Data analysis and visualization (DAV)

Lecture 10

Łukasz P. Kozłowski

Warsaw, 2025

l.kozlowski@mimuw.edu.pl

University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Uniwersytet Warszawski

Data analysis and visualization (DAV)

Lecture 10
Statistics & machine learning
Part 2

Łukasz P. Kozłowski

Warsaw, 2025

l.kozlowski@mimuw.edu.pl

Before we go to „Statistical classification aka supervised learning” (SVM, ANN, etc.) Let’s spend some time on …

Before we go to „Statistical classification aka supervised learning” (SVM, ANN, etc.) Let’s spend some time on … **the time**

Before we go to „Statistical classification aka supervised learning” (SVM, ANN, etc.) Let’s spend some time on … **the time**

On many plots the main subject is the change of some pattern over the time. Usually, we just refer to past (analysis). Yet, if we have enough data we can try to predict the trend in the future (forecast).

this is called

Before we go to „Statistical classification aka supervised learning” (SVM, ANN, etc.) Let’s spend some time on … **the time**

On many plots the main subject is the change of some pattern over the time. Usually, we just refer to past (analysis). Yet, if we have enough data we can try to predict the trend in the future (forecast).

this is called

Time Series Forecast

Before we go to „Statistical classification aka supervised learning” (SVM, ANN, etc.) Let’s spend some time on … **the time**

On many plots the main subject is the change of some pattern over the time. Usually, we just refer to past (analysis). Yet, if we have enough data we can try to predict the trend in the future (forecast).

