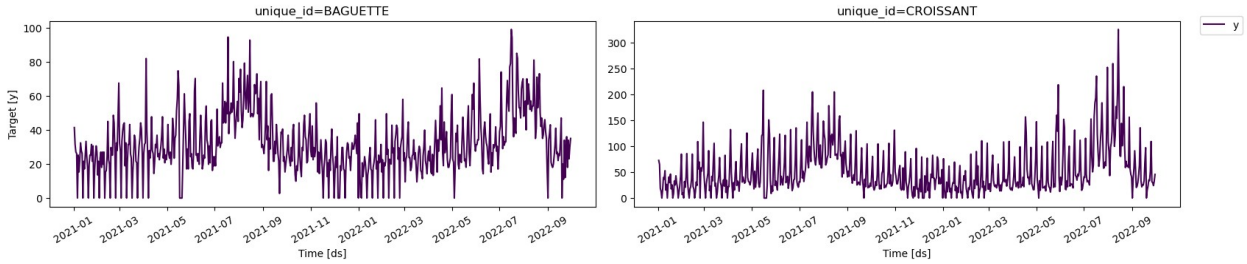
```
pip install statsforecast utilsforecast

Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: statsforecast in c:\users\sheri\
appdata\roaming\python\python311\site-packages (2.0.2)
Requirement already satisfied: utilsforecast in c:\users\sheri\
appdata\roaming\python\python311\site-packages (0.2.12)
Requirement already satisfied: cloudpickle in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (2.2.1)
Requirement already satisfied: coreforecast>=0.0.12 in c:\users\sheri\
appdata\roaming\python\python311\site-packages (from statsforecast)
(0.0.16)
Requirement already satisfied: numba>=0.55.0 in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (0.59.0)
Requirement already satisfied: numpy>=1.21.6 in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (1.26.4)
Requirement already satisfied: pandas>=1.3.5 in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (2.1.4)
Requirement already satisfied: scipy<1.16.0,>=1.7.3 in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (1.11.4)
Requirement already satisfied: statsmodels>=0.13.2 in c:\programdata\
anaconda3\lib\site-packages (from statsforecast) (0.14.0)
Requirement already satisfied: tqdm in c:\programdata\anaconda3\lib\
site-packages (from statsforecast) (4.65.0)
Requirement already satisfied: fugue>=0.8.1 in c:\users\sheri\appdata\
roaming\python\python311\site-packages (from statsforecast) (0.9.1)
Requirement already satisfied: threadpoolctl>=3 in c:\users\sheri\
appdata\roaming\python\python311\site-packages (from statsforecast)
(3.6.0)
Requirement already satisfied: packaging in c:\programdata\anaconda3\
lib\site-packages (from utilsforecast) (23.1)
Requirement already satisfied: triad>=0.9.7 in c:\users\sheri\appdata\
roaming\python\python311\site-packages (from fugue>=0.8.1-
>statsforecast) (0.9.8)
Requirement already satisfied: adagio>=0.2.4 in c:\users\sheri\
appdata\roaming\python\python311\site-packages (from fugue>=0.8.1-
>statsforecast) (0.2.6)
Requirement already satisfied: llvmlite<0.43,>=0.42.0dev0 in c:\
programdata\anaconda3\lib\site-packages (from numba>=0.55.0-
>statsforecast) (0.42.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\
programdata\anaconda3\lib\site-packages (from pandas>=1.3.5-
>statsforecast) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=1.3.5->statsforecast)
(2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas>=1.3.5->statsforecast)
(2023.3)
```

```
Requirement already satisfied: patsy>=0.5.2 in c:\programdata\
anaconda3\lib\site-packages (from statsmodels>=0.13.2->statsforecast)
(0.5.3)
Requirement already satisfied: colorama in c:\programdata\anaconda3\
lib\site-packages (from tqdm->statsforecast) (0.4.6)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\
site-packages (from patsy>=0.5.2->statsmodels>=0.13.2->statsforecast)
(1.16.0)
Requirement already satisfied: pyarrow>=6.0.1 in c:\programdata\
anaconda3\lib\site-packages (from triad>=0.9.7->fugue>=0.8.1-
>statsforecast) (14.0.2)
Requirement already satisfied: fsspec>=2022.5.0 in c:\programdata\
anaconda3\lib\site-packages (from triad>=0.9.7->fugue>=0.8.1-
>statsforecast) (2023.10.0)
Requirement already satisfied: fs in c:\users\sheri\appdata\roaming\
python\python311\site-packages (from triad>=0.9.7->fugue>=0.8.1-
>statsforecast) (2.4.16)
Requirement already satisfied: appdirs~=1.4.3 in c:\programdata\
anaconda3\lib\site-packages (from fs->triad>=0.9.7->fugue>=0.8.1-
>statsforecast) (1.4.4)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\
lib\site-packages (from fs->triad>=0.9.7->fugue>=0.8.1->statsforecast)
(68.2.2)
Note: you may need to restart the kernel to use updated packages.
```

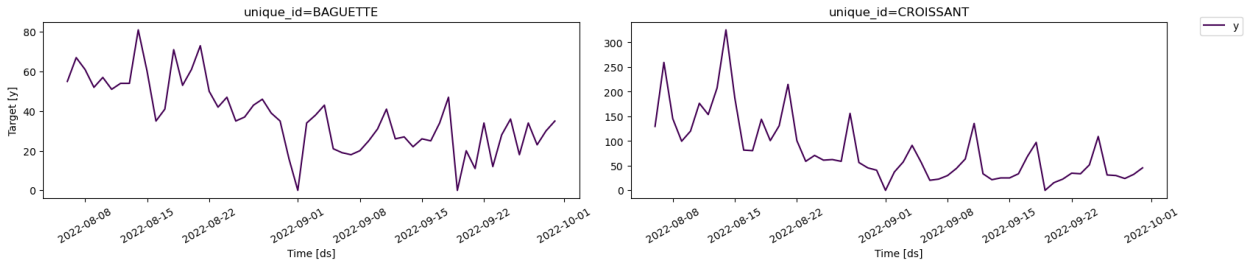
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from utilsforecast.plotting import plot_series
from utilsforecast.evaluation import evaluate
from utilsforecast.losses import *

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv(r"C:\Users\sheri\Downloads\daily_sales_french_bakery
(3).csv", parse_dates=["ds"])
df = df.groupby('unique_id').filter(lambda x: len(x) >= 28)
df = df.drop(["unit_price"], axis=1)
df.head()
```

	unique_id	ds	y
0	12 MACARON	2022-07-13	10.0
1	12 MACARON	2022-07-14	0.0
2	12 MACARON	2022-07-15	0.0
3	12 MACARON	2022-07-16	10.0
4	12 MACARON	2022-07-17	30.0

