False,
                        verbose            = False,
                        show_progress      = False
                    )

    items.append(item)
    mae_values.append(metric)
    dictes[item] = results_grid


uni_series_mae = pd.Series(
                     data  = mae_values,
                     index = items,
                     name  = 'uni_series_mae'
```

```
                  )
```

```
Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.01, 'max_depth': 10, 'max_iter': 100}
  Backtesting metric: 264.4963276077366

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'max_iter': 100}
  Backtesting metric: 26.48353771142115

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.01, 'max_depth': 10, 'max_iter': 100}
  Backtesting metric: 28.431921601843328

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
  Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'max_iter': 500}
  Backtesting metric: 15.036508336730442

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
  Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'max_iter': 500}
  Backtesting metric: 15.153688182893955

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'max_iter': 100}
  Backtesting metric: 19.76892031343544

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
```

```
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 100}
  Backtesting metric: 12.822421108786475

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.01, 'max_depth': 10, 'max_iter': 100}
  Backtesting metric: 3.714957869835493

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 6.363547144362676

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.1, 'max_depth': 10, 'max_iter': 500}
  Backtesting metric: 8.466386809096104

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 100}
  Backtesting metric: 7.267362601120225

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 100}
  Backtesting metric: 6.327995185232399

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
  Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'max_iter': 100}
  Backtesting metric: 4.410337363014955

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
```

```
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 100}
  Backtesting metric: 4.243145044573199

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 2.6834590610917495

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 0.9615273672153041

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21]
  Parameters: {'learning_rate': 0.01, 'max_depth': 10, 'max_iter': 100}
  Backtesting metric: 6.327498928354935

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 5.143836575534395

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [1 2 3 4 5 6 7]
  Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 3.826729151861387

Number of models compared: 36.
`Forecaster` refitted using the best-found lags and parameters, and the whole
data set:
  Lags: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14]
  Parameters: {'learning_rate': 0.01, 'max_depth': 5, 'max_iter': 500}
  Backtesting metric: 0.6832690608391511
```

4 Se procede a hacer un modelo para cada uno de ellos.

Se puede guardar cada modelo en un csv, que puede ser releido para usarse en el futuro, pero es recomendable reentrenar cada semna

[ ]:
```
for i in dictes.keys():
    #dictes[i].to_csv(f"dictes_{i}.csv")
    dictes[i]["Producto"] = i
```

[ ]:
```

```

Se crea un dataframe con cada uno de los parametros necesarios

[ ]:
```
models = pd.DataFrame()
for i in dictes.keys():
    models = pd.concat([models,␣
 ↪dictes[i][["lags","learning_rate","max_depth","max_iter","Producto","mean_absolute_error"]]
 ↪head(1)], axis=0)
models.index = models["Producto"]
models["max_depth"] = models["max_depth"].astype(int)
models["max_iter"] = models["max_iter"].astype(int)
models.loc["Producto 273","max_iter"]= 500
#models.to_csv("modelsHGB.csv")
```

Se entrena cada modelo y se predice para una semana y para un mes, se guarda en un dataframe cada predicción

[ ]:
```
month_pred = pd.DataFrame()
week_pred = pd.DataFrame()
```

[ ]:
```
for i in models.index:
    forecaster = ForecasterAutoreg(
                     regressor     =␣
 ↪HistGradientBoostingRegressor(random_state=123, max_depth=models.
 ↪loc[i,"max_depth"], max_iter=models.loc[i,"max_iter"], learning_rate=models.
 ↪loc[i,"learning_rate"]),
                     lags          = [models.loc[i,"lags"].max()],
                     transformer_y = StandardScaler()
                 )
    forecaster.fit(y=data.loc[:"2023-11-30",i], exog=exog.loc[:"2023-11-30", :])
    month_pred[i] = forecaster.predict(steps=31, exog=exog.loc["2023-12-01":])
    week_pred[i] = forecaster.predict(steps=8, exog=exog.loc["2023-12-01":])
```

Se ayuda un poco al modelo dado que los valores negativos no existen en los productos

[ ]:
```
# cambiar valores negativos a 0
month_pred[month_pred<0] = 0
week_pred[week_pred<0] = 0
```

[ ]:
```
from sklearn.metrics import mean_absolute_error,␣
↪mean_absolute_percentage_error, mean_squared_error, r2_score
```

Se consiguen las estadisticas que tiene el modelo para predecir durante un mes y durante una semana

[ ]:
```
stats_mes = pd.DataFrame()
for i in month_pred.columns:
    y_pred = month_pred[i]
    y_test = data.loc["2023-12-01":"2023-12-31",i]
    mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
    mape = mean_absolute_percentage_error(y_true=y_test, y_pred=y_pred)
    mse = mean_squared_error(y_true=y_test, y_pred=y_pred)
    r2 = r2_score(y_true=y_test, y_pred=y_pred)
    mape2 = abs((y_pred - y_test)/y_test).replace([np.inf, -np.inf], np.log(0.
↪99999999999999999999999999)).dropna().sum()/30
    mape3 = (np.abs((y_test - y_pred)/np.where(y_test==0, 1, y_test))).mean()
    smape = 1/len(y_test) * np.sum(2*np.abs(y_pred - y_test)/(np.abs(y_pred) +␣
↪np.abs(y_test))*100)
    valor_real = y_test.sum()
    valor_pred = y_pred.sum()
    error = (valor_real - valor_pred)/valor_real
    error_semanal = error/4 * 100
    stats_mes = pd.concat([stats_mes, pd.DataFrame({"Producto":i, "MAE":mae,␣
↪"MSE":mse, "R2":r2, "SMAPE": smape,"MAPE lib":mape,"MAPE2":mape2,"MAPE3":
↪mape3, "valor real":
        valor_real, "valor predecido": valor_pred, "error":error*100, "error␣
↪por semana": error_semanal}, index=[0])], axis=0)
stats_mes
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_5440\1630151674.py:14: RuntimeWarning:
divide by zero encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real
```

[ ]:

	Producto	MAE	MSE	R2	SMAPE	MAPE lib	\
0	Producto 273	95.793754	15217.656038	-0.171916	38.714044	1.451618e+00	
0	Producto 0	18.286359	1003.474879	-0.075167	45.931559	8.522957e-01	
0	Producto 1	12.395970	220.389846	-0.029128	56.568876	3.898879e+15	
0	Producto 5	7.568829	101.837021	-0.120478	38.743569	3.745069e+15	
0	Producto 8	6.164464	59.024440	0.256040	57.239842	5.250339e+15	
0	Producto 21	3.218037	15.420351	-0.326437	115.200718	3.110624e+15	
0	Producto 12	2.020074	7.477569	0.053859	76.989850	4.449879e+15	
0	Producto 22	3.103092	13.229094	-0.333945	130.604673	1.018220e+16	
0	Producto 186	5.286273	38.350348	-0.320696	90.903108	9.108969e+15	
0	Producto 20	1.415677	2.990410	0.137520	145.006665	3.537359e+15	
0	Producto 33	3.168126	16.999202	0.040597	93.022267	5.977556e+15	
0	Producto 245	4.503595	35.648434	-0.166215	118.320799	7.030731e+15	

```
0   Producto 16   3.445023     27.862517 -0.065198  113.134387  4.851344e+15
0   Producto 17   3.143415     13.631636  0.038320  124.150710  7.073511e+15
0   Producto 38   1.783880      3.649313 -0.908566  174.613341  6.531856e+15
0  Producto 134   1.175395      2.242783 -4.864802  198.849395  4.834820e+15
0   Producto 37   1.810427      3.681765 -0.504326  183.100538  6.778935e+15
0   Producto 59   2.290128      9.889100 -0.204643  153.703921  2.667133e+15
0  Producto 248   4.312378     24.862918 -1.172016  103.239386  9.044035e+15
0  Producto 122   1.135570      1.371256  0.000000  200.000000  5.114153e+15

      MAPE2     MAPE3  valor real  valor predecido       error  \
0  1.500005  1.451618      8452.5      6935.372539   17.948861
0  0.880706  0.852296      1186.5      1316.897688  -10.990113
0  0.894437  1.731309       731.5       810.692673  -10.826066
0  0.473572  1.289868       651.0       697.175864   -7.093067
0  0.647157  1.792090       399.0       444.534648  -11.412192
0  0.397368  1.075247       133.0        76.064086   42.808958
0  0.148671  1.131946       108.5       107.332802    1.075758
0  0.115641  2.372813        70.0       118.648808  -69.498297
0  0.439951  2.448357       203.0       233.814536  -15.179575
0  0.163922  0.944086        35.0        39.812001  -13.748574
0  0.249243  1.568487       129.5       131.609779   -1.629173
0  0.295801  1.847395       150.5       113.321209   24.703516
0  0.253910  1.322934       126.0        85.991593   31.752704
0  0.206699  1.770666       101.5       102.558913   -1.043264
0  0.098467  1.545654        21.0        55.622259 -164.867898
0  0.030070  1.102646         3.5        33.622572 -860.644902
0  0.067094  1.570155        17.5        54.700755 -212.575743
0  0.324307  0.906067        73.5        39.223829   46.634246
0  0.541025  2.531752       112.0       198.224928  -76.986543
0  0.000000  1.135570         0.0        35.202673        -inf

   error por semana
0          4.487215
0         -2.747528
0         -2.706517
0         -1.773267
0         -2.853048
0         10.702239
0          0.268940
0        -17.374574
0         -3.794894
0         -3.437143
0         -0.407293
0          6.175879
0          7.938176
0         -0.260816
0        -41.216975
```

```
0       -215.161225
0        -53.143936
0         11.658562
0        -19.246636
0              -inf
```

[ ]:
```
stats_semana = pd.DataFrame()
for i in month_pred.columns:
    y_pred = week_pred[i]
    y_test = data.loc["2023-12-01":"2023-12-8",i]
    mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
    mape = mean_absolute_percentage_error(y_true=y_test, y_pred=y_pred)
    mse = mean_squared_error(y_true=y_test, y_pred=y_pred)
    r2 = r2_score(y_true=y_test, y_pred=y_pred)
    mape2 = abs((y_pred - y_test)/y_test).replace([np.inf, -np.inf], np.log(0.
 ↪99999999999999999999999)).dropna().sum()/30
    mape3 = (np.abs((y_test - y_pred)/np.where(y_test==0, 1, y_test))).mean()
    smape = 1/len(y_test) * np.sum(2*np.abs(y_pred - y_test)/(np.abs(y_pred) +␣
 ↪np.abs(y_test))*100)
    valor_real = y_test.sum()
    valor_pred = y_pred.sum()
    error = (valor_real - valor_pred)/valor_real *100
    stats_semana = pd.concat([stats_semana, pd.DataFrame({"Producto":i, "MAE":
 ↪mae, "MSE":mse, "R2":r2, "SMAPE": smape,"MAPE lib":mape,"MAPE2":
 ↪mape2,"MAPE3":mape3, "valor real":
        valor_real, "valor predecido": valor_pred, "error":error}, index=[0])],␣
 ↪axis=0)
stats_semana
```

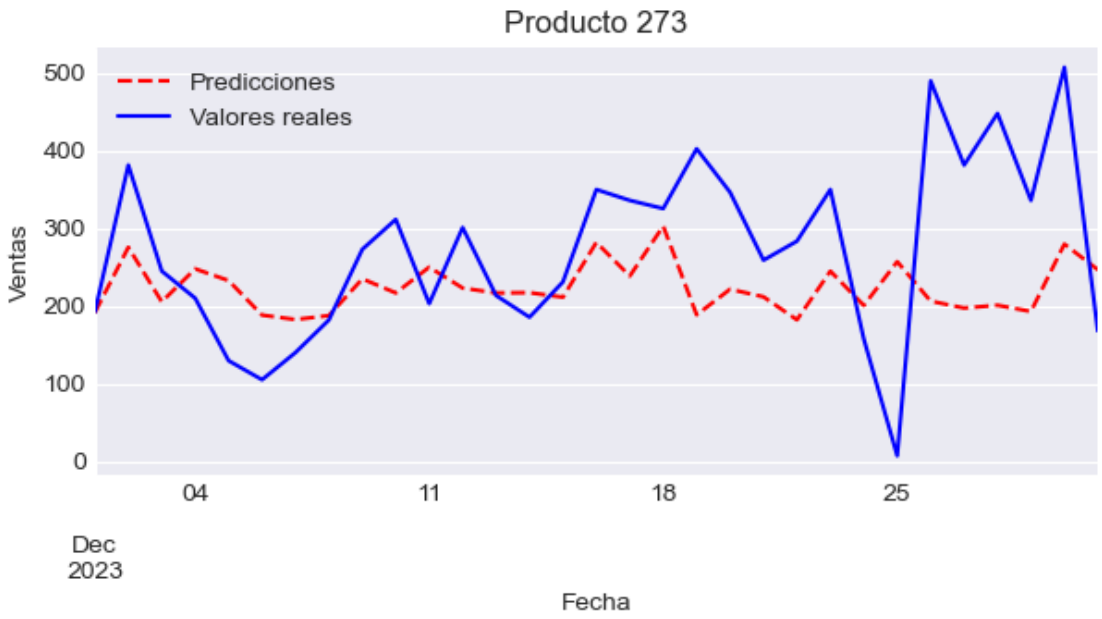
```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_5440\2497161326.py:14: RuntimeWarning:
divide by zero encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real *100
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_5440\2497161326.py:14: RuntimeWarning:
divide by zero encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real *100
```

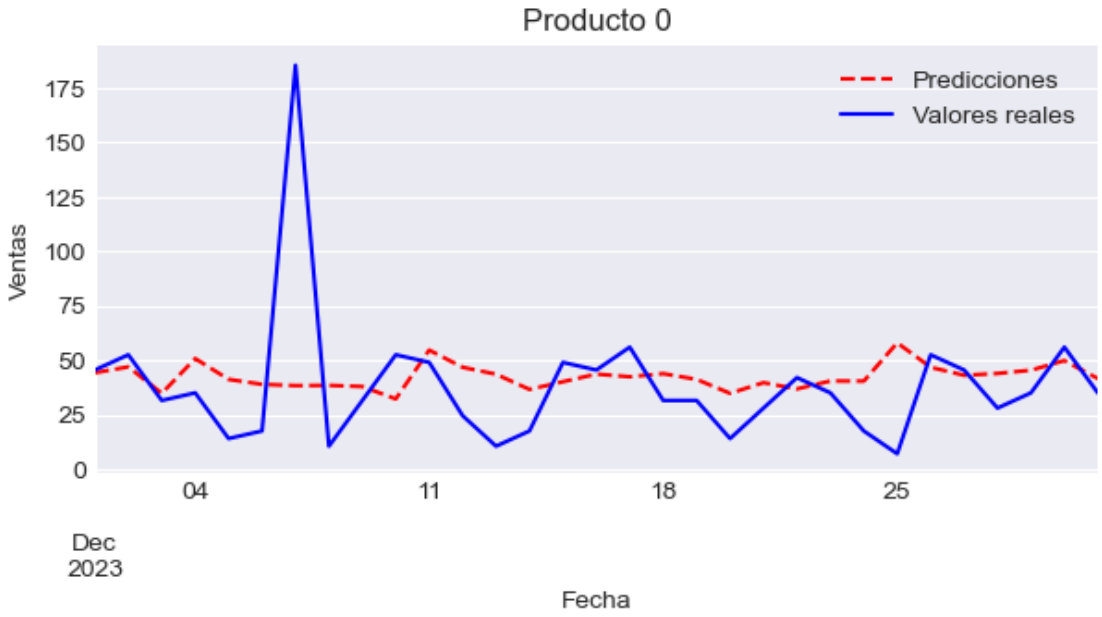
[ ]:

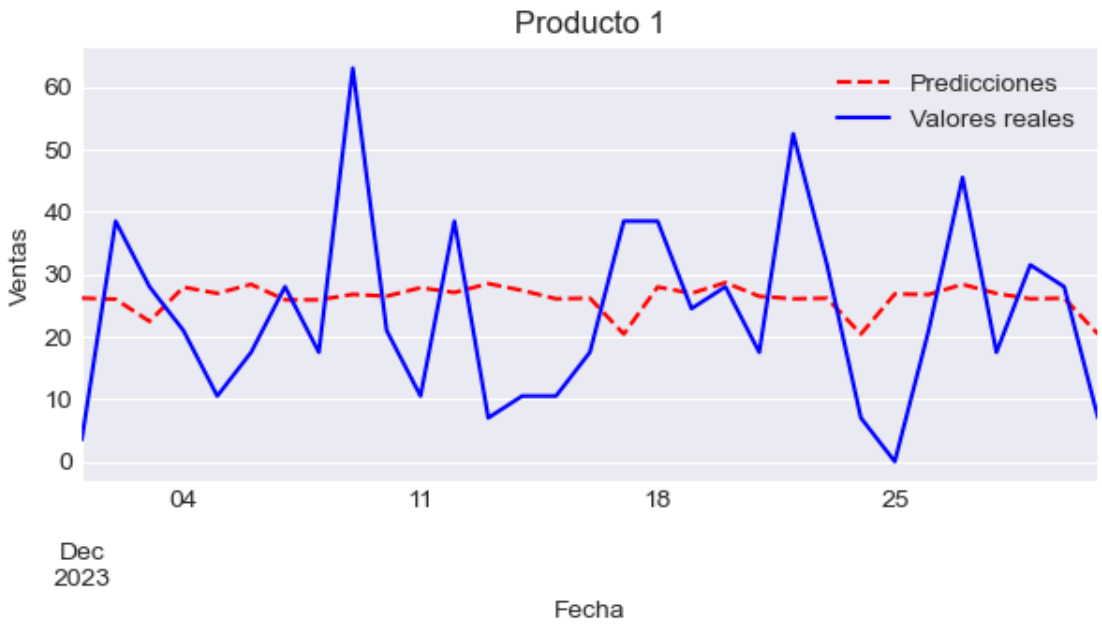
	Producto	MAE	MSE	R2	SMAPE	MAPE lib	\
0	Producto 273	52.346240	4202.757693	0.365819	26.245740	3.185085e-01	
0	Producto 0	31.196143	2991.057848	-0.046808	60.095193	9.134583e-01	
0	Producto 1	10.685492	151.653224	-0.437950	52.958992	1.259597e+00	
0	Producto 5	4.024606	31.415018	-0.032248	24.541844	3.363593e-01	
0	Producto 8	6.666298	68.199411	-1.240925	91.036889	1.441358e+16	
0	Producto 21	2.754933	9.081938	-0.482765	111.583601	3.493695e+15	
0	Producto 12	1.300566	4.308337	-0.500591	24.839629	1.902599e-01	
0	Producto 22	3.524867	14.119302	-9.538021	176.440046	1.563373e+16	

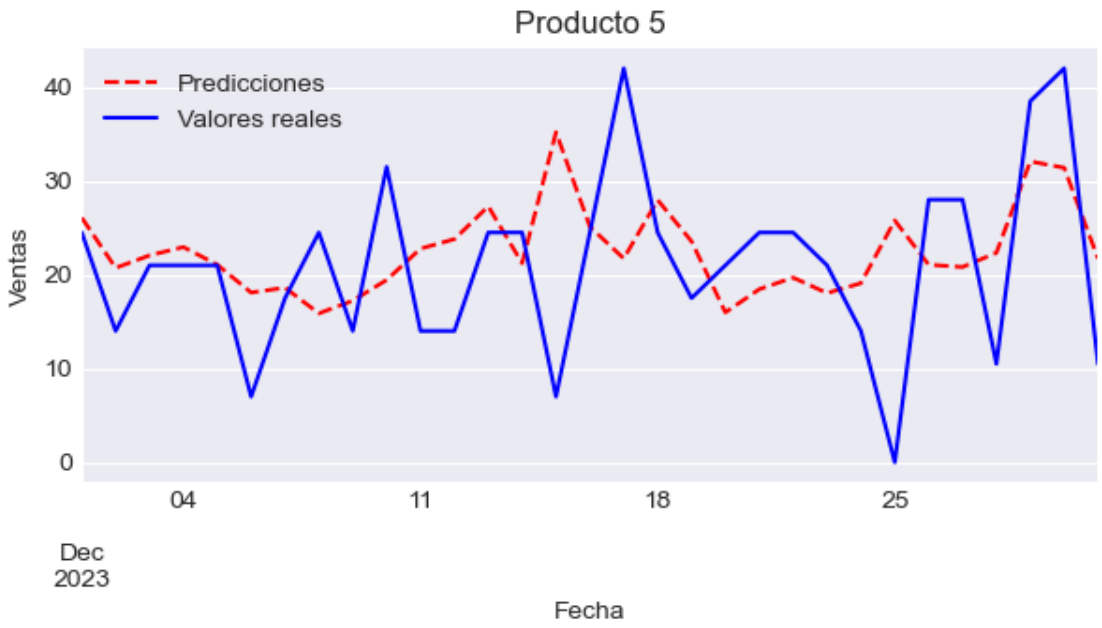
```
0  Producto 186   7.046090    60.093318 -0.794040  136.798306  1.827215e+16
0   Producto 20   0.804632     0.846046  0.368549  156.693373  2.791506e+15
0   Producto 33   3.085606    15.505472  0.147282   81.959046  4.479499e+15
0  Producto 245   2.513071     9.211696 -0.503950   83.623549  4.729783e+15
0   Producto 16   1.999625     4.863603 -0.693990  134.830181  7.638052e+15
0   Producto 17   3.407305    19.205853  0.104100  126.238992  7.154921e+15
0   Producto 38   1.793914     3.530183 -0.536950  171.223835  5.731280e+15
0  Producto 134   0.951639     1.432301  0.000000  200.000000  4.285799e+15
0   Producto 37   2.033616     5.287519  0.108884  171.048696  5.494542e+15
0   Producto 59   1.199590     1.863651  0.350892  150.701957  2.546453e+15
0  Producto 248   4.276385    26.748851 -1.183580  105.560156  1.131302e+16
0  Producto 122   1.084490     1.263457  0.000000  200.000000  4.884110e+15