this is called

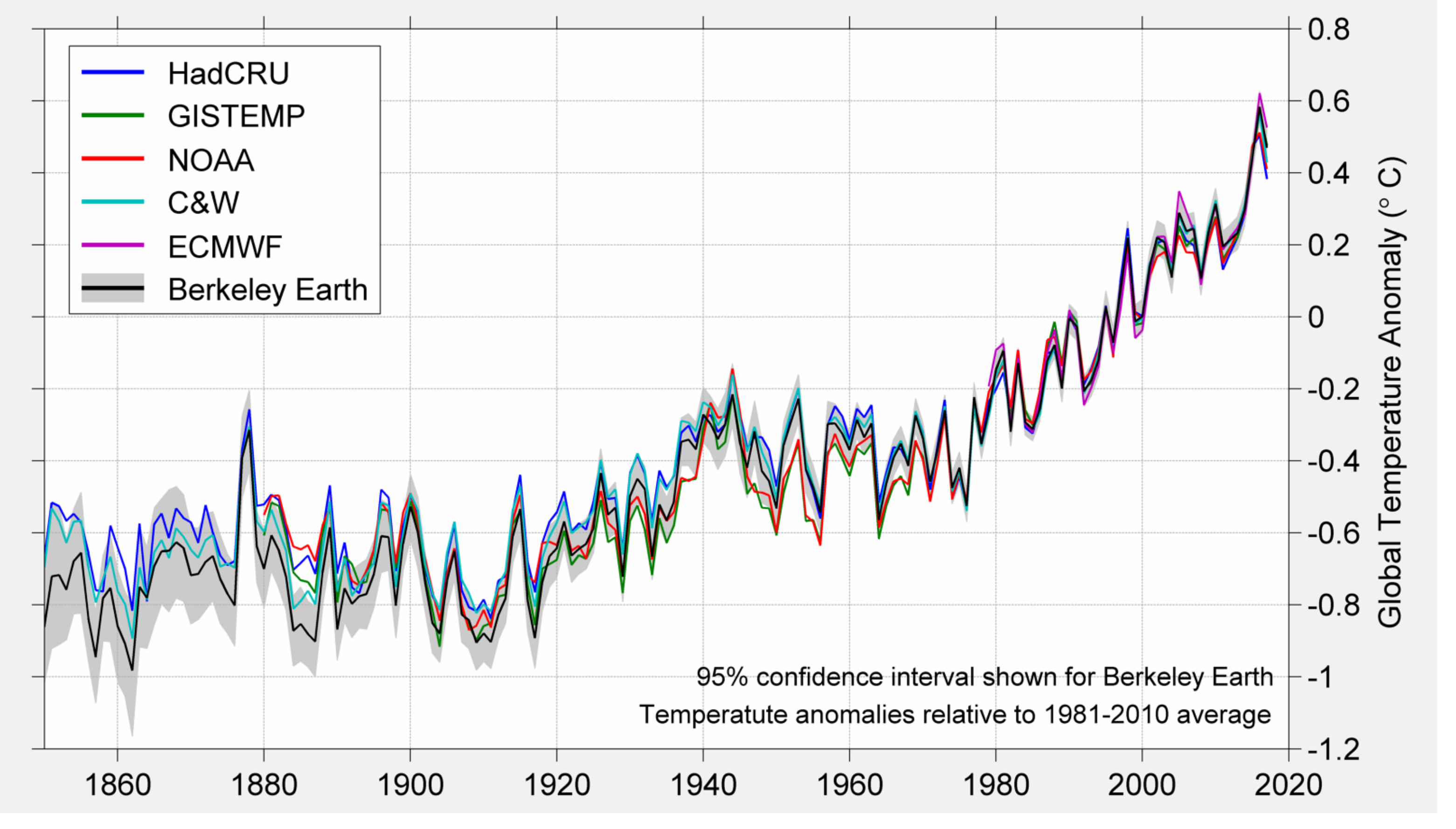
Time Series Forecast

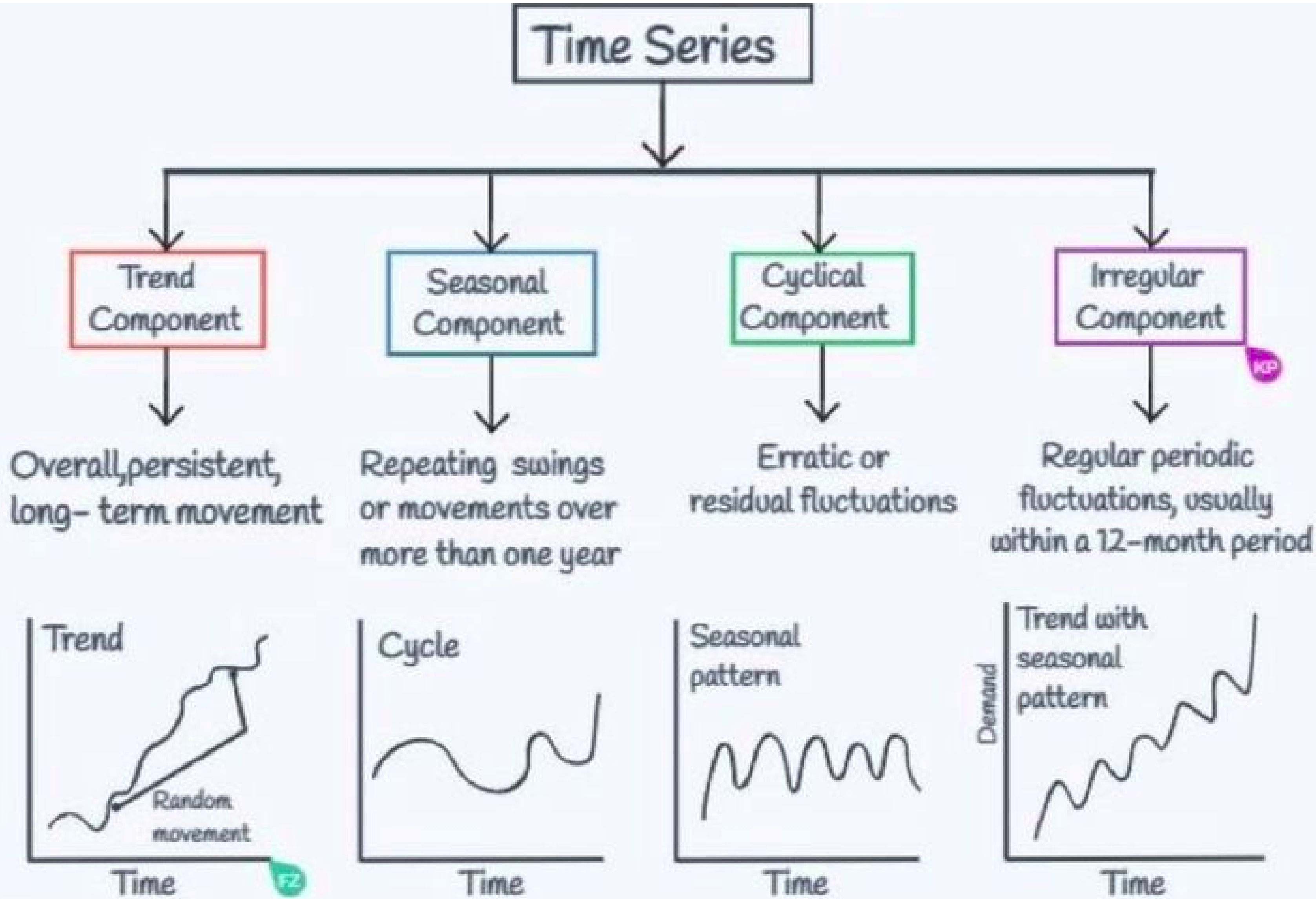
For start see: https://en.wikipedia.org/wiki/Time_series

Definition:

Time series is a series of data points indexed (or listed or graphed) in time order

There are many approaches for analyze and forecast time series. We will mention just few.





Applications:

- Process and Quality Control
- Inventory Studies
- Workload Projections
- Utility Studies
- Census Analysis

- Economic Forecasting
- Sales Forecasting
- Budgetary Analysis
- Stock Market Analysis
- Yield Projections

Autoregression (AR)

In the autoregressive model we assume that the output variable depends linearly on its own previous values and on a stochastic term (an imperfectly predictable term)

Definition

The notation $AR(p)$ indicates an autoregressive model of order *p*.

The AR(*p*) model is defined as:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

where $\varphi_1, \ldots, \varphi_p$ are the *parameters* of the model, c is a constant, and ε_t is white noise.

https://en.wikipedia.org/wiki/Autoregressive_model

Autoregression (AR)

In short in AR(p) model p (called also lag) decides how many previous time steps are taken into account

simple linear regression vs autoregression model

```
y = a + b*X
```

```
y = a + b1*X(t-1) + b2*X(t-2) + b3*X(t-3)
```

https://en.wikipedia.org/wiki/Autoregressive_model

Autoregression (AR)

For more mathematical details see:

