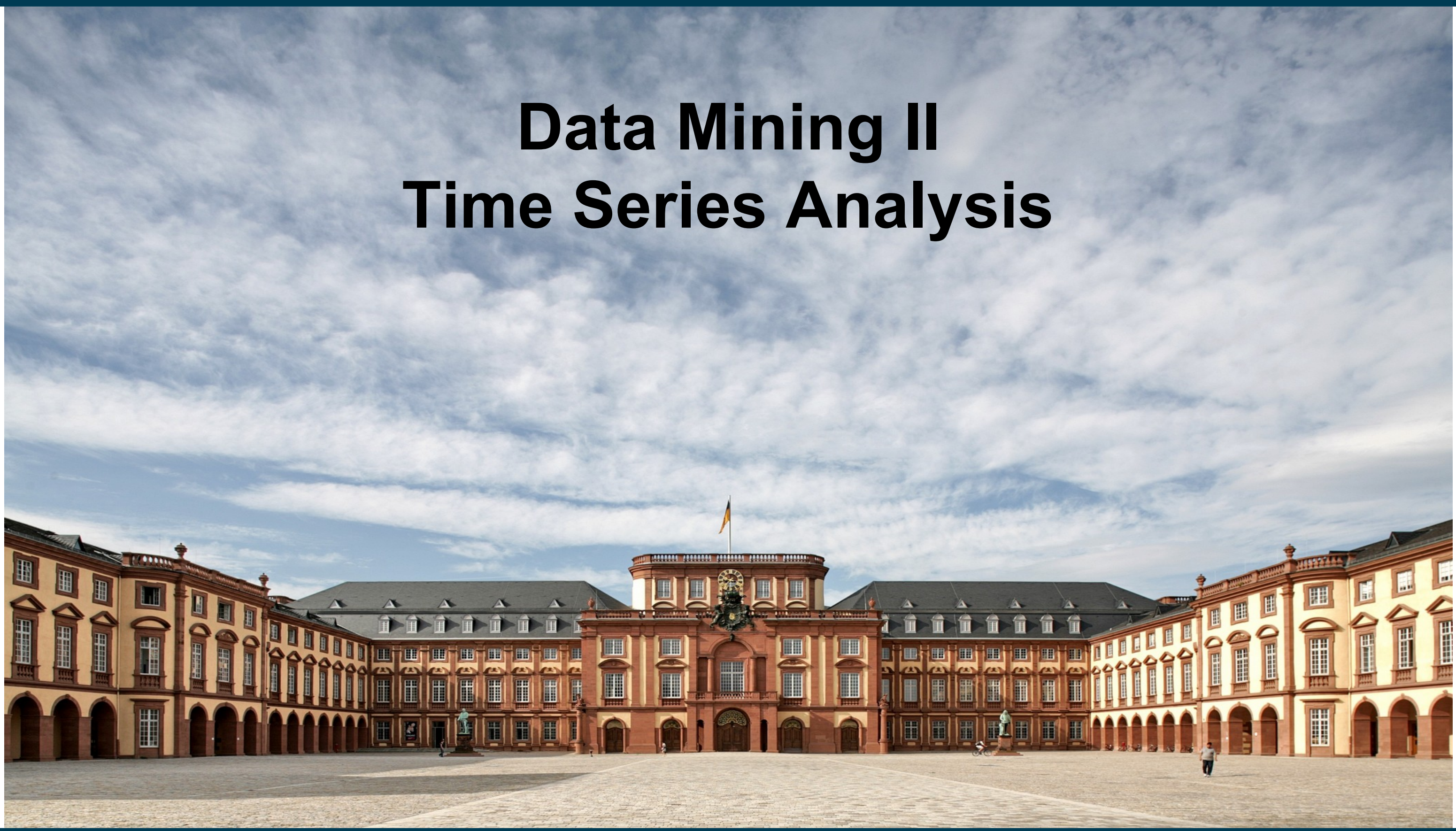
UNIVERSITÄT MANNHEIM

Data Mining II
Time Series Analysis



Heiko Paulheim

Introduction

- So far, we have only looked at data without a time dimension
  - or simply ignored the temporal aspect