      MAPE2     MAPE3  valor real  valor predecido       error
0  0.084936  0.318509      1585.5      1711.417806   -7.941836
0  0.243589  0.913458       392.0       333.567139   14.906342
0  0.335893  1.259597       164.5       209.707703  -27.481886
0  0.089696  0.336359       150.5       165.441314   -9.927783
0  0.223963  4.040318        45.5        98.830380 -117.209627
0  0.113508  1.201412        28.0        18.372639   34.383434
0  0.050736  0.190260        38.5        28.106424   26.996300
0  0.004075  3.486665         3.5        31.698934 -805.683841
0  0.081019  4.361053        38.5        54.992778  -42.838384
0  0.014079  0.672637         3.5         6.980370  -99.439149
0  0.069807  1.256426        38.5        32.961087   14.386787
0  0.072020  1.320300        28.0        28.461034   -1.646549
0  0.023134  1.782742        10.5        21.638808 -106.083888
0  0.049787  1.775414        28.0        26.160943    6.568062
0  0.039719  1.421547         7.0        13.010288  -85.861252
0  0.000000  0.951639         0.0         7.613109        -inf
0  0.038974  1.366185        10.5        13.751606  -30.967674
0  0.048317  0.746616        10.5         9.950101    5.237135
0  0.099250  2.884182        28.0        47.180184  -68.500656
0  0.000000  1.084490         0.0         8.675922        -inf
```

[ ]:
```python
#Save figure#
for i in month_pred.columns:
    fig, ax=plt.subplots(figsize=(7, 3))
    month_pred[i].plot(ax=ax,color='red', linestyle='--', label='Predicciones')
    data.loc["2023-12-01":"2023-12-31",i].plot(ax=ax,color='blue',␣
 ↪linestyle='-', label='Valores reales')
    ax.set_title(i)
    ax.legend()
    ax.set_ylabel('Ventas')
    ax.set_xlabel('Fecha')
    #plt.savefig(f"pred_mensual_{i}.png")
```


Producto 273
Predicciones
Valores reales
Ventas
500
400
300
200
100
0
04
11
18
25
Dec
2023
Fecha

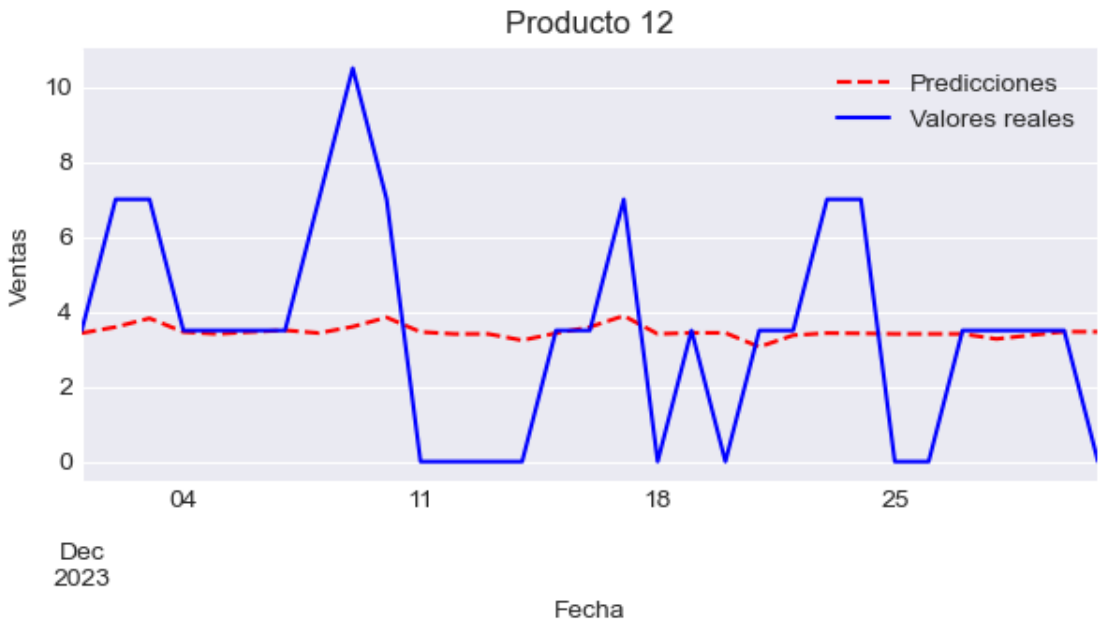

Producto 0
Predicciones
Valores reales
Ventas
175
150
125
100
75
50
25
0
04
11
18
25
Dec
2023
Fecha


Producto 1
Predicciones
Valores reales
Ventas
60
50
40
30
20
10
0
04
11
18
25
Dec
2023
Fecha

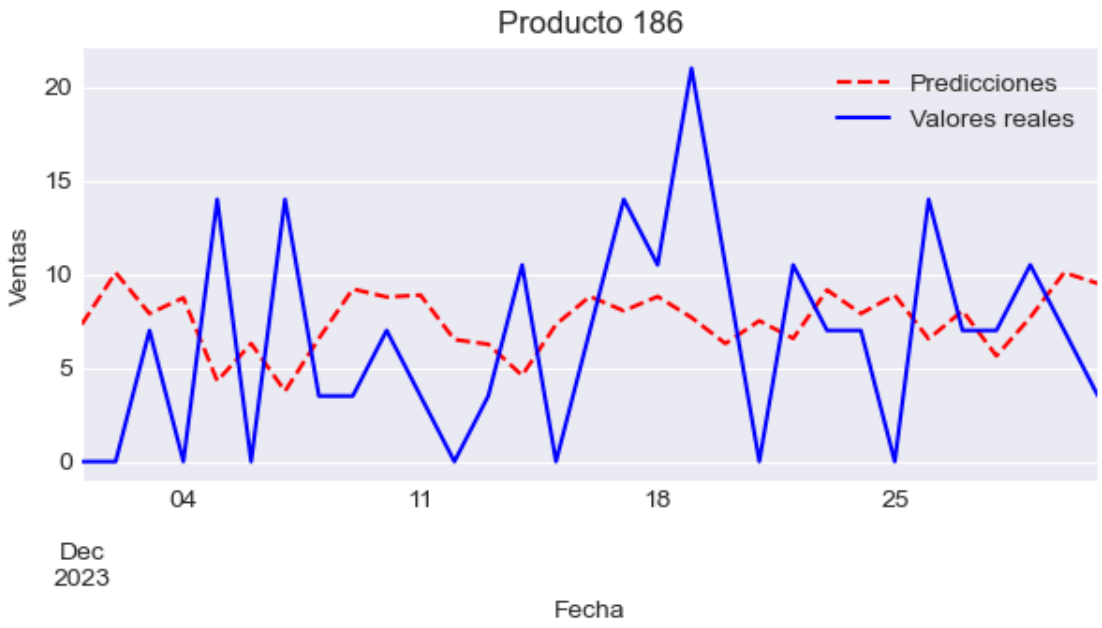

Producto 5
Predicciones
Valores reales
Ventas
40
30
20
10
0
04
11
18
25
Dec
2023
Fecha

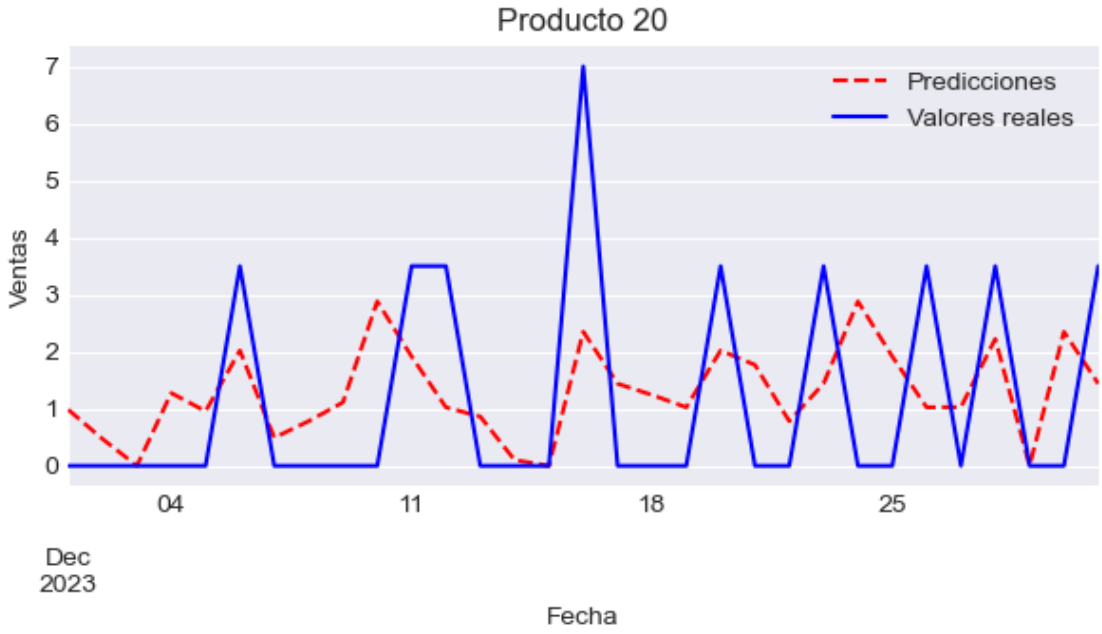

Producto 8
Predicciones
Valores reales
Ventas
35
30
25
20
15
10
5
0
04
11
18
25
Dec
2023
Fecha

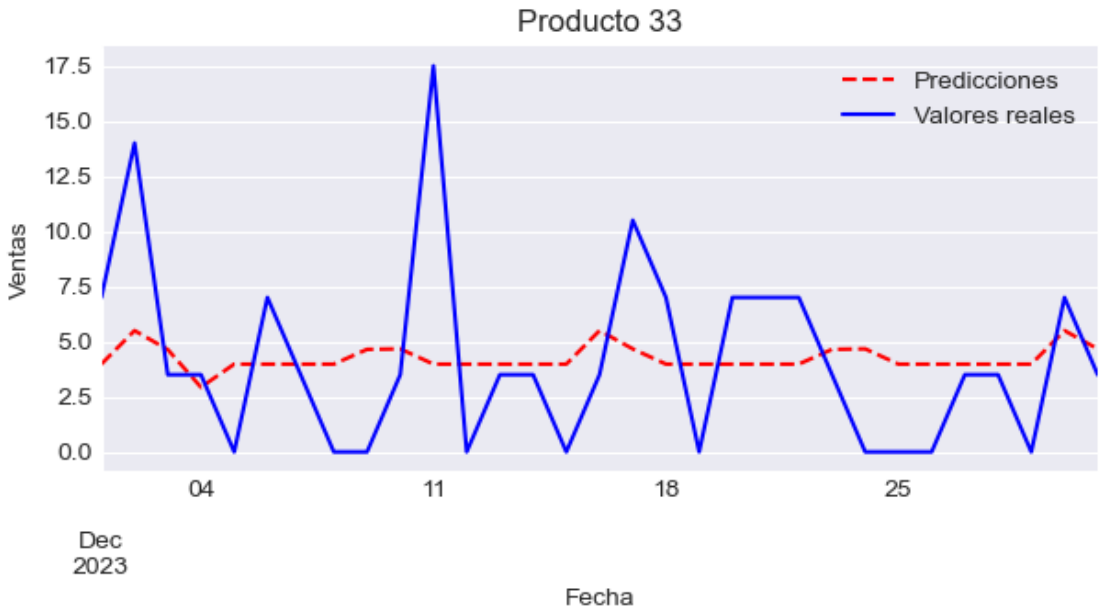

Producto 21
Predicciones
Valores reales
Ventas
10
8
6
4
2
0
04
11
18
25
Dec
2023
Fecha

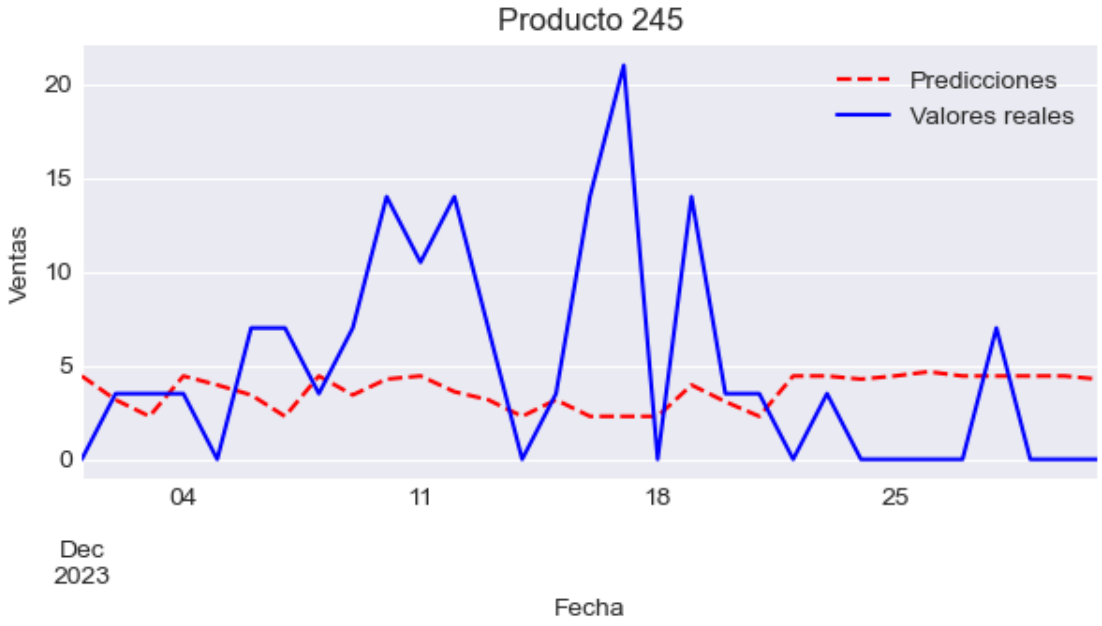

Producto 12
Predicciones
Valores reales
Ventas
10
8
6
4
2
0
04
11
18
25
Dec
2023
Fecha


Producto 22
Predicciones
Valores reales
Ventas
10
8
6
4
2
0
04
11
18
25
Dec
2023
Fecha

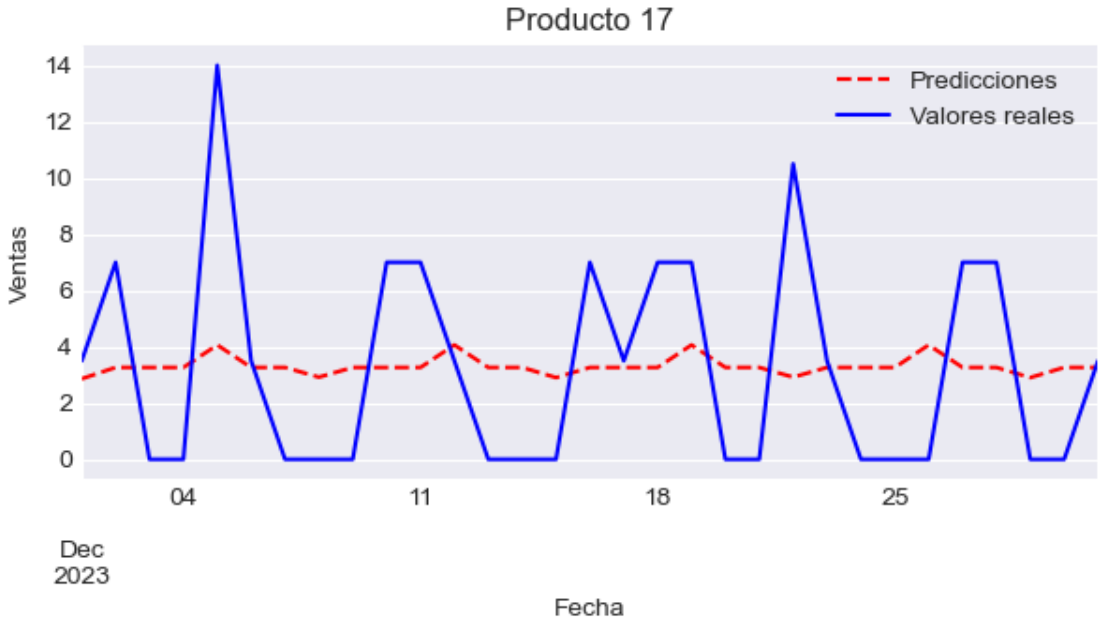

Producto 186
Predicciones
Valores reales
Ventas
20
15
10
5
0
04
11
18
25
Dec
2023
Fecha

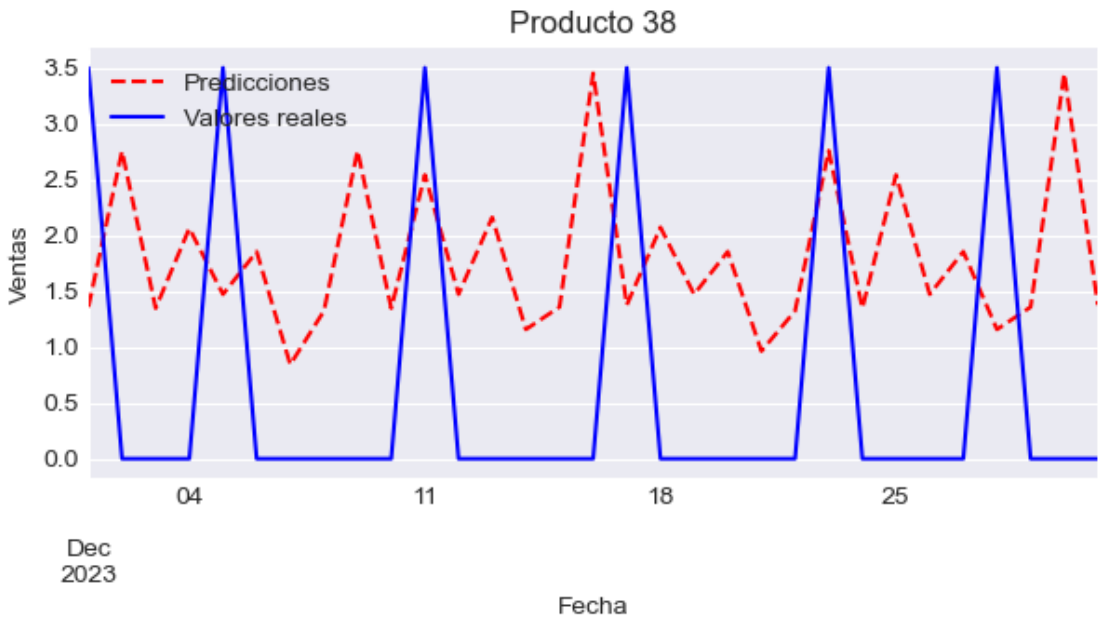

Producto 20
Predicciones
Valores reales
Ventas
7
6
5
4
3
2
1
0
04
11
18
25
Dec
2023
Fecha

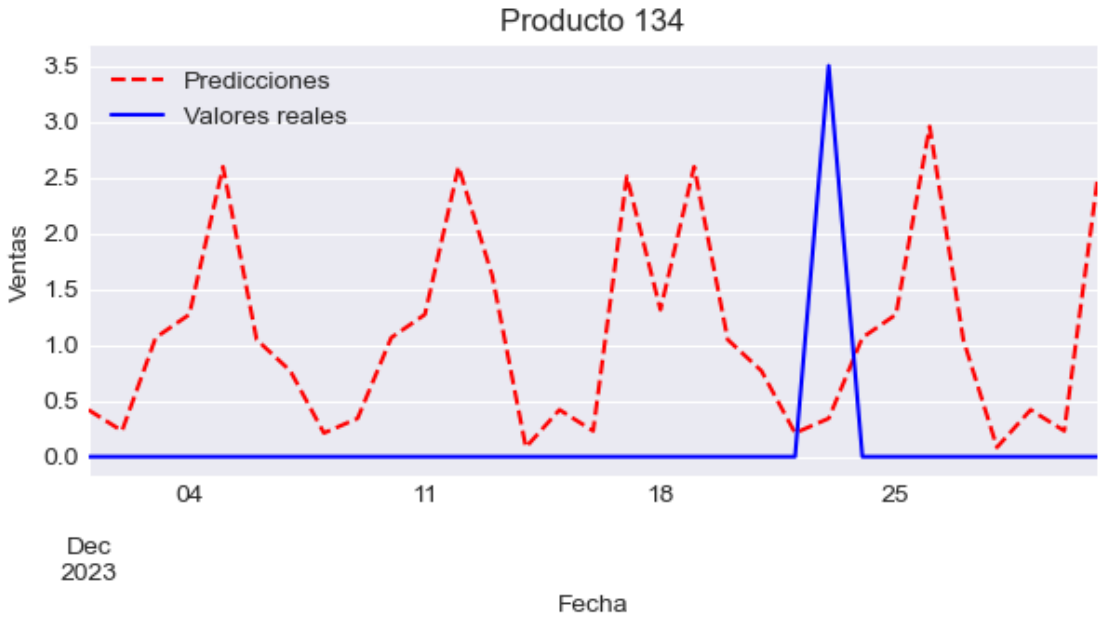

Producto 33
Predicciones
Valores reales
Ventas
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
04
11
18
25
Dec
2023
Fecha

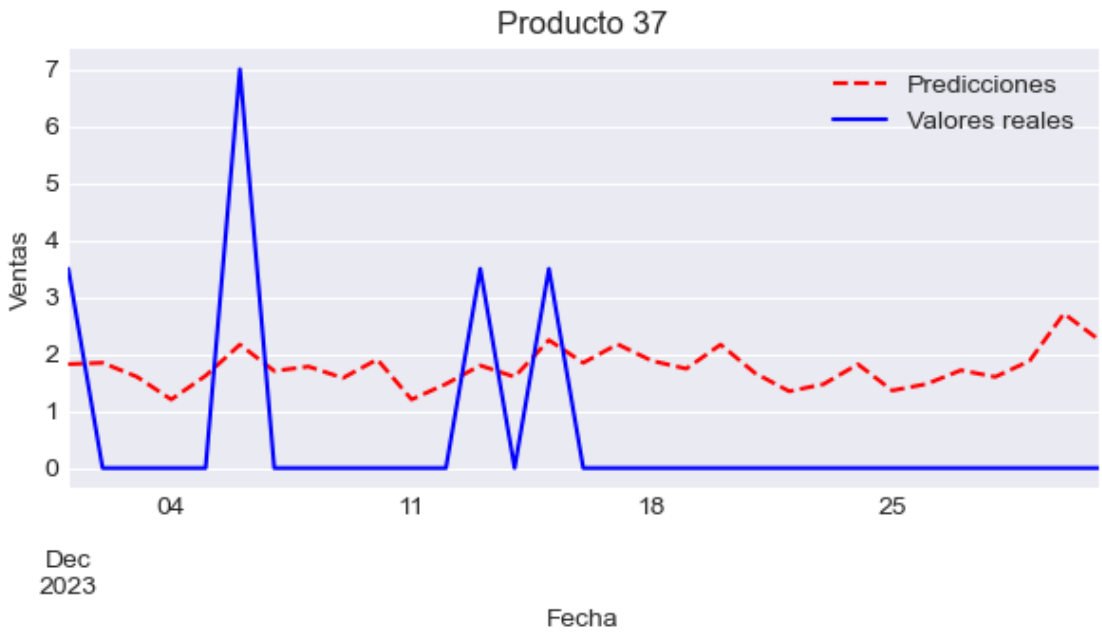

Producto 245
Predicciones
Valores reales
Ventas
20
15
10
5
0
04
11
18
25
Dec
2023
Fecha

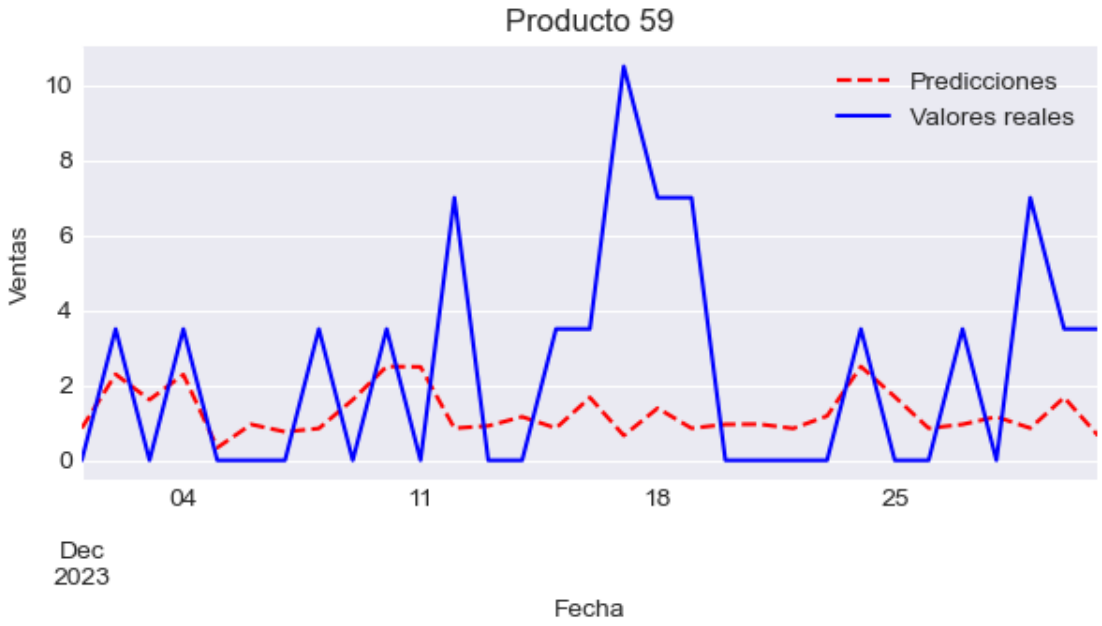

Producto 16
Predicciones
Valores reales
Ventas
20
15
10
5
0
04
11
18
25
Dec
2023
Fecha


Producto 17
Predicciones
Valores reales
Ventas
14
12
10
8
6
4
2
0
04
11
18
25
Dec
2023
Fecha

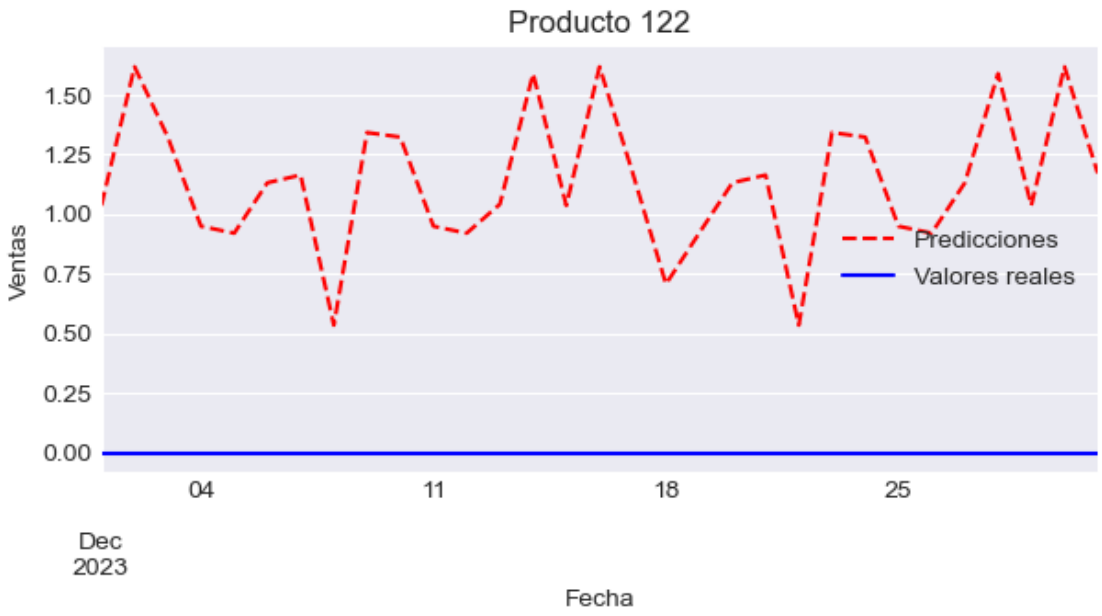

Producto 38
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
04
11
18
25
Dec
2023
Fecha


Producto 134
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
04
11
18
25
Dec
2023
Fecha


Producto 37
Predicciones
Valores reales
Ventas
0
1
2
3
4
5
6
7
04
11
18
25
Dec
2023
Fecha

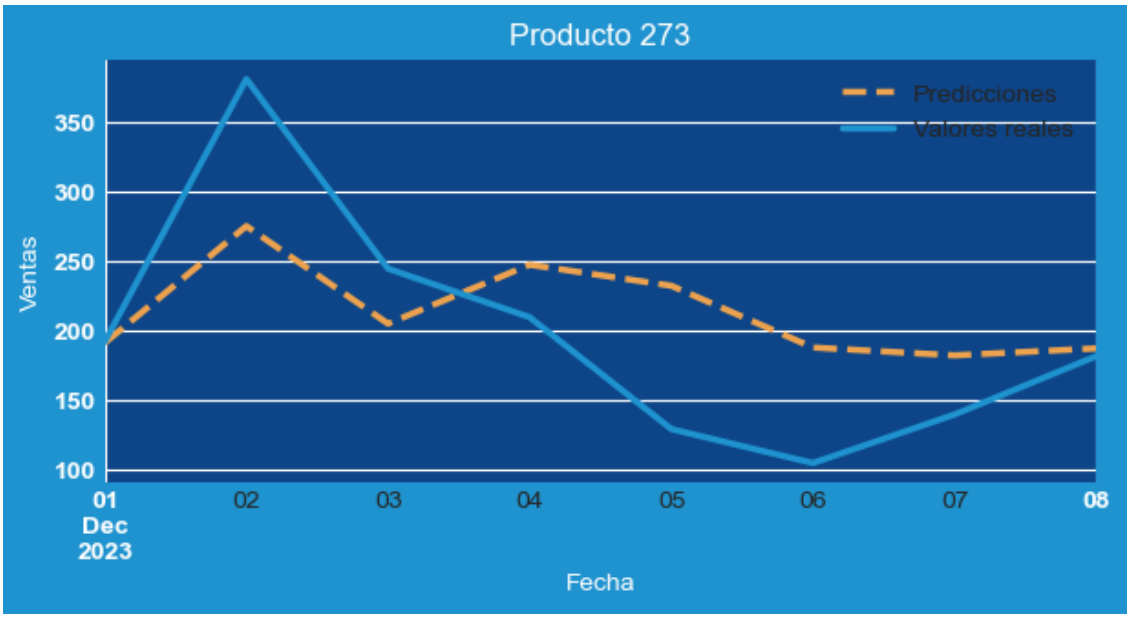

Producto 59
Predicciones
Valores reales
Ventas
0
2
4
6
8
10
04
11
18
25
Dec
2023
Fecha

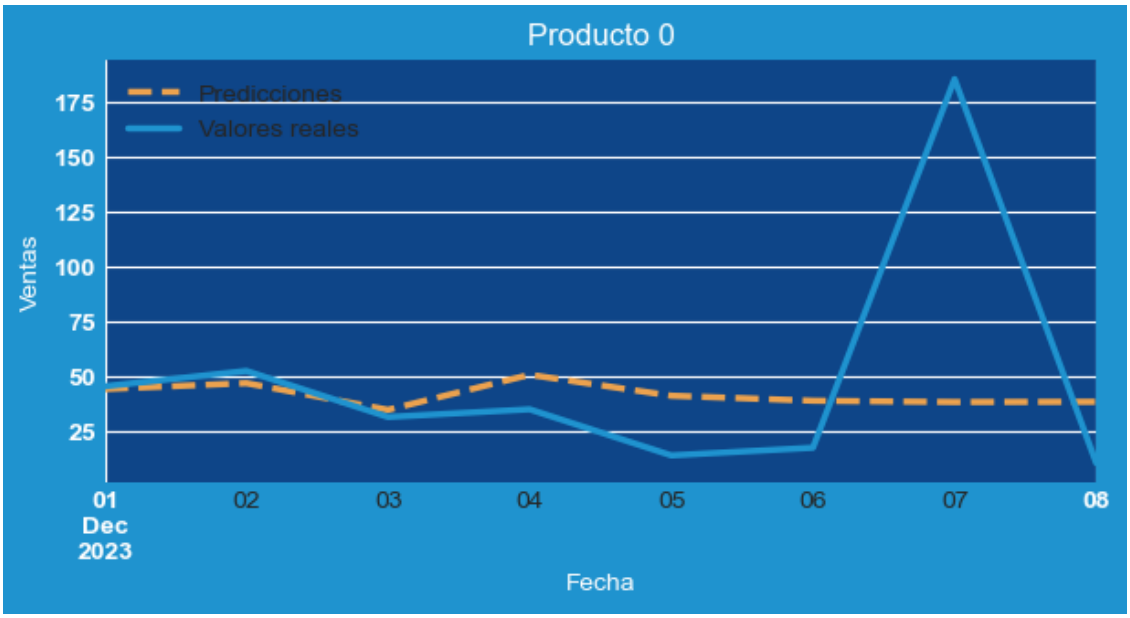

Producto 248
Ventas
10
8
6
4
2
0
04
11
18
25
Dec
2023
Fecha
Predicciones
Valores reales

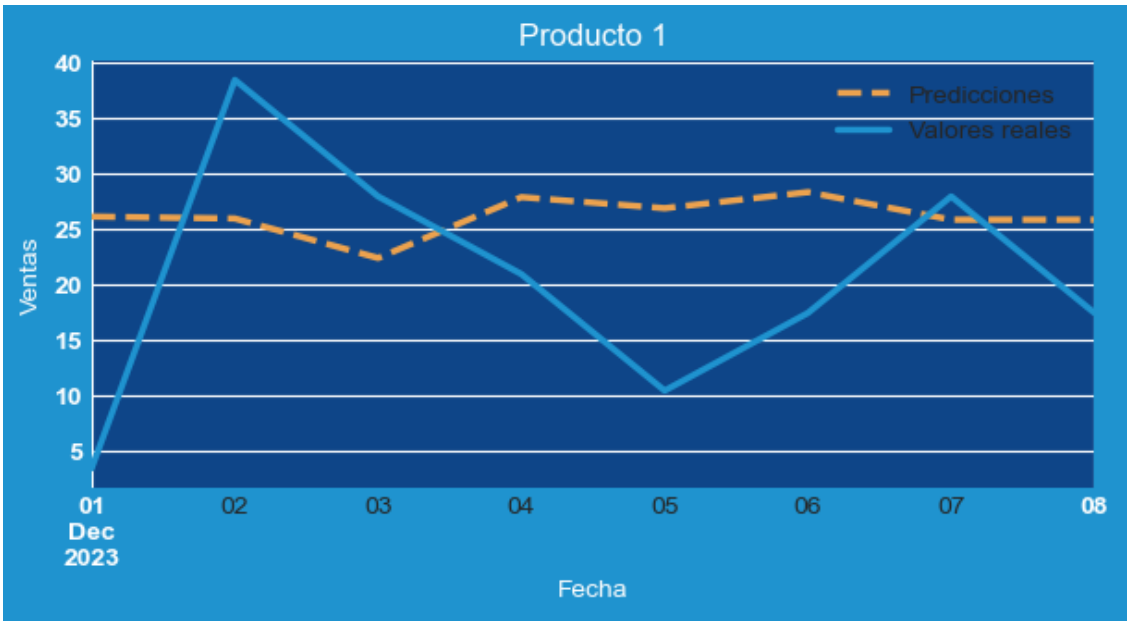

Producto 122
Ventas
1.50
1.25
1.00
0.75
0.50
0.25
0.00
04
11
18
25
Dec
2023
Fecha
Predicciones
Valores reales

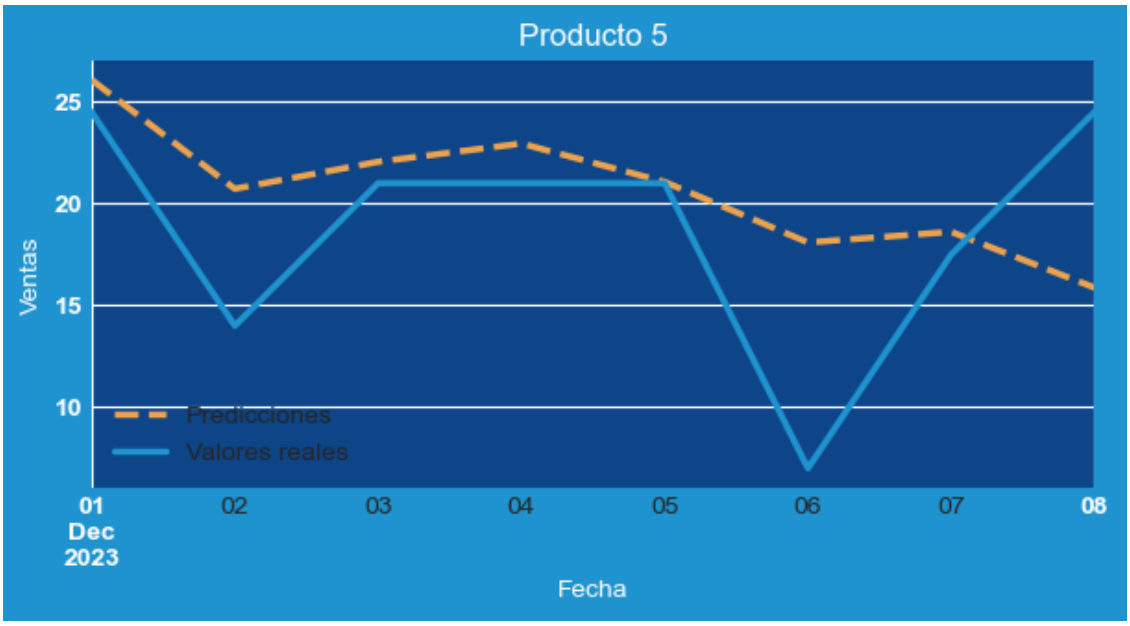
4.1 Agregar modelo del otro producto y comparar

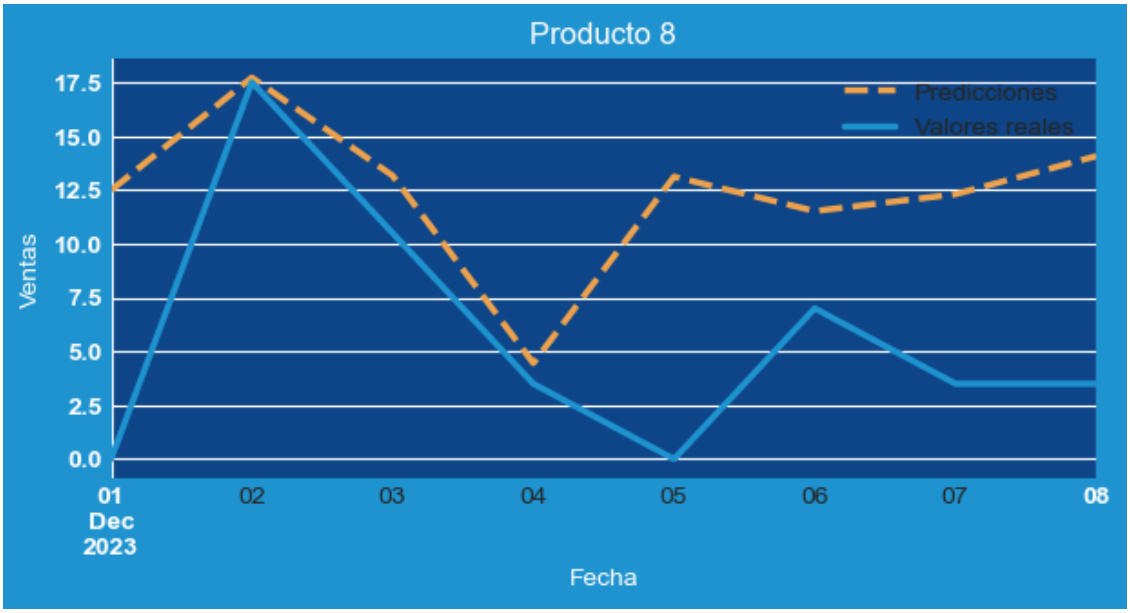
```
[ ]: for i in week_pred.columns:
        fig, ax=plt.subplots(figsize=(7, 3))
        week_pred[i].plot(ax=ax,color='#ECA24E', linestyle='--',␣
     ↪label='Predicciones', linewidth=2.5)
        data.loc["2023-12-01":"2023-12-08",i].plot(ax=ax,color='#1f93cf',␣
     ↪linestyle='-', label='Valores reales', linewidth=2.5)
        ax.set_title(i, color = 'white')
        ax.set_facecolor('#0e4588')
        ax.grid(color='white')
        #ax.tick_params(axis='x', colors='white')
        #ax.spines['bottom'].set_color('white')
        #ax.spines['top'].set_color('white')
        #ax.xaxis.label.set_color('white')
        #ax.tick_params(axis='x', colors='white', labelsize = 10, which = 'major')
        fig.set_facecolor('#1f93cf')
        ax.set_ylabel('Ventas',color='white')
        ax.set_xlabel('Fecha',color='white')
        plt.xticks(color='white', weight='bold')
        plt.yticks(color='white', weight='bold')
        #plt.legend(color='white')
        ax.legend()
```

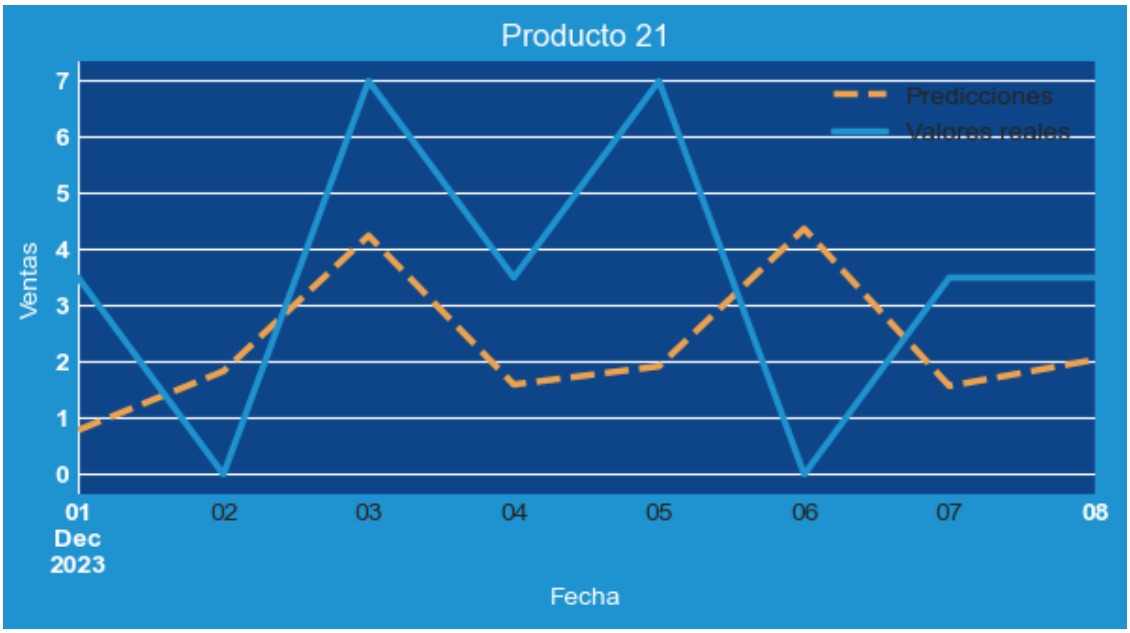


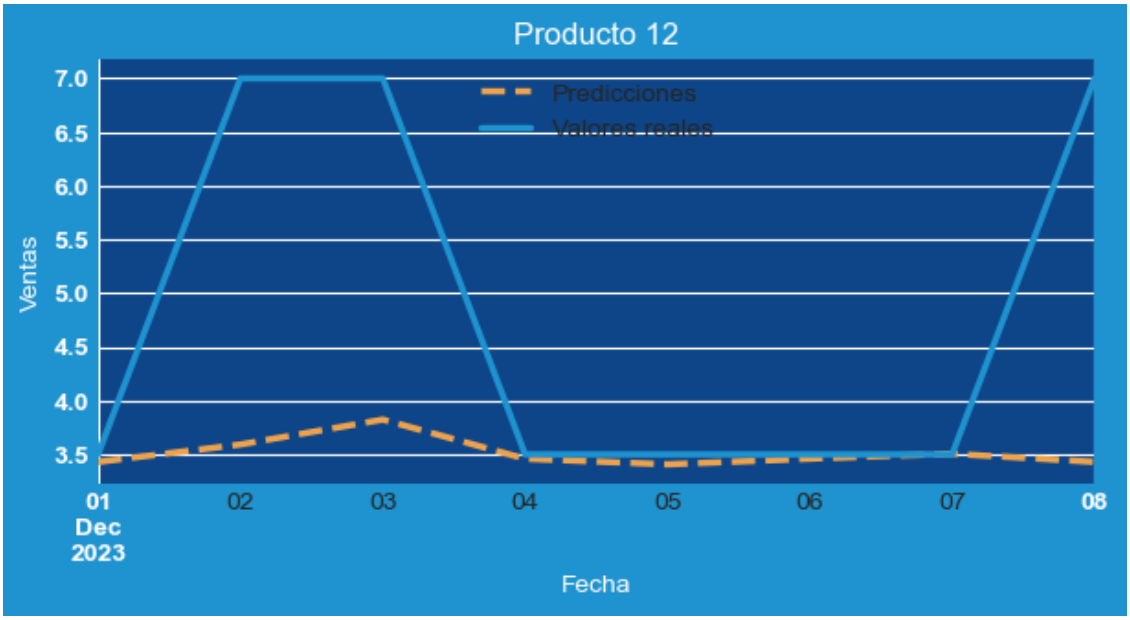

Producto 0
Predicciones
Valores reales
175
150
125
100
75
50
25
Ventas
01
Dec
2023
02
03
04
05
06
07
08
Fecha

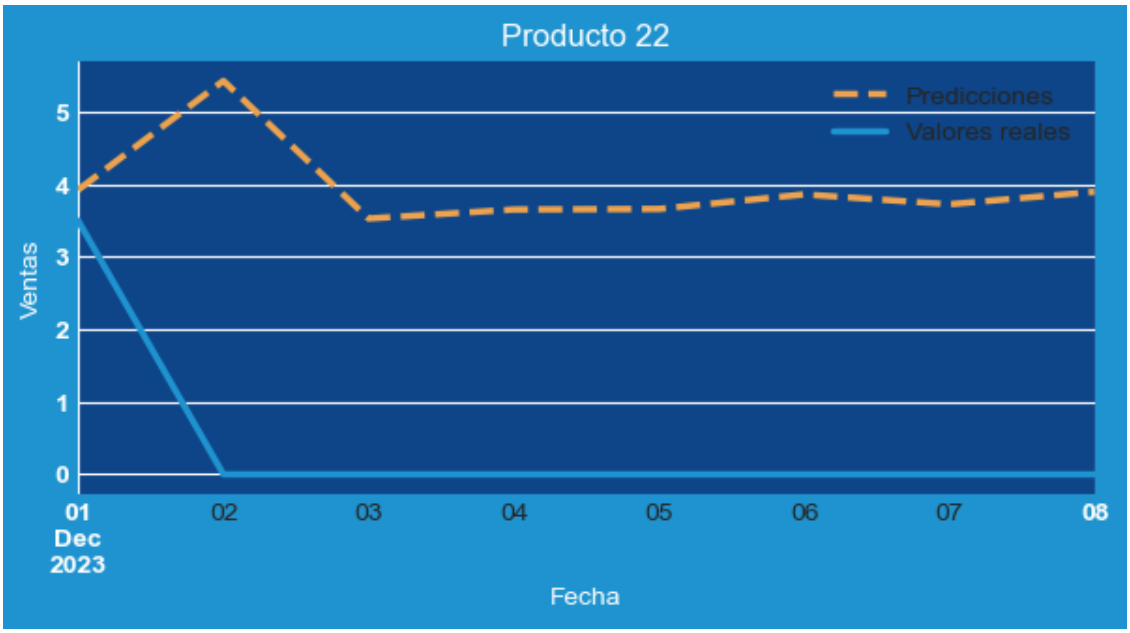

Producto 1
Predicciones
Valores reales
40
35
30
25
20
15
10
5
Ventas
01
Dec
2023
02
03
04
05
06
07
08
Fecha

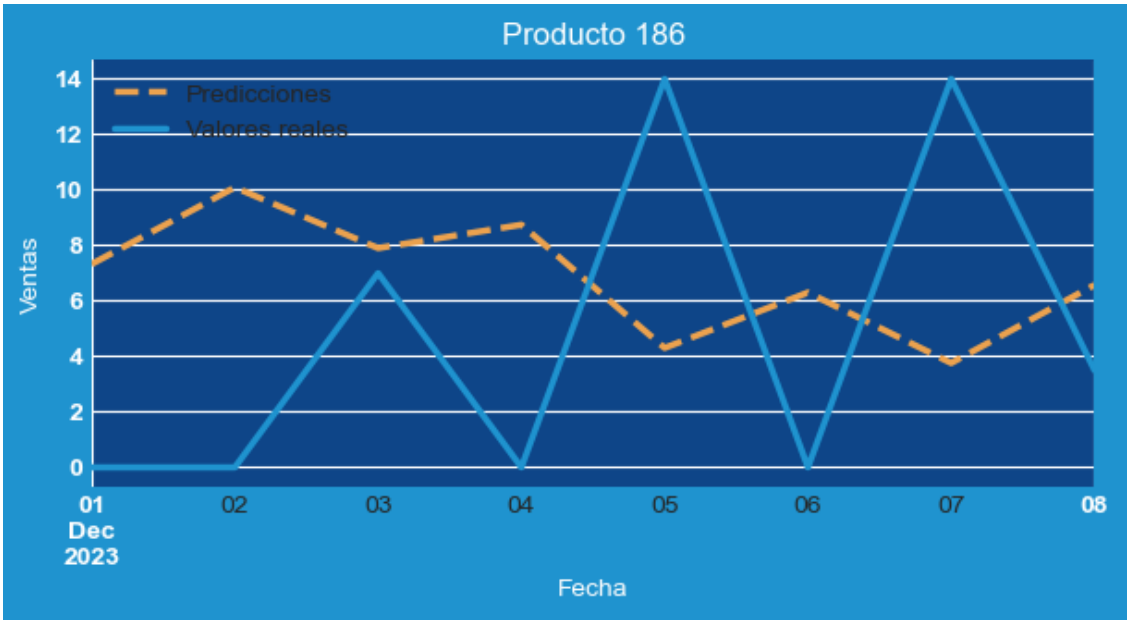

Producto 5
25
20
15
10
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
Fecha

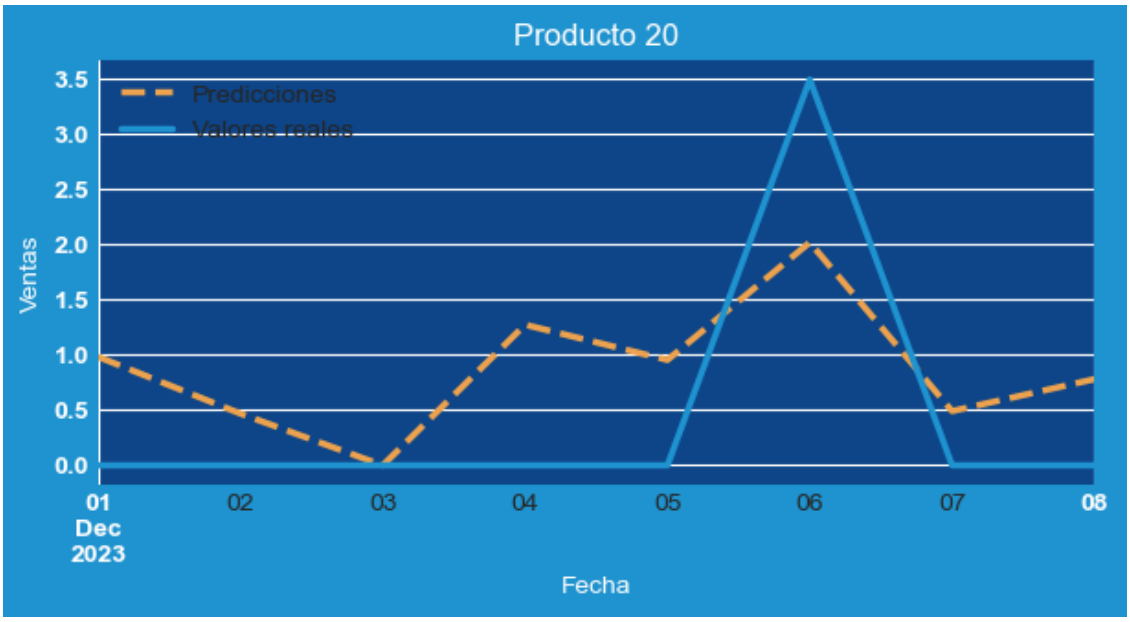

Producto 8
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
Fecha

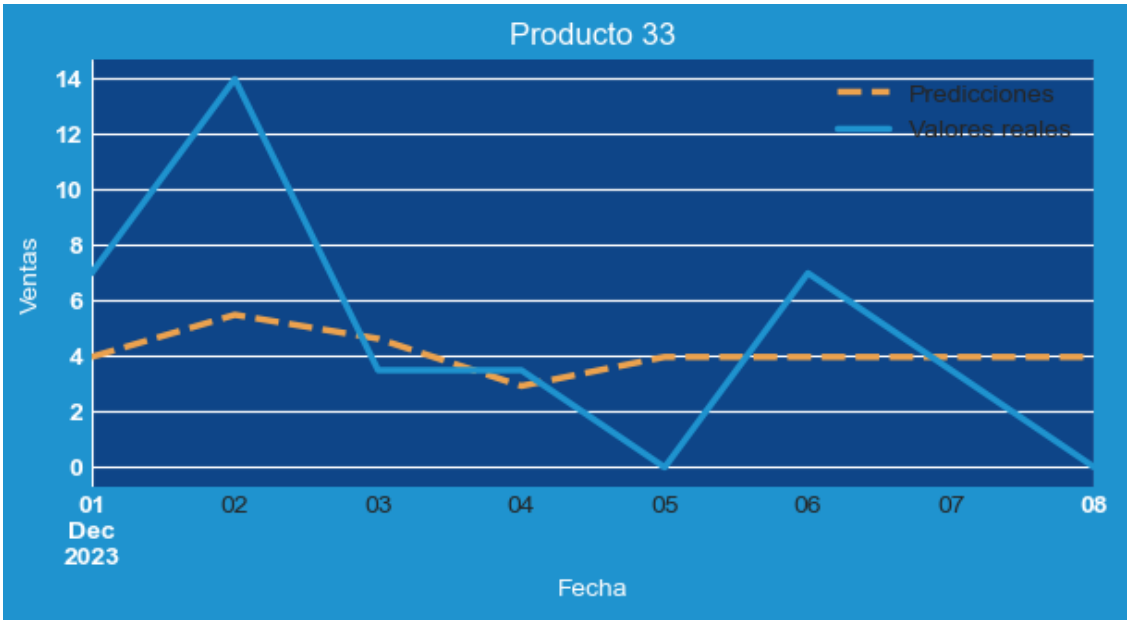

Producto 21
Predicciones
Valores reales
Ventas
7
6
5
4
3
2
1
0
01
Dec
2023
02
03
04
05
06
07
08
Fecha

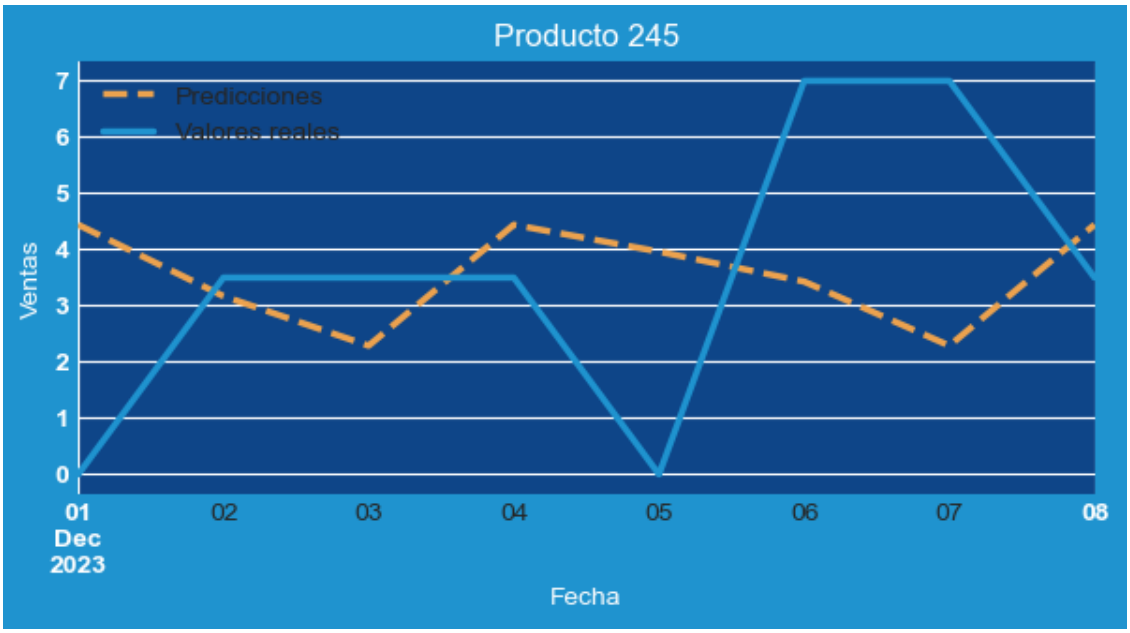

Producto 12
Predicciones
Valores reales
Ventas
7.0
6.5
6.0
5.5
5.0
4.5
4.0
3.5
01
Dec
2023
02
03
04
05
06
07
08
Fecha

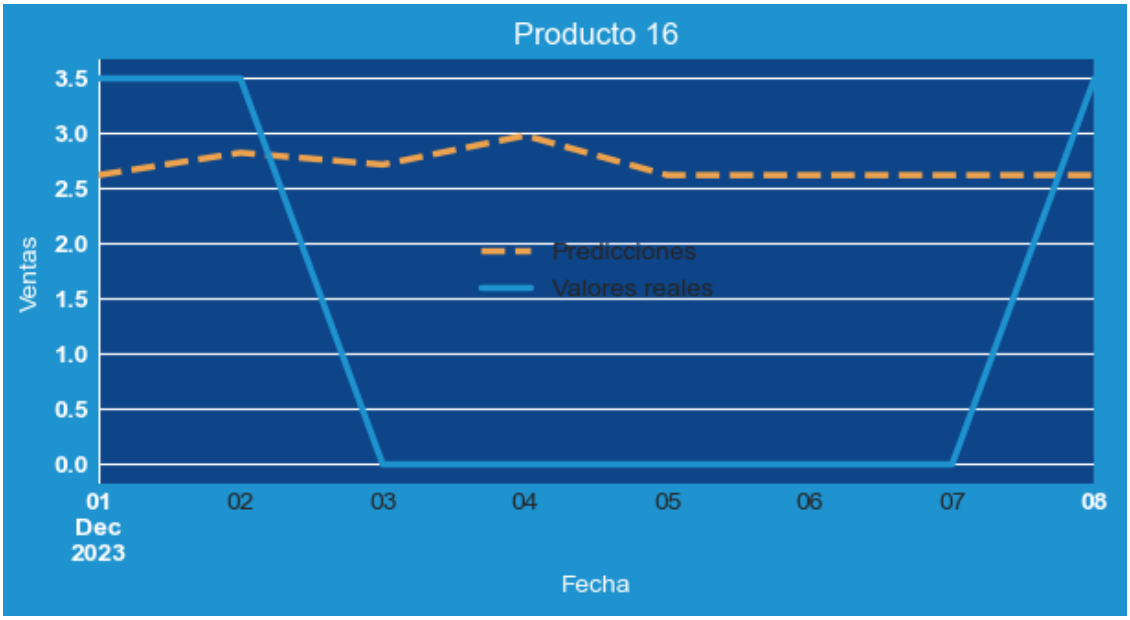

Producto 22
Predicciones
Valores reales
Ventas
0
1
2
3
4
5
01
Dec
2023
02
03
04
05
06
07
08
Fecha

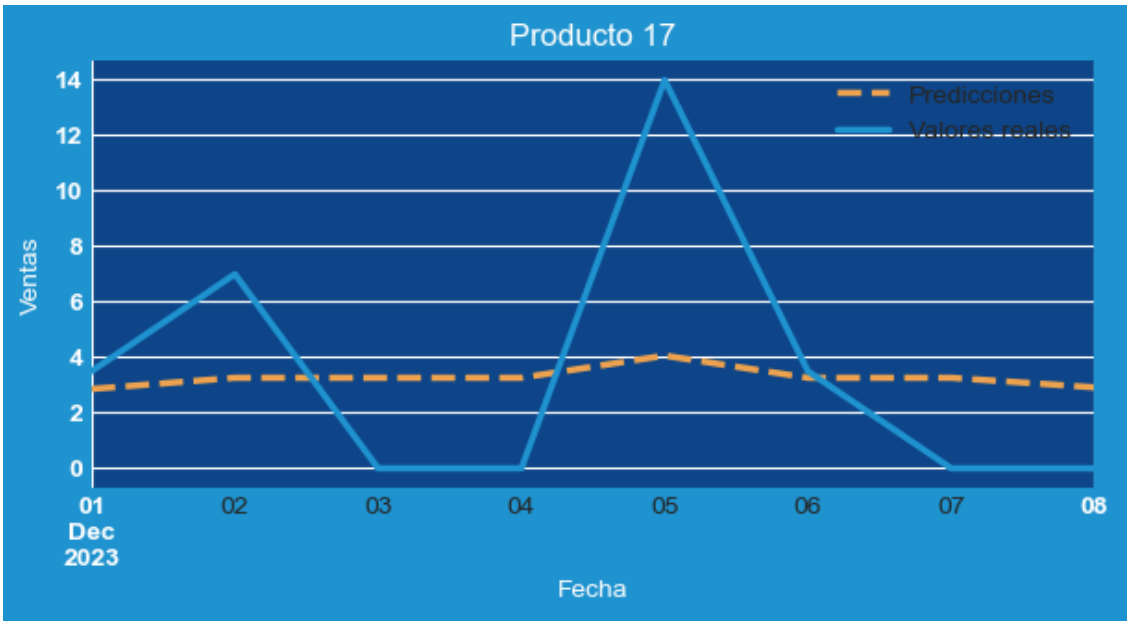

Producto 186
Predicciones
Valores reales
Ventas
0
2
4
6
8
10
12
14
01
Dec
2023
02
03
04
05
06
07
08
Fecha

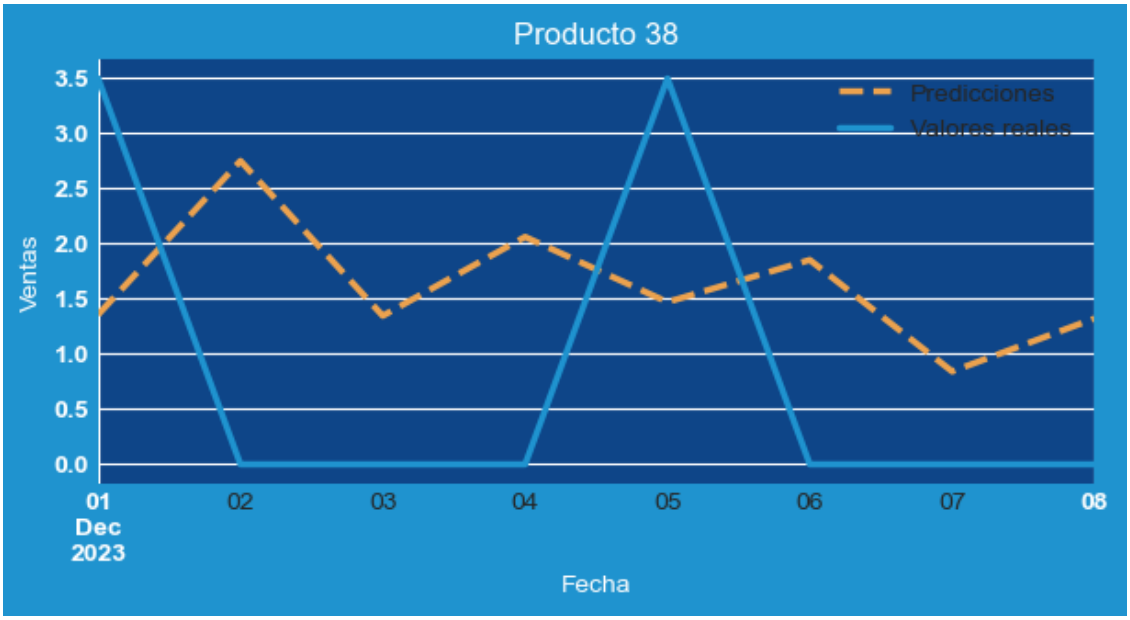

Producto 20
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
Dec
2023
02
03
04
05
06
07
08
Fecha

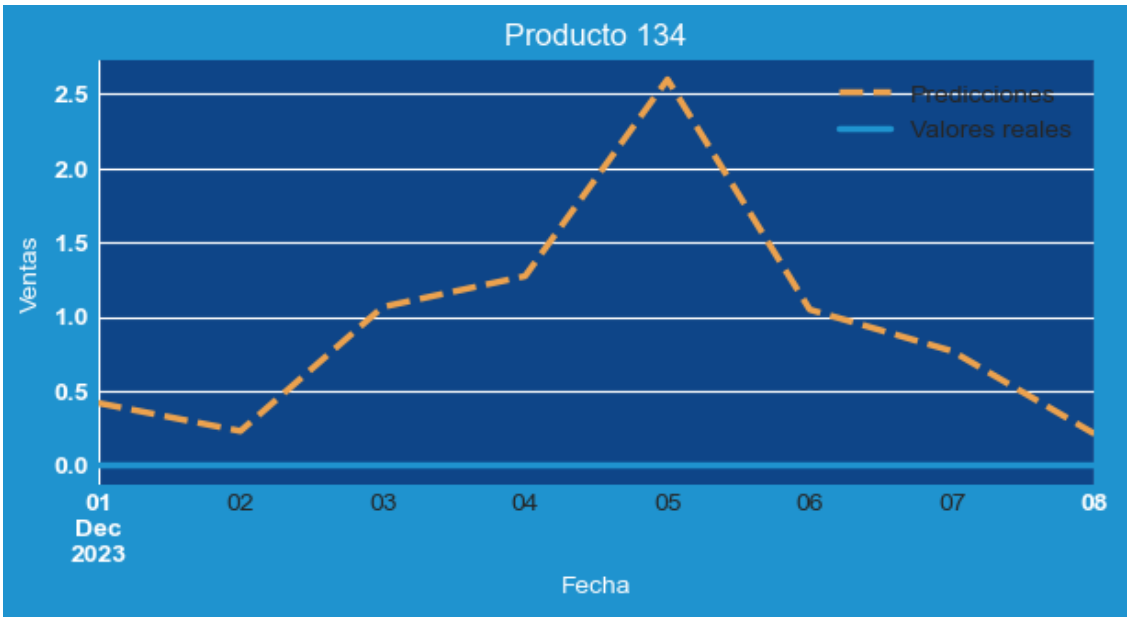

Producto 33
Predicciones
Valores reales
Ventas
14
12
10
8
6
4
2
0
01
Dec
2023
02
03
04
05
06
07
08
Fecha

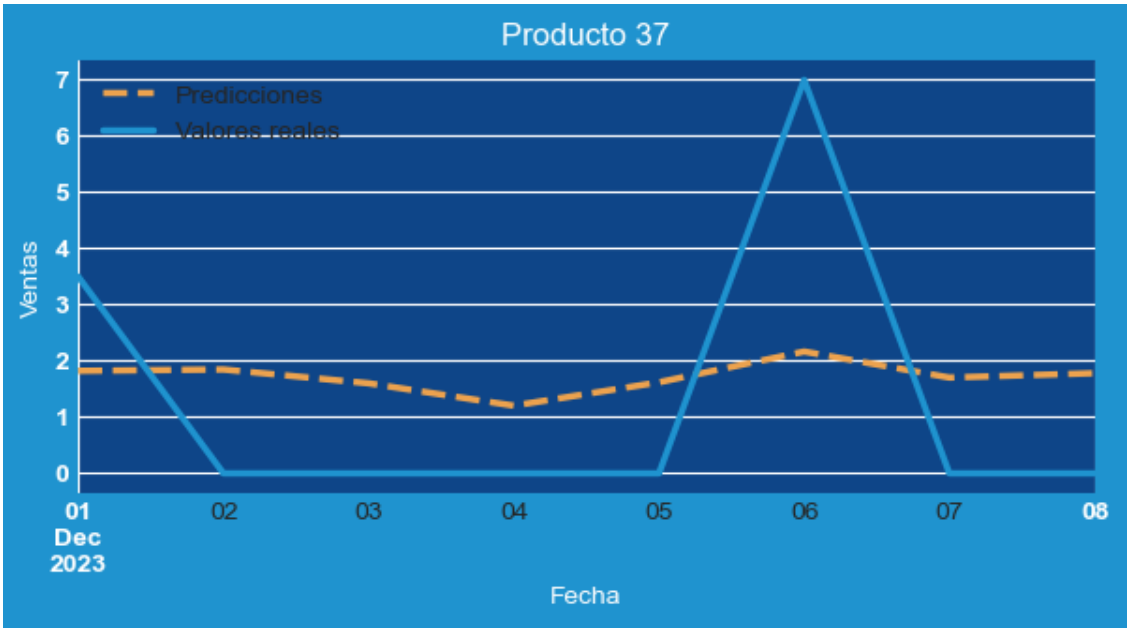

Producto 245
Predicciones
Valores reales
Ventas
0
1
2
3
4
5
6
7
01
Dec
2023
02
03
04
05
06
07
08
Fecha

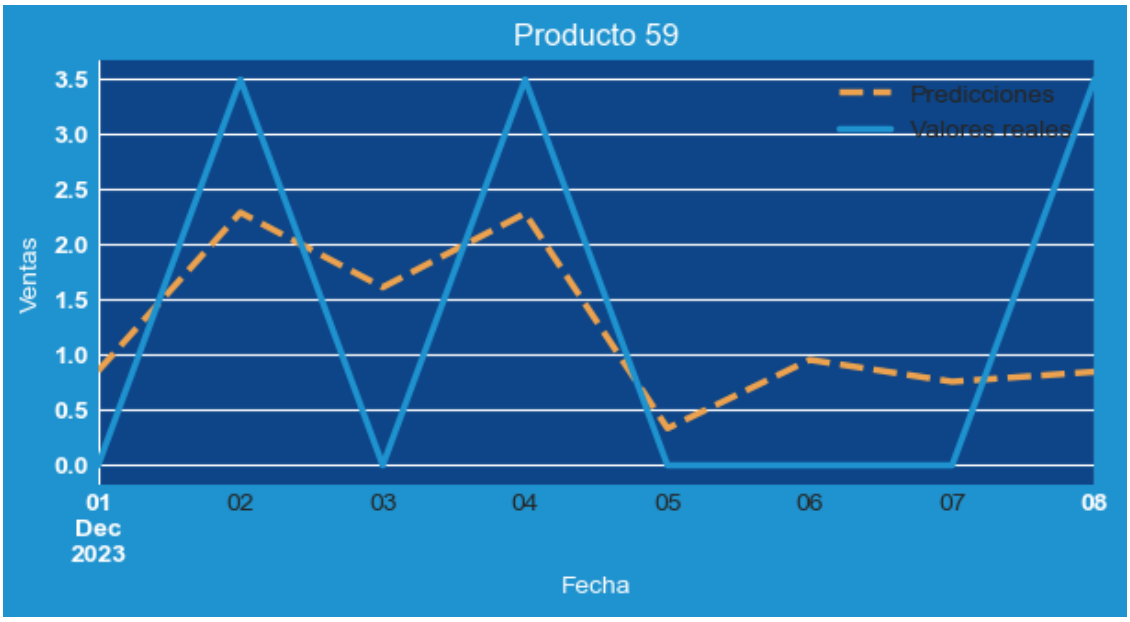

Producto 16
Predicciones
Valores reales
Ventas
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
01
Dec
2023
02
03
04
05
06
07
08
Fecha

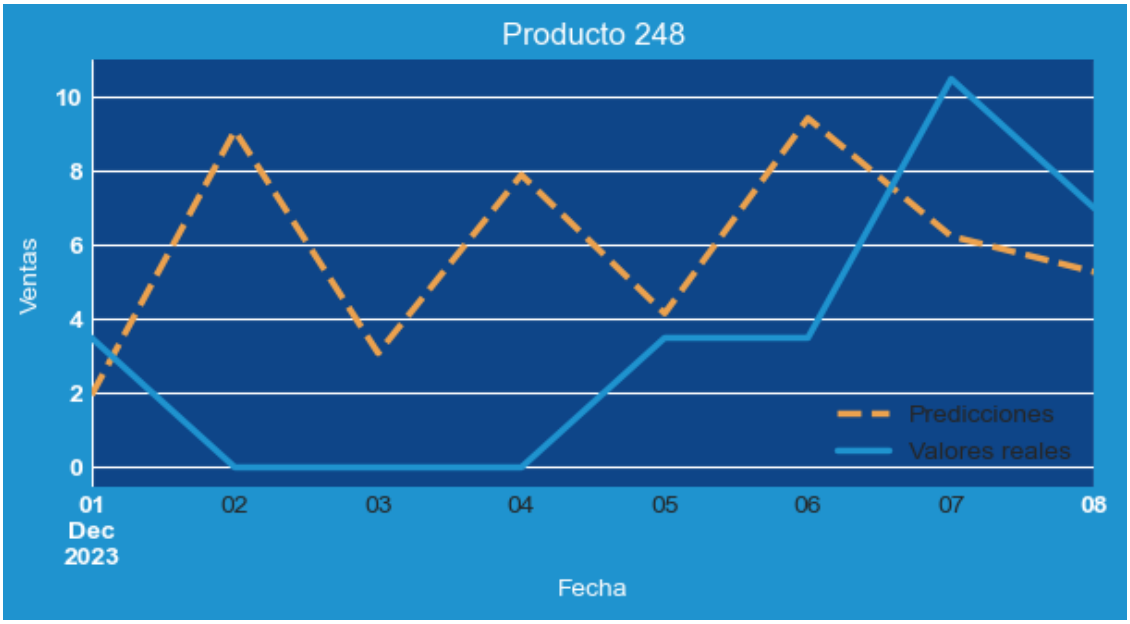

Producto 17
Predicciones
Valores reales
Ventas
14
12
10
8
6
4
2
0
01
Dec
2023
02
03
04
05
06
07
08
Fecha

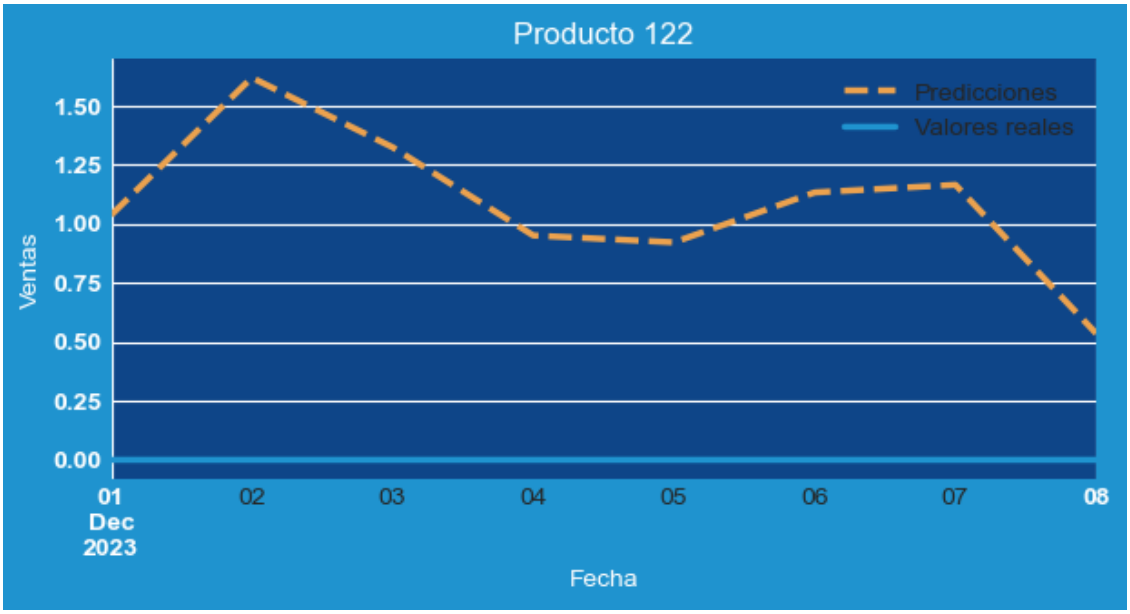

Producto 38
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
Dec
2023
02
03
04
05
06
07
08
Fecha


Producto 134
Predicciones
Valores reales
Ventas
2.5
2.0
1.5
1.0
0.5
0.0
01
Dec
2023
02
03
04
05
06
07
08
Fecha


Producto 37
Predicciones
Valores reales
Ventas
7
6
5
4
3
2
1
0
01
Dec
2023
02
03
04
05
06
07
08
Fecha


Producto 59
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
Fecha


Producto 248
10
8
6
4
2
0
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
Fecha



[ ]:
```
resid = data["Producto 0"].loc["2023-10-21":"2023-10-28"] - predicts