https://en.wikipedia.org/wiki/Autoregressive_model

Python Code

```
# AR example
from statsmodels.tsa.ar_model import AutoReg
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = AutoReg(data, lags=1)
model_fit = model.fit()
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Moving Average (MA)

The moving-average model specifies that the output variable depends linearly on the current and various past values of a stochastic (imperfectly predictable) term.

Definition

The notation MA(q) refers to the moving average model of order q:

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}$$

where μ is the mean of the series, the θ_1, ..., θ_q are the parameters of the model and

the ε_t, ε_{t-1},..., ε_{t-q} are white noise error terms. The value of q is called the order of the MA model.

Moving Average (MA)

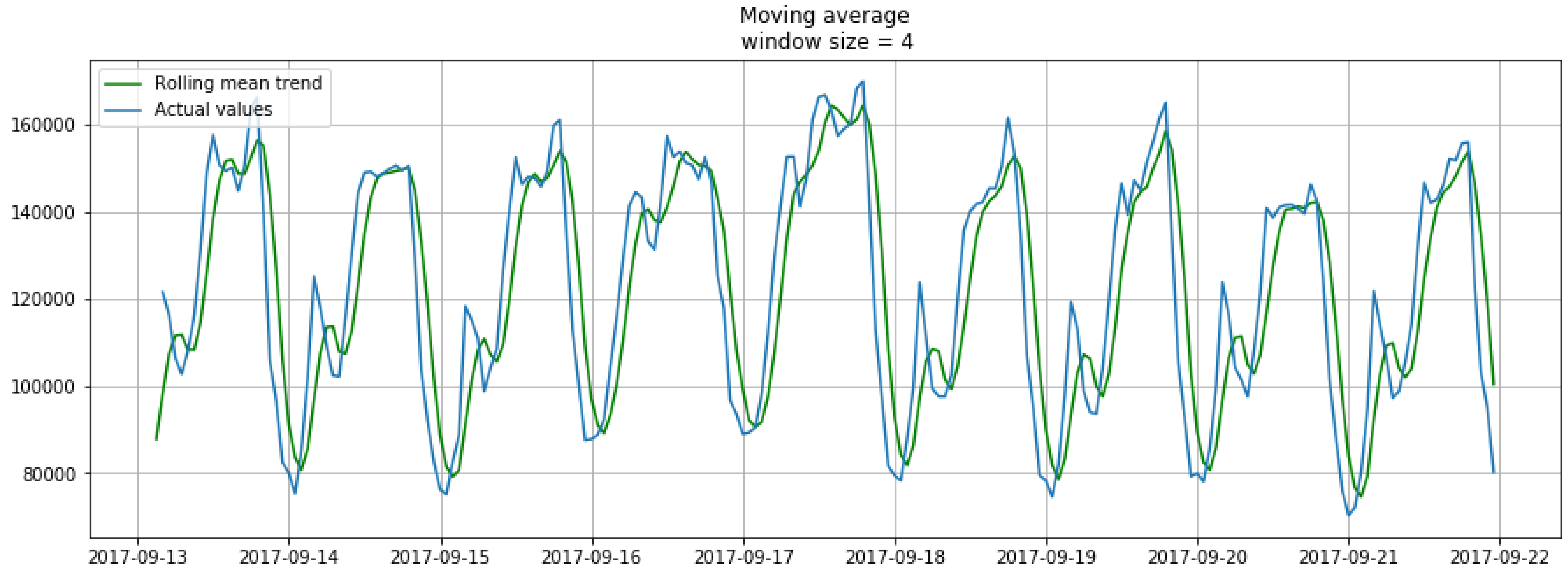
For more mathematical details see:

https://en.wikipedia.org/wiki/Moving-average_model

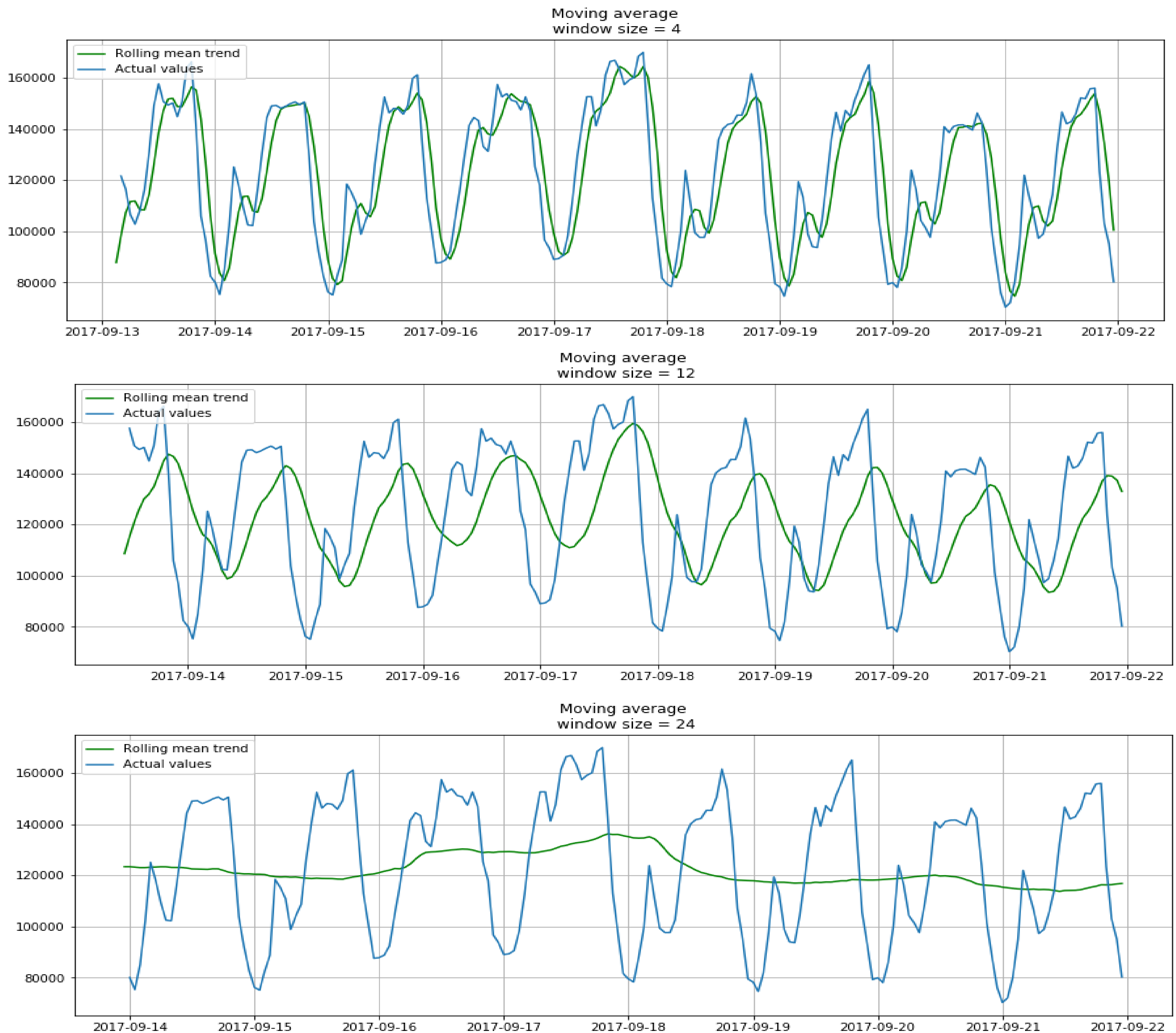
PYTHON

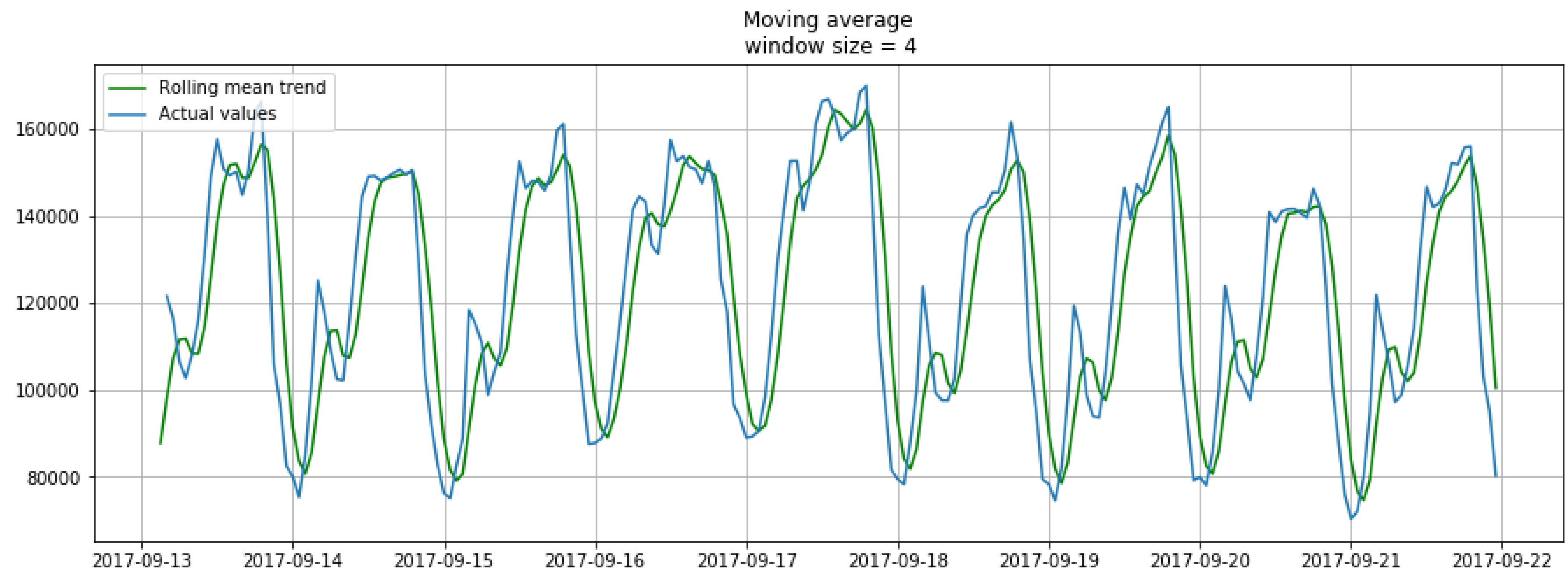
NUMPY np.average(series[-n:])

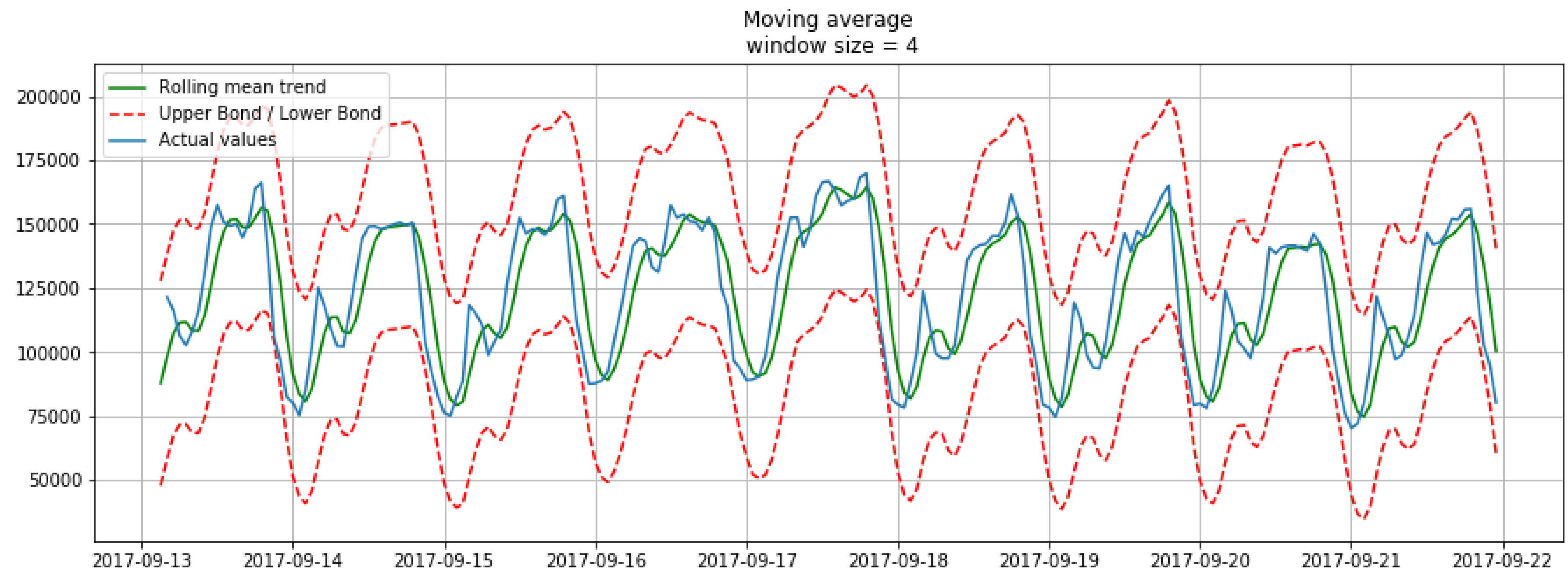
PANDAS DataFrame.rolling(window).mean().



https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/
https://www.kaggle.com/code/kashnitsky/topic-9-part-1-time-series-analysis-in-python/notebook


Moving average
window size = 4
Rolling mean trend
Actual values
160000
140000
120000
100000
80000
2017-09-13
2017-09-14
2017-09-15
2017-09-16
2017-09-17
2017-09-18
2017-09-19
2017-09-20
2017-09-21
2017-09-22
Moving average
window size = 12
Rolling mean trend
Actual values
160000
140000
120000
100000
80000
2017-09-14
2017-09-15
2017-09-16
2017-09-17
2017-09-18
2017-09-19
2017-09-20
2017-09-21
2017-09-22
Moving average
window size = 24
Rolling mean trend
Actual values
160000
140000
120000
100000
80000
2017-09-14
2017-09-15
2017-09-16
2017-09-17
2017-09-18
2017-09-19
2017-09-20
2017-09-21
2017-09-22





https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/
https://www.kaggle.com/code/kashnitsky/topic-9-part-1-time-series-analysis-in-python/notebook

Moving Average (MA)

For more mathematical details see:

https://en.wikipedia.org/wiki/Moving-average_model

Python Code

We can use the ARMA class to create an MA model and setting a zeroth-order AR model. We must specify the order of the MA model in the order argument.