```
plot_series(df=df, ids=["BAGUETTE", "CROISSANT"], palette="viridis")
```




```
plot_series(df=df, ids=["BAGUETTE", "CROISSANT"],
max_insample_length=56, palette="viridis")
```

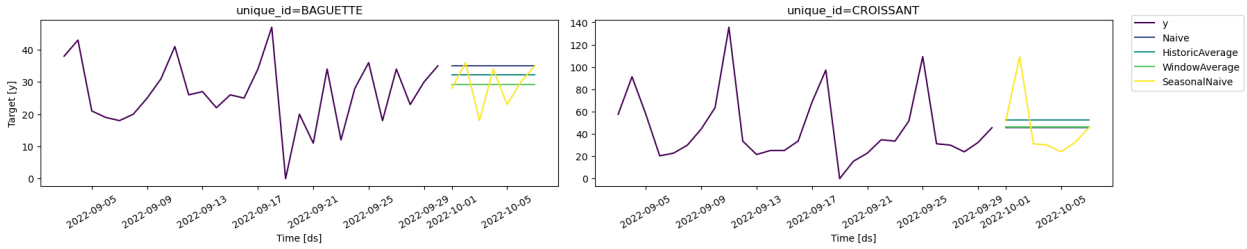



```
from statsforecast import StatsForecast
from statsforecast.models import Naive, HistoricAverage,
WindowAverage, SeasonalNaive

horizon = 7

models = [
    Naive(),
    HistoricAverage(),
    WindowAverage(window_size=7),
    SeasonalNaive(season_length=7)
]

sf = StatsForecast(models=models, freq="D")
sf.fit(df=df)
preds = sf.predict(h=horizon)

preds.head()
```

```
    unique_id         ds  Naive  HistoricAverage  WindowAverage
SeasonalNaive
0  12 MACARON 2022-09-29   10.0         8.974359       2.857143
0.0
1  12 MACARON 2022-09-30   10.0         8.974359       2.857143
0.0
2  12 MACARON 2022-10-01   10.0         8.974359       2.857143
10.0
3  12 MACARON 2022-10-02   10.0         8.974359       2.857143
0.0
```

```
4  12 MACARON 2022-10-03   10.0         8.974359       2.857143
0.0
```

```
plot_series(
    df=df,
    forecasts_df=preds,
    ids=["BAGUETTE", "CROISSANT"],
    max_insample_length=28,
    palette="viridis")
```



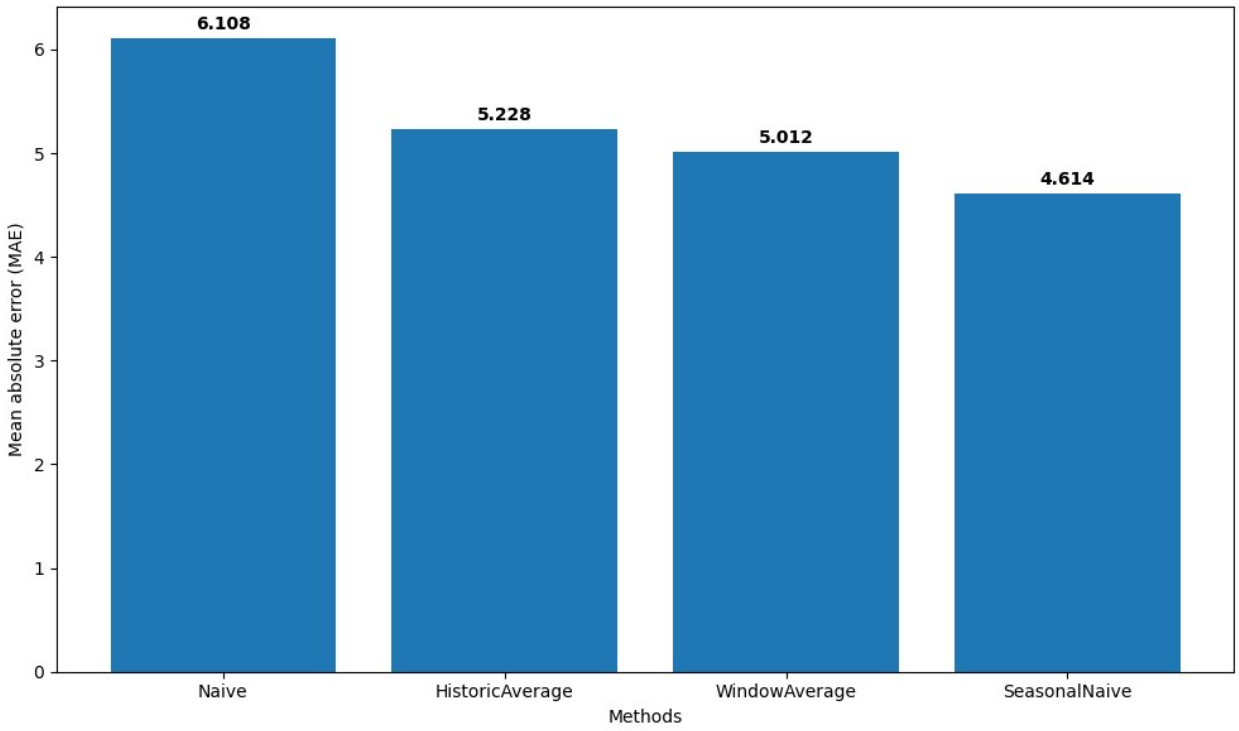

```
test = df.groupby("unique_id").tail(7)
train = df.drop(test.index).reset_index(drop=True)

sf.fit(df=train)

preds = sf.predict(h=horizon)

eval_df = pd.merge(test, preds, 'left', ['ds', 'unique_id'])

evaluation = evaluate(
    eval_df,
    metrics=[mae],
)
evaluation.head()
```

```
         unique_id metric      Naive  HistoricAverage
WindowAverage  \
0       12 MACARON    mae   2.857143         6.961771       3.469388

1         BAGUETTE    mae  17.142857         5.455193       7.877551

2   BAGUETTE APERO    mae   0.000000         0.537572       0.642857

3  BAGUETTE GRAINE    mae   9.800000         4.612271       2.942857

4          BANETTE    mae   1.314286         5.421984       6.008163


   SeasonalNaive
0       4.285714
1      12.571429
2       0.642857
```

```
3       0.200000
4       7.885714

evaluation = evaluation.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
evaluation
```

	metric	Naive	HistoricAverage	WindowAverage	SeasonalNaive
0	mae	6.107556	5.228439	5.011663	4.613636