```

[ ]:
```
week_pred.to_csv("predicciones_semanales_HGB.csv")
month_pred.to_csv("predicciones_mensuales_HGB.csv")
data.to_csv("datatop20.csv")
```

[ ]:
```

```

[ ]:

	Producto 273	Producto 0	Producto 1	Producto 5	Producto 8	\
2023-12-01	191.568635	44.241920	26.184245	26.074028	12.488096	
2023-12-02	275.633156	46.925787	26.008929	20.725320	17.719338	
2023-12-03	205.372151	34.773765	22.449637	22.052071	13.164492	
2023-12-04	247.810189	50.755710	27.934841	22.948310	4.455952	
2023-12-05	232.683095	41.185338	26.939218	21.064282	13.115563	
2023-12-06	188.224421	38.913195	28.384194	18.091801	11.517635	
2023-12-07	182.533458	38.331289	25.903319	18.613266	12.298853	
2023-12-08	187.592701	38.440135	25.903319	15.872234	14.070452	
2023-12-09	234.984092	37.902690	26.774874	17.228063	15.596747	
2023-12-10	216.779599	32.201267	26.488295	19.437686	13.669274	
2023-12-11	249.974796	54.635387	27.837849	22.777097	18.126364	
2023-12-12	223.101762	46.750805	27.073017	23.787848	13.115563	
2023-12-13	216.733205	43.558633	28.503288	27.289403	12.147120	
2023-12-14	217.154813	36.571764	27.403040	21.224439	14.178806	
2023-12-15	211.305301	40.107968	26.072431	35.189572	15.691226	
2023-12-16	281.908942	43.621405	26.167755	25.112460	17.130006	
2023-12-17	238.512364	42.386933	20.420017	21.726030	20.159162	
2023-12-18	302.890780	43.811118	27.964438	27.962468	10.665316	

2023-12-19	188.579399	41.185338	26.939218	23.558058	14.962108
2023-12-20	221.568520	34.773765	28.661276	15.977934	12.550490
2023-12-21	211.786207	39.679339	26.470359	18.450799	11.896032
2023-12-22	182.142781	36.797155	26.072431	19.690732	14.070452
2023-12-23	244.860702	40.451500	26.167755	18.036483	19.134814
2023-12-24	200.721037	40.402294	20.420017	19.093844	21.089978
2023-12-25	257.111622	57.979763	26.837474	25.778746	10.536435
2023-12-26	206.236211	46.750805	26.746486	21.064282	14.962108
2023-12-27	197.355147	43.068882	28.384194	20.817497	12.550490
2023-12-28	200.936543	43.909711	26.920555	22.332116	13.759193
2023-12-29	192.901699	45.392082	26.072431	32.074153	15.691226
2023-12-30	279.709095	49.826862	26.167755	31.398807	17.130006
2023-12-31	246.700116	41.565085	20.420017	21.726030	16.891349

	Producto 21	Producto 12	Producto 22	Producto 186	Producto 20	\
2023-12-01	0.787651	3.429695	3.927858	7.324022	0.984379	
2023-12-02	1.835158	3.591847	5.430823	10.089389	0.469039	
2023-12-03	4.247331	3.823965	3.530881	7.900793	0.000000	
2023-12-04	1.598692	3.459292	3.654907	8.743454	1.275361	
2023-12-05	1.922587	3.407166	3.663162	4.303401	0.955947	
2023-12-06	4.370891	3.459292	3.864032	6.300994	2.021659	
2023-12-07	1.571765	3.505475	3.727272	3.758627	0.492125	
2023-12-08	2.038564	3.429695	3.899998	6.572098	0.781859	
2023-12-09	2.481960	3.601068	3.938911	9.211254	1.111823	
2023-12-10	3.239626	3.848183	2.836979	8.786546	2.881347	
2023-12-11	1.598692	3.459292	3.504267	8.886675	1.916004	
2023-12-12	1.922587	3.406078	3.578214	6.513655	1.025181	
2023-12-13	6.895957	3.407166	4.140249	6.261653	0.863215	
2023-12-14	2.400717	3.236893	3.788007	4.597474	0.101235	
2023-12-15	1.518902	3.427099	3.776033	7.324022	0.000000	
2023-12-16	2.200181	3.591847	5.430823	8.817117	2.349684	
2023-12-17	2.856256	3.890980	3.079360	8.047114	1.438940	
2023-12-18	1.316913	3.407166	3.488994	8.813059	1.242320	
2023-12-19	1.922587	3.435073	3.578214	7.687903	1.025181	
2023-12-20	3.947257	3.435073	4.124977	6.300994	2.021659	
2023-12-21	1.571765	3.061190	3.727272	7.518311	1.768624	
2023-12-22	1.999346	3.371246	3.899998	6.572098	0.781859	
2023-12-23	2.512175	3.429181	3.938911	9.161567	1.442220	
2023-12-24	2.643176	3.421425	2.836979	7.900793	2.881347	
2023-12-25	2.556159	3.407166	3.504267	8.886675	1.916004	
2023-12-26	3.106679	3.407166	3.578214	6.547640	1.025181	
2023-12-27	2.503868	3.407166	4.124977	8.053186	1.028703	
2023-12-28	1.658591	3.273835	3.788007	5.642710	2.222481	
2023-12-29	1.729786	3.371246	3.776033	7.699010	0.000000	
2023-12-30	2.539430	3.466619	5.430823	10.089389	2.349684	
2023-12-31	2.568839	3.464222	3.079360	9.502914	1.438940	

	Producto 33	Producto 245	Producto 16	Producto 17	Producto 38	\
2023-12-01	3.978594	4.440104	2.622576	2.860921	1.357833	
2023-12-02	5.498016	3.170399	2.825751	3.263814	2.751930	
2023-12-03	4.637185	2.289104	2.718295	3.263814	1.346992	
2023-12-04	2.932914	4.440104	2.981883	3.263814	2.062868	
2023-12-05	3.978594	3.961679	2.622576	4.062701	1.471656	
2023-12-06	3.978594	3.429625	2.622576	3.263814	1.853515	
2023-12-07	3.978594	2.289104	2.622576	3.263814	0.842675	
2023-12-08	3.978594	4.440915	2.622576	2.918250	1.322817	
2023-12-09	4.647223	3.429625	3.442045	3.263814	2.757661	
2023-12-10	4.669825	4.274207	2.718295	3.263814	1.346992	
2023-12-11	3.978594	4.440915	2.815799	3.263814	2.540752	
2023-12-12	3.978594	3.602029	2.622576	4.062701	1.471656	
2023-12-13	3.978594	3.170399	2.622576	3.263814	2.160392	
2023-12-14	3.978594	2.289104	2.622576	3.263814	1.156967	
2023-12-15	3.978594	3.170399	2.622576	2.903388	1.357833	
2023-12-16	5.498016	2.289104	3.242054	3.263814	3.449952	
2023-12-17	4.669825	2.289104	2.718295	3.263814	1.376128	
2023-12-18	3.978594	2.289104	2.842814	3.263814	2.071075	
2023-12-19	3.978594	3.961679	2.622576	4.062701	1.471656	
2023-12-20	3.978594	3.069976	2.622576	3.263814	1.853515	
2023-12-21	3.978594	2.289104	2.622576	3.263814	0.963118	
2023-12-22	3.978594	4.440104	2.622576	2.918250	1.322817	
2023-12-23	4.647223	4.440104	3.442045	3.263814	2.757661	
2023-12-24	4.669825	4.273396	2.718295	3.263814	1.346992	
2023-12-25	3.978594	4.440104	2.981883	3.263814	2.540752	
2023-12-26	3.978594	4.667907	2.622576	4.062701	1.471656	
2023-12-27	3.978594	4.440104	2.622576	3.263814	1.853515	
2023-12-28	3.978594	4.440104	2.622576	3.263814	1.156967	
2023-12-29	3.978594	4.440104	2.622576	2.903388	1.357833	
2023-12-30	5.498016	4.440104	3.242054	3.263814	3.449952	
2023-12-31	4.669825	4.273396	2.718295	3.263814	1.376128	

	Producto 134	Producto 37	Producto 59	Producto 248	Producto 122
2023-12-01	0.419574	1.824171	0.856879	1.954742	1.035936
2023-12-02	0.230753	1.847885	2.293071	9.100468	1.617229
2023-12-03	1.065830	1.602324	1.614743	3.088791	1.322783
2023-12-04	1.272860	1.203935	2.285042	7.906694	0.949515
2023-12-05	2.596176	1.620143	0.335706	4.162075	0.920300
2023-12-06	1.047474	2.167168	0.956964	9.437604	1.131519
2023-12-07	0.768602	1.704188	0.759117	6.247438	1.163769
2023-12-08	0.211841	1.781791	0.848580	5.282371	0.534871
2023-12-09	0.342657	1.581131	1.613565	6.831312	1.341793
2023-12-10	1.065830	1.906910	2.500628	6.488423	1.322783
2023-12-11	1.272860	1.203935	2.489122	7.906694	0.949515
2023-12-12	2.596176	1.475594	0.854379	8.218482	0.920300
2023-12-13	1.622609	1.801570	0.917737	9.208536	1.041755

2023-12-14	0.083628	1.600685	1.159407	6.867339	1.588550
2023-12-15	0.419574	2.245843	0.856879	1.744518	1.035936
2023-12-16	0.230753	1.847885	1.674439	6.611136	1.617229
2023-12-17	2.508414	2.173583	0.660028	5.167270	1.171812
2023-12-18	1.315572	1.882273	1.388573	7.954511	0.707182
2023-12-19	2.596176	1.744374	0.854379	4.162075	0.920300
2023-12-20	1.047474	2.167168	0.956964	7.708876	1.131519
2023-12-21	0.768602	1.667964	0.963190	6.424329	1.163769
2023-12-22	0.211841	1.347252	0.848580	3.890502	0.534871
2023-12-23	0.342657	1.473019	1.179633	5.140814	1.341793
2023-12-24	1.065830	1.822427	2.500628	6.488423	1.322783
2023-12-25	1.272860	1.361188	1.693503	7.906694	0.949515
2023-12-26	2.956111	1.475594	0.854379	8.218482	0.920300
2023-12-27	1.047474	1.716491	0.956964	9.437604	1.131519
2023-12-28	0.083628	1.600685	1.159407	6.867339	1.588550
2023-12-29	0.419574	1.881297	0.856879	2.798389	1.035936
2023-12-30	0.230753	2.712065	1.674439	9.835723	1.617229
2023-12-31	2.508414	2.260217	0.660028	5.167270	1.171812

sarimax20

March 15, 2024

[ ]:
```
# Libraries
#␣
 ↪==========================================================================================
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-darkgrid')
from statsmodels.graphics.tsaplots import plot_acf