```
# MA example
from statsmodels.tsa.arima_model import ARMA
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = ARMA(data, order=(0, 1))
model_fit = model.fit(disp=False)
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Weighted average

Weighted average is a simple modification to the moving average. The weights sum up to `1` with larger weights assigned to more recent observations.

$$\hat{y}_t = \sum_{n=1}^{k} \omega_n y_{t+1-n}$$

:

```
def exponential_smoothing(series, alpha):
    """
        series - dataset with timestamps
        alpha - float [0.0, 1.0], smoothing parameter
    """
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result
[n-1])
    return result
```

Autoregressive–moving-average model (ARMA)

Autoregressive–moving-average (ARMA) models provide a parsimonious description of a (weakly) stationary stochastic process in terms of two polynomials, one for the autoregression (AR) and the second for the moving average (MA)

ARMA = AR + MA

ARMA model

The notation ARMA(p, q) refers to the model with p autoregressive terms and q moving-average terms. This model contains the AR(p) and MA(q) models,

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}.$$

Autoregressive–moving-average model

For more mathematical details see:

https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model

Python Code

```
# ARMA example
from statsmodels.tsa.arima_model import ARMA
from random import random
# contrived dataset
data = [random() for x in range(1, 100)]
# fit model
model = ARMA(data, order=(2, 1))
model_fit = model.fit(disp=False)
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Autoregressive Integrated Moving Average (ARIMA)

ARIMA models are generally denoted **ARIMA(p,d,q)** where parameters p, d, and q are non-negative integers, p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the moving-average model

The parameters of the ARIMA model are defined as follows:

p: The number of lag observations included in the model, also called the lag order.
d: The number of times that the raw observations are differenced, also called the degree of differencing.
q: The size of the moving average window, also called the order of moving average

ARIMA = AR + MA + I

I: Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.

Autoregressive Integrated Moving Average (ARIMA)

For more mathematical details see:

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

Python Code

```
# ARIMA example
from statsmodels.tsa.arima_model import ARIMA
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = ARIMA(data, order=(1, 1, 1))
model_fit = model.fit(disp=False)
# make prediction
yhat = model_fit.predict(len(data), len(data), typ='levels')
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Seasonal Autoregressive Integrated Moving-Average (SARIMA)

Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component

SARIMA(p,d,q)(P,D,Q)m

p: Trend autoregression order
d: Trend difference order
q: Trend moving average order

P: Seasonal autoregressive order
D: Seasonal difference order
Q: Seasonal moving average order
m: The number of time steps for a single seasonal period

Seasonal Autoregressive Integrated Moving-Average (SARIMA)

For more mathematical details see:

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

Python Code

```
# SARIMA example
from statsmodels.tsa.statespace.sarimax import SARIMAX
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = SARIMAX(data, order=(1, 1, 1), seasonal_order=(1, 1, 1, 1))
model_fit = model.fit(disp=False)
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)

SARIMAX is an extension of the SARIMA model that also includes the modeling of *exogenous variables*

Exogenous variables(covariates) can be thought of as parallel input sequences that have observations at the same time steps as the original series

The method is suitable for univariate time series with trend and/or seasonal components and exogenous variables

Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)

For more mathematical details see:

https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

Python Code