```
methods = evaluation.columns[1:].tolist()
values = evaluation.iloc[0, 1:].tolist()

plt.figure(figsize=(10, 6))
bars = plt.bar(methods, values)

for bar, value in zip(bars, values):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,

             f'{value:.3f}', ha='center', va='bottom',
fontweight='bold')

plt.xlabel('Methods')
plt.ylabel('Mean absolute error (MAE)')
plt.tight_layout()

plt.show()
```



```
from statsforecast.models import AutoARIMA

unique_ids = ["BAGUETTE", "CROISSANT"]

small_train = train[train["unique_id"].isin(unique_ids)]
small_test = test[test["unique_id"].isin(unique_ids)]

models = [
    AutoARIMA(seasonal=False, alias="ARIMA"),
    AutoARIMA(season_length=7, alias="SARIMA")
]

sf = StatsForecast(models=models, freq="D")
sf.fit(df=small_train)
arima_preds = sf.predict(h=horizon)

arima_eval_df = pd.merge(arima_preds, eval_df, 'inner', ['ds',
'unique_id'])
arima_eval = evaluate(
    arima_eval_df,
    metrics=[mae],
)
arima_eval
```

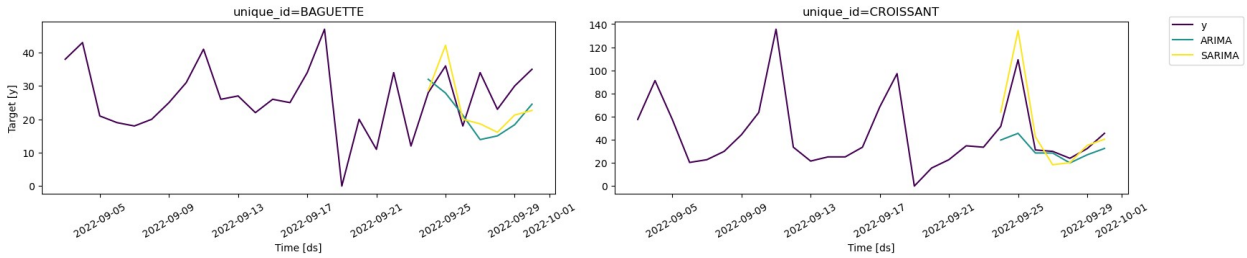
```
   unique_id metric      ARIMA     SARIMA      Naive  HistoricAverage
\
0   BAGUETTE    mae   9.353153   7.449084  17.142857         5.455193

1  CROISSANT    mae  14.565395  10.359143  17.485714        22.618934


   WindowAverage  SeasonalNaive
0       7.877551      12.571429
1      18.244898      12.857143
```

```
arima_eval = arima_eval.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
arima_eval
```

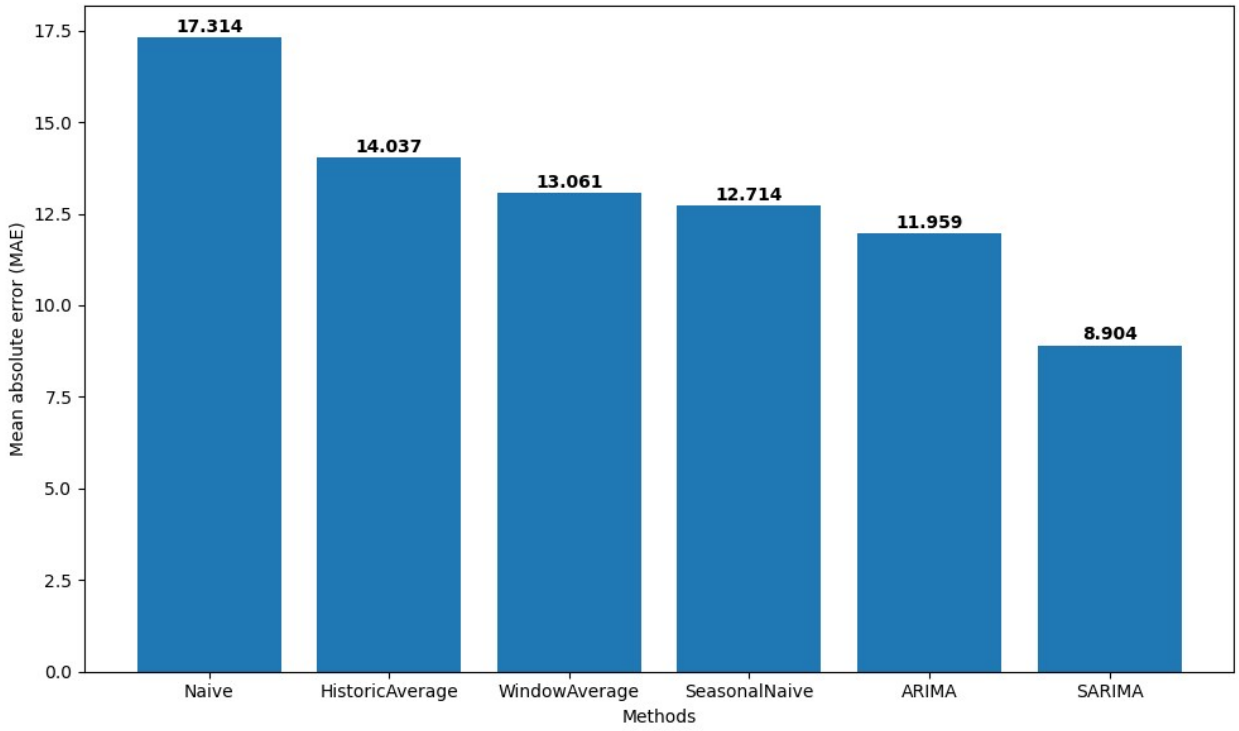
```
  metric      ARIMA    SARIMA      Naive  HistoricAverage
WindowAverage  \
0    mae  11.959274  8.904113  17.314286        14.037063
13.061224

   SeasonalNaive
0      12.714286
```

```
plot_series(
    df=df,
    forecasts_df=arima_preds,
    ids=["BAGUETTE", "CROISSANT"],
```

