Summary

Window features

Motivation for window features

1. Want to predict future values of the target.

2. Past values of the target are likely to be predictive.

3. Past values of a feature could also be predictive (e.g., the sales on a day is related to advertising (ad) spend on prior days).

target ↓ feature ↓

Date	Sales	Ad spend
2020-02-12	23	100
2020-02-13	30	120
2020-02-14	35	90
2020-02-15	30	80
2020-02-16	?	100

Lag feature

- A past value of the target or feature is used to create a lag feature.

- Can we summarise more than one past value into a single feature?

target

features

Date	Sales	Sales Lag 1	Ad spend
2020-02-12	23	NaN	100
2020-02-13	30	23	120
2020-02-14	35	30	90
2020-02-15	30	35	80
2020-02-16	?	30	?

Window feature

- A past value of the target or feature is used to create a lag feature
- Can we summarise more than one past value into a single feature?
- A window feature is where we compute statistics (e.g., mean, min, max) using a window over the past data.

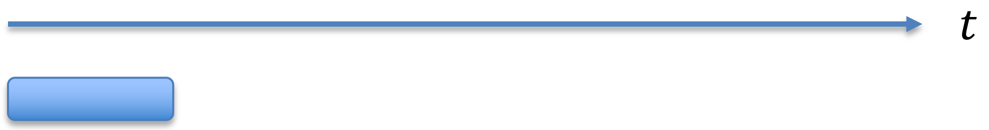
target

features

Date	Sales	Sales Window	Ad spend
2020-02-12	23		100
2020-02-13	30		120
2020-02-14	35		90
2020-02-15	30		80
2020-02-16	?		?

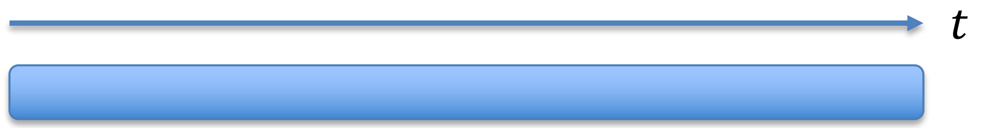
Rolling vs expanding windows

Rolling window



- Uses only some of the previous values at any step.
- Useful to capture recent behaviors of the time series.
- E.g., the average sales over the past one month.