```python
# SARIMAX example
from statsmodels.tsa.statespace.sarimax import SARIMAX
from random import random
# contrived dataset
data1 = [x + random() for x in range(1, 100)]
data2 = [x + random() for x in range(101, 200)]
# fit model
model = SARIMAX(data1, exog=data2, order=(1, 1, 1), seasonal_order=(0, 0, 0, 0))
model_fit = model.fit(disp=False)
# make prediction
exog2 = [200 + random()]
yhat = model_fit.predict(len(data1), len(data1), exog=[exog2])
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Vector Autoregression (VAR)

VAR is the generalization of AR to multiple parallel time series, e.g. multivariate time series

The notation for the model involves specifying the order for the AR(p) model as parameters to a VAR function, e.g. VAR(p)

The method is suitable for multivariate time series without trend and seasonal components

Vector Autoregression (VAR)

For more mathematical details see:

https://en.wikipedia.org/wiki/Vector_autoregression

Python Code

```
# VAR example
from statsmodels.tsa.vector_ar.var_model import VAR
from random import random
# contrived dataset with dependency
data = list()
for i in range(100):
    v1 = i + random()
    v2 = v1 + random()
    row = [v1, v2]
    data.append(row)
# fit model
model = VAR(data)
model_fit = model.fit()
# make prediction
yhat = model_fit.forecast(model_fit.y, steps=1)
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Vector Autoregression Moving-Average (VARMA)

VARMA method models the next step in each time series using an ARMA model

VARMA is the generalization of ARMA to multiple parallel time series, e.g. multivariate time series

VARMA(p, q) = AR(p) + MA(q)

Suitable for multivariate time series without trend and seasonal components

Vector Autoregression Moving-Average (VARMA)

For more mathematical details see:

https://en.wikipedia.org/wiki/Vector_autoregression

Python Code

```python
# VARMA example
from statsmodels.tsa.statespace.varmax import VARMAX
from random import random
# contrived dataset with dependency
data = list()
for i in range(100):
    v1 = random()
    v2 = v1 + random()
    row = [v1, v2]
    data.append(row)
# fit model
model = VARMAX(data, order=(1, 1))
model_fit = model.fit(disp=False)
# make prediction
yhat = model_fit.forecast()
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)

VARMAX is an extension of the VARMA model that also includes the modeling of exogenous variables

VARMAX is a multivariate version of the ARMAX method

Suitable for multivariate time series without trend and seasonal components with exogenous variables

Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)

For more mathematical details see:

https://en.wikipedia.org/wiki/Vector_autoregression

Python Code

```
# VARMAX example
from statsmodels.tsa.statespace.varmax import VARMAX
from random import random
# contrived dataset with dependency
data = list()
for i in range(100):
    v1 = random()
    v2 = v1 + random()
    row = [v1, v2]
    data.append(row)
data_exog = [x + random() for x in range(100)]
# fit model
model = VARMAX(data, exog=data_exog, order=(1, 1))
model_fit = model.fit(disp=False)
# make prediction
data_exog2 = [[100]]
yhat = model_fit.forecast(exog=data_exog2)
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Simple Exponential Smoothing (SES)

Next time step in SES is modeled as an exponentially weighted linear function of observations at prior time steps

SES is suitable for univariate time series without trend and seasonal components

Simple Exponential Smoothing (SES)

For more mathematical details see:

https://en.wikipedia.org/wiki/Exponential_smoothing

Python Code

```
# SES example
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = SimpleExpSmoothing(data)
model_fit = model.fit()
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Holt Winter’s Exponential Smoothing (HWES)

HWES is also called the **Triple** Exponential Smoothing method

HWES models the next time step as an exponentially **weighted linear function** of observations at prior time steps, taking **trends** and **seasonality** into account

Suitable for univariate time series with trend and/or seasonal components.

Holt Winter’s Exponential Smoothing (HWES)

For more mathematical details see:

https://en.wikipedia.org/wiki/Exponential_smoothing

Python Code

```python
# HWES example
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from random import random
# contrived dataset
data = [x + random() for x in range(1, 100)]
# fit model
model = ExponentialSmoothing(data)
model_fit = model.fit()
# make prediction
yhat = model_fit.predict(len(data), len(data))
print(yhat)
```

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Forecast quality metrics

After forecasting, we need to measure the quality of our predictions. To do so you need calculate some quality metrics. The most popular are:

- R squared (R^2)
- Mean Absolute Error (MAE)
- Median Absolute Error (MedAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Squared Logarithmic Error (MSLE)
- Mean Absolute Percentage Error (MAPE)

R squared (R^2) called also coefficient of determination

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value for total n samples, the estimated R^2 is defined as:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$.

Possible values: $(-\infty, 1]$

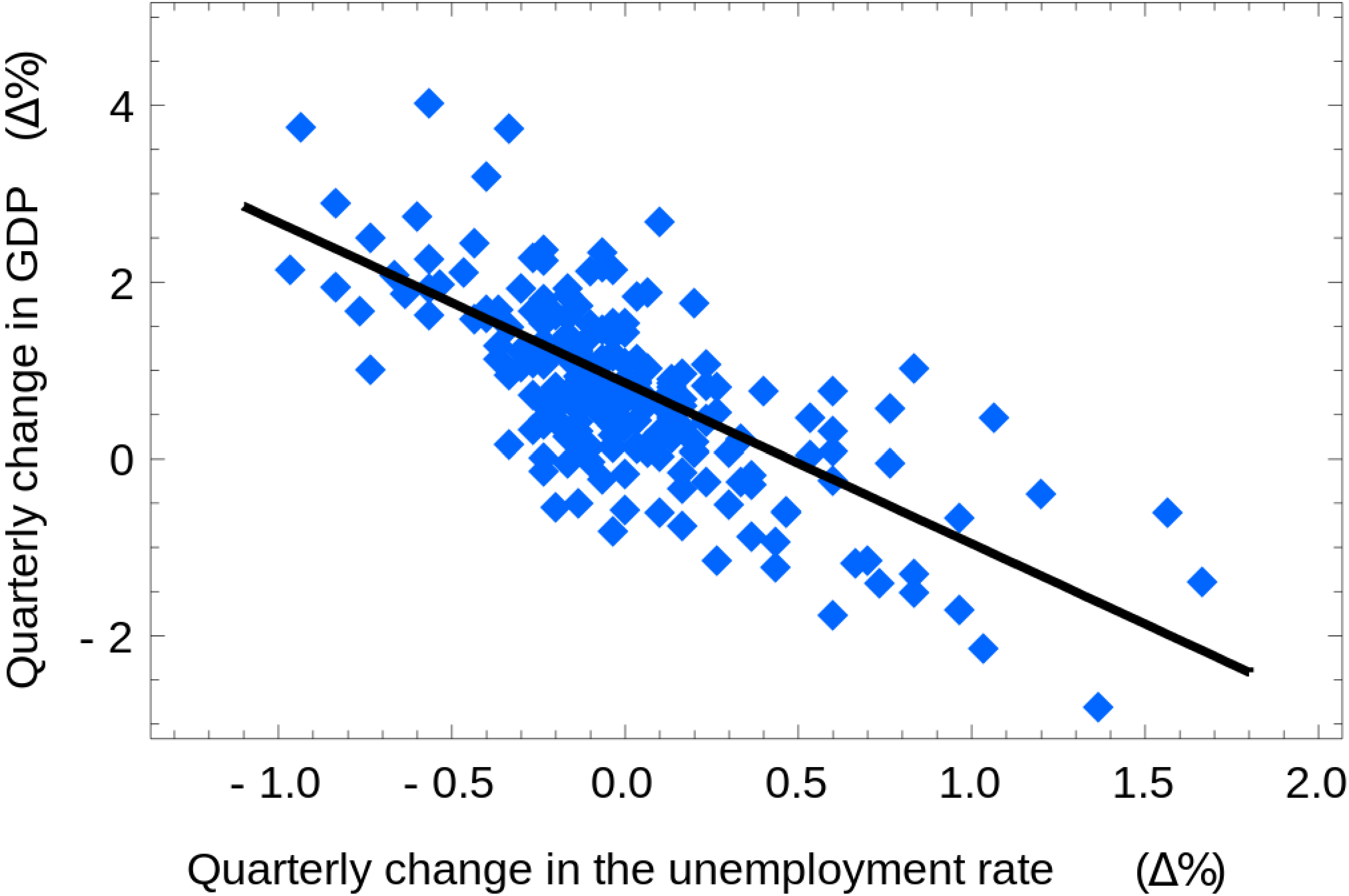
Interpretation: 0 – baseline, 1 – prefect, <0 very poor