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import  HistGradientBoostingRegressor
from lightgbm import LGBMRegressor


# pmdarima
from pmdarima import ARIMA
from pmdarima import auto_arima

# statsmodels
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
# skforecast
from skforecast.Sarimax import Sarimax
from skforecast.ForecasterSarimax import ForecasterSarimax
from skforecast.model_selection_sarimax import backtesting_sarimax
from skforecast.model_selection_sarimax import grid_search_sarimax
from sklearn.metrics import mean_absolute_error,␣
 ↪mean_absolute_percentage_error, mean_squared_error, r2_score

import warnings
```

```
c:\Users\progra.DESKTOP-GV4Q93K\miniconda3\envs\last\lib\site-
packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
```

```
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1 Se leen y transforman los datos

[ ]:
```python
df = pd.read_csv("Final-db.csv")
# Eliminar los espacios adicionales en las fechas
df['date'] = df['date'].str.strip()

# Mapeo de los nombres de los meses en español a los nombres en inglés
meses = {
    'ene': 'Jan',
    'feb': 'Feb',
    'mar': 'Mar',
    'abr': 'Apr',
    'may': 'May',
    'jun': 'Jun',
    'jul': 'Jul',
    'ago': 'Aug',
    'sep': 'Sep',
    'oct': 'Oct',
    'nov': 'Nov',
    'dic': 'Dec'
}

# Función para convertir los nombres de los meses en español a inglés
def convertir_meses(fecha):
    for mes_es, mes_en in meses.items():
        fecha = fecha.replace(mes_es, mes_en)
    return fecha

# Aplicar la función a la columna de fecha
df['date'] = df['date'].apply(convertir_meses)

# Convertir la columna de fecha a datetime
df['date'] = pd.to_datetime(df['date'], format='%d %b %Y')
```

Se crean las variables exogenas y se limpia el dataframe

[ ]:
```python
df = df.drop(columns=['Unnamed: 0',"index","Total libre de␣
 ↪impuestos","Indefinido total $","Indefinido ctdad"])
df = df.rename(columns={"date":"Fecha","Encoded Products":"Producto"})
df['Fecha'] = pd.to_datetime(df['Fecha'], format="mixed")
df.columns = df.columns.str.replace("total $", "Precio por unidad")
df.columns = df.columns.str.replace("ctdad", "Cantidad")
def div(numerator, denominator):
```

```
  return lambda row: 0.0 if row[denominator] == 0 else float(row[numerator]/
 ↪row[denominator])
for i in range(2, len(df.columns)-1,2):
    df[df.columns[i]] = df.apply(div(df.columns[i], df.columns[i+1]), axis=1)
df = df.set_index('Fecha')
```

De igual manera se eliminan los valores con menos de 50 ventas en 2023

[ ]:
```
data = pd.DataFrame()
for i in df["Producto"].unique():
        x =pd.DataFrame(df[df["Producto"]==i].loc[:].groupby("Fecha").
 ↪sum()["Ctdad Ordenada"].asfreq("D", fill_value=0)).rename(columns={"Ctdad␣
 ↪Ordenada":i})
        data = pd.concat([data,x], axis=1)
data.fillna(0, inplace=True)
exog = pd.DataFrame()
exog["Mes"] = data.index.month
data["Dia"] = data.index.day
exog["Dia de la semana"] = data.index.dayofweek
exog.index = data.index
exog = pd.get_dummies(exog, columns=["Mes","Dia de la semana"],dtype=int)
exog = pd.concat([exog, data["Dia"]], axis=1)
exog["Dia"].replace(to_replace=[13,14,15,16,17,18,28,29,30,31,1,2], value=1,␣
 ↪inplace=True)
exog["Dia"].
 ↪replace(to_replace=[3,4,5,6,7,8,9,10,11,12,19,20,21,22,23,24,25,26,27,28],␣
 ↪value=0, inplace=True)
for i in data.columns:
    if data.loc["2023-01-01":,i].sum() < 50:
        data = data.drop(columns=i,axis=1)
data.drop(columns="Dia", inplace=True)
data = data[data.sum().sort_values(ascending=False).index[0:20]]
data.head()
```

[ ]:
```
            Producto 273  Producto 0  Producto 1  Producto 5  Producto 8  \
Fecha
2022-01-02           0.0        10.5        35.0        14.0        21.0
2022-01-03           0.0        38.5        17.5        21.0         0.0
2022-01-04           0.0        56.0         3.5        21.0        10.5
2022-01-05           0.0        49.0        14.0        10.5        10.5
2022-01-06           0.0        17.5         7.0        10.5         3.5

            Producto 21  Producto 12  Producto 22  Producto 186  Producto 20  \
Fecha
2022-01-02          7.0          7.0          0.0           0.0          3.5
2022-01-03          7.0         10.5          3.5           0.0          0.0
2022-01-04          3.5          3.5          7.0           0.0          3.5
```

```
2022-01-05         0.0          7.0          3.5          0.0          0.0
2022-01-06         3.5          7.0          0.0          0.0          0.0

            Producto 33  Producto 245  Producto 16  Producto 17  Producto 38  \
Fecha
2022-01-02          0.0           0.0          3.5          0.0          0.0
2022-01-03          0.0           0.0          3.5          0.0          0.0
2022-01-04          7.0           0.0          3.5          0.0          0.0
2022-01-05          0.0           0.0          3.5          3.5          0.0
2022-01-06          0.0           0.0          7.0          3.5          0.0

            Producto 134  Producto 37  Producto 59  Producto 248  Producto 122
Fecha
2022-01-02           0.0          7.0          0.0           0.0           0.0
2022-01-03           0.0          0.0          3.5           0.0           0.0
2022-01-04           0.0          3.5          0.0           0.0           0.0
2022-01-05           0.0          0.0          0.0           0.0           0.0
2022-01-06           0.0          0.0          3.5           0.0           0.0
```