```
    max_insample_length=28,
    palette="viridis")
```




```
methods = arima_eval.columns[1:].tolist()
values = arima_eval.iloc[0, 1:].tolist()

sorted_data = sorted(zip(methods, values), key=lambda x: x[1],
reverse=True)
methods_sorted, values_sorted = zip(*sorted_data)

plt.figure(figsize=(10, 6))
bars = plt.bar(methods_sorted, values_sorted)

for bar, value in zip(bars, values_sorted):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,

             f'{value:.3f}', ha='center', va='bottom',
fontweight='bold')

plt.xlabel('Methods')
plt.ylabel('Mean absolute error (MAE)')
plt.tight_layout()

plt.show()
```

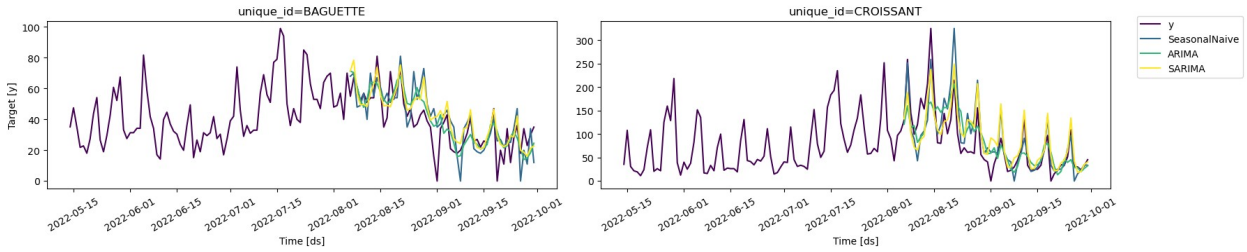



```
small_df = df[df["unique_id"].isin(unique_ids)]

models = [
    SeasonalNaive(season_length=7),
    AutoARIMA(seasonal=False, alias="ARIMA"),
    AutoARIMA(season_length=7, alias="SARIMA")
]

sf = StatsForecast(models=models, freq="D")
cv_df = sf.cross_validation(
    h=horizon, # 7 days
    df=small_df,
    n_windows=8,
    step_size=horizon,
    refit=True
)

cv_df.head()
```

```
  unique_id         ds     cutoff     y  SeasonalNaive      ARIMA
SARIMA
0  BAGUETTE 2022-08-06 2022-08-05  55.0           68.0  71.355198
71.584715
1  BAGUETTE 2022-08-07 2022-08-05  67.0           70.0  70.337981
78.458885
2  BAGUETTE 2022-08-08 2022-08-05  61.0           48.0  61.195006
57.001733
```

```
3  BAGUETTE 2022-08-09 2022-08-05  52.0           49.0  52.649015
49.401146
4  BAGUETTE 2022-08-10 2022-08-05  57.0           57.0  47.785877
49.725280
```

```
plot_series(
    df=small_df,
    forecasts_df=cv_df.drop(["y", "cutoff"], axis=1),
    ids=["BAGUETTE", "CROISSANT"],
    max_insample_length=140,
    palette="viridis")
```

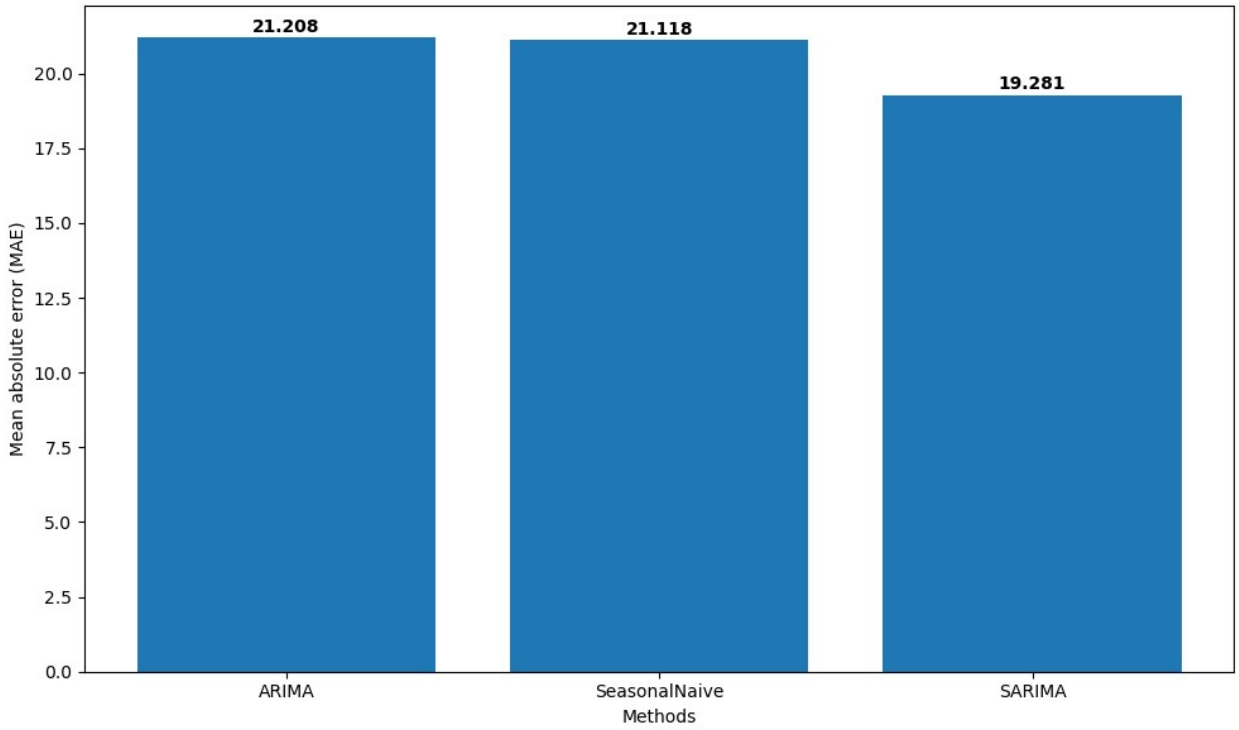



```
cv_eval = evaluate(
    cv_df.drop(["cutoff"], axis=1),
    metrics=[mae],
)
cv_eval = cv_eval.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
cv_eval
```

```
  metric  SeasonalNaive      ARIMA     SARIMA
0    mae      21.117857  21.208499  19.281295
```

```
methods = cv_eval.columns[1:].tolist()
values = cv_eval.iloc[0, 1:].tolist()

sorted_data = sorted(zip(methods, values), key=lambda x: x[1],
reverse=True)
methods_sorted, values_sorted = zip(*sorted_data)

plt.figure(figsize=(10, 6))
bars = plt.bar(methods_sorted, values_sorted)

for bar, value in zip(bars, values_sorted):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,

             f'{value:.3f}', ha='center', va='bottom',
fontweight='bold')

plt.xlabel('Methods')
plt.ylabel('Mean absolute error (MAE)')
plt.tight_layout()
```