Expanding window

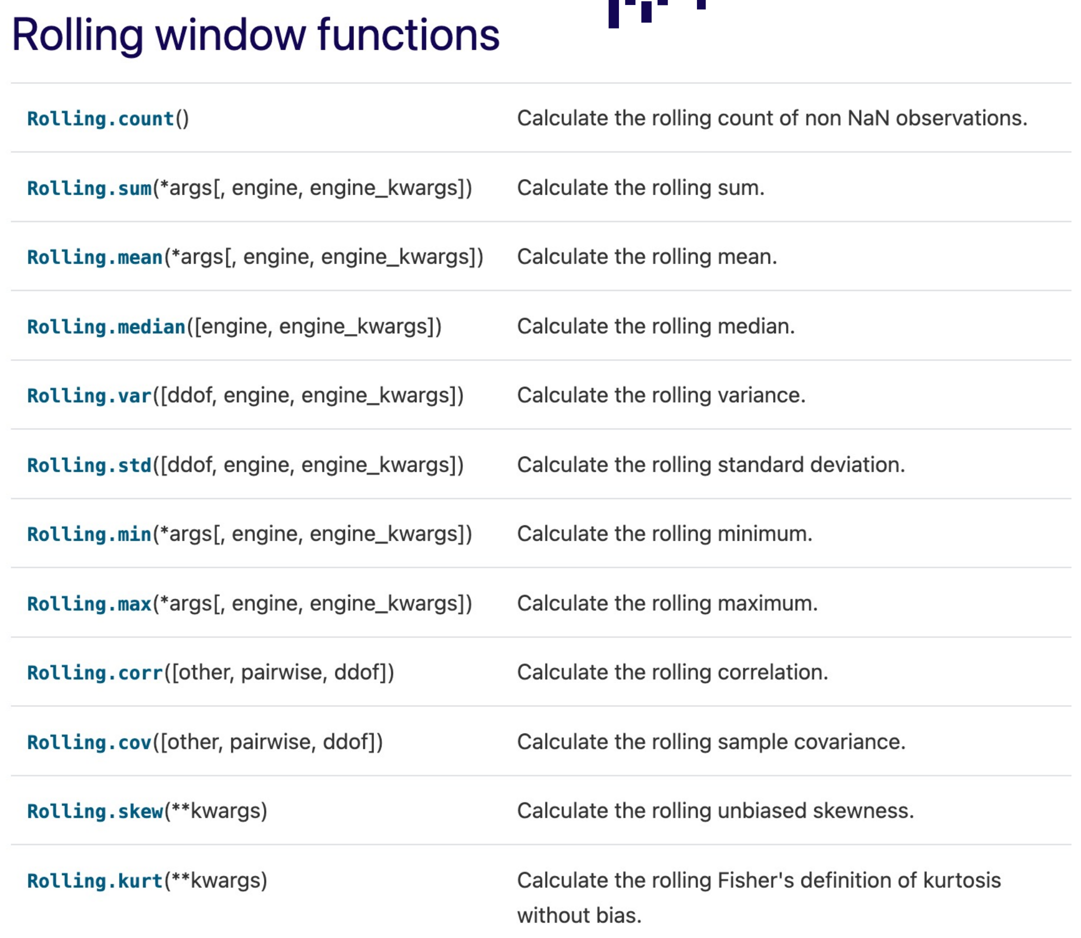


- Uses all previous values at any step.
- Useful when you need access to the entire time series.
- E.g., aggregating anything which has an accumulative effect. Target encoding.

How to pick which statistics?

- Keep it simple and use mean and standard deviation.
- Is accuracy good enough? Don’t make more complex than needed.
- Try a variety of features and use feature selection & modelling techniques (e.g., LASSO to reduce number of features).