[ ]:
```
exog.head()
```

[ ]:
```
            Mes_1  Mes_2  Mes_3  Mes_4  Mes_5  Mes_6  Mes_7  Mes_8  Mes_9  \
Fecha
2022-01-02      1      0      0      0      0      0      0      0      0
2022-01-03      1      0      0      0      0      0      0      0      0
2022-01-04      1      0      0      0      0      0      0      0      0
2022-01-05      1      0      0      0      0      0      0      0      0
2022-01-06      1      0      0      0      0      0      0      0      0

            Mes_10  Mes_11  Mes_12  Dia de la semana_0  Dia de la semana_1  \
Fecha
2022-01-02       0       0       0                   0                   0
2022-01-03       0       0       0                   1                   0
2022-01-04       0       0       0                   0                   1
2022-01-05       0       0       0                   0                   0
2022-01-06       0       0       0                   0                   0

            Dia de la semana_2  Dia de la semana_3  Dia de la semana_4  \
Fecha
2022-01-02                   0                   0                   0
2022-01-03                   0                   0                   0
2022-01-04                   0                   0                   0
2022-01-05                   1                   0                   0
2022-01-06                   0                   1                   0

            Dia de la semana_5  Dia de la semana_6  Dia
Fecha
```

2022-01-02	0	1	1
2022-01-03	0	0	0
2022-01-04	0	0	0
2022-01-05	0	0	0
2022-01-06	0	0	0

2 Se divide el test en validacion, test y entrenamiento

[ ]:
```
# Split data into train-validation-test
#␣
 ↪==============================================================================
end_train = '2023-10-30'
end_val = '2023-11-30'

data_train = data.loc[:end_train, :].copy()
data_val   = data.loc[end_train:end_val, :].copy()
data_test  = data.loc[end_val:, :].copy()
exog_train = exog.loc[:end_train, :].copy()
exog_val   = exog.loc[end_train:end_val, :].copy()
exog_test  = exog.loc[end_val:, :].copy()

print(f"Train dates      : {data_train.index.min()} --- {data_train.index.
 ↪max()}  (n={len(data_train)})")
print(f"Validation dates : {data_val.index.min()} --- {data_val.index.max()} ␣
 ↪(n={len(data_val)})")
print(f"Test dates       : {data_test.index.min()} --- {data_test.index.max()} ␣
 ↪(n={len(data_test)})")
```

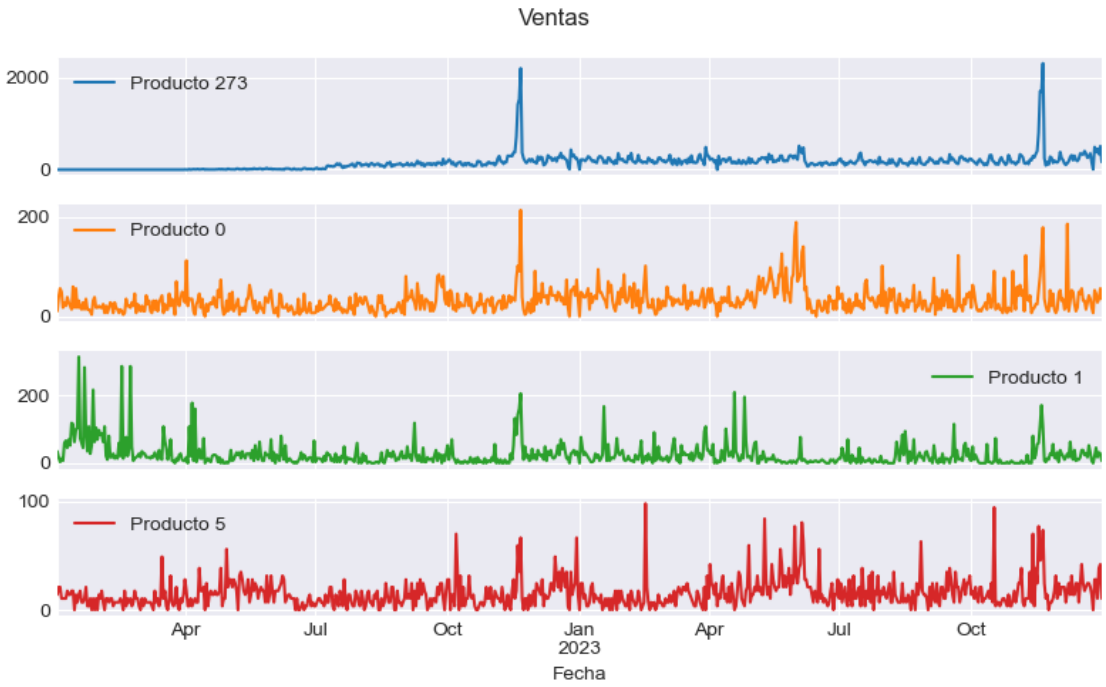
```
Train dates      : 2022-01-02 00:00:00 --- 2023-10-30 00:00:00  (n=667)
Validation dates : 2023-10-30 00:00:00 --- 2023-11-30 00:00:00  (n=32)
Test dates       : 2023-11-30 00:00:00 --- 2023-12-31 00:00:00  (n=32)
```

Algunas de los trends que hay

[ ]:
```
# Plot time series
#␣
 ↪==============================================================================
fig, ax = plt.subplots(figsize=(8, 5))
data.iloc[:, :4].plot(
    legend   = True,
    subplots = True,
    sharex   = True,
    title    = 'Ventas',
    ax       = ax,
)
fig.tight_layout();
```

```
C:\Users\progra.DESKTOP-
```

```
GV4Q93K\AppData\Local\Temp\ipykernel_19048\1578984712.py:4: UserWarning: To
output multiple subplots, the figure containing the passed axes is being
cleared.
  data.iloc[:, :4].plot(
```

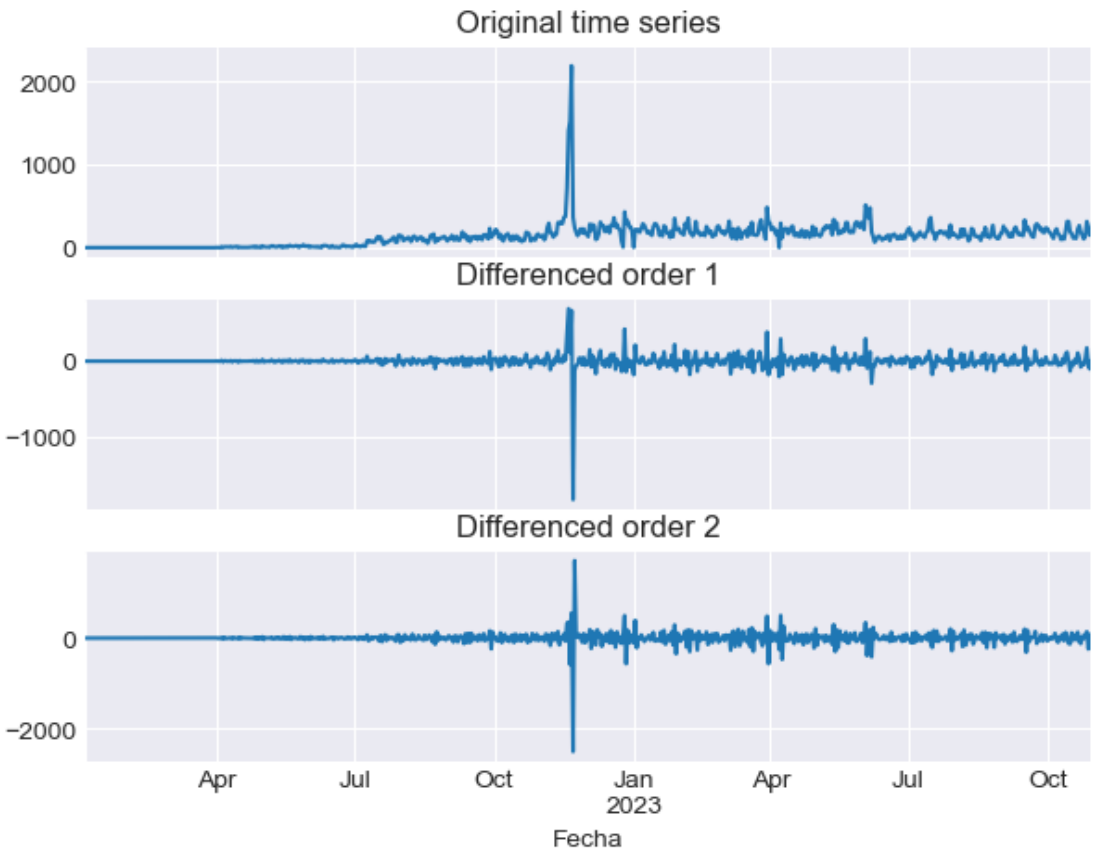


3 Se hacen las pruebas de estacionalidad

[ ]:
```python
# Test stationarity
# ============================================================================
warnings.filterwarnings("ignore")

data_diff_1 = data_train.loc[:,"Producto 273"].diff().dropna()
data_diff_2 = data_diff_1.diff().dropna()

print('Test stationarity for original series')
print('-------------------------------------')
adfuller_result = adfuller(data.loc[:,"Producto 273"])
kpss_result = kpss(data.loc[:,"Producto 273"])
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')
print(f'KPSS Statistic: {kpss_result[0]}, p-value: {kpss_result[1]}')

print('\nTest stationarity for differenced series (order=1)')
print('---------------------------------------------------')
adfuller_result = adfuller(data_diff_1)
```

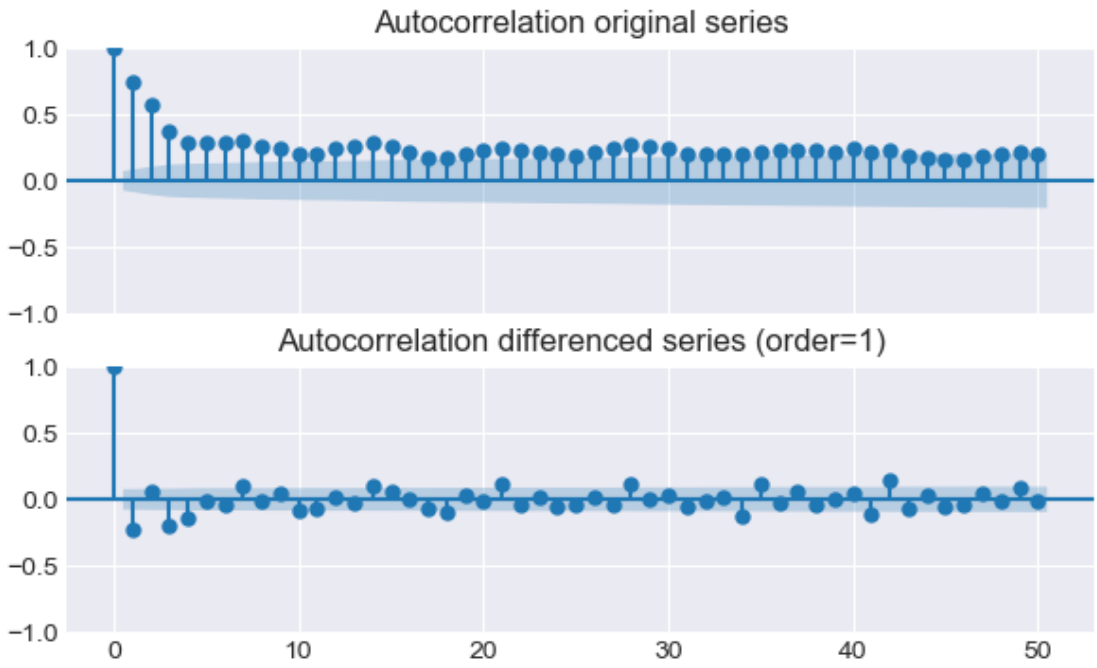
```
kpss_result = kpss(data.loc[:,"Producto 273"].diff().dropna())
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')
print(f'KPSS Statistic: {kpss_result[0]}, p-value: {kpss_result[1]}')

print('\nTest stationarity for differenced series (order=2)')
print('-------------------------------------------------')
adfuller_result = adfuller(data_diff_2)
kpss_result = kpss(data.loc[:,"Producto 273"].diff().diff().dropna())
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')
print(f'KPSS Statistic: {kpss_result[0]}, p-value: {kpss_result[1]}')

warnings.filterwarnings("default")

# Plot series
# ============================================================================
fig, axs = plt.subplots(nrows=3, ncols=1, figsize=(7, 5), sharex=True)
data.loc[:,"Producto 273"].plot(ax=axs[0], title='Original time series')
data_diff_1.plot(ax=axs[1], title='Differenced order 1')
data_diff_2.plot(ax=axs[2], title='Differenced order 2');
```

```
Test stationarity for original series
-------------------------------------
ADF Statistic: -6.8466637778759205, p-value: 1.7358130411528209e-09
KPSS Statistic: 2.372477848531467, p-value: 0.01

Test stationarity for differenced series (order=1)
--------------------------------------------------
ADF Statistic: -9.642266345299063, p-value: 1.5098298316387997e-16
KPSS Statistic: 0.17723174229793687, p-value: 0.1

Test stationarity for differenced series (order=2)
--------------------------------------------------
ADF Statistic: -12.074263451651005, p-value: 2.309630710552708e-22
KPSS Statistic: 0.09212782654564701, p-value: 0.1
```



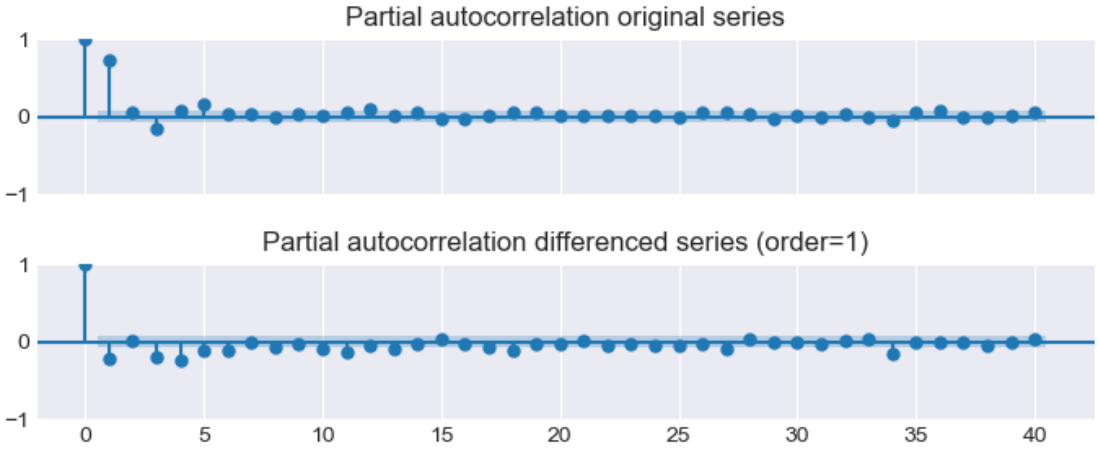
Y de autocorrelacion

[ ]:
```
# Autocorrelation plot for original and differenced series
# ================================================================================
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(7, 4), sharex=True)
plot_acf(data.loc[:,"Producto 273"], ax=axs[0], lags=50, alpha=0.05)
axs[0].set_title('Autocorrelation original series')
plot_acf(data_diff_1, ax=axs[1], lags=50, alpha=0.05)
axs[1].set_title('Autocorrelation differenced series (order=1)');
```



De igual manera de correlación parcial

[ ]:
```
# Partial autocorrelation plot for original and differenced series
# ==============================================================================
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(7, 3), sharex=True)
plot_pacf(data.loc[:,"Producto 273"], ax=axs[0], lags=40, alpha=0.05)
axs[0].set_title('Partial autocorrelation original series')
plot_pacf(data_diff_1, ax=axs[1], lags=40, alpha=0.05)
axs[1].set_title('Partial autocorrelation differenced series (order=1)');
plt.tight_layout();
```