It can be interpreted as the percentage of variance explained by the model

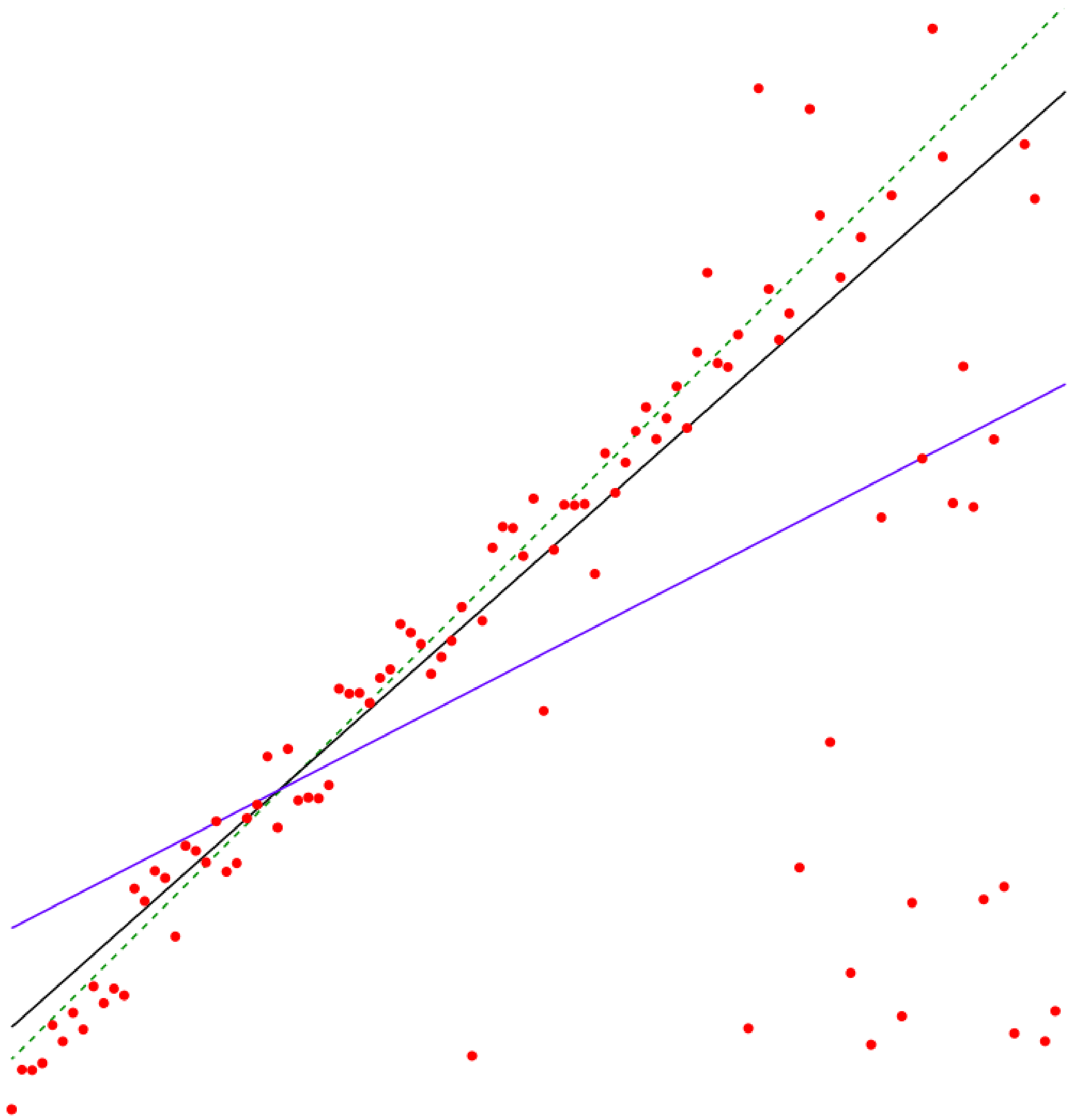
PYTHON: sklearn.metrics.r2_score

https://en.wikipedia.org/wiki/Coefficient_of_determination

R squared (R^2) called also coefficient of determination

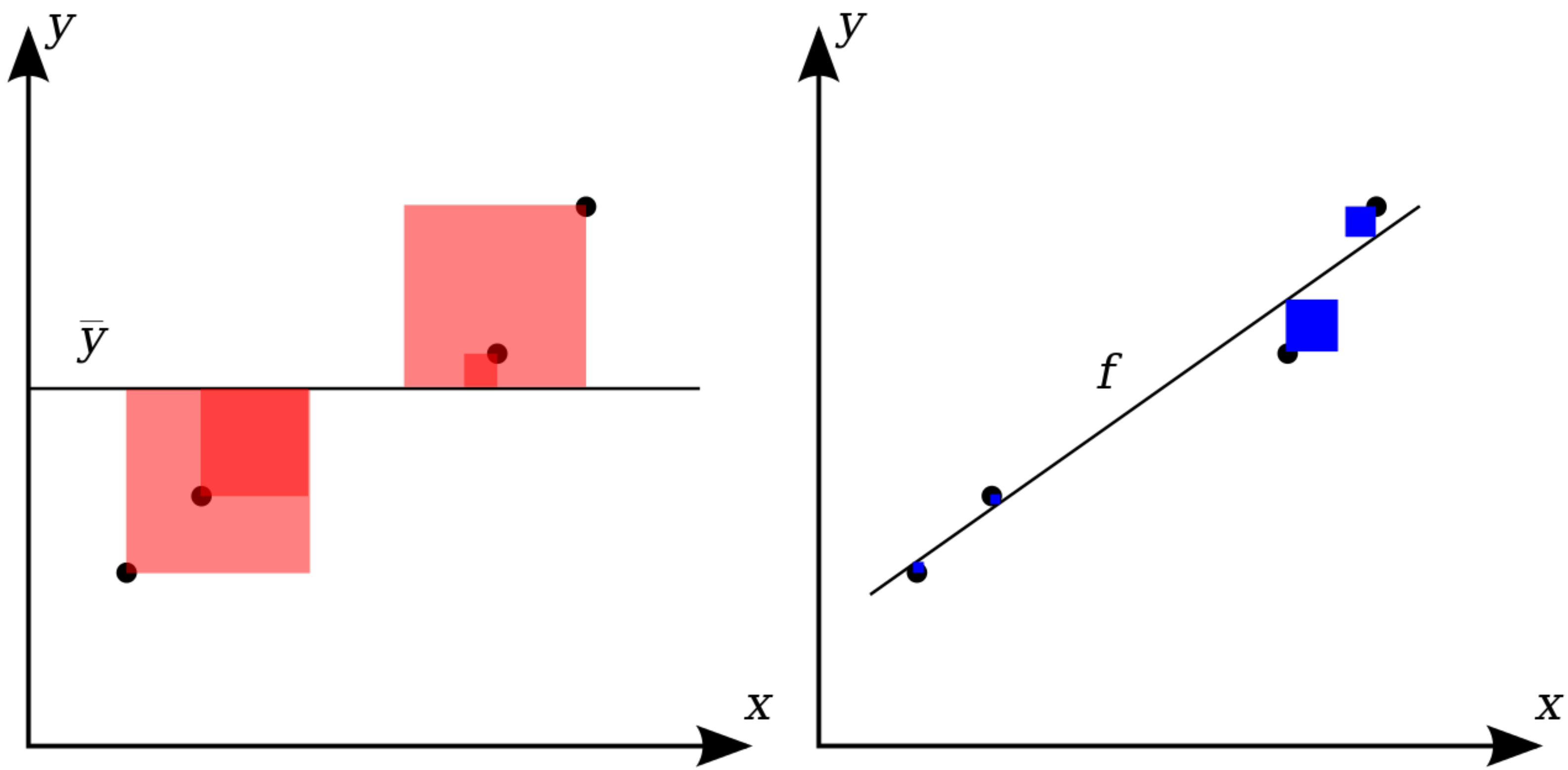


R squared (R^2) called also coefficient of determination



https://en.wikipedia.org/wiki/Coefficient_of_determination

R squared (R^2) called also coefficient of determination



$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

The better the linear regression (on the right) fits the data in comparison to the simple average (on the left graph), the closer the value of R2 is to 1. The areas of the blue squares represent the squared residuals with respect to the linear regression. The areas of the red squares represent the squared residuals with respect to the average value.

https://en.wikipedia.org/wiki/Coefficient_of_determination

Mean Absolute Error

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over n_{samples} is defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \left| y_i - \hat{y}_i \right|.$$

This is an interpretable metric because it has the same unit of measurment as the initial series

Possible values: $[0,+\infty)$

Interpretation: 0 – perfect, the bigger, the worse

PYTHON: sklearn.metrics.mean_absolute_error

https://en.wikipedia.org/wiki/Mean_absolute_error

Median absolute error (MedAE)

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the median absolute error (MedAE) estimated over n_{samples} is defined as

$$\text{MedAE}(y, \hat{y}) = \text{median}(\mid y_1 - \hat{y}_1 \mid, \ldots, \mid y_n - \hat{y}_n \mid).$$

An interpretable metric that is **robust to outliers**

Possible values: [0,+∞)

Interpretation: 0 – perfect, the bigger, the worse

PYTHON: sklearn.metrics.median_absolute_error

https://en.wikipedia.org/wiki/Median_absolute_deviation

Mean Squared Error (MSE)

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over n_{samples} is defined as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

The most commonly used metric that gives a higher penalty to large errors and vice versa

Possible values: [0,+∞)

Interpretation: 0 – perfect, the bigger, the worse

PYTHON: sklearn.metrics.mean_squared_error

https://en.wikipedia.org/wiki/Mean_squared_error

Root Mean Squared Error (RMSE)

The RMSD of predicted values $\hat{y}_t$ for times t of a regression's dependent variable y_t, with variables observed over T times, is computed for T different predictions as the square root of the mean of the squares of the deviations:

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{T} (\hat{y}_t - y_t)^2}{T}}.$$

Pros: has the same units as the quantity being estimated

Cons: The effect of each error on RMSD is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSD. Consequently, RMSD is sensitive to outliers

Possible values: [0,+∞)

Interpretation: 0 – perfect, the bigger, the worse

https://en.wikipedia.org/wiki/Root-mean-square_deviation

Mean Squared Logarithmic Error (MSLE)

If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared logarithmic error (MSLE) estimated over n_{samples} is defined as

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e(1 + y_i) - \log_e(1 + \hat{y}_i))^2.$$

Almost the same as MSE, but we take the logarithm of the series. Thus, we give more weight to small mistakes as well

Usually, used when the data has exponential trends

Possible values: [0,+∞)