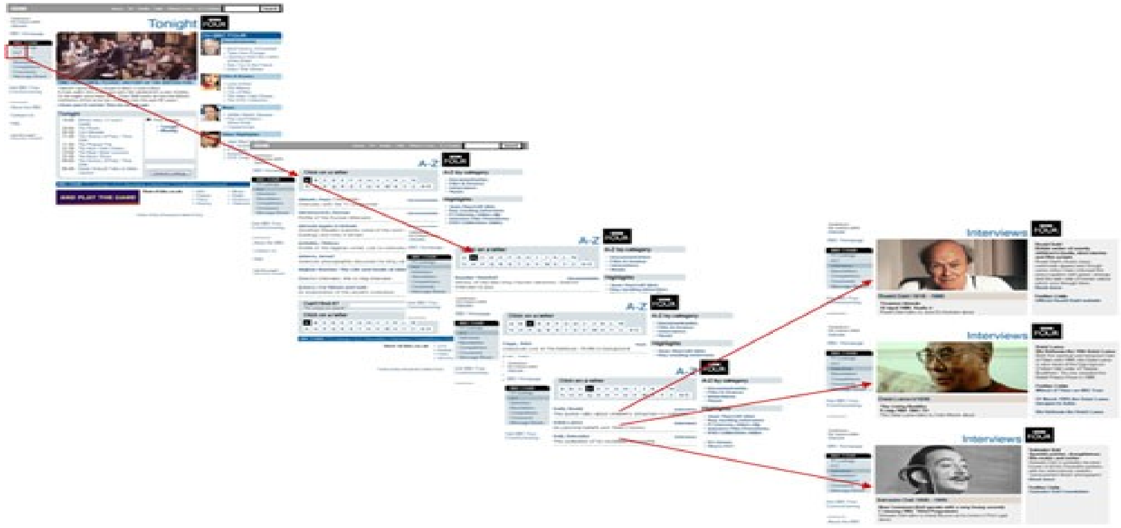
- Many “classic” DM problems have variants that respect time
  - frequent pattern mining → sequential pattern mining
  - classification → predicting sequences of nominals
  - regression → predicting the continuation of a numeric series

Contents

- Sequential Pattern Mining
  - Finding frequent subsequences in set of sequences
  - the GSP algorithm
- Trend analysis
  - Is a time series moving up or down?
  - Simple models and smoothing
  - Identifying seasonal effects
- Forecasting
  - Predicting future developments from the past
  - Autoregressive models and windowing
  - Exponential smoothing and its extensions

Sequential Pattern Mining: Application 1

- Web usage mining (navigation analysis)
- Input
  - Server logs
- Patterns
  - typical sequences of pages
- Usage
  - restructuring web sites



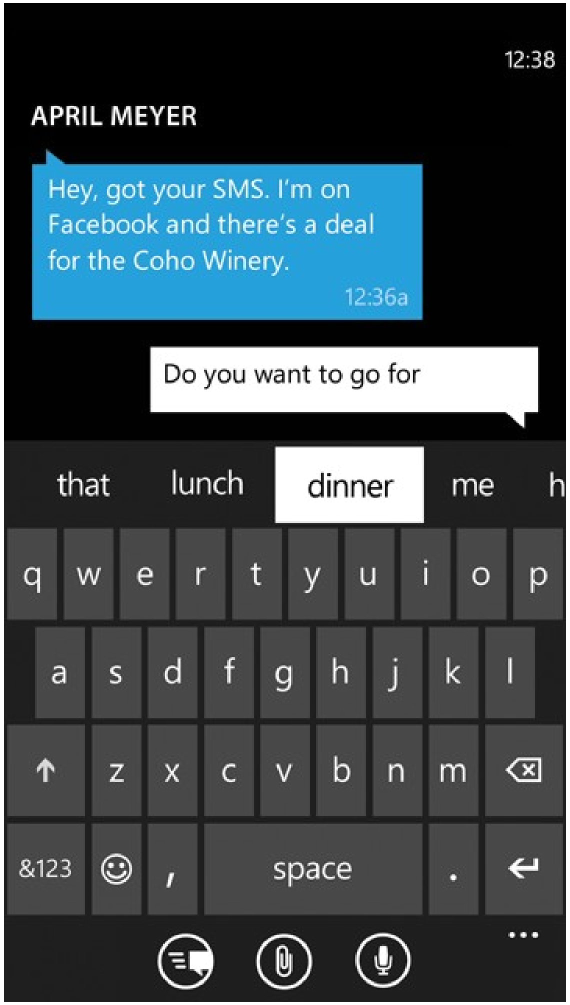
Sequential Pattern Mining: Application 2

- Recurring customers
  - Typical book store example:
    - (Twilight) (New Moon) → (Eclipse)

- Recommendation in online stores
- Allows more fine grained suggestions than frequent pattern mining
- Example:
  - *mobile phone* → *charger* vs. *charger* → *mobile phone*
    - are indistinguishable by frequent pattern mining
  - customers will select a charger after a mobile phone
    - but not the other way around!
    - however, Amazon does not respect sequences...