[ ]:
```
from tqdm import tqdm
from functools import partialmethod
```

3.1 Se obtienen los mejores hiperparametros atraves de autosarima y se guardan en un archivo

[ ]:
```
# Hyperparameter search and backtesting of each item's model
#␣
  ↪==========================================================================================
items = []
mae_values  = []
dictes = {}
models = []
lags_grid = [7, 14, 21]
param_grid = {'order': [(1, 0, 1), (12, 2, 0)],
              'seasonal_order': [(1, 0, 1, 54),(1, 0, 1, 12)],
              'trend': [None, 'n', 'c']}

for i, item in enumerate(data.columns):
    model = auto_arima(
            y                 = data.loc[:end_val, item],
            start_p           = 0,
            start_q           = 0,
            max_p             = 3,
            max_q             = 3,
            seasonal          = True,
            test              = 'adf',
            exog = exog.loc[:end_val],
            m                 = 12, # Seasonal period
            d                 = None, # The algorithm will determine 'd'
            D                 = None, # The algorithm will determine 'D'
            trace             = True,
            error_action      = 'ignore',
            suppress_warnings = True,
            stepwise          = True
        )
    models.append(model)


    uni_series_mae = pd.Series(
                     data  = mae_values,
                     index = items,
                     name  = 'uni_series_mae'
                 )
```

```
Performing stepwise search to minimize aic
```

```
ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=9203.028, Time=0.89 sec
ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=9302.979, Time=0.02 sec
ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=8738.579, Time=0.43 sec
ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=8963.173, Time=0.54 sec
ARIMA(0,0,0)(0,0,0)[12]             : AIC=9634.031, Time=0.02 sec
ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=8737.793, Time=0.04 sec
ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=8738.634, Time=0.16 sec
ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=8740.613, Time=0.37 sec
ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=8738.072, Time=0.16 sec
ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=8738.578, Time=0.29 sec
ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=8979.485, Time=0.26 sec
ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=8733.026, Time=0.41 sec
ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=8734.305, Time=1.06 sec
ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=8734.325, Time=1.03 sec
ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=8736.094, Time=0.55 sec
ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=8722.114, Time=0.53 sec
ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=8722.265, Time=1.27 sec
ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=8722.402, Time=1.07 sec
ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=8719.585, Time=1.45 sec
ARIMA(3,0,1)(2,0,1)[12] intercept   : AIC=8724.653, Time=4.07 sec
ARIMA(3,0,1)(1,0,2)[12] intercept   : AIC=8723.910, Time=3.31 sec
ARIMA(3,0,1)(0,0,2)[12] intercept   : AIC=8723.770, Time=2.82 sec
ARIMA(3,0,1)(2,0,0)[12] intercept   : AIC=8723.305, Time=4.26 sec
ARIMA(3,0,1)(2,0,2)[12] intercept   : AIC=inf, Time=4.38 sec
ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=8724.229, Time=0.55 sec
ARIMA(3,0,2)(1,0,1)[12] intercept   : AIC=8707.340, Time=1.65 sec
ARIMA(3,0,2)(0,0,1)[12] intercept   : AIC=8709.293, Time=2.20 sec
ARIMA(3,0,2)(1,0,0)[12] intercept   : AIC=8708.013, Time=2.10 sec
ARIMA(3,0,2)(2,0,1)[12] intercept   : AIC=8707.557, Time=4.04 sec
ARIMA(3,0,2)(1,0,2)[12] intercept   : AIC=8707.626, Time=4.00 sec
ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=8707.933, Time=0.79 sec
ARIMA(3,0,2)(0,0,2)[12] intercept   : AIC=8709.444, Time=4.58 sec
ARIMA(3,0,2)(2,0,0)[12] intercept   : AIC=8711.459, Time=4.31 sec
ARIMA(3,0,2)(2,0,2)[12] intercept   : AIC=inf, Time=5.64 sec
ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=8704.284, Time=2.49 sec
ARIMA(2,0,2)(0,0,1)[12] intercept   : AIC=8710.246, Time=1.91 sec
ARIMA(2,0,2)(1,0,0)[12] intercept   : AIC=8710.004, Time=2.10 sec
ARIMA(2,0,2)(2,0,1)[12] intercept   : AIC=8709.210, Time=4.73 sec
ARIMA(2,0,2)(1,0,2)[12] intercept   : AIC=8705.156, Time=5.56 sec
ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=8710.487, Time=0.77 sec
ARIMA(2,0,2)(0,0,2)[12] intercept   : AIC=8711.075, Time=4.42 sec
ARIMA(2,0,2)(2,0,0)[12] intercept   : AIC=8710.088, Time=4.40 sec
ARIMA(2,0,2)(2,0,2)[12] intercept   : AIC=inf, Time=6.16 sec
ARIMA(1,0,2)(1,0,1)[12] intercept   : AIC=8700.083, Time=2.05 sec
ARIMA(1,0,2)(0,0,1)[12] intercept   : AIC=8708.311, Time=1.24 sec
ARIMA(1,0,2)(1,0,0)[12] intercept   : AIC=8708.052, Time=1.60 sec
ARIMA(1,0,2)(2,0,1)[12] intercept   : AIC=8702.383, Time=4.83 sec
ARIMA(1,0,2)(1,0,2)[12] intercept   : AIC=8707.490, Time=2.66 sec
```

```
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=8708.665, Time=0.72 sec
 ARIMA(1,0,2)(0,0,2)[12] intercept   : AIC=8709.081, Time=2.89 sec
 ARIMA(1,0,2)(2,0,0)[12] intercept   : AIC=8708.089, Time=2.86 sec
 ARIMA(1,0,2)(2,0,2)[12] intercept   : AIC=inf, Time=5.85 sec
 ARIMA(0,0,2)(1,0,1)[12] intercept   : AIC=8751.524, Time=1.89 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=8741.713, Time=0.84 sec
 ARIMA(1,0,3)(1,0,1)[12] intercept   : AIC=8702.341, Time=1.91 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=8926.041, Time=1.72 sec
 ARIMA(0,0,3)(1,0,1)[12] intercept   : AIC=8721.263, Time=1.27 sec
 ARIMA(2,0,3)(1,0,1)[12] intercept   : AIC=inf, Time=2.83 sec
 ARIMA(1,0,2)(1,0,1)[12]             : AIC=8715.125, Time=1.58 sec

Best model:  ARIMA(1,0,2)(1,0,1)[12] intercept
Total fit time: 128.644 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=0.71 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=6450.647, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=6331.794, Time=0.62 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=6366.779, Time=0.71 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=7142.258, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=6336.216, Time=0.17 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=6333.218, Time=2.82 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=6333.373, Time=1.34 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=6332.208, Time=0.60 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=6335.187, Time=3.41 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=6437.252, Time=0.55 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=6307.971, Time=1.15 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=6308.419, Time=0.21 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=6309.679, Time=2.59 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=6309.974, Time=0.88 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=6308.084, Time=0.63 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=6311.671, Time=2.24 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=6309.481, Time=1.09 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=6307.052, Time=1.62 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=6305.241, Time=0.85 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=6307.239, Time=1.75 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=6311.543, Time=2.02 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=6305.833, Time=0.33 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=6300.364, Time=0.83 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=6311.509, Time=0.63 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=6311.535, Time=0.59 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=6313.172, Time=1.21 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=6309.719, Time=0.10 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=inf, Time=0.93 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=6309.523, Time=0.68 sec
 ARIMA(3,0,1)(0,0,0)[12]             : AIC=6306.087, Time=0.31 sec

Best model:  ARIMA(3,0,1)(0,0,0)[12] intercept
```

```
Total fit time: 31.752 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=7009.566, Time=0.35 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=7013.494, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=6945.223, Time=0.36 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=6963.852, Time=0.36 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=7288.647, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=6946.560, Time=0.03 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=6946.871, Time=1.37 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=1.01 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=6945.062, Time=0.34 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=6946.460, Time=0.57 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=inf, Time=2.40 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=7007.573, Time=0.20 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=6918.145, Time=0.45 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=6917.009, Time=0.07 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=6918.241, Time=0.65 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=1.09 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=6904.788, Time=0.20 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=6906.603, Time=0.60 sec
 ARIMA(3,0,0)(0,0,1)[12] intercept   : AIC=6906.582, Time=0.35 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=inf, Time=1.48 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=6890.026, Time=0.73 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=6891.809, Time=1.76 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=6891.769, Time=1.67 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.59 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=6888.026, Time=0.36 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=6889.813, Time=0.89 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=6889.775, Time=0.83 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.33 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=6886.040, Time=0.27 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=6887.824, Time=0.72 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=6887.785, Time=0.61 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.20 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=6967.304, Time=0.22 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=6888.026, Time=0.38 sec
 ARIMA(0,0,2)(0,0,0)[12] intercept   : AIC=6943.754, Time=0.19 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=6886.943, Time=0.69 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=6899.465, Time=0.08 sec

Best model:  ARIMA(1,0,1)(0,0,0)[12] intercept
Total fit time: 25.436 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=5586.961, Time=0.95 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=5595.533, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=5555.053, Time=0.37 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=5564.497, Time=0.34 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=6170.133, Time=0.01 sec
```

```
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=5558.787, Time=0.07 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=5553.962, Time=1.43 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=5555.922, Time=1.44 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=5554.433, Time=0.91 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=5584.280, Time=0.78 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=5535.807, Time=1.42 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=5536.046, Time=0.53 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=5537.438, Time=3.04 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=5536.194, Time=1.10 sec
 ARIMA(3,0,0)(2,0,0)[12] intercept   : AIC=5533.810, Time=2.02 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=5533.119, Time=0.69 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=5533.963, Time=0.16 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=5533.973, Time=1.47 sec
 ARIMA(3,0,0)(0,0,1)[12] intercept   : AIC=5533.357, Time=0.44 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=5535.846, Time=1.90 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=5519.048, Time=1.52 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=5517.565, Time=0.64 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=5519.062, Time=1.37 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=5539.398, Time=1.69 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=5516.927, Time=0.44 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=5518.274, Time=1.16 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=5518.317, Time=1.03 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.49 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=5515.146, Time=0.34 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=5516.394, Time=0.71 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=5516.442, Time=0.73 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=1.36 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=5569.318, Time=0.15 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=5516.953, Time=0.43 sec
 ARIMA(0,0,2)(0,0,0)[12] intercept   : AIC=5549.940, Time=0.18 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=5537.471, Time=0.12 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=inf, Time=0.51 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=5529.152, Time=0.12 sec

Best model:  ARIMA(1,0,1)(0,0,0)[12] intercept
Total fit time: 33.086 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=5218.408, Time=0.86 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=5232.552, Time=0.03 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=5052.906, Time=0.22 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=5076.602, Time=0.32 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=5625.969, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=5051.128, Time=0.09 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=5052.913, Time=0.23 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=5047.487, Time=1.05 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=5048.338, Time=2.14 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=5048.166, Time=2.47 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=5054.782, Time=0.51 sec
```

```
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=5054.690, Time=0.77 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=inf, Time=2.69 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=5048.786, Time=1.21 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=5048.707, Time=1.30 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=5068.818, Time=1.06 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=5046.941, Time=1.73 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=5052.614, Time=1.08 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=5052.585, Time=1.29 sec
 ARIMA(2,0,1)(2,0,1)[12] intercept   : AIC=5046.907, Time=3.25 sec
 ARIMA(2,0,1)(2,0,0)[12] intercept   : AIC=5054.242, Time=3.05 sec
 ARIMA(2,0,1)(2,0,2)[12] intercept   : AIC=inf, Time=3.38 sec
 ARIMA(2,0,1)(1,0,2)[12] intercept   : AIC=5047.739, Time=2.76 sec
 ARIMA(1,0,1)(2,0,1)[12] intercept   : AIC=5049.477, Time=2.45 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=5049.589, Time=2.67 sec
 ARIMA(3,0,1)(2,0,1)[12] intercept   : AIC=5049.345, Time=3.37 sec
 ARIMA(2,0,2)(2,0,1)[12] intercept   : AIC=5048.887, Time=3.34 sec
 ARIMA(1,0,2)(2,0,1)[12] intercept   : AIC=5057.994, Time=2.91 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=5050.788, Time=3.24 sec
 ARIMA(3,0,2)(2,0,1)[12] intercept   : AIC=5051.297, Time=3.69 sec
 ARIMA(2,0,1)(2,0,1)[12]             : AIC=inf, Time=2.38 sec

Best model:  ARIMA(2,0,1)(2,0,1)[12] intercept
Total fit time: 55.548 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=5345.945, Time=0.48 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=5365.706, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=5151.209, Time=0.31 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=5226.215, Time=0.28 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=5556.693, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=5165.729, Time=0.06 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=5152.571, Time=0.79 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=5152.699, Time=0.56 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=5152.697, Time=0.23 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=5154.552, Time=1.26 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=5344.446, Time=0.24 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=5127.306, Time=0.40 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=5140.989, Time=0.13 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=5128.888, Time=1.08 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=5128.955, Time=0.63 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=5128.540, Time=0.27 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=5130.851, Time=1.82 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=5129.264, Time=0.54 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=5124.116, Time=0.67 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=5124.123, Time=0.42 sec
 ARIMA(2,0,1)(2,0,0)[12] intercept   : AIC=5125.828, Time=1.63 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=5125.838, Time=0.99 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=5125.232, Time=0.89 sec
 ARIMA(2,0,1)(2,0,1)[12] intercept   : AIC=5127.827, Time=3.02 sec
```

```
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=5123.692, Time=0.37 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=5133.877, Time=0.20 sec
 ARIMA(1,0,1)(2,0,0)[12] intercept   : AIC=5125.018, Time=1.03 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=5125.166, Time=0.72 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=5124.807, Time=0.33 sec
 ARIMA(1,0,1)(2,0,1)[12] intercept   : AIC=5126.918, Time=1.87 sec
 ARIMA(0,0,1)(1,0,0)[12] intercept   : AIC=5224.075, Time=0.24 sec
 ARIMA(1,0,2)(1,0,0)[12] intercept   : AIC=5121.710, Time=0.53 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=5128.287, Time=0.29 sec
 ARIMA(1,0,2)(2,0,0)[12] intercept   : AIC=5122.835, Time=1.48 sec
 ARIMA(1,0,2)(1,0,1)[12] intercept   : AIC=5123.070, Time=0.86 sec
 ARIMA(1,0,2)(0,0,1)[12] intercept   : AIC=5122.579, Time=0.52 sec
 ARIMA(1,0,2)(2,0,1)[12] intercept   : AIC=5124.761, Time=2.12 sec
 ARIMA(0,0,2)(1,0,0)[12] intercept   : AIC=5156.022, Time=0.41 sec
 ARIMA(2,0,2)(1,0,0)[12] intercept   : AIC=5118.693, Time=0.82 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=5128.527, Time=0.36 sec
 ARIMA(2,0,2)(2,0,0)[12] intercept   : AIC=5120.275, Time=1.94 sec
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=5120.308, Time=1.11 sec
 ARIMA(2,0,2)(0,0,1)[12] intercept   : AIC=5119.554, Time=0.70 sec
 ARIMA(2,0,2)(2,0,1)[12] intercept   : AIC=5122.286, Time=3.24 sec
 ARIMA(3,0,2)(1,0,0)[12] intercept   : AIC=5105.971, Time=1.50 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=5109.550, Time=0.81 sec
 ARIMA(3,0,2)(2,0,0)[12] intercept   : AIC=5107.476, Time=3.71 sec
 ARIMA(3,0,2)(1,0,1)[12] intercept   : AIC=5107.614, Time=1.76 sec
 ARIMA(3,0,2)(0,0,1)[12] intercept   : AIC=5106.386, Time=1.49 sec
 ARIMA(3,0,2)(2,0,1)[12] intercept   : AIC=5109.567, Time=3.58 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=5122.414, Time=1.00 sec
 ARIMA(3,0,3)(1,0,0)[12] intercept   : AIC=5107.033, Time=1.14 sec
 ARIMA(2,0,3)(1,0,0)[12] intercept   : AIC=5105.466, Time=0.88 sec
 ARIMA(2,0,3)(0,0,0)[12] intercept   : AIC=5111.310, Time=0.48 sec
 ARIMA(2,0,3)(2,0,0)[12] intercept   : AIC=5106.847, Time=2.88 sec
 ARIMA(2,0,3)(1,0,1)[12] intercept   : AIC=5106.921, Time=1.50 sec
 ARIMA(2,0,3)(0,0,1)[12] intercept   : AIC=5106.137, Time=0.79 sec
 ARIMA(2,0,3)(2,0,1)[12] intercept   : AIC=5108.848, Time=3.53 sec
 ARIMA(1,0,3)(1,0,0)[12] intercept   : AIC=5111.958, Time=0.67 sec
 ARIMA(2,0,3)(1,0,0)[12]             : AIC=5111.403, Time=0.47 sec

Best model:  ARIMA(2,0,3)(1,0,0)[12] intercept
Total fit time: 62.202 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4924.559, Time=0.31 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4920.909, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4748.553, Time=0.26 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4797.557, Time=0.25 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=5164.909, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4746.678, Time=0.07 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4748.558, Time=0.20 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4750.390, Time=0.56 sec
```

```
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4739.346, Time=0.13 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4741.335, Time=0.33 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4741.336, Time=0.29 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4743.184, Time=0.82 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4741.225, Time=0.20 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4741.278, Time=0.21 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4739.726, Time=0.15 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=4743.071, Time=0.47 sec
 ARIMA(2,0,0)(0,0,0)[12]             : AIC=4789.264, Time=0.07 sec

Best model:  ARIMA(2,0,0)(0,0,0)[12] intercept
Total fit time: 4.380 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4237.880, Time=0.37 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4234.017, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4224.326, Time=0.27 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4224.467, Time=0.20 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4567.050, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4222.465, Time=0.06 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4224.325, Time=0.20 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4226.374, Time=0.40 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4224.462, Time=0.14 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4224.459, Time=0.12 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=4222.648, Time=0.11 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4225.211, Time=0.68 sec
 ARIMA(1,0,0)(0,0,0)[12]             : AIC=4397.411, Time=0.03 sec

Best model:  ARIMA(1,0,0)(0,0,0)[12] intercept
Total fit time: 2.631 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4304.638, Time=0.47 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4369.987, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4268.703, Time=0.25 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4287.377, Time=0.27 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4626.398, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4283.811, Time=0.07 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=4269.158, Time=0.73 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4259.019, Time=0.45 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4269.973, Time=0.25 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=4260.288, Time=1.26 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=4260.178, Time=1.37 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=4271.082, Time=0.53 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=inf, Time=2.82 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4250.262, Time=0.59 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4256.207, Time=0.29 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4256.006, Time=0.31 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=4251.668, Time=1.65 sec
 ARIMA(2,0,0)(1,0,2)[12] intercept   : AIC=4251.555, Time=1.22 sec
```

```
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4264.095, Time=0.14 sec
 ARIMA(2,0,0)(0,0,2)[12] intercept   : AIC=4258.186, Time=0.72 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=4257.829, Time=1.00 sec
 ARIMA(2,0,0)(2,0,2)[12] intercept   : AIC=4250.573, Time=2.76 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=4251.843, Time=0.79 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=4213.933, Time=1.29 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=4210.591, Time=0.94 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4210.144, Time=0.49 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=4210.718, Time=1.15 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4219.424, Time=0.40 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=4211.577, Time=0.67 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=4211.863, Time=0.73 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=4211.099, Time=0.39 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4262.399, Time=0.17 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=4214.115, Time=0.43 sec
 ARIMA(2,0,1)(0,0,0)[12]             : AIC=4210.407, Time=0.20 sec

Best model:  ARIMA(2,0,1)(0,0,0)[12] intercept
Total fit time: 24.845 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4461.684, Time=0.61 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4458.612, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4384.787, Time=0.30 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4404.400, Time=0.29 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4682.375, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4383.252, Time=0.05 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4384.729, Time=0.18 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4386.152, Time=0.74 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4370.600, Time=0.09 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4372.230, Time=0.32 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4372.190, Time=0.21 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4374.577, Time=0.39 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4371.364, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4371.493, Time=0.27 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4369.644, Time=0.14 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=4371.454, Time=0.33 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=4371.432, Time=0.32 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=4373.640, Time=0.53 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=4402.893, Time=0.10 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=4371.530, Time=0.24 sec
 ARIMA(0,0,2)(0,0,0)[12] intercept   : AIC=4382.174, Time=0.17 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=4373.532, Time=0.30 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=4404.675, Time=0.06 sec

Best model:  ARIMA(1,0,1)(0,0,0)[12] intercept
Total fit time: 5.793 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4539.765, Time=0.76 sec
```

```
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4543.347, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4442.717, Time=0.22 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4481.346, Time=0.24 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4722.684, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4442.493, Time=0.07 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4442.618, Time=0.13 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4444.911, Time=0.29 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4402.920, Time=0.13 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4403.593, Time=0.31 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4403.469, Time=0.25 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4405.001, Time=0.94 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4395.809, Time=0.12 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=4395.692, Time=0.41 sec
 ARIMA(3,0,0)(2,0,0)[12] intercept   : AIC=4396.487, Time=1.40 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=4394.648, Time=1.35 sec
 ARIMA(3,0,0)(0,0,1)[12] intercept   : AIC=4395.475, Time=0.36 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=4396.589, Time=2.40 sec
 ARIMA(3,0,0)(1,0,2)[12] intercept   : AIC=4396.593, Time=2.09 sec
 ARIMA(3,0,0)(0,0,2)[12] intercept   : AIC=4396.042, Time=0.82 sec
 ARIMA(3,0,0)(2,0,2)[12] intercept   : AIC=4397.712, Time=2.79 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=4396.518, Time=1.56 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=4398.749, Time=1.26 sec
 ARIMA(3,0,0)(1,0,1)[12]             : AIC=inf, Time=0.56 sec

Best model:  ARIMA(3,0,0)(1,0,1)[12] intercept
Total fit time: 18.526 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4480.033, Time=0.64 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4488.427, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4459.334, Time=0.28 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4467.692, Time=0.20 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4661.048, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4457.457, Time=0.09 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4459.350, Time=0.19 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4456.505, Time=0.74 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=4457.479, Time=1.86 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=4461.245, Time=0.67 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=4458.628, Time=0.59 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=4458.359, Time=0.77 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=inf, Time=2.05 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4440.254, Time=0.72 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4440.937, Time=0.27 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4440.935, Time=0.35 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=4441.457, Time=1.88 sec
 ARIMA(2,0,0)(1,0,2)[12] intercept   : AIC=4441.476, Time=2.23 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4438.962, Time=0.13 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4438.979, Time=0.26 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4428.202, Time=0.49 sec
```

```
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=4428.453, Time=0.95 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=4428.475, Time=1.01 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=4430.241, Time=1.35 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4429.970, Time=0.24 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=4423.476, Time=0.75 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=4424.687, Time=1.53 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=4424.693, Time=1.34 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=4440.619, Time=1.49 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=inf, Time=0.78 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=4432.376, Time=0.61 sec
 ARIMA(3,0,1)(0,0,0)[12]             : AIC=4423.686, Time=0.27 sec

Best model:  ARIMA(3,0,1)(0,0,0)[12] intercept
Total fit time: 24.765 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3983.103, Time=0.80 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3980.008, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3965.463, Time=0.28 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3968.540, Time=0.21 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4252.514, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3964.018, Time=0.05 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3965.453, Time=0.20 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3967.586, Time=0.37 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=3960.084, Time=0.13 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=3961.422, Time=0.38 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=3961.417, Time=0.24 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3963.502, Time=0.77 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=3962.016, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=3962.018, Time=0.23 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=3961.249, Time=0.19 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=3964.016, Time=0.17 sec
 ARIMA(2,0,0)(0,0,0)[12]             : AIC=4061.238, Time=0.04 sec

Best model:  ARIMA(2,0,0)(0,0,0)[12] intercept
Total fit time: 4.213 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4615.887, Time=0.36 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4615.262, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=4601.893, Time=0.21 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=4604.211, Time=0.23 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4730.378, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=4601.196, Time=0.06 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=4601.953, Time=0.27 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=4603.745, Time=0.53 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=4596.900, Time=0.14 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=4598.207, Time=0.33 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=4598.210, Time=0.31 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=4600.206, Time=0.74 sec
```

```
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=4595.163, Time=0.17 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=4596.820, Time=0.45 sec
 ARIMA(3,0,0)(0,0,1)[12] intercept   : AIC=4596.818, Time=0.39 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=4598.815, Time=1.31 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=4582.732, Time=0.53 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=4584.647, Time=1.12 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=4584.645, Time=1.09 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=4586.692, Time=1.43 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=4580.786, Time=0.44 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=4582.697, Time=0.85 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=4582.695, Time=0.83 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=4584.193, Time=1.48 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=4579.838, Time=0.30 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=4581.734, Time=0.63 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=4581.730, Time=0.65 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=4582.258, Time=1.15 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=4603.767, Time=0.09 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=4580.809, Time=0.40 sec
 ARIMA(0,0,2)(0,0,0)[12] intercept   : AIC=4600.070, Time=0.13 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=4582.977, Time=0.80 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=4586.125, Time=0.10 sec

Best model:  ARIMA(1,0,1)(0,0,0)[12] intercept
Total fit time: 17.559 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3587.391, Time=0.84 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3586.790, Time=0.01 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3588.170, Time=0.26 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3588.269, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=3895.008, Time=0.01 sec
 ARIMA(0,0,0)(1,0,0)[12] intercept   : AIC=3586.786, Time=0.16 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=3588.786, Time=0.40 sec
 ARIMA(0,0,0)(0,0,1)[12] intercept   : AIC=3586.780, Time=0.12 sec
 ARIMA(0,0,0)(0,0,2)[12] intercept   : AIC=3588.776, Time=0.27 sec
 ARIMA(0,0,0)(1,0,2)[12] intercept   : AIC=3589.059, Time=2.05 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3588.167, Time=0.17 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=3588.558, Time=0.50 sec
 ARIMA(0,0,0)(0,0,1)[12]             : AIC=3818.320, Time=0.08 sec

Best model:  ARIMA(0,0,0)(0,0,1)[12] intercept
Total fit time: 5.072 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3785.299, Time=0.56 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3789.993, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3726.570, Time=0.20 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3737.910, Time=0.19 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=3999.206, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3726.031, Time=0.06 sec
```

```
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3726.477, Time=0.22 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3726.067, Time=1.01 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=3722.209, Time=0.10 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=3723.595, Time=0.32 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=3723.549, Time=0.27 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3723.520, Time=0.95 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=3723.009, Time=0.13 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=3701.425, Time=0.62 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=3703.156, Time=1.39 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=3703.099, Time=1.22 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=3705.112, Time=1.51 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=3719.744, Time=0.29 sec
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=3703.471, Time=1.00 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=3703.276, Time=0.77 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=3703.565, Time=0.67 sec
 ARIMA(3,0,2)(0,0,0)[12] intercept   : AIC=3703.922, Time=0.80 sec
 ARIMA(2,0,1)(0,0,0)[12]             : AIC=3701.024, Time=0.14 sec
 ARIMA(2,0,1)(1,0,0)[12]             : AIC=3702.767, Time=0.37 sec
 ARIMA(2,0,1)(0,0,1)[12]             : AIC=3702.715, Time=0.34 sec
 ARIMA(2,0,1)(1,0,1)[12]             : AIC=inf, Time=1.25 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=3725.873, Time=0.14 sec
 ARIMA(2,0,0)(0,0,0)[12]             : AIC=3788.253, Time=0.05 sec
 ARIMA(3,0,1)(0,0,0)[12]             : AIC=3702.634, Time=0.26 sec
 ARIMA(2,0,2)(0,0,0)[12]             : AIC=3702.714, Time=0.37 sec
 ARIMA(1,0,0)(0,0,0)[12]             : AIC=3815.700, Time=0.03 sec
 ARIMA(1,0,2)(0,0,0)[12]             : AIC=3703.393, Time=0.13 sec
 ARIMA(3,0,0)(0,0,0)[12]             : AIC=3778.199, Time=0.06 sec
 ARIMA(3,0,2)(0,0,0)[12]             : AIC=3703.540, Time=0.36 sec

Best model:  ARIMA(2,0,1)(0,0,0)[12]
Total fit time: 15.852 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=4006.308, Time=0.40 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=4006.149, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3926.184, Time=0.20 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3943.242, Time=0.19 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4157.138, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3925.435, Time=0.07 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3926.139, Time=0.19 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3928.072, Time=0.40 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=3919.013, Time=0.10 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=3919.541, Time=0.33 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=3919.495, Time=0.23 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3921.447, Time=0.60 sec
 ARIMA(3,0,0)(0,0,0)[12] intercept   : AIC=3917.295, Time=0.19 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=3917.905, Time=0.38 sec
 ARIMA(3,0,0)(0,0,1)[12] intercept   : AIC=3917.851, Time=0.33 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=3919.770, Time=1.14 sec
```

```
 ARIMA(3,0,1)(0,0,0)[12] intercept   : AIC=3916.633, Time=0.80 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=3918.262, Time=1.20 sec
 ARIMA(3,0,1)(0,0,1)[12] intercept   : AIC=3918.226, Time=0.94 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=3921.913, Time=1.57 sec
 ARIMA(2,0,1)(0,0,0)[12] intercept   : AIC=3915.062, Time=0.31 sec
 ARIMA(2,0,1)(1,0,0)[12] intercept   : AIC=3916.263, Time=0.63 sec
 ARIMA(2,0,1)(0,0,1)[12] intercept   : AIC=3916.227, Time=0.57 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=3918.146, Time=1.07 sec
 ARIMA(1,0,1)(0,0,0)[12] intercept   : AIC=3914.607, Time=0.23 sec
 ARIMA(1,0,1)(1,0,0)[12] intercept   : AIC=3915.327, Time=0.35 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=3915.281, Time=0.38 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=3917.218, Time=0.72 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=3942.902, Time=0.13 sec
 ARIMA(1,0,2)(0,0,0)[12] intercept   : AIC=3915.137, Time=0.31 sec
 ARIMA(0,0,2)(0,0,0)[12] intercept   : AIC=3929.880, Time=0.18 sec
 ARIMA(2,0,2)(0,0,0)[12] intercept   : AIC=3916.914, Time=0.47 sec
 ARIMA(1,0,1)(0,0,0)[12]             : AIC=3941.844, Time=0.09 sec

Best model:  ARIMA(1,0,1)(0,0,0)[12] intercept
Total fit time: 14.764 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3827.391, Time=0.54 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3852.117, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3799.368, Time=0.24 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3814.731, Time=0.27 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=4019.361, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3802.302, Time=0.07 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=3800.704, Time=0.61 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3790.507, Time=0.45 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3799.586, Time=0.22 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=3791.972, Time=1.02 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=3791.754, Time=1.06 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=3801.507, Time=0.45 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=3794.262, Time=1.74 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3771.118, Time=0.57 sec
 ARIMA(2,0,0)(0,0,1)[12] intercept   : AIC=3775.943, Time=0.25 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=3776.030, Time=0.28 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=3773.092, Time=1.40 sec
 ARIMA(2,0,0)(1,0,2)[12] intercept   : AIC=3773.082, Time=1.19 sec
 ARIMA(2,0,0)(0,0,0)[12] intercept   : AIC=3778.207, Time=0.07 sec
 ARIMA(2,0,0)(0,0,2)[12] intercept   : AIC=3777.792, Time=0.56 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=3778.028, Time=0.77 sec
 ARIMA(2,0,0)(2,0,2)[12] intercept   : AIC=3774.207, Time=1.70 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=3772.585, Time=0.61 sec
 ARIMA(2,0,1)(1,0,1)[12] intercept   : AIC=3772.632, Time=0.72 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=3776.731, Time=0.66 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=3774.585, Time=1.21 sec
 ARIMA(2,0,0)(1,0,1)[12]             : AIC=3787.470, Time=0.40 sec
```

```
Best model:  ARIMA(2,0,0)(1,0,1)[12] intercept
Total fit time: 17.149 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3798.452, Time=0.36 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3840.177, Time=0.02 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3778.982, Time=0.25 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3794.649, Time=0.25 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=3997.158, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3788.768, Time=0.06 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=3768.320, Time=0.61 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=3768.829, Time=1.22 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3771.711, Time=0.45 sec
 ARIMA(0,0,0)(2,0,0)[12] intercept   : AIC=3795.037, Time=0.45 sec
 ARIMA(2,0,0)(2,0,0)[12] intercept   : AIC=3739.526, Time=0.75 sec
 ARIMA(2,0,0)(1,0,0)[12] intercept   : AIC=3745.336, Time=0.30 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=3739.544, Time=1.73 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3743.784, Time=0.57 sec
 ARIMA(3,0,0)(2,0,0)[12] intercept   : AIC=3737.121, Time=1.13 sec
 ARIMA(3,0,0)(1,0,0)[12] intercept   : AIC=3741.534, Time=0.50 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=3737.207, Time=2.13 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=3741.144, Time=0.72 sec
 ARIMA(3,0,1)(2,0,0)[12] intercept   : AIC=3711.968, Time=3.15 sec
 ARIMA(3,0,1)(1,0,0)[12] intercept   : AIC=3712.267, Time=1.62 sec
 ARIMA(3,0,1)(2,0,1)[12] intercept   : AIC=inf, Time=3.83 sec
 ARIMA(3,0,1)(1,0,1)[12] intercept   : AIC=3715.098, Time=1.85 sec
 ARIMA(2,0,1)(2,0,0)[12] intercept   : AIC=3716.549, Time=1.95 sec
 ARIMA(3,0,2)(2,0,0)[12] intercept   : AIC=3717.244, Time=3.87 sec
 ARIMA(2,0,2)(2,0,0)[12] intercept   : AIC=3718.957, Time=2.66 sec
 ARIMA(3,0,1)(2,0,0)[12]             : AIC=3712.934, Time=1.14 sec

Best model:  ARIMA(3,0,1)(2,0,0)[12] intercept
Total fit time: 31.610 seconds
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(1,0,1)[12] intercept   : AIC=3534.103, Time=0.41 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=3558.181, Time=0.03 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=3525.043, Time=0.23 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=3532.895, Time=0.21 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=3814.574, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[12] intercept   : AIC=3527.289, Time=0.06 sec
 ARIMA(1,0,0)(2,0,0)[12] intercept   : AIC=3523.485, Time=0.65 sec
 ARIMA(1,0,0)(2,0,1)[12] intercept   : AIC=3519.056, Time=1.02 sec
 ARIMA(1,0,0)(1,0,1)[12] intercept   : AIC=3518.712, Time=0.41 sec
 ARIMA(1,0,0)(0,0,1)[12] intercept   : AIC=3525.563, Time=0.18 sec
 ARIMA(1,0,0)(1,0,2)[12] intercept   : AIC=3518.443, Time=0.90 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=3525.412, Time=0.50 sec
 ARIMA(1,0,0)(2,0,2)[12] intercept   : AIC=3517.983, Time=1.43 sec
 ARIMA(0,0,0)(2,0,2)[12] intercept   : AIC=3532.229, Time=1.31 sec
```

```
 ARIMA(2,0,0)(2,0,2)[12] intercept   : AIC=3496.994, Time=1.80 sec
 ARIMA(2,0,0)(1,0,2)[12] intercept   : AIC=3497.467, Time=1.03 sec
 ARIMA(2,0,0)(2,0,1)[12] intercept   : AIC=3497.869, Time=1.47 sec
 ARIMA(2,0,0)(1,0,1)[12] intercept   : AIC=3497.189, Time=0.56 sec
 ARIMA(3,0,0)(2,0,2)[12] intercept   : AIC=3490.077, Time=2.95 sec
 ARIMA(3,0,0)(1,0,2)[12] intercept   : AIC=3491.452, Time=1.80 sec
 ARIMA(3,0,0)(2,0,1)[12] intercept   : AIC=3491.811, Time=1.77 sec
 ARIMA(3,0,0)(1,0,1)[12] intercept   : AIC=3490.998, Time=0.78 sec
 ARIMA(3,0,1)(2,0,2)[12] intercept   : AIC=inf, Time=3.73 sec
 ARIMA(2,0,1)(2,0,2)[12] intercept   : AIC=inf, Time=3.34 sec
 ARIMA(3,0,0)(2,0,2)[12]             : AIC=inf, Time=3.19 sec

Best model:  ARIMA(3,0,0)(2,0,2)[12] intercept
Total fit time: 29.813 seconds
```

[ ]:
```
datamod = pd.DataFrame(models)
datamod.index = data.columns
datamod.rename(columns={0:"Model"}, inplace=True)
datamod.to_csv("ModelsSari.csv")
```

4 Se procede a hacer un modelo para cada uno de ellos.

Se prueba para un producto

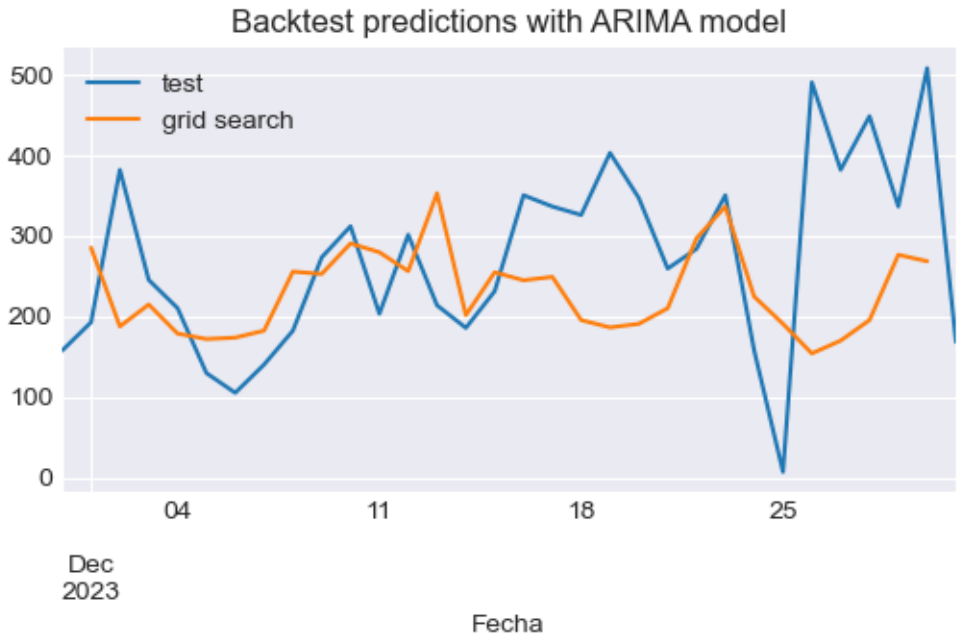
[ ]:
```
forecaster = ForecasterSarimax(
                regressor=Sarimax(order=datamod.loc["Producto 273"][0].order,␣
 ↪seasonal_order=datamod.loc["Producto 273"][0].seasonal_order, maxiter=500),
             )

metric_m1, predictions_m1 = backtesting_sarimax(
                                forecaster            = forecaster,
                                y                     = data.loc[:,"Producto␣
 ↪273"],
                                initial_train_size    = len(data.loc[:end_val]),
                                steps                 = 12,
                                exog = exog,
                                metric                = 'mean_squared_error',
                                refit                 = True,
                                n_jobs                = "auto",
                                suppress_warnings_fit = True,
                                verbose               = False,
                                show_progress         = True
                            )
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2345847679.py:2: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
```

```
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc["Producto 273"][0].order,
seasonal_order=datamod.loc["Producto 273"][0].seasonal_order, maxiter=500),
100%|      | 3/3 [00:39<00:00, 13.15s/it]
```

[ ]:
```
# Compare predictions
# ==============================================================================
print(f"Metric (mean_absolute_error) for grid search model: {metric_m1}")

fig, ax = plt.subplots(figsize=(6, 3))
data.loc[end_val:,"Producto 273"].plot(ax=ax, label='test')
predictions_m1 = predictions_m1.shift(-1).rename(columns={'pred': 'grid␣
 ↪search'})
predictions_m1.plot(ax=ax)
ax.set_title('Backtest predictions with ARIMA model')
ax.legend();
```

```
Metric (mean_absolute_error) for grid search model: 17949.30396715978
```



Se corre cada modelo personalizado para cada producto

[ ]:
```
month_pred = pd.DataFrame()
week_pred = pd.DataFrame()
```

[ ]:
```
for i in datamod.index:
    forecaster = ForecasterSarimax(
```

```
                 regressor=Sarimax(order=datamod.loc[i][0].order,␣
↪seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
             )
   forecaster.fit(y=data.loc[:end_val,i], exog=exog[:end_val])
   month_pred[i] = forecaster.predict(steps=31, exog=exog.loc["2023-12-01":, :
↪])
   week_pred[i] = forecaster.predict(steps=8, exog=exog.loc["2023-12-01":, :])
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
```

```
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
```

```
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2969500660.py:3: FutureWarning:
```

```
Series.__getitem__ treating keys as positions is deprecated. In a future
version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  regressor=Sarimax(order=datamod.loc[i][0].order,
seasonal_order=datamod.loc[i][0].seasonal_order, maxiter=500),
```

Se ayuda al modelo por si llega a estimar valores negativos

[ ]:
```
# cambiar valores negativos a 0
month_pred[month_pred<0] = 0
week_pred[week_pred<0] = 0
```

[ ]:
```
from sklearn.metrics import mean_absolute_error,␣
 ↪mean_absolute_percentage_error, mean_squared_error, r2_score
```