Interpretation: 0 – perfect, the bigger, the worse

PYTHON: sklearn.metrics.mean_squared_log_error

Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{y_i}$$

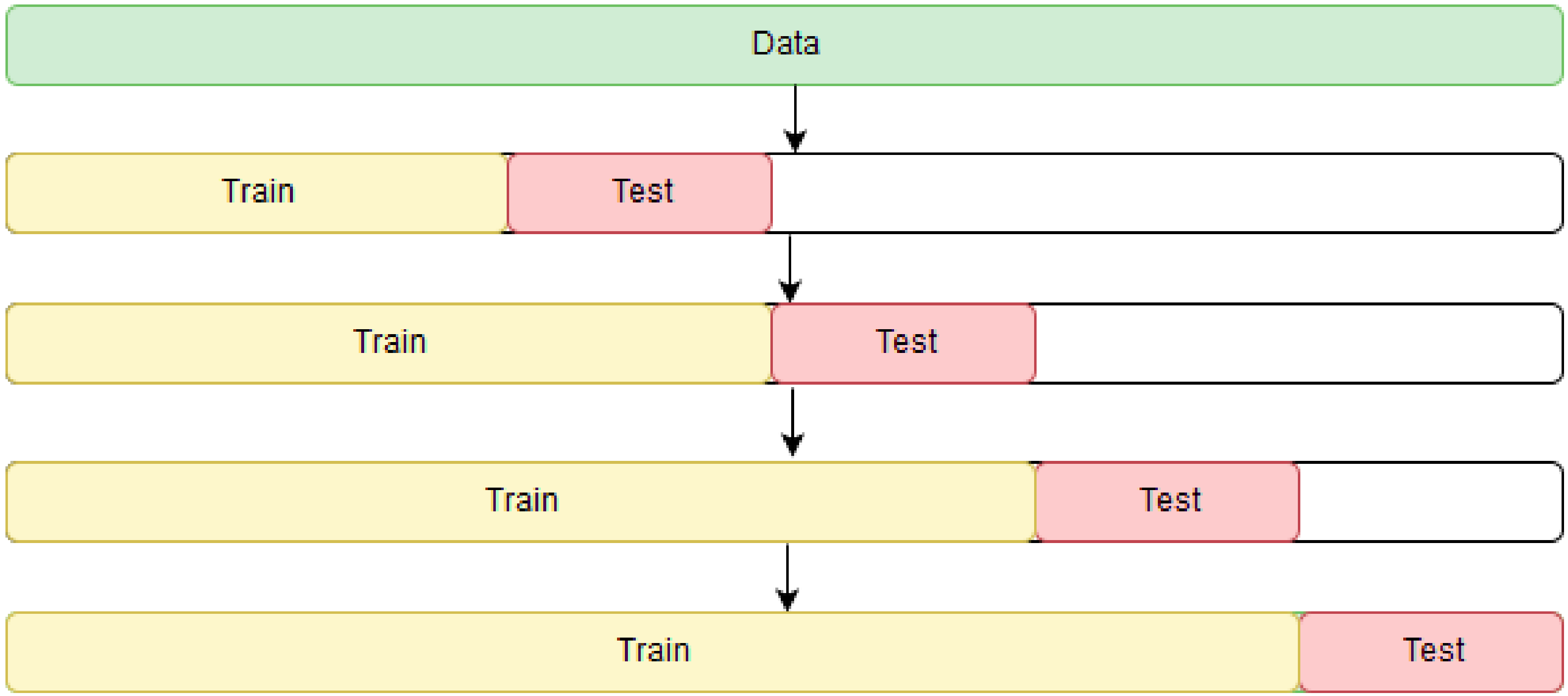
The same as MAE, but is computed as a percentage (very convenient when you want to explain the quality of the model)

Possible values: [0,+∞)

Interpretation: 0 – perfect, the bigger, the worse

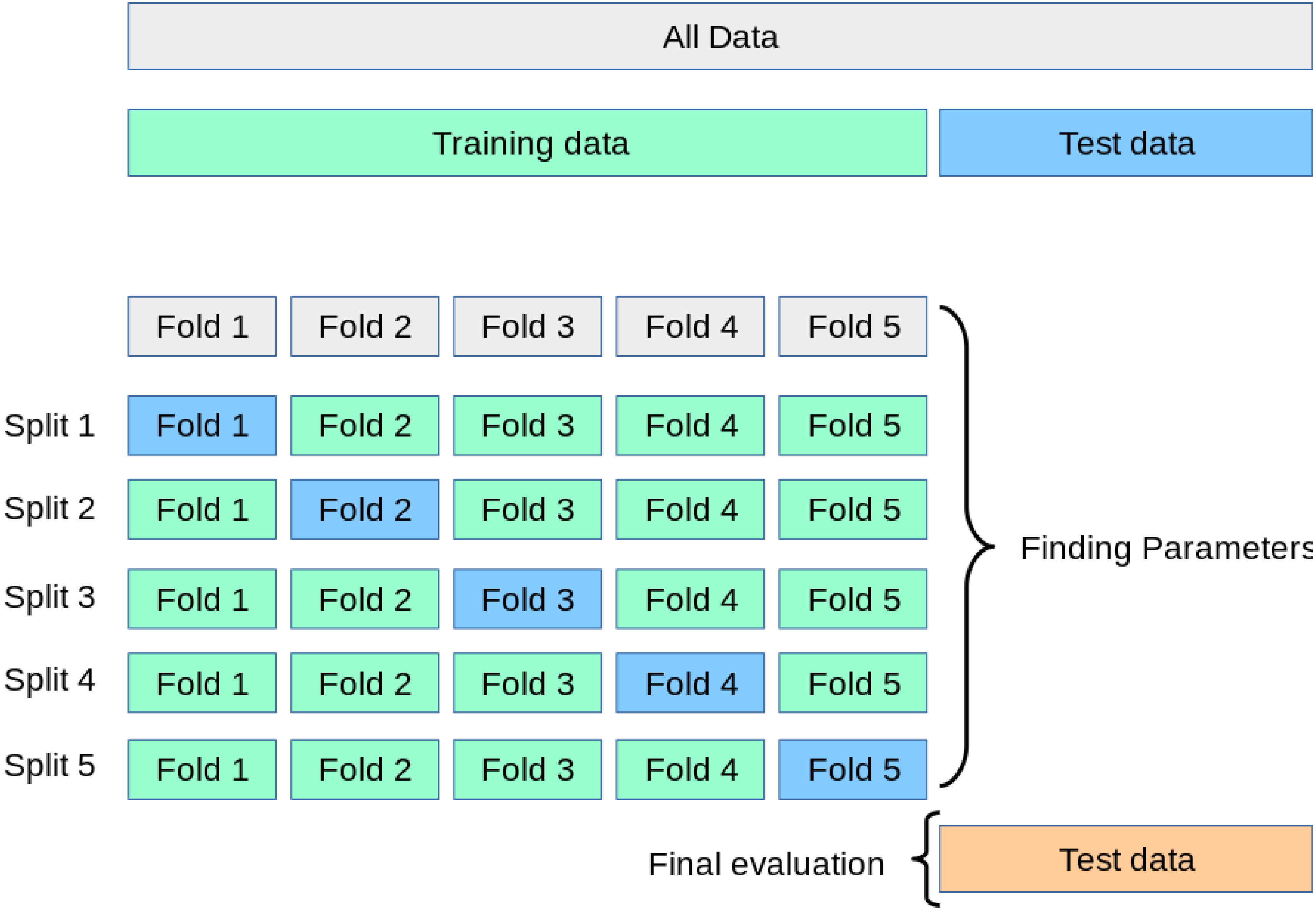
```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

Time series cross validation




```
from sklearn.model_selection import TimeSeriesSplit
```

(standard) cross validation




```
from sklearn.model_selection import cross_validate
```

Time series cross validation

```
import numpy as np
from sklearn.model_selection import TimeSeriesSplit
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4, 5, 6])
tscv = TimeSeriesSplit()
print(tscv)

for train_index, test_index in tscv.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

For mote details see:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

Useful resources:

https://en.wikipedia.org/wiki/Time_series

https://campus.datacamp.com/courses/time-series-analysis-in-python/

https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

https://towardsdatascience.com/time-series-analysis-in-python-an-introduction-70d5a5b1d52a

https://stackoverflow.com/questions/49712037/trend-predictor-in-python

https://machinelearningmastery.com/make-predictions-time-series-forecasting-python/

https://machinelearningmastery.com/multi-step-time-series-forecasting/

https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

https://www.kaggle.com/kashnitsky/topic-9-part-1-time-series-analysis-in-python/notebook

https://www.datacamp.com/courses/introduction-to-time-series-analysis-in-python

Thank you for your time
and
See you at the next lecture

Any other
questions & comments

l.kozlowski@mimuw.edu.pl

www: bioinformatic.netmark.pl