pandas

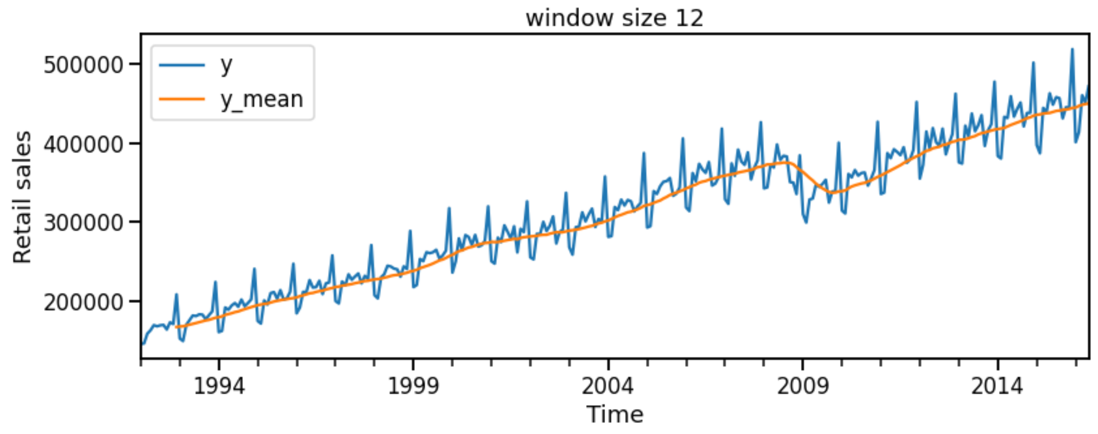


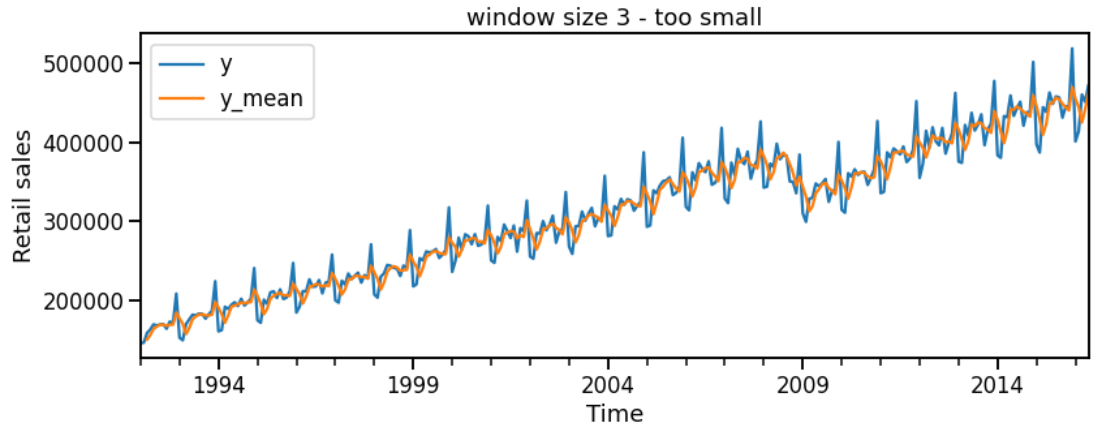
Rolling window functions

`Rolling.count`()	Calculate the rolling count of non NaN observations.
`Rolling.sum`(*args[, engine, engine_kwargs])	Calculate the rolling sum.
`Rolling.mean`(*args[, engine, engine_kwargs])	Calculate the rolling mean.
`Rolling.median`([engine, engine_kwargs])	Calculate the rolling median.
`Rolling.var`([ddof, engine, engine_kwargs])	Calculate the rolling variance.
`Rolling.std`([ddof, engine, engine_kwargs])	Calculate the rolling standard deviation.
`Rolling.min`(*args[, engine, engine_kwargs])	Calculate the rolling minimum.
`Rolling.max`(*args[, engine, engine_kwargs])	Calculate the rolling maximum.
`Rolling.corr`([other, pairwise, ddof])	Calculate the rolling correlation.
`Rolling.cov`([other, pairwise, ddof])	Calculate the rolling sample covariance.
`Rolling.skew`(**kwargs)	Calculate the rolling unbiased skewness.
`Rolling.kurt`(**kwargs)	Calculate the rolling Fisher's definition of kurtosis without bias.

How to pick the window size?

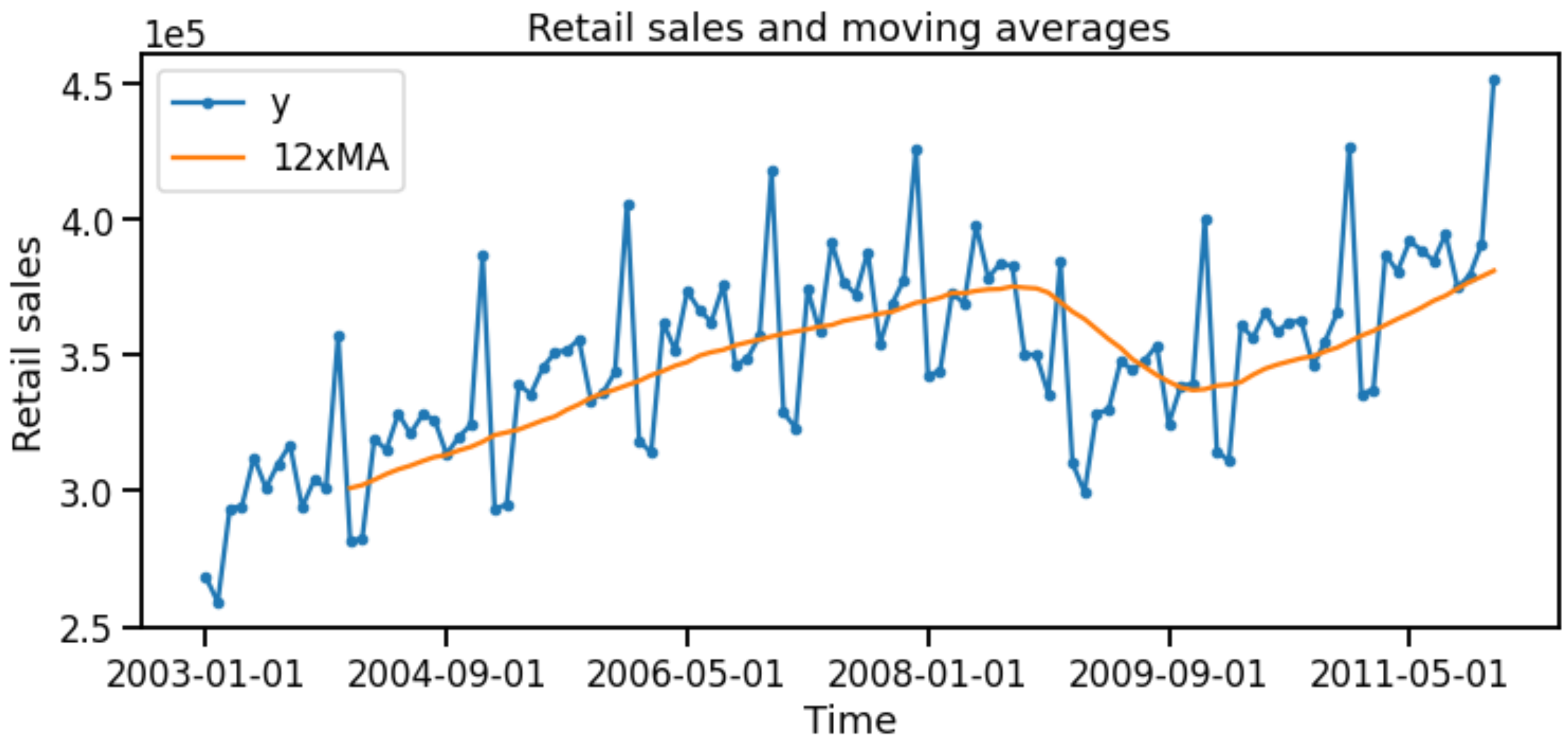
- Heuristic: Pick window size the same as seasonal period to smooth over seasonality.
- Different window sizes can capture different behaviors in the data (e.g., short term vs long term trends) → try different window sizes.
- Try multiple window sizes, use feature selection methods, and assess performance.