```
plt.show()
```

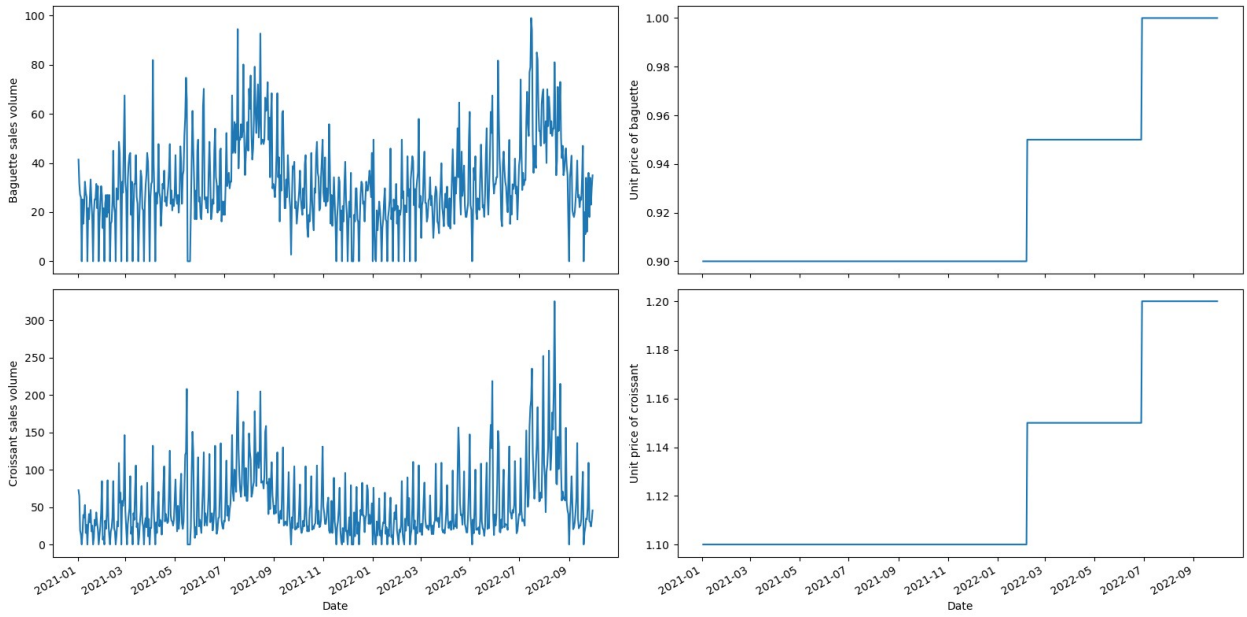



```
df = pd.read_csv(r"C:\Users\sheri\Downloads\daily_sales_french_bakery
(3).csv", parse_dates=["ds"])
df = df.groupby('unique_id').filter(lambda x: len(x) >= 28)
df.head()
```

	unique_id	ds	y	unit_price
0	12 MACARON	2022-07-13	10.0	10.0
1	12 MACARON	2022-07-14	0.0	10.0
2	12 MACARON	2022-07-15	0.0	10.0
3	12 MACARON	2022-07-16	10.0	10.0
4	12 MACARON	2022-07-17	30.0	10.0

```
baguette_plot_df = df[df["unique_id"] == "BAGUETTE"]
croissant_plot_df = df[df["unique_id"] == "CROISSANT"]

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2,
figsize=(16,8))

ax1.plot(baguette_plot_df["ds"], baguette_plot_df["y"])
ax1.set_xlabel("Date")
ax1.set_ylabel("Baguette sales volume")

ax2.plot(baguette_plot_df["ds"], baguette_plot_df["unit_price"])
ax2.set_xlabel("Date")
```

```
ax2.set_ylabel("Unit price of baguette")

ax3.plot(croissant_plot_df["ds"], croissant_plot_df["y"])
ax3.set_xlabel("Date")
ax3.set_ylabel("Croissant sales volume")

ax4.plot(croissant_plot_df["ds"], croissant_plot_df["unit_price"])
ax4.set_xlabel("Date")
ax4.set_ylabel("Unit price of croissant")

fig.autofmt_xdate()
plt.tight_layout()
```




```
unique_ids = ["BAGUETTE", "CROISSANT"]
small_df = df[df["unique_id"].isin(unique_ids)]
test = small_df.groupby("unique_id").tail(7)
train = small_df.drop(test.index).reset_index(drop=True)

futr_exog_df = test.drop(["y"], axis=1)
futr_exog_df.head()
```

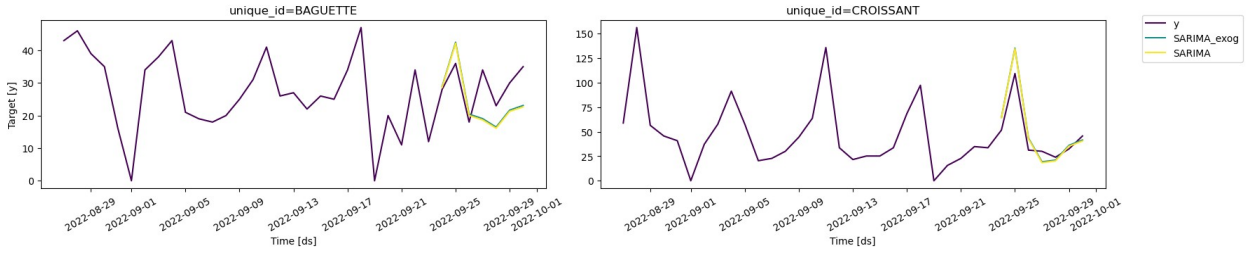
	unique_id	ds	unit_price
714	BAGUETTE	2022-09-24	1.0
715	BAGUETTE	2022-09-25	1.0
716	BAGUETTE	2022-09-26	1.0
717	BAGUETTE	2022-09-27	1.0
718	BAGUETTE	2022-09-28	1.0

```
models = [
    AutoARIMA(season_length=7, alias="SARIMA_exog")
]
```

```
sf = StatsForecast(models=models, freq="D")
sf.fit(df=train)
arima_exog_preds = sf.predict(h=7, X_df=futr_exog_df)

models = [
    AutoARIMA(season_length=7, alias="SARIMA")
]

sf = StatsForecast(models=models, freq="D")
sf.fit(df=train.drop(["unit_price"], axis=1))
arima_preds = sf.predict(h=horizon)

test_df = test.merge(arima_exog_preds, on=["unique_id", "ds"],
how="left")\
              .merge(arima_preds, on=["unique_id", "ds"], how="left")
test_df
```

	unique_id	ds	y	unit_price	SARIMA_exog	SARIMA
0	BAGUETTE	2022-09-24	28.0	1.0	28.898805	28.657114
1	BAGUETTE	2022-09-25	36.0	1.0	42.513133	42.182373
2	BAGUETTE	2022-09-26	18.0	1.0	20.423812	20.013655
3	BAGUETTE	2022-09-27	34.0	1.0	19.066095	18.646490
4	BAGUETTE	2022-09-28	23.0	1.0	16.468747	16.114926
5	BAGUETTE	2022-09-29	30.0	1.0	21.656586	21.288016
6	BAGUETTE	2022-09-30	35.0	1.0	23.101064	22.660125
7	CROISSANT	2022-09-24	51.6	1.2	64.424299	64.011003
8	CROISSANT	2022-09-25	109.2	1.2	135.123242	134.488203
9	CROISSANT	2022-09-26	31.2	1.2	43.524909	42.735408
10	CROISSANT	2022-09-27	30.0	1.2	19.288689	18.400718
11	CROISSANT	2022-09-28	24.0	1.2	21.043339	20.222399
12	CROISSANT	2022-09-29	32.4	1.2	36.118929	35.192346
13	CROISSANT	2022-09-30	45.6	1.2	41.484971	40.489843