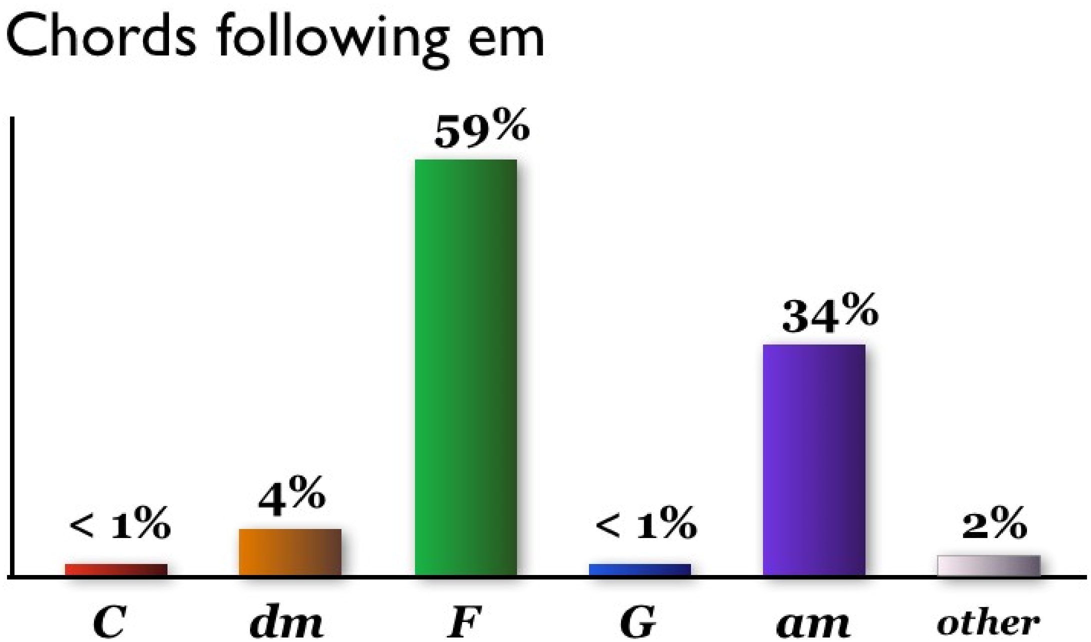
Sequential Pattern Mining: Application 3

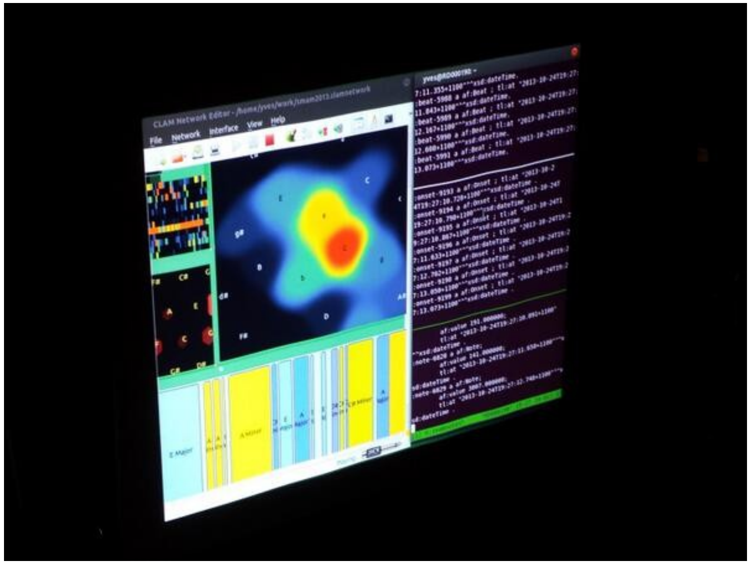
- Using texts as a corpus
  - looking for common sequences of words
  - allows for intelligent suggestions for autocompletion