Motivation for weighted window functions

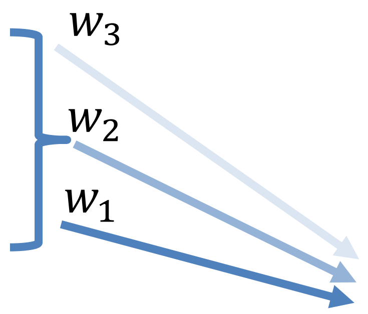
- What if we want to be more sensitive to recent observations, e.g., to quickly pick up changes in trend?



Weighted rolling mean

- We can specify weights to our window function to give a weighted average.

Date	Sales
…	…
2020-02-12	23
2020-02-13	30
2020-02-14	35
2020-02-15	?

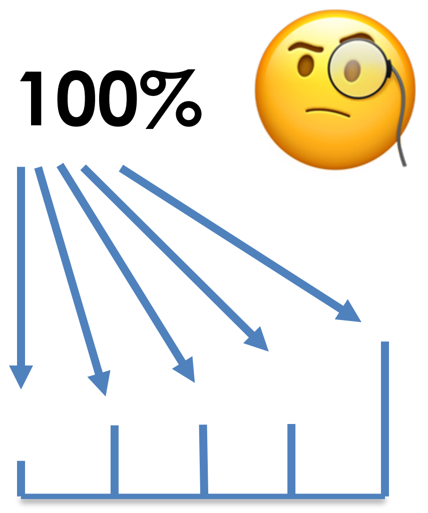


Rolling window (mean)
…
x

$$= \frac{w_1 35 + w_2 30 + w_3 23}{\sum_i w_i}$$

How to pick the weights?

- We can think of starting with 100% weight. We're spreading this over the window.
- How do we spread the weight?
  - Domain knowledge.
  - Linear & exponential weights.

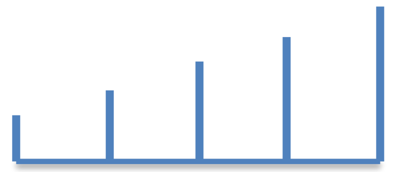


[10%, 20%, 20%, 20%, 30%]

How to pick the weights?