```
plot_series(
    df=train,
    forecasts_df=test_df,
    ids=["BAGUETTE", "CROISSANT"],
    max_insample_length=28,
    models=["SARIMA_exog", "SARIMA"],
    palette="viridis"
)
```



```
models = [
    AutoARIMA(season_length=7, alias="SARIMA_exog")
]

sf = StatsForecast(models=models, freq="D")
cv_exog_df = sf.cross_validation(
    h=horizon, # 7 days
    df=small_df,
    n_windows=8,
    step_size=7,
    refit=True
)

cv_exog_df.head()
```

	unique_id	ds	cutoff	y	SARIMA_exog
0	BAGUETTE	2022-08-06	2022-08-05	55.0	71.511253
1	BAGUETTE	2022-08-07	2022-08-05	67.0	78.457742
2	BAGUETTE	2022-08-08	2022-08-05	61.0	57.062387
3	BAGUETTE	2022-08-09	2022-08-05	52.0	49.525489
4	BAGUETTE	2022-08-10	2022-08-05	57.0	49.485269

```
cv_exog_eval = evaluate(
    cv_exog_df.drop(["cutoff"], axis=1),
    metrics=[mae],
)
cv_exog_eval = cv_exog_eval.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
cv_exog_eval
```

	metric	SARIMA_exog
0	mae	19.210235

```
from functools import partial
from utilsforecast.feature_engineering import fourier, time_features,
pipeline

features = [
    partial(fourier, season_length=7, k=2),
    partial(time_features, features=["day", "week", "month"])
]

small_exog_df, futr_exog_df = pipeline(
    df=small_df,
    features=features,
    freq="D",
    h=horizon
)

small_exog_df.head()
```

```
   unique_id         ds     y  unit_price    sin1_7    sin2_7
cos1_7  \
84  BAGUETTE 2021-01-02  41.4         0.9  0.781832  0.974928
0.623490
85  BAGUETTE 2021-01-03  31.5         0.9  0.974928 -0.433884 -
0.222521
86  BAGUETTE 2021-01-04  27.0         0.9  0.433884 -0.781831 -
0.900969
87  BAGUETTE 2021-01-05  26.1         0.9 -0.433884  0.781832 -
0.900969
88  BAGUETTE 2021-01-06   0.0         0.9 -0.974928  0.433884 -
0.222521

      cos2_7  day week  month
84 -0.222521    2   53      1
85 -0.900969    3   53      1
86  0.623490    4    1      1
87  0.623490    5    1      1
88 -0.900969    6    1      1
```

futr_exog_df

```
    unique_id         ds    sin1_7    sin2_7    cos1_7    cos2_7  day
week  \
0    BAGUETTE 2022-10-01  0.781844  0.974919  0.623474 -0.222559    1
39
1    BAGUETTE 2022-10-02  0.974927 -0.433892 -0.222526 -0.900965    2
39
2    BAGUETTE 2022-10-03  0.433893 -0.781844 -0.900964  0.623474    3
40
3    BAGUETTE 2022-10-04 -0.433861  0.781800 -0.900980  0.623529    4
40
4    BAGUETTE 2022-10-05 -0.974933  0.433846 -0.222500 -0.900987    5
40
5    BAGUETTE 2022-10-06 -0.781828 -0.974931  0.623495 -0.222509    6
40
6    BAGUETTE 2022-10-07 -0.000009 -0.000017  1.000000  1.000000    7
40
7   CROISSANT 2022-10-01  0.781844  0.974919  0.623474 -0.222559    1
39
8   CROISSANT 2022-10-02  0.974927 -0.433892 -0.222526 -0.900965    2
39
9   CROISSANT 2022-10-03  0.433893 -0.781844 -0.900964  0.623474    3
40
10  CROISSANT 2022-10-04 -0.433861  0.781800 -0.900980  0.623529    4
40
11  CROISSANT 2022-10-05 -0.974933  0.433846 -0.222500 -0.900987    5
40
12  CROISSANT 2022-10-06 -0.781828 -0.974931  0.623495 -0.222509    6
40
```

```
13  CROISSANT 2022-10-07 -0.000009 -0.000017  1.000000  1.000000    7
40
```

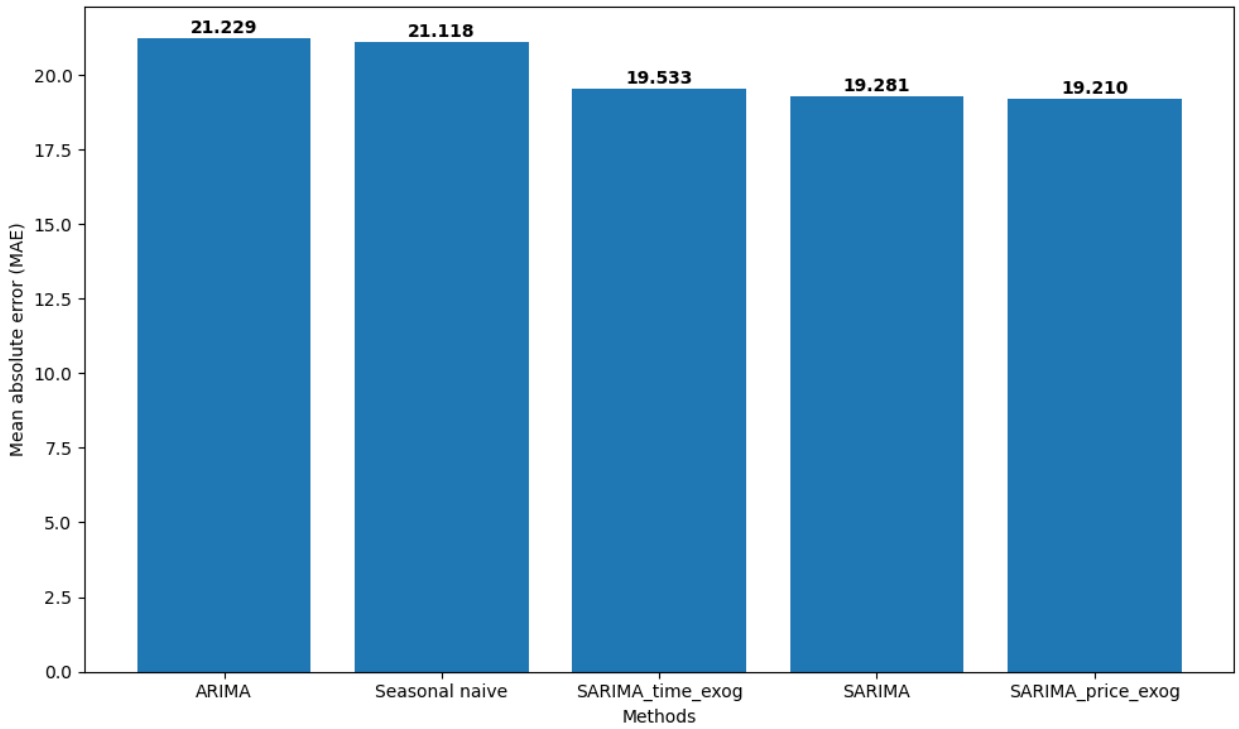
```
    month
0      10
1      10
2      10
3      10
4      10
5      10
6      10
7      10
8      10
9      10
10     10
11     10
12     10
13     10
```