Se obtienen las estadisticas descriptivas de cada modelo por mes y por semana

[ ]:
```
stats_mes = pd.DataFrame()
for i in month_pred.columns:
    y_pred = month_pred[i]
    y_test = data.loc["2023-12-01":"2023-12-31",i]
    mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
    mape = mean_absolute_percentage_error(y_true=y_test, y_pred=y_pred)
    mse = mean_squared_error(y_true=y_test, y_pred=y_pred)
    r2 = r2_score(y_true=y_test, y_pred=y_pred)
    mape2 = abs((y_pred - y_test)/y_test).replace([np.inf, -np.inf], np.log(0.
 ↪99999999999999999999999)).dropna().sum()/30
    mape3 = (np.abs((y_test - y_pred)/np.where(y_test==0, 1, y_test))).mean()
    smape = 1/len(y_test) * np.sum(2*np.abs(y_pred - y_test)/(np.abs(y_pred) +␣
 ↪np.abs(y_test))*100)
    valor_real = y_test.sum()
    valor_pred = y_pred.sum()
    error = (valor_real - valor_pred)/valor_real
    error_semanal = error/4 * 100
    stats_mes = pd.concat([stats_mes, pd.DataFrame({"Producto":i, "MAE":mae,␣
 ↪"MSE":mse, "R2":r2, "SMAPE": smape,"MAPE lib":mape,"MAPE2":mape2,"MAPE3":
 ↪mape3, "valor real":
        valor_real, "valor predecido": valor_pred, "error":error*100, "error␣
 ↪por semana": error_semanal}, index=[0])], axis=0)
stats_mes
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\1630151674.py:14: RuntimeWarning:
divide by zero encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real
```

[ ]:

	Producto	MAE	MSE	R2	SMAPE	MAPE lib	\
0	Producto 273	107.309473	18019.026755	-0.387650	43.063818	1.855757e+00	
0	Producto 0	20.262914	1048.008900	-0.122883	50.102742	9.415877e-01	

```
0    Producto 1   14.262463    291.643859 -0.361855   60.481885  5.240341e+15
0    Producto 5    7.463292     93.859860 -0.032708   38.733354  3.451022e+15
0    Producto 8    9.031194    110.910652 -0.397948   70.534206  7.991283e+15
0   Producto 21    5.705833     92.586240 -6.964140   97.167616  1.276984e+16
0   Producto 12    2.658235      9.387431 -0.187797  120.277525  2.972964e+15
0   Producto 22    2.959856     12.400126 -0.250356  126.690667  1.031810e+16
0  Producto 186    4.708370     33.479094 -0.152942   87.772322  7.752684e+15
0   Producto 20    1.725111      4.467942 -0.288623  159.704041  4.550007e+15
0   Producto 33    3.451659     17.555547  0.009198  101.397902  6.210484e+15
0  Producto 245    4.317202     38.349614 -0.254582  136.063830  3.531934e+15
0   Producto 16    3.590845     28.073254 -0.073254  110.908266  5.871153e+15
0   Producto 17    3.330647     14.280163 -0.007432  119.864308  8.864364e+15
0   Producto 38    1.678278      3.217299 -0.682625  176.522398  5.883560e+15
0  Producto 134    0.112903      0.395161 -0.033333    6.451613  3.225806e-02
0   Producto 37    1.659654      3.676552 -0.502197  185.696108  5.835430e+15
0   Producto 59    2.298116      8.963984 -0.091949  147.326436  3.298162e+15
0  Producto 248    3.992845     19.317107 -0.687536   99.020015  9.092906e+15
0  Producto 122    0.968119      1.136711  0.000000  193.548387  4.360019e+15

      MAPE2     MAPE3  valor real  valor predecido       error  \
0  1.917615  1.855757      8452.5      7084.204968   16.188051
0  0.972974  0.941588      1186.5      1416.896506  -19.418163
0  1.081707  2.210403       731.5       940.289479  -28.542649
0  0.440212  1.192293       651.0       689.115308   -5.854886
0  0.961057  2.704476       399.0       525.947695  -31.816465
0  0.634613  3.449616       133.0       231.235757  -73.861472
0  0.416663  1.063353       108.5        67.316443   37.957196
0  0.086434  2.374725        70.0       123.206525  -76.009321
0  0.363726  2.073435       203.0       203.983398   -0.484433
0  0.187940  1.192182        35.0        44.160442  -26.172692
0  0.339021  1.707090       129.5       138.033908   -6.589890
0  0.387070  1.158831       150.5        65.290033   56.617918
0  0.245072  1.540824       126.0       101.077185   19.780012
0  0.203343  2.165067       101.5       131.717314  -29.770752
0  0.109789  1.412660        21.0        49.970980 -137.957046
0  0.033333  0.032258         3.5         0.000000  100.000000
0  0.080597  1.373723        17.5        46.385723 -165.061272
0  0.298460  1.021171        73.5        50.059013   31.892500
0  0.459699  2.463901       112.0       195.734211  -74.762689
0  0.000000  0.968119         0.0        30.011682        -inf

   error por semana
0          4.047013
0         -4.854541
0         -7.135662
0         -1.463722
0         -7.954116
```

```
0        -18.465368
0          9.489299
0        -19.002330
0         -0.121108
0         -6.543173
0         -1.647473
0         14.154480
0          4.945003
0         -7.442688
0        -34.489262
0         25.000000
0        -41.265318
0          7.973125
0        -18.690672
0              -inf
```

[ ]:
```
stats_semana = pd.DataFrame()
for i in month_pred.columns:
    y_pred = week_pred[i]
    y_test = data.loc["2023-12-01":"2023-12-8",i]
    mae = mean_absolute_error(y_true=y_test, y_pred=y_pred)
    mape = mean_absolute_percentage_error(y_true=y_test, y_pred=y_pred)
    mse = mean_squared_error(y_true=y_test, y_pred=y_pred)
    r2 = r2_score(y_true=y_test, y_pred=y_pred)
    mape2 = abs((y_pred - y_test)/y_test).replace([np.inf, -np.inf], np.log(0.
 ↪99999999999999999999999)).dropna().sum()/30
    mape3 = (np.abs((y_test - y_pred)/np.where(y_test==0, 1, y_test))).mean()
    smape = 1/len(y_test) * np.sum(2*np.abs(y_pred - y_test)/(np.abs(y_pred) +␣
 ↪np.abs(y_test))*100)
    valor_real = y_test.sum()
    valor_pred = y_pred.sum()
    error = (valor_real - valor_pred)/valor_real *100
    stats_semana = pd.concat([stats_semana, pd.DataFrame({"Producto":i, "MAE":
 ↪mae, "MSE":mse, "R2":r2, "SMAPE": smape,"MAPE lib":mape,"MAPE2":
 ↪mape2,"MAPE3":mape3, "valor real":
        valor_real, "valor predecido": valor_pred, "error":error}, index=[0])],␣
 ↪axis=0)
stats_semana
```

```
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2497161326.py:14: RuntimeWarning:
invalid value encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real *100
C:\Users\progra.DESKTOP-
GV4Q93K\AppData\Local\Temp\ipykernel_19048\2497161326.py:14: RuntimeWarning:
divide by zero encountered in scalar divide
  error = (valor_real - valor_pred)/valor_real *100
```

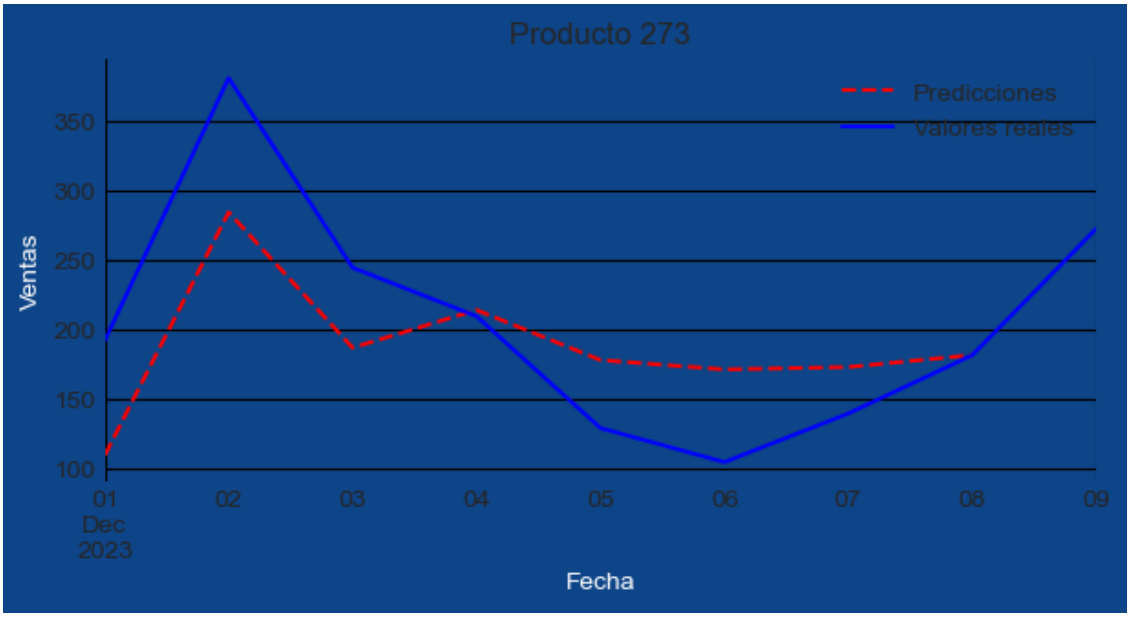
[ ]:

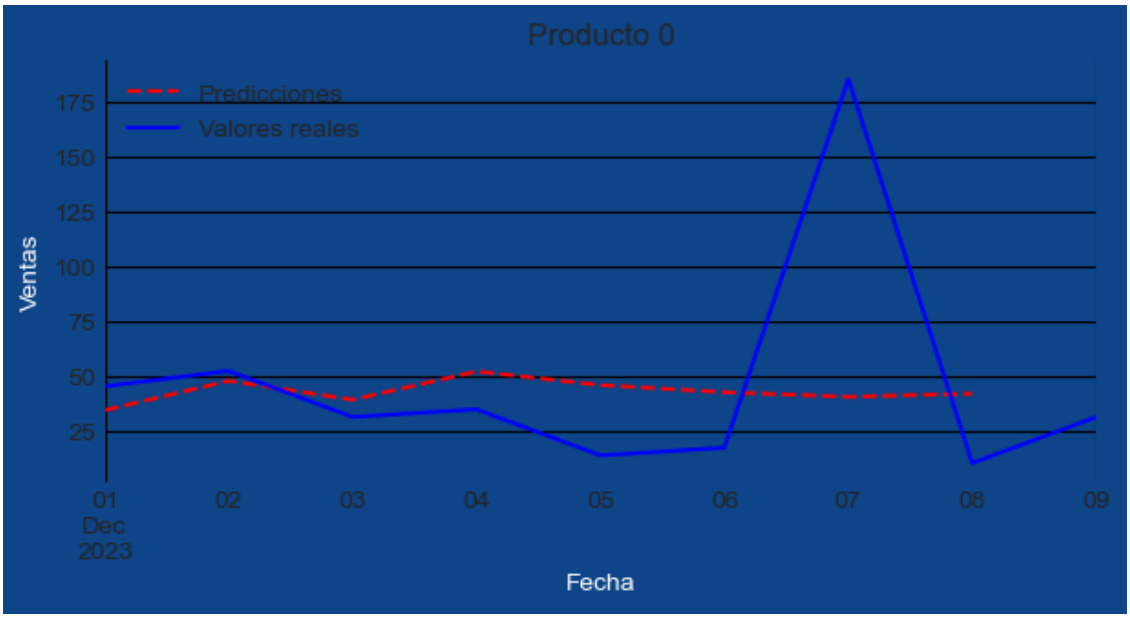
	Producto	MAE	MSE	R2	SMAPE	MAPE lib	\
0	Producto 273	48.840042	3435.755726	0.481556	26.741805	2.740573e-01	
0	Producto 0	34.295611	3017.404290	-0.056029	67.117765	1.074584e+00	
0	Producto 1	13.392477	211.902942	-1.009228	61.312046	1.420857e+00	
0	Producto 5	4.585538	41.088805	-0.350113	26.421688	4.100188e-01	
0	Producto 8	10.087327	117.351141	-2.855974	111.970440	1.727572e+16	
0	Producto 21	13.464725	318.785556	-51.046621	118.020719	3.150247e+16	
0	Producto 12	3.277976	13.451169	-3.685033	109.881526	6.680354e-01	
0	Producto 22	3.423491	13.692747	-9.219660	175.378907	1.535739e+16	
0	Producto 186	6.141932	46.296216	-0.382138	130.359322	1.608724e+16	
0	Producto 20	1.672703	3.994915	-1.981627	175.000000	5.562860e+15	
0	Producto 33	3.438004	15.128475	0.168015	94.681014	4.093078e+15	
0	Producto 245	2.454919	8.602838	-0.404545	107.758296	1.938684e+15	
0	Producto 16	2.347238	6.946615	-1.419501	136.303067	9.039540e+15	
0	Producto 17	3.420979	16.351081	0.237267	121.285972	8.752811e+15	
0	Producto 38	1.837654	3.949731	-0.719611	174.955290	5.676613e+15	
0	Producto 134	0.000000	0.000000	1.000000	0.000000	0.000000e+00	
0	Producto 37	1.947253	6.163764	-0.038791	176.980483	4.459825e+15	
0	Producto 59	1.494169	2.567795	0.105639	148.739128	3.893828e+15	
0	Producto 248	3.880493	19.836095	-0.619273	101.338506	1.075578e+16	
0	Producto 122	0.778015	0.798120	0.000000	175.000000	3.503866e+15	

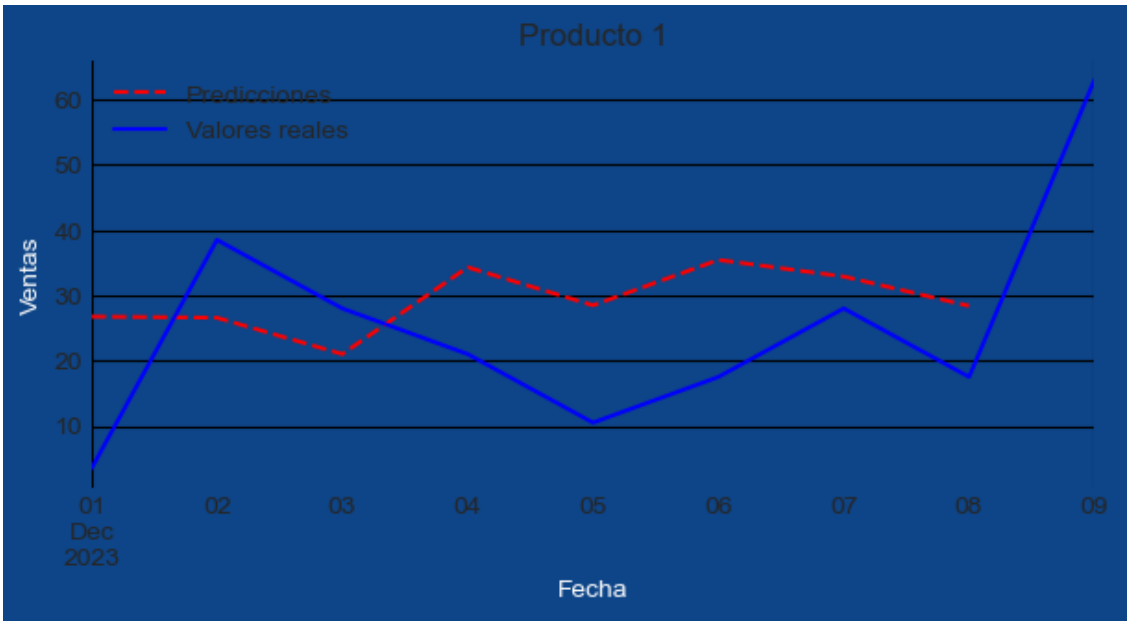
	MAPE2	MAPE3	valor real	valor predecido	error
0	0.073082	0.274057	1585.5	1502.171660	5.255651
0	0.286556	1.074584	392.0	345.868444	11.768254
0	0.378895	1.420857	164.5	233.788619	-42.120741
0	0.109338	0.410019	150.5	173.331844	-15.170660
0	0.364641	5.203384	45.5	126.198619	-177.359602
0	0.459531	8.718193	28.0	120.143714	-329.084691
0	0.178143	0.668035	38.5	12.276192	68.113787
0	0.001026	3.413873	3.5	30.887930	-782.512296
0	0.069439	3.832481	38.5	52.124169	-35.387452
0	0.033333	1.360203	3.5	9.881625	-182.332146
0	0.105858	1.305814	38.5	36.108119	6.212679
0	0.105762	0.827082	28.0	15.248238	45.542009
0	0.025909	2.104340	10.5	24.891023	-137.057358
0	0.044776	2.111426	28.0	32.356434	-15.558692
0	0.043977	1.425374	7.0	12.466146	-78.087800
0	0.000000	0.000000	0.0	0.000000	NaN
0	0.046064	1.163018	10.5	10.766458	-2.537695
0	0.047967	1.044480	10.5	14.775886	-40.722721
0	0.080995	2.691997	28.0	48.356870	-72.703106
0	0.000000	0.778015	0.0	6.224116	-inf

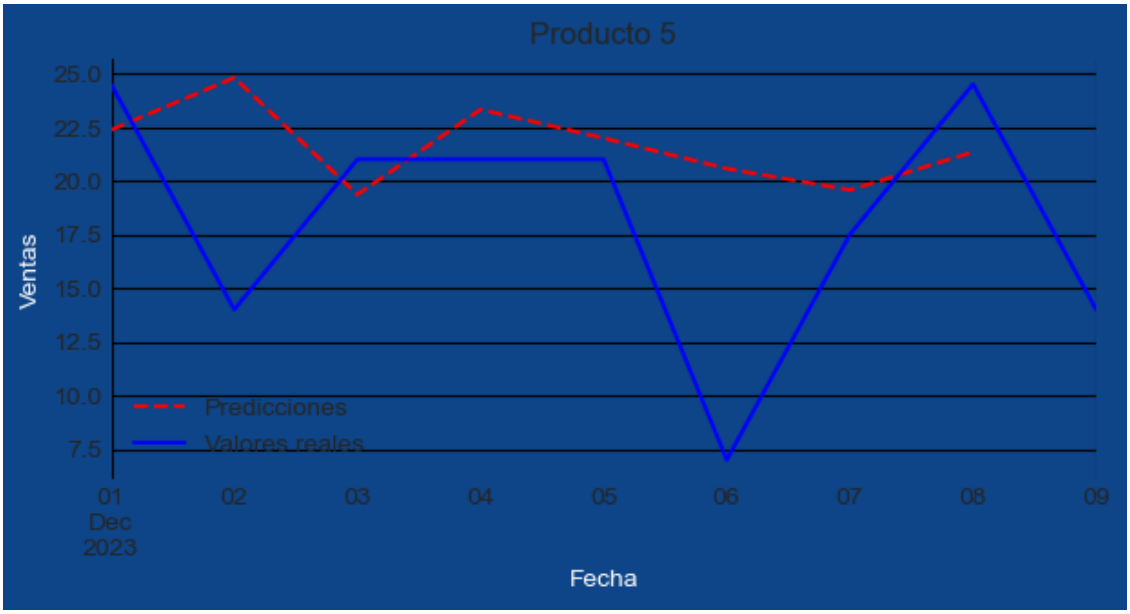
[ ]:
```
#Save figure
for i in week_pred.columns:
    fig, ax=plt.subplots(figsize=(7, 3))
```

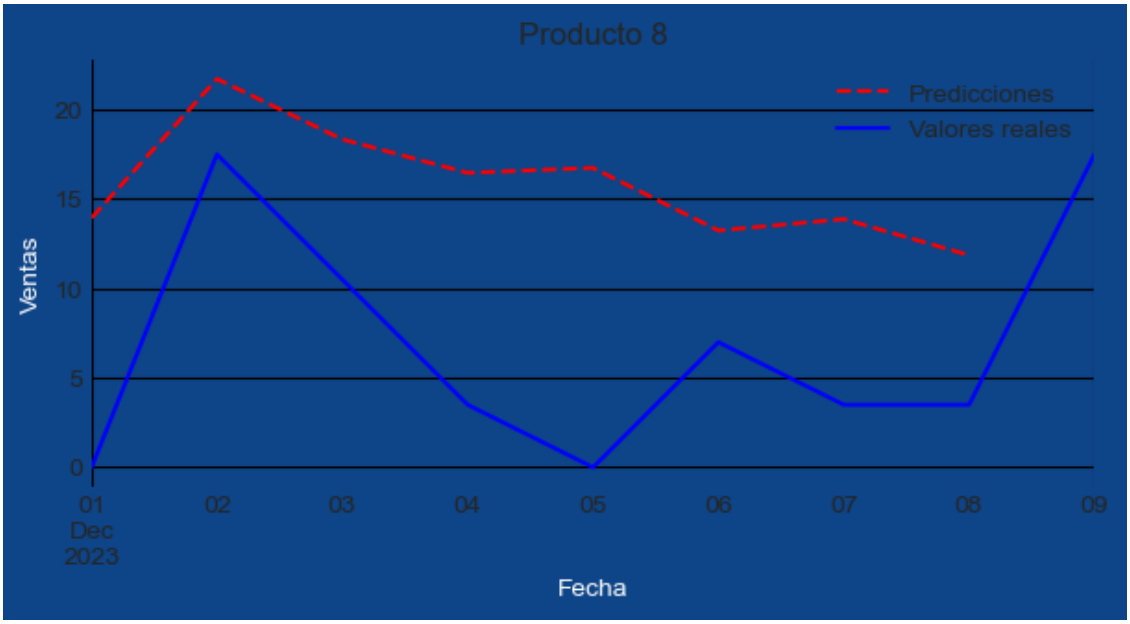
```
    week_pred[i].plot(ax=ax,color='red', linestyle='--', label='Predicciones')
    data.loc["2023-12-01":"2023-12-09",i].plot(ax=ax,color='blue',␣
 ↪linestyle='-', label='Valores reales')
    ax.set_title(i)
    ax.set_facecolor('#0e4588')
    ax.legend()
    ax.grid(color='black')
    fig.set_facecolor('#0e4588')
    ax.set_ylabel('Ventas',color='white')
    ax.set_xlabel('Fecha',color='white')
#plt.savefig(f"pred_semanal_sari{i}.png")
```

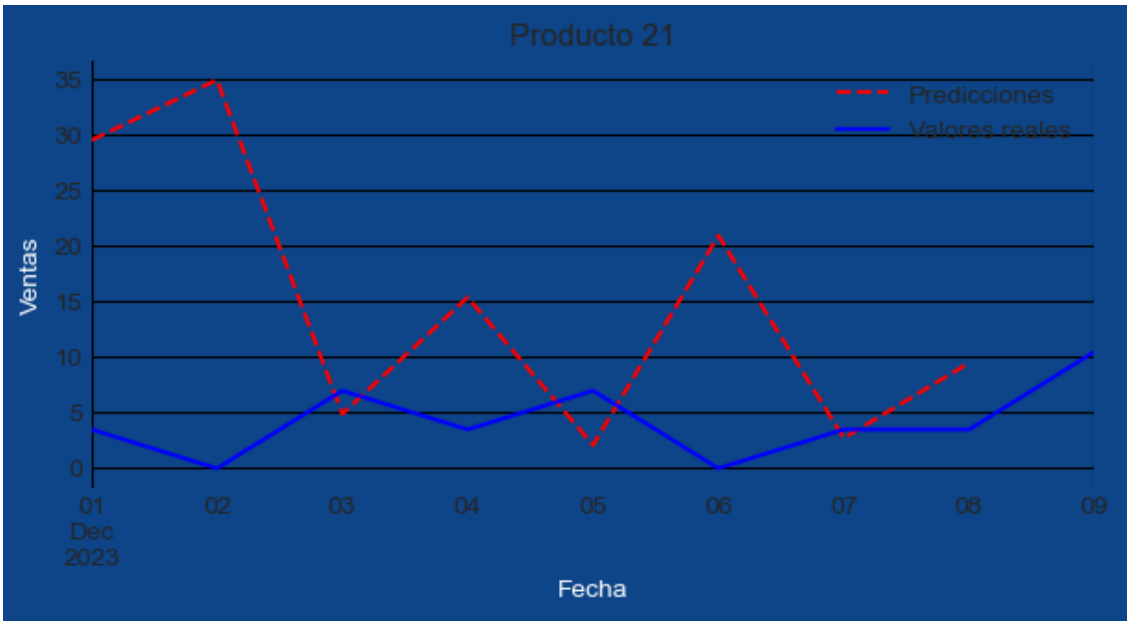


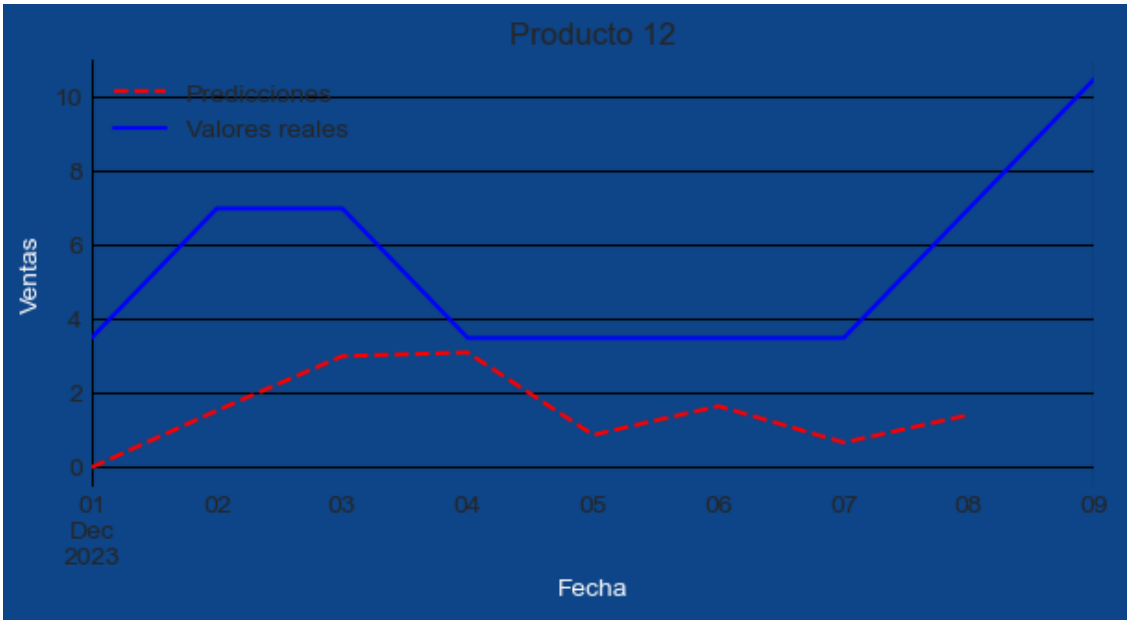

Producto 0
Predicciones
Valores reales
175
150
125
100
75
50
25
Ventas
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

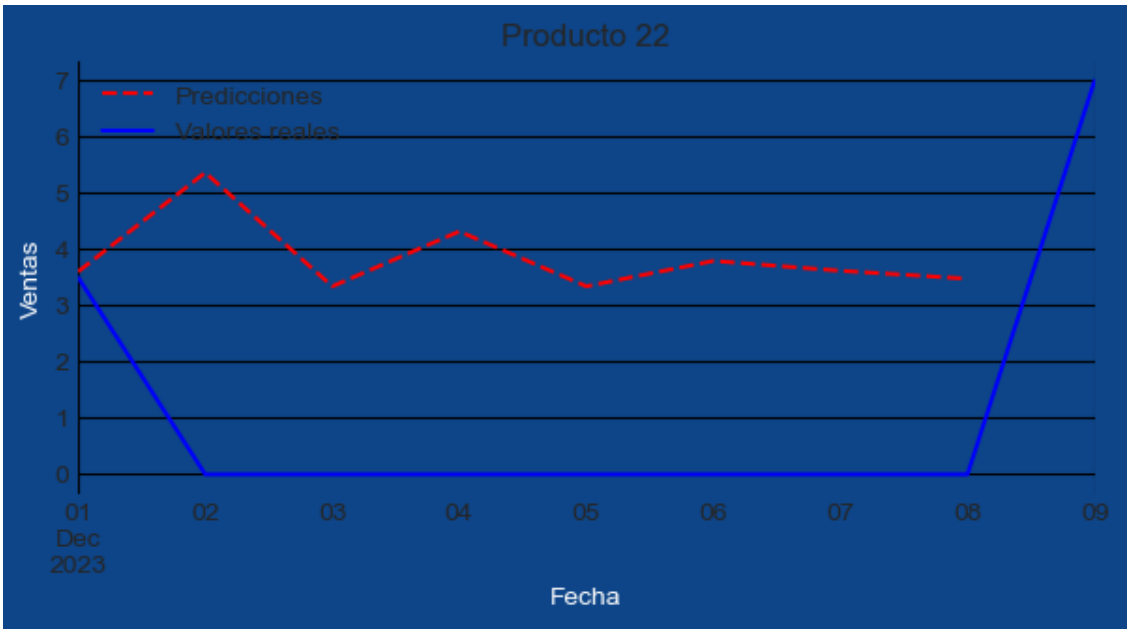

Producto 1
Predicciones
Valores reales
60
50
40
30
20
10
Ventas
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