- We can think of starting with 100% weight.
  We're spreading this over the window.
- How do we spread the weight?
  - Domain knowledge.
  - Linear & exponential weights.

Linear



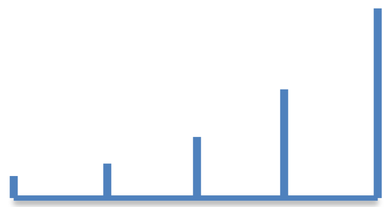
[6.5%, 13%, 19.5%, 26%, 32.5%]

Weight decays
linearly

How to pick the weights?

- We can think of starting with 100% weight. We're spreading this over the window.
- How do we spread the weight?
  - Domain knowledge.
  - Linear & exponential weights.

Exponential



[3.2%, 6.5%, 13%, 26%, 52%]

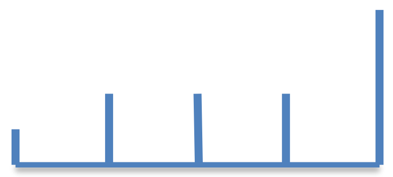
Weight decays exponentially

Parameter:
Rate of decay, α

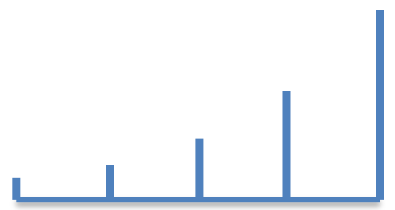
How to pick the weights?

- We can think of starting with 100% weight. We're spreading this over the window.
- How do we spread the weight?
  - Domain knowledge.
  - Linear & exponential weights.
  - Try multiple weighting schemes and test performance.
- The lack of a principled way to select weights is a downside of this approach.