```
models = [
    AutoARIMA(season_length=7, alias="SARIMA_time_exog")
]

sf = StatsForecast(models=models, freq="D")
cv_time_exog_df = sf.cross_validation(
    h=horizon, # 7 days
    df=small_exog_df,
    n_windows=8,
    step_size=horizon,
    refit=True
)

cv_time_exog_eval = evaluate(
    cv_time_exog_df.drop(["cutoff"], axis=1),
    metrics=[mae],
)
cv_time_exog_eval = cv_time_exog_eval.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
cv_time_exog_eval
```

```
  metric  SARIMA_time_exog
0    mae         19.495375
```

```
methods = ["ARIMA", "Seasonal naive", "SARIMA", "SARIMA_price_exog",
"SARIMA_time_exog"]
values = [21.229, 21.118, 19.281, 19.210, 19.533]

sorted_data = sorted(zip(methods, values), key=lambda x: x[1],
reverse=True)
methods_sorted, values_sorted = zip(*sorted_data)
```

```
plt.figure(figsize=(10, 6))
bars = plt.bar(methods_sorted, values_sorted)

for bar, value in zip(bars, values_sorted):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.05,

             f'{value:.3f}', ha='center', va='bottom',
fontweight='bold')

plt.xlabel('Methods')
plt.ylabel('Mean absolute error (MAE)')
plt.tight_layout()

plt.show()
```

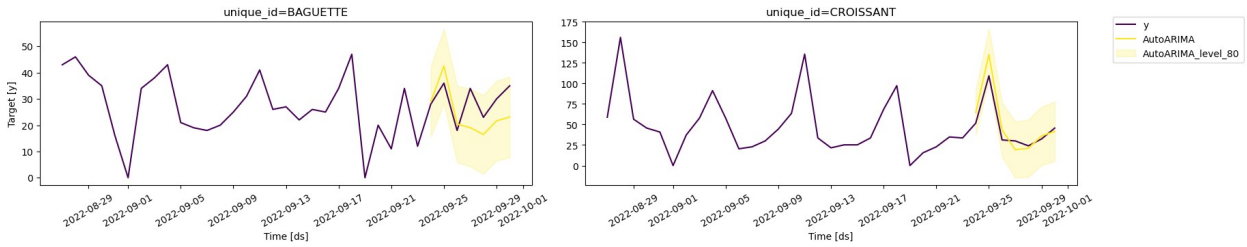



```
unique_ids = ["BAGUETTE", "CROISSANT"]
small_df = df[df["unique_id"].isin(unique_ids)]
test = small_df.groupby("unique_id").tail(7)
train = small_df.drop(test.index).reset_index(drop=True)

train.head()
```

	unique_id	ds	y	unit_price
0	BAGUETTE	2021-01-02	41.4	0.9
1	BAGUETTE	2021-01-03	31.5	0.9
2	BAGUETTE	2021-01-04	27.0	0.9

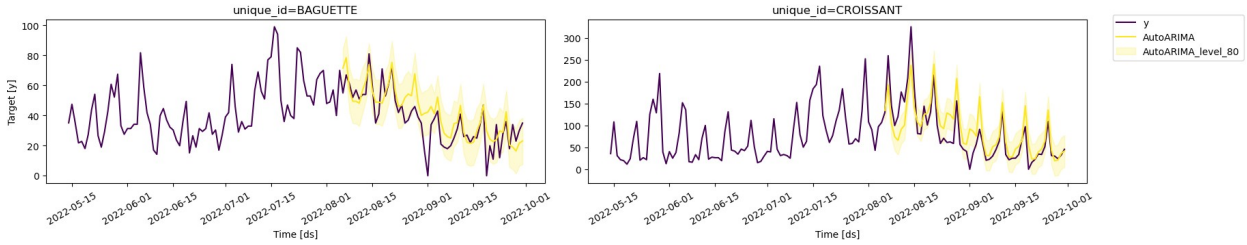
```
3  BAGUETTE 2021-01-05  26.1          0.9
4  BAGUETTE 2021-01-06   0.0          0.9
```

```
models = [
    AutoARIMA(season_length=7)
]

sf = StatsForecast(models=models, freq="D")
sf.fit(df=train)
prob_preds = sf.predict(h=horizon, X_df=test.drop(["y"], axis=1),
level=[80])

test_df = test.merge(prob_preds, on=["unique_id", "ds"], how="left")

plot_series(
    df=train,
    forecasts_df=test_df,
    ids=["BAGUETTE", "CROISSANT"],
    max_insample_length=28,
    models=["AutoARIMA"],
    level=[80],
    palette="viridis"
)
```




```
models = [
    AutoARIMA(season_length=7)
]

sf = StatsForecast(models=models, freq="D")
cv_prob_df = sf.cross_validation(
    h=horizon,
    df=small_df,
    n_windows=8,
    step_size=7,
    refit=True,
    level=[80],
)

plot_series(
    df=small_df,
    forecasts_df=cv_prob_df.drop(["y", "cutoff"], axis=1),
    ids=["BAGUETTE", "CROISSANT"],
```

```
    models=["AutoARIMA"],
    max_insample_length=140,
    level=[80],
    palette="viridis"
)
```