Sequential Pattern Mining: Application 4

- Chord progressions in music
  - supporting musicians (or even computers) in jam sessions
  - supporting producers in writing top 10 hits :-)



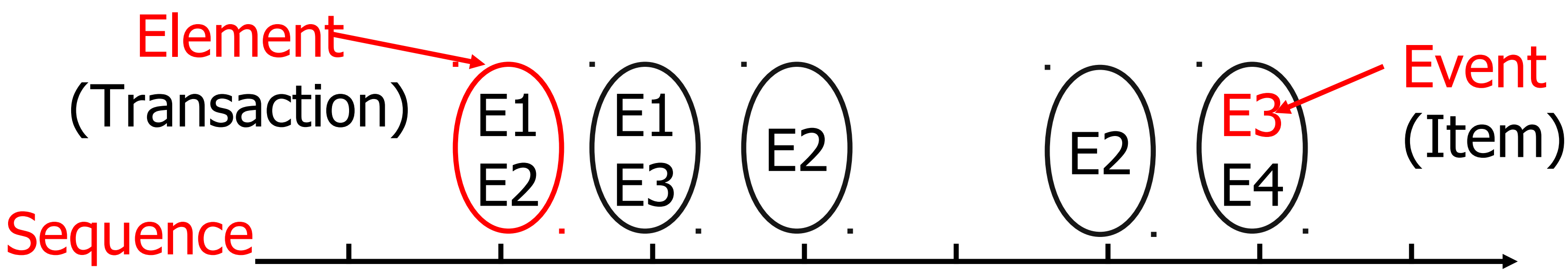


http://www.hooktheory.com/blog/i-analyzed-the-chords-of-1300-popular-songs-for-patterns-this-is-what-i-found/

Sequence Data

- Data Model: transactions containing items

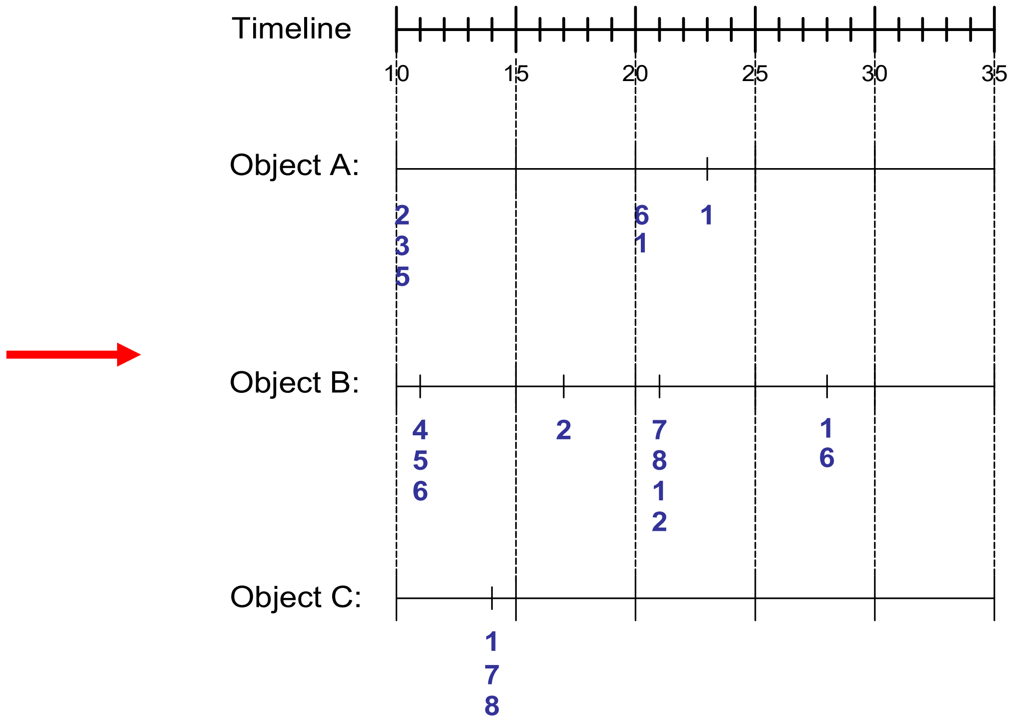
Sequence Database	Sequence	Element (Transaction)	Event (Item)
Customer Data	Purchase history of a given customer	A set of items bought by a customer at time t	Books, dairy products, CDs, etc
Web Server Logs	Browsing activity of a particular Web visitor	A collection of files viewed by a Web visitor after a single mouse click	Home page, index page, contact info, etc
Chord Progressions	Chords played in a song	Individual notes hit at a time	Notes (C, C#, D, ...)