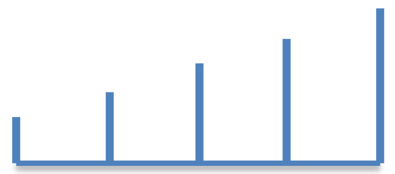
Custom



Exponential

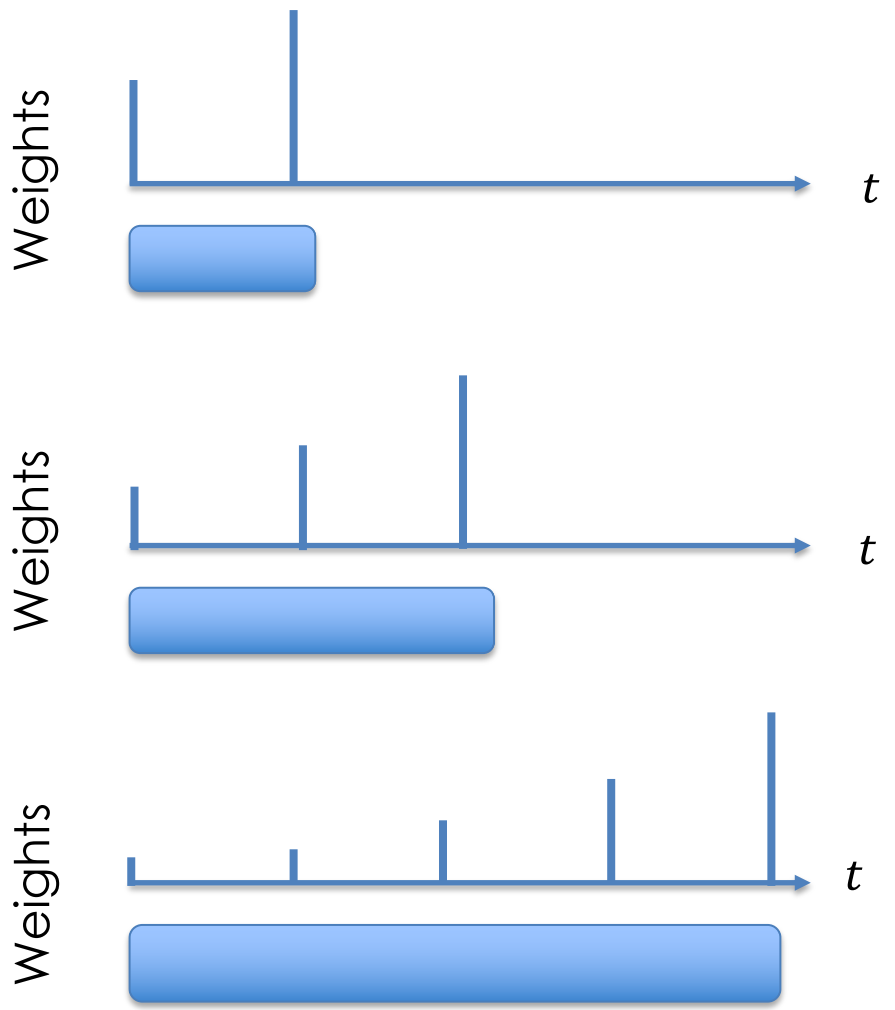


Linear

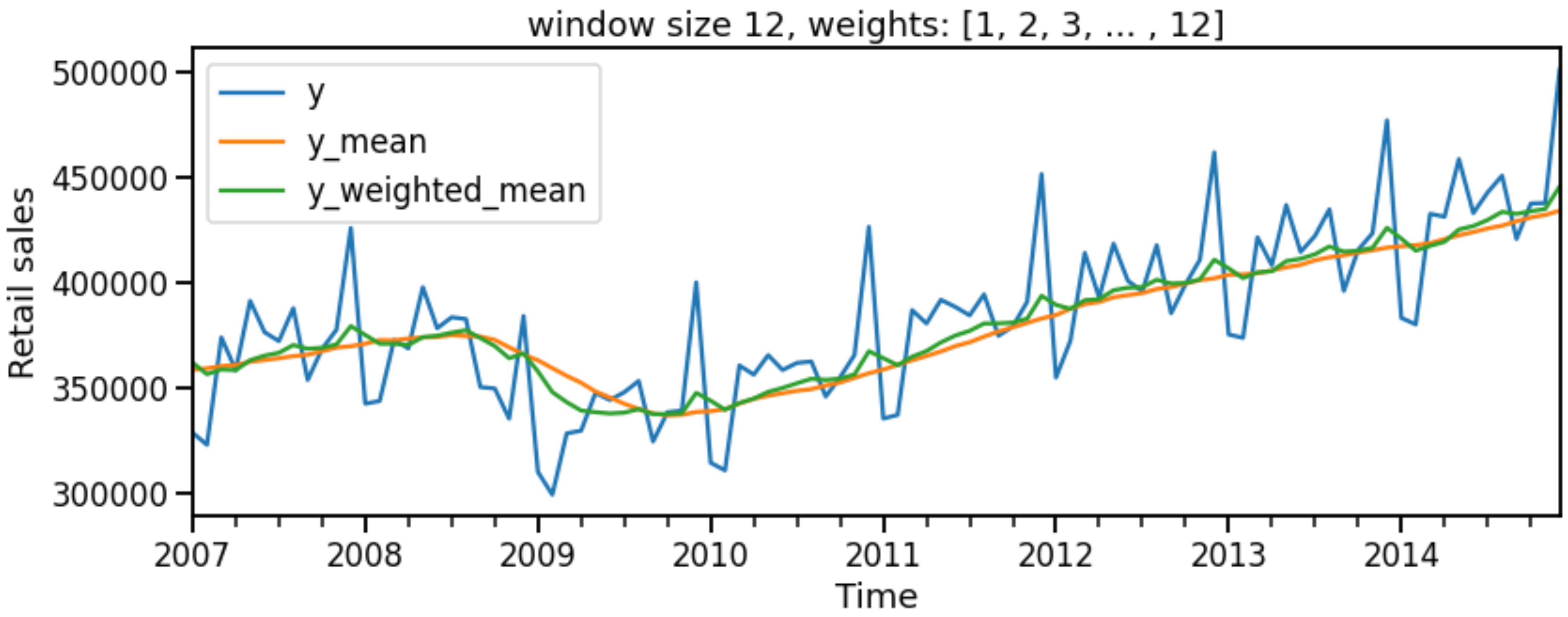


Exponential weights and expanding windows

- Exponential weights are commonly applied to expanding windows.
- Allows use of the whole history but weighs recent values more than the distant past.
- The weights change as the window expands.