```
models = [
    AutoARIMA(season_length=7, alias="SARIMA_exog"),
    SeasonalNaive(season_length=7)
]

sf = StatsForecast(models=models, freq="D")
final_cv_df = sf.cross_validation(
    h=horizon,
    df=small_df,
    n_windows=8,
    step_size=7,
    refit=True,
    level=[80],
)

final_cv_df.head()
```

```
  unique_id         ds     cutoff     y  SARIMA_exog  SARIMA_exog-lo-
80  \
0  BAGUETTE 2022-08-06 2022-08-05  55.0    71.511253
58.278934
1  BAGUETTE 2022-08-07 2022-08-05  67.0    78.457742
64.353800
2  BAGUETTE 2022-08-08 2022-08-05  61.0    57.062387
42.621832
3  BAGUETTE 2022-08-09 2022-08-05  52.0    49.525489
34.846926
4  BAGUETTE 2022-08-10 2022-08-05  57.0    49.485269
34.606417

   SARIMA_exog-hi-80  SeasonalNaive  SeasonalNaive-lo-80
SeasonalNaive-hi-80
0          84.743573           68.0            50.158042
85.841958
1          92.561684           70.0            52.158042
```

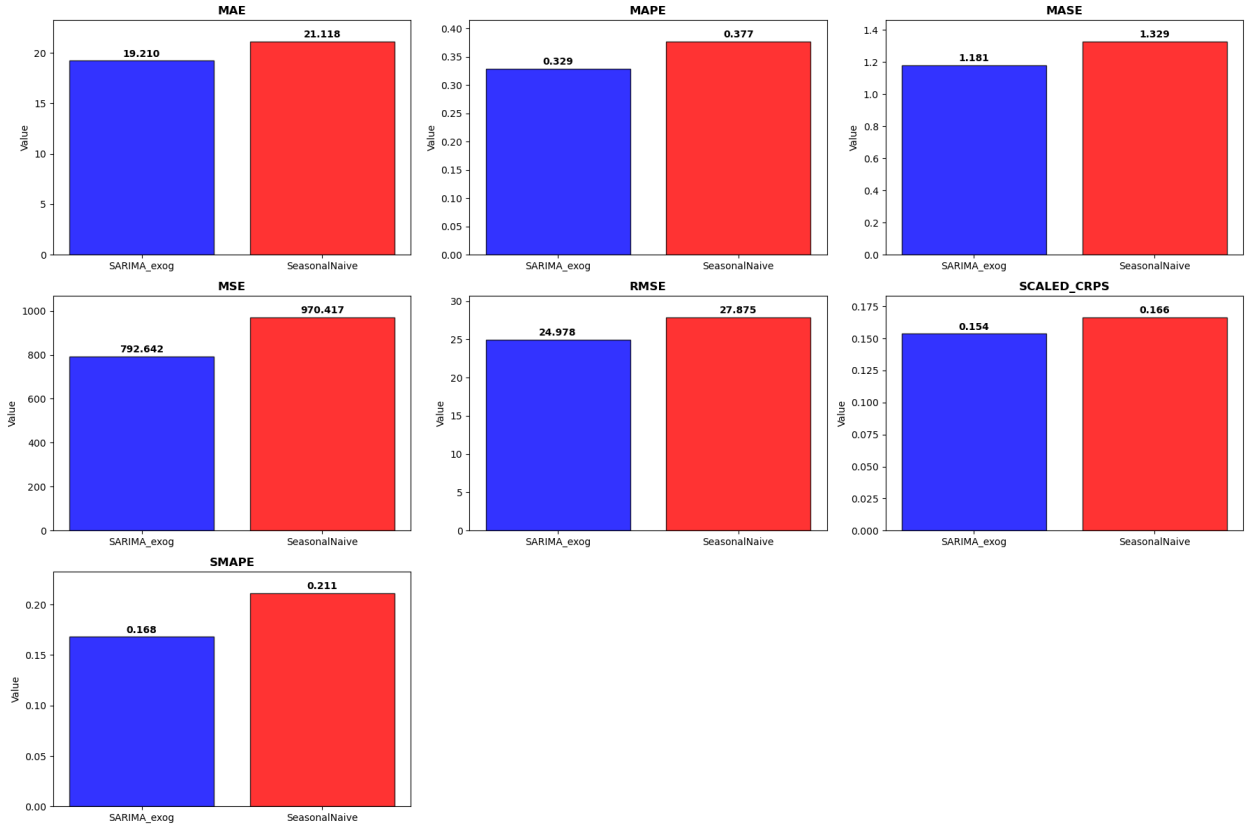
```
87.841958
2          71.502942            48.0             30.158042
65.841958
3          64.204052            49.0             31.158042
66.841958
4          64.364121            57.0             39.158042
74.841958
```

```
temp_test = small_df.groupby("unique_id").tail(7*8)
temp_train = small_df.drop(temp_test.index).reset_index(drop=True)

models = ["SARIMA_exog", "SeasonalNaive"]
metrics = [
    mae,
    mse,
    rmse,
    mape,
    smape,
    partial(mase, seasonality=7),
    scaled_crps
]

final_eval = evaluate(
    final_cv_df.drop(["ds", "cutoff"], axis=1),
    metrics=metrics,
    models=models,
    train_df=temp_train,
    level=[80]
)
final_eval = final_eval.drop(['unique_id'],
axis=1).groupby('metric').mean().reset_index()
final_eval
```

```
        metric  SARIMA_exog  SeasonalNaive
0          mae    19.210235      21.117857
1         mape     0.328596       0.376819
2         mase     1.181425       1.328592
3          mse   792.641646     970.417143
4         rmse    24.977638      27.875413
5  scaled_crps     0.153621       0.166451
6        smape     0.168232       0.211317
```

```
fig, axes = plt.subplots(3, 3, figsize=(18, 12))

axes_flat = axes.flatten()

models = ['SARIMA_exog', 'SeasonalNaive']
x_pos = [0, 1]
colors = ['blue', 'red']

for i, row in final_eval.iterrows():
```

```
    ax = axes_flat[i]

    model_values = [row['SARIMA_exog'], row['SeasonalNaive']]

    bars = ax.bar(x_pos, model_values, color=colors, alpha=0.8,
edgecolor='black', linewidth=1)

    for j, (bar, value) in enumerate(zip(bars, model_values)):
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height +
height*0.01,
                f'{value:.3f}', ha='center', va='bottom',
fontweight='bold', fontsize=10)

    ax.set_title(row['metric'].upper(), fontweight='bold',
fontsize=12)
    ax.set_xticks(x_pos)
    ax.set_xticklabels(models, ha='center')
    ax.set_ylabel('Value')
    max_value = max(model_values)
    ax.set_ylim(0, max_value * 1.1)

fig.delaxes(axes_flat[7])

axes_flat[8].set_visible(False)

plt.tight_layout()
plt.show()
```


MAE
19.210
21.118
MAPE
0.329
0.377
MASE
1.181
1.329
MSE
792.642
970.417
RMSE
24.978
27.875
SCALED_CRPS
0.154
0.166
SMAPE
0.168
0.211
Value
SARIMA_exog
SeasonalNaive