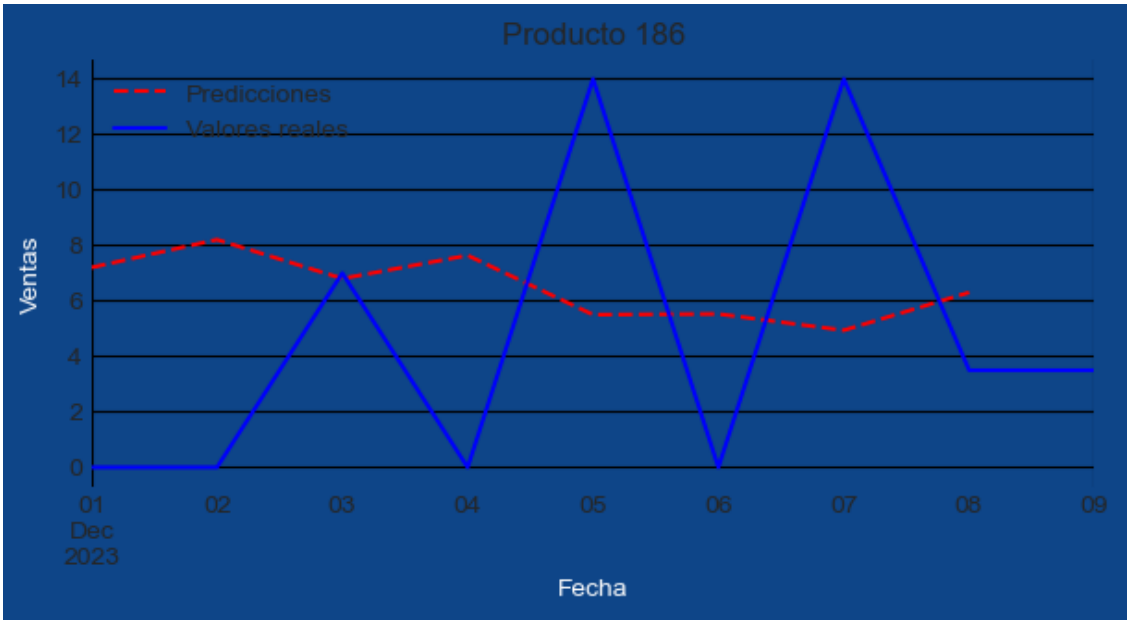

Producto 5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

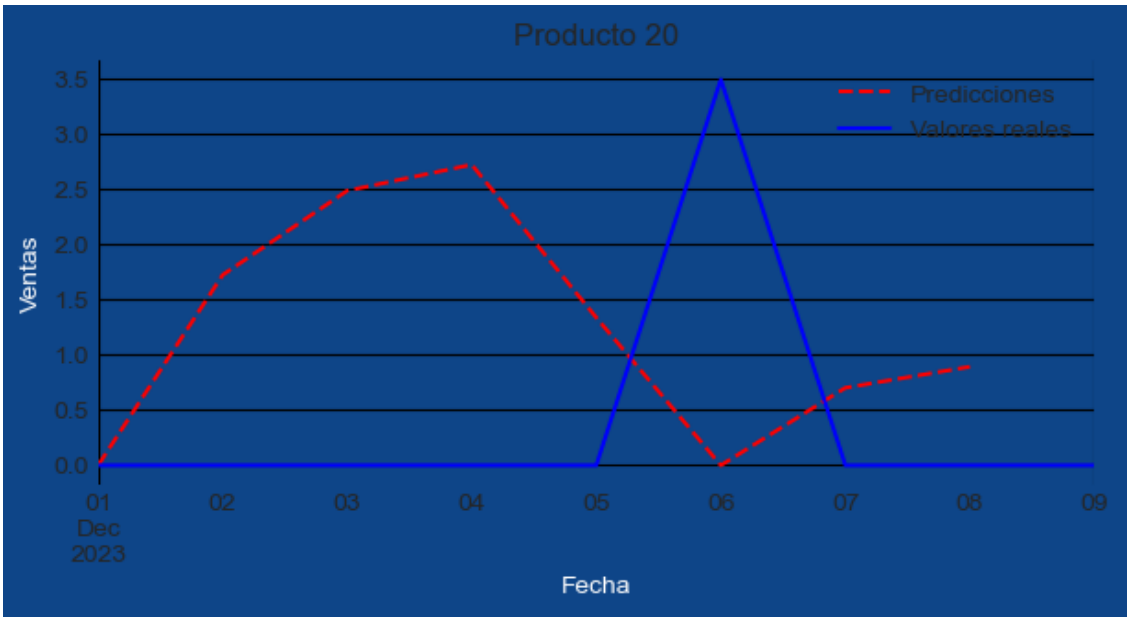

Producto 8
20
15
10
5
0
Ventas
Predicciones
Valores reales
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

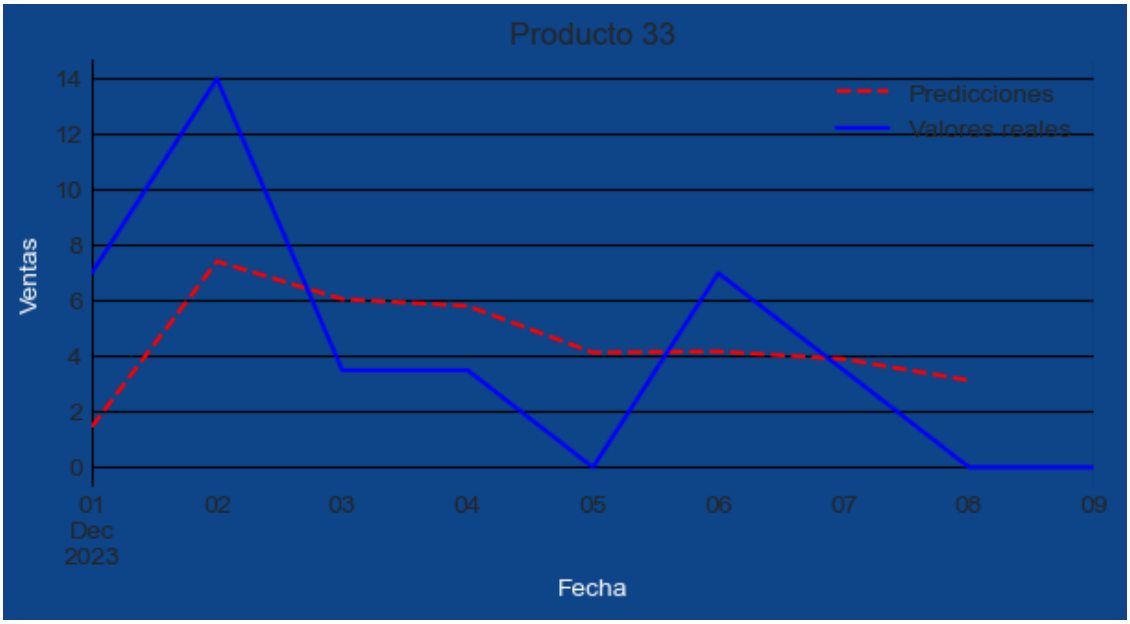

Producto 21
Predicciones
Valores reales
Ventas
35
30
25
20
15
10
5
0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

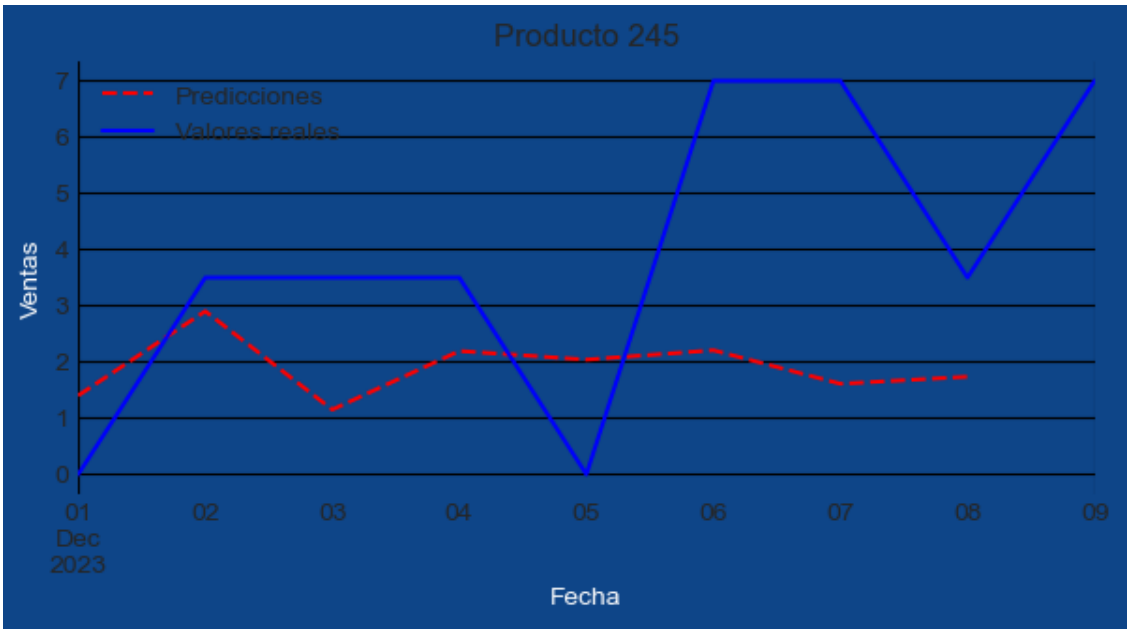

Producto 12
Predicciones
Valores reales
Ventas
10
8
6
4
2
0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

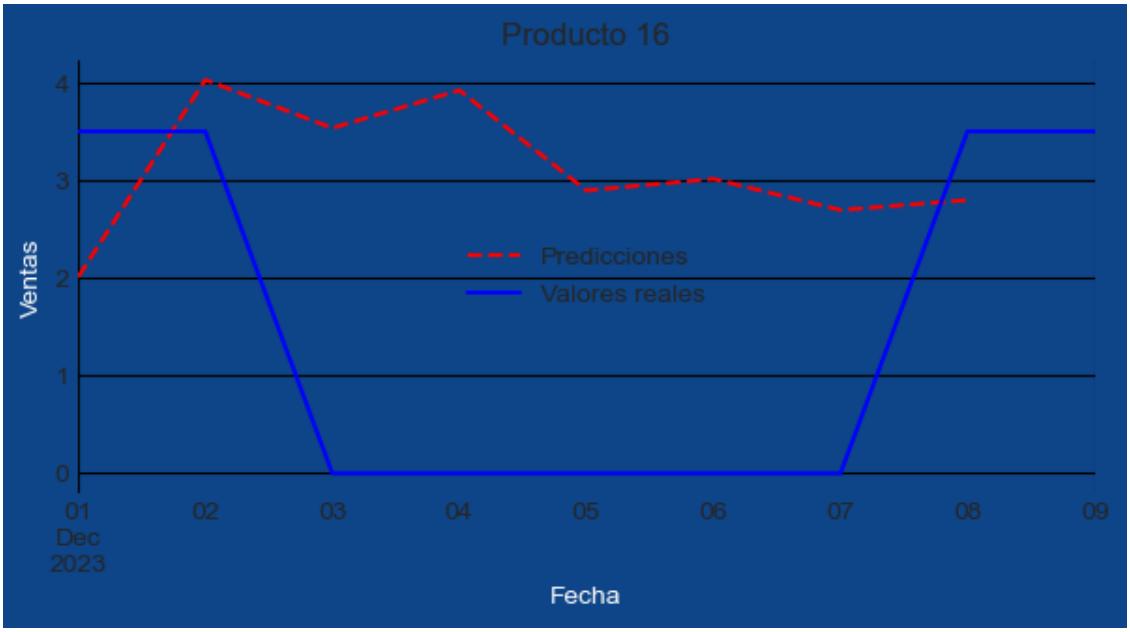

Producto 22
Predicciones
Valores reales
Ventas
0
1
2
3
4
5
6
7
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

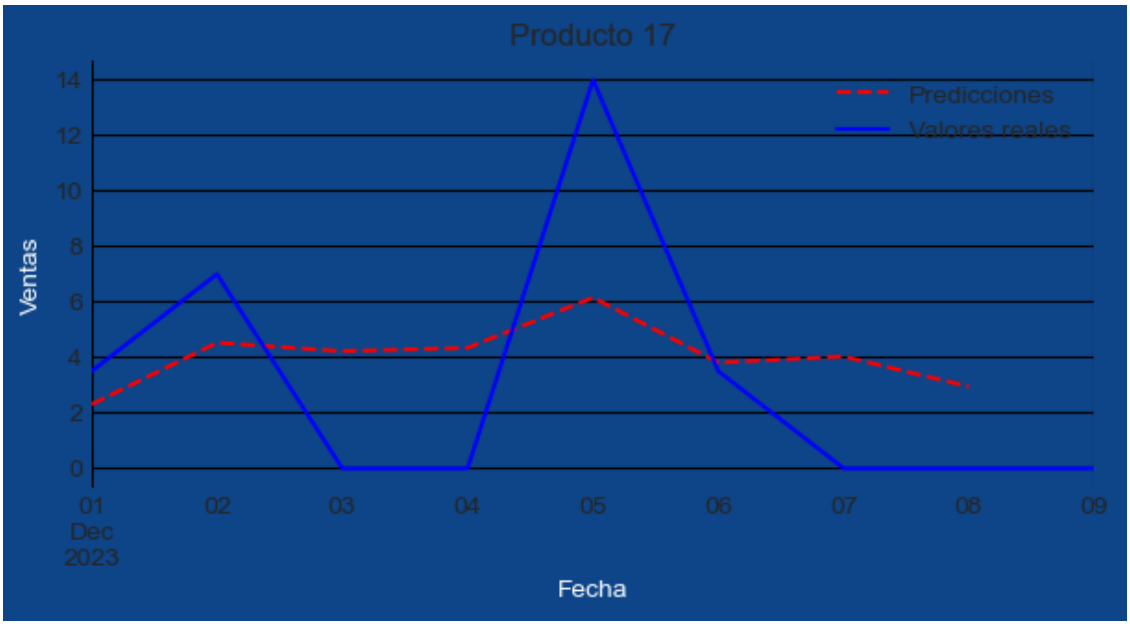

Producto 186
Predicciones
Valores reales
Ventas
0
2
4
6
8
10
12
14
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

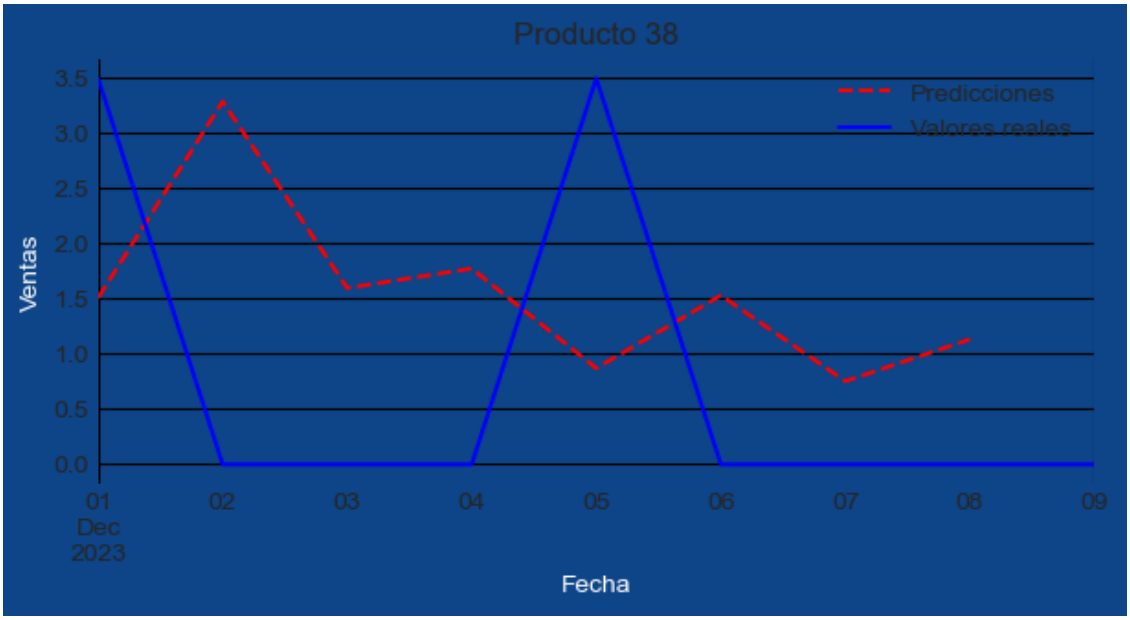

Producto 20
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

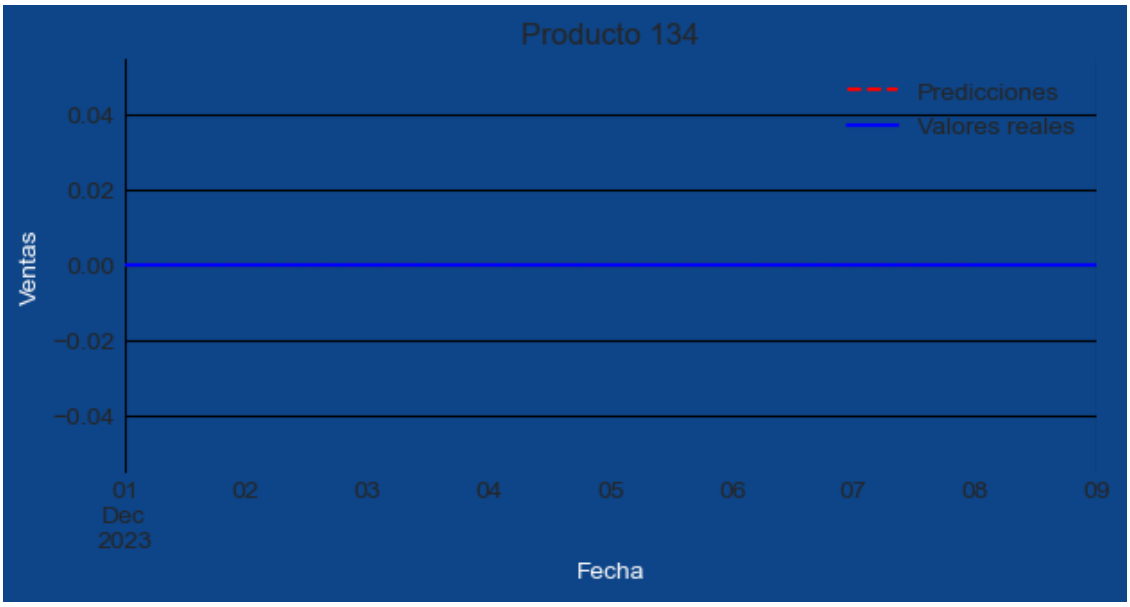

Producto 33
Predicciones
Valores reales
Ventas
14
12
10
8
6
4
2
0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

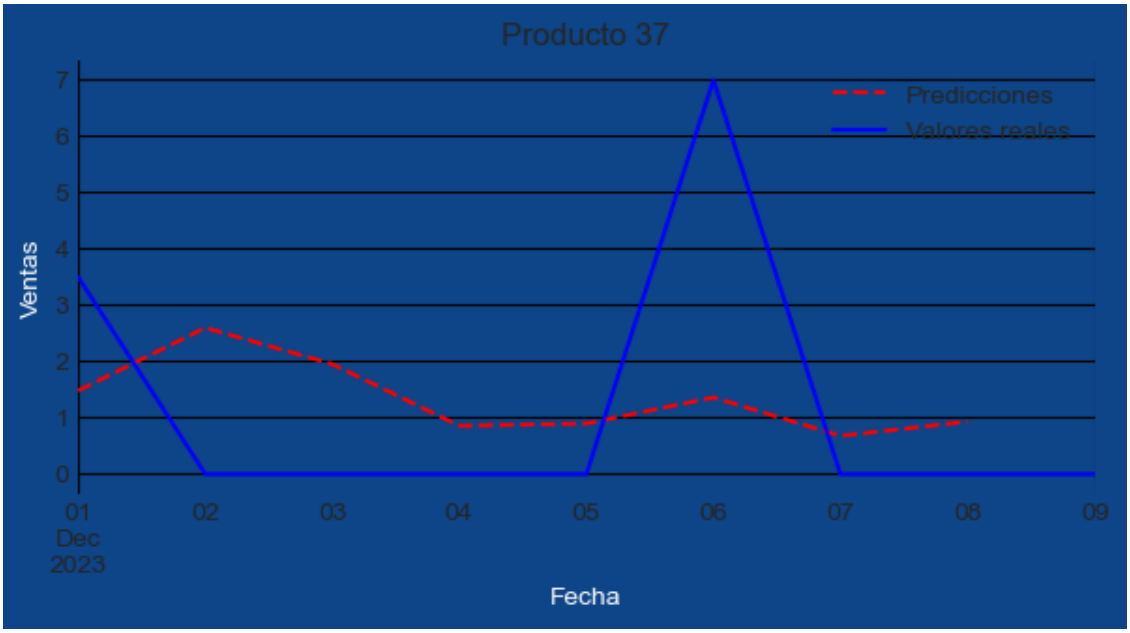

Producto 245
Predicciones
Valores reales
Ventas
0
1
2
3
4
5
6
7
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

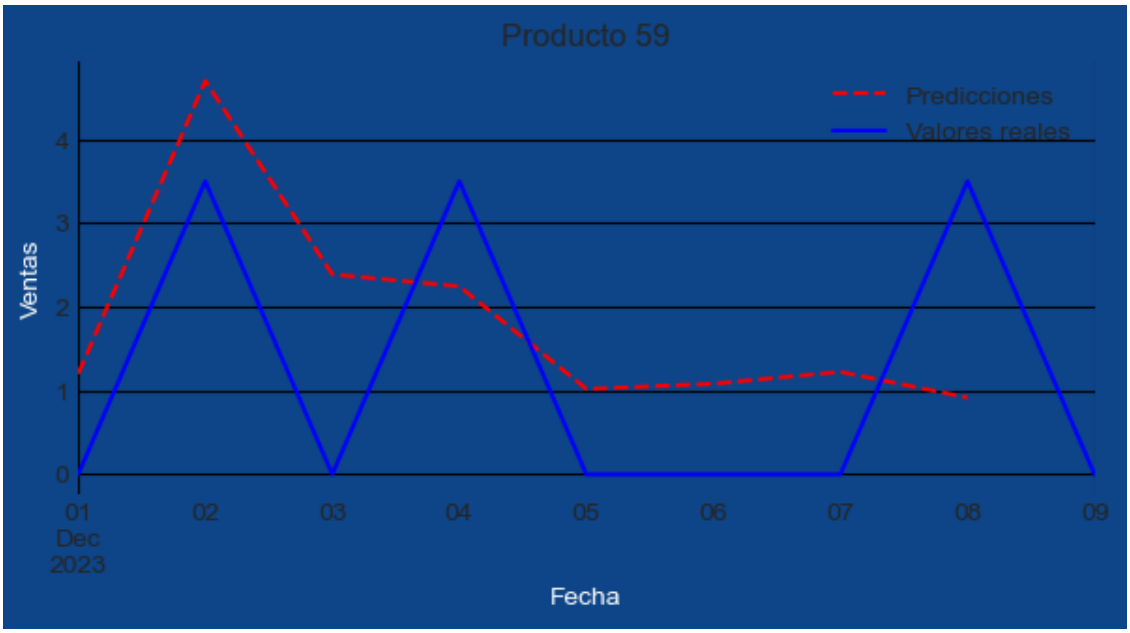

Producto 16
Predicciones
Valores reales
Ventas
0
1
2
3
4
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

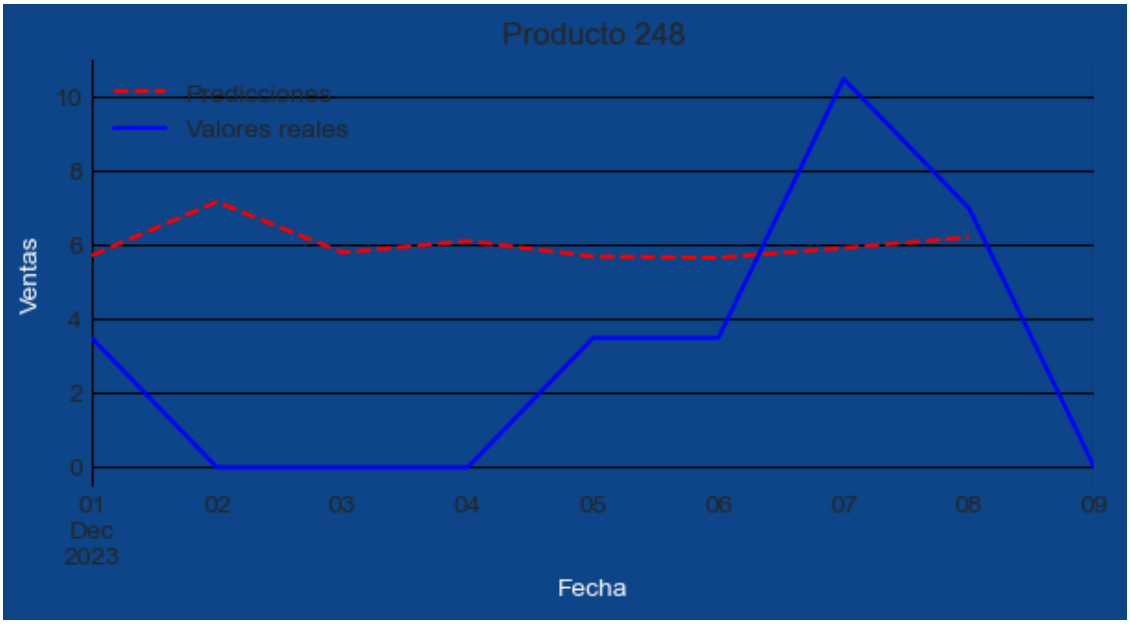

Producto 17
Predicciones
Valores reales
Ventas
14
12
10
8
6
4
2
0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha

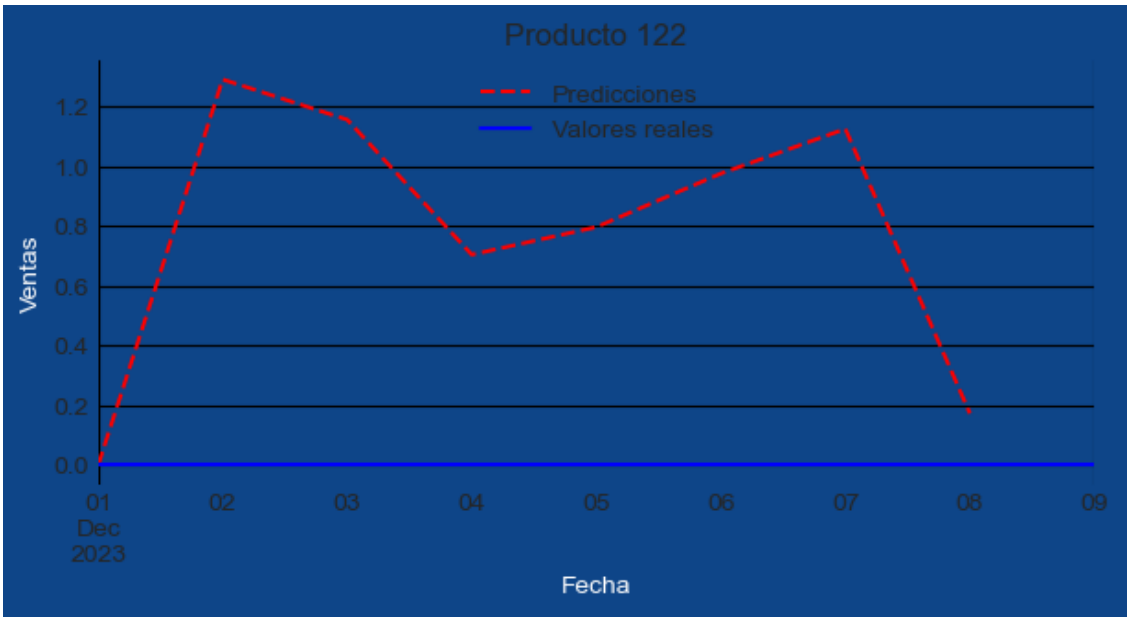

Producto 38
Predicciones
Valores reales
Ventas
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha


Producto 134
Predicciones
Valores reales
0.04
0.02
0.00
−0.02
−0.04
Ventas
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha


Producto 37
Predicciones
Valores reales
7
6
5
4
3
2
1
0
Ventas
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha


Producto 59
Predicciones
Valores reales
Ventas
0
1
2
3
4
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha


Producto 248
Predicciones
Valores reales
Ventas
0
2
4
6
8
10
01
Dec
2023
02
03
04
05
06
07
08
09
Fecha



4.1 Agregar modelo del otro producto y comparar

[ ]:
```
week_pred.to_csv("predicciones_semanales_Sari.csv")
month_pred.to_csv("predicciones_mensuales_Sari.csv")
data.to_csv("datatop20.csv")
```