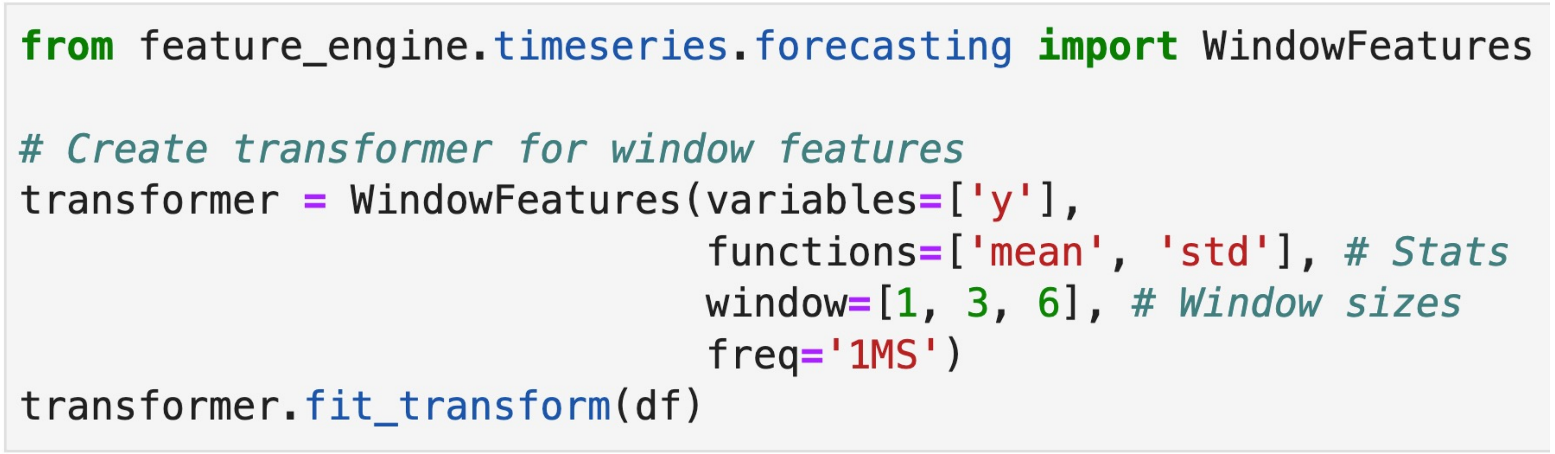


Retail sales dataset example



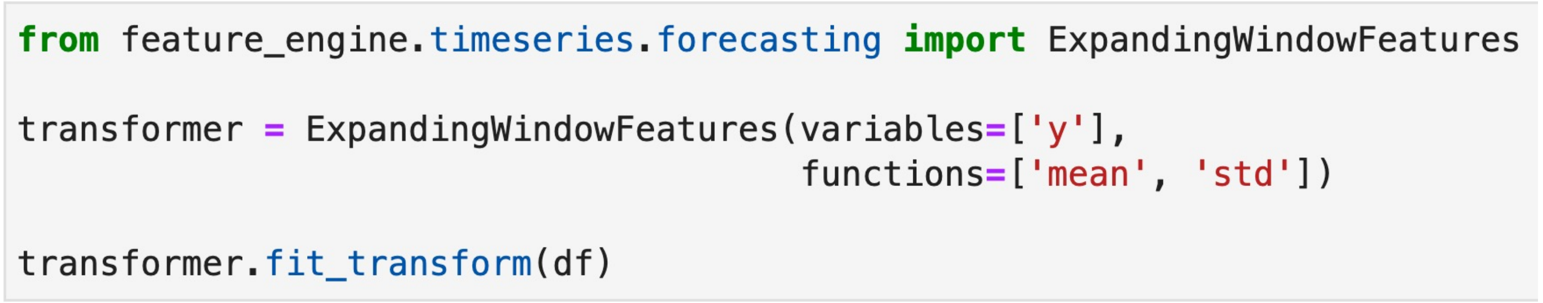
The weighted mean was more sensitive to the change in the trend in 2009.

Implementation in Feature-engine



```
from feature_engine.timeseries.forecasting import WindowFeatures

# Create transformer for window features
transformer = WindowFeatures(variables=['y'],
                             functions=['mean', 'std'], # Stats
                             window=[1, 3, 6], # Window sizes
                             freq='1MS')
transformer.fit_transform(df)
```

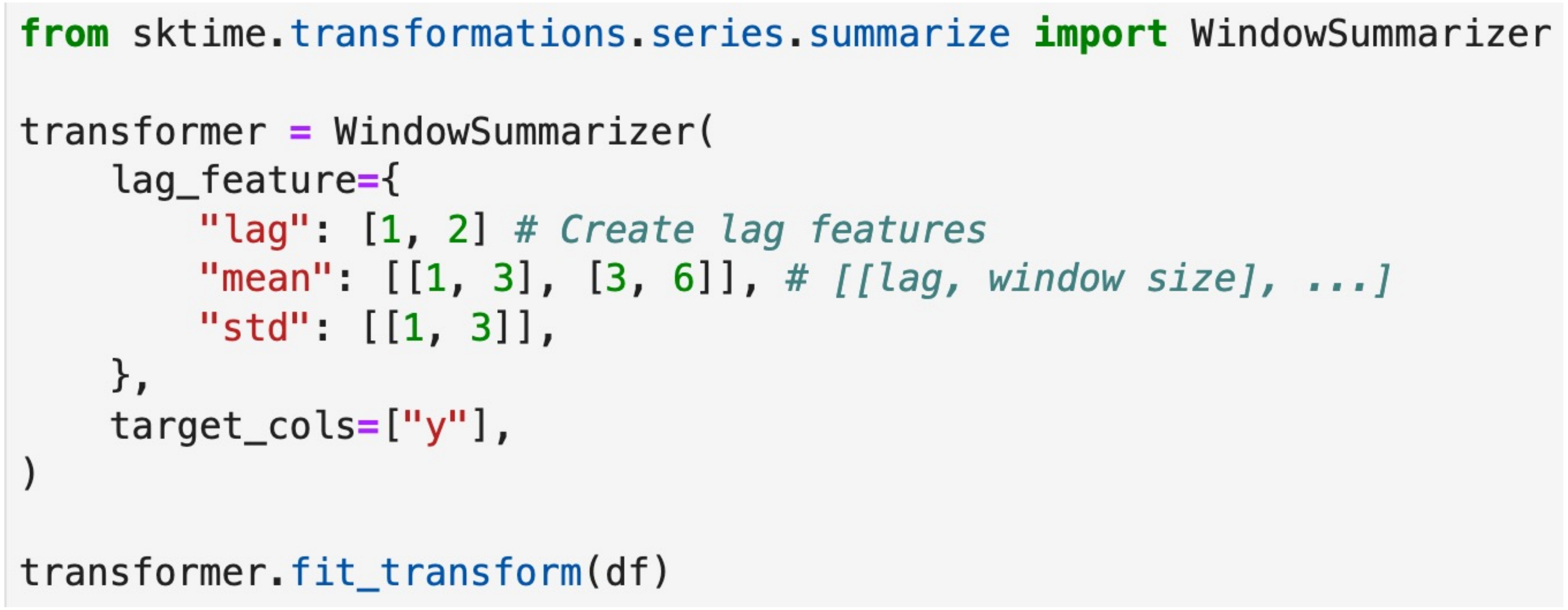


```
from feature_engine.timeseries.forecasting import ExpandingWindowFeatures

transformer = ExpandingWindowFeatures(variables=['y'],
                                      functions=['mean', 'std'])

transformer.fit_transform(df)
```

Implementation in sktime



```python
from sktime.transformations.series.summarize import WindowSummarizer

transformer = WindowSummarizer(
    lag_feature={
        "lag": [1, 2] # Create lag features
        "mean": [[1, 3], [3, 6]], # [[lag, window size], ...]
        "std": [[1, 3]],
    },
    target_cols=["y"],
)

transformer.fit_transform(df)
```

Summary

- Window features can be derived from past values of the target and other features.
- There are rolling windows and expanding windows. Rolling windows are used more often.
- Most commonly the mean and standard deviation are used as statistics.
- Weights can be added to the window to make the window feature more sensitive to recent values.