Sequence Data

Sequence Database:

Object	Timestamp	Events
A	10	2, 3, 5
A	20	6, 1
A	23	1
B	11	4, 5, 6
B	17	2
B	21	7, 8, 1, 2
B	28	1, 6
C	14	1, 8, 7



Formal Definition of a Sequence

- A sequence is an ordered list of elements (transactions)

$$s = < e_1 \, e_2 \, e_3 \ldots >$$

  - Each element contains a collection of items (events)

$$e_i = \{i_1, i_2, \ldots, i_k\}$$

  - Each element is attributed to a specific time

- Length of a sequence |s| is given by the number of elements of the sequence.

- A k-sequence is a sequence that contains k events (items).

Further Examples of Sequences

- Web browsing sequence:

  < {Homepage} {Electronics} {Digital Cameras} {Canon Digital Camera} {Shopping Cart} {Order Confirmation} {Homepage} >

- Sequence of books checked out at a library:

  < {Fellowship of the Ring} {The Two Towers, Return of the King} >

- Sequence of initiating events causing the nuclear accident at 3-mile Island:

  < {clogged resin} {outlet valve closure} {loss of feedwater} {condenser polisher outlet valve shut} {booster pumps stop} {main waterpump stops, main turbine stops} {reactor pressure increases} >

Formal Definition of a Subsequence

- A sequence $<a_1\, a_2 \ldots a_n>$ is contained in another sequence $<b_1\, b_2 \ldots b_m>$ ($m \geq n$) if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, …, $a_n \subseteq b_{i_n}$

Data sequence <b>	Subsequence <a>	Contain?
< {2,4} {3,5,6} {8} >	< {2} {3,5} >	Yes
< {1,2} {3,4} >	< {1} {2} >	No
< {2,4} {2,4} {2,5} >	< {2} {4} >	Yes

- The *support* of a subsequence w is defined as the fraction of data sequences that contain w
- A *sequential pattern* is a frequent subsequence (i.e., a subsequence whose support is ≥ *minsup*)

Examples of Sequential Patterns

Table 1. A set of transactions sorted by customer ID and transaction time

Customer ID	Transaction Time	Transaction (items bought)
1	July 20, 2005	30
1	July 25, 2005	90
2	July 9, 2005	10, 20
2	July 14, 2005	30
2	July 20, 2005	40, 60, 70
3	July 25, 2005	30, 50, 70
4	July 25, 2005	30
4	July 29, 2005	40, 70
4	August 2, 2005	90
5	July 12, 2005	90

Examples of Sequential Patterns

Table 2. Data sequences produced from the transaction database in Table 1.

Customer ID	Data Sequence
1	⟨{30} {90}⟩
2	⟨{10, 20} {30} {40, 60, 70}⟩
3	⟨{30, 50, 70}⟩
4	⟨{30} {40, 70} {90}⟩
5	⟨{90}⟩

Table 3. The final output sequential patterns

	Sequential Patterns with Support ≥ 25%
1-sequences	⟨{30}⟩, ⟨{40}⟩, ⟨{70}⟩, ⟨{90}⟩
2-sequences	⟨{30} {40}⟩, ⟨{30} {70}⟩, ⟨{30} {90}⟩, ⟨{40, 70}⟩
3-sequences	⟨{30} {40, 70}⟩

Sequential Pattern Mining

- Given:
  - a database of sequences
  - a user-specified minimum support threshold, *minsup*

- Task:
  - Find all subsequences with support ≥ *minsup*

- Challenge:
  - Very large number of candidate subsequences that need to be checked against the sequence database
  - By applying the Apriori principle, the number of candidates can be pruned significantly

Determining the Candidate Subsequences

- Given n events: $i_1, i_2, i_3, \ldots, i_n$
- Candidate 1-subsequences:
  $<\{i_1\}>, <\{i_2\}>, <\{i_3\}>, \ldots, <\{i_n\}>$
- Candidate 2-subsequences:
  $<\{i_1, i_2\}>, <\{i_1, i_3\}>, \ldots, <\{i_{n-1}, i_n\}>, <\{i_1\}\ \{i_1\}>, <\{i_1\}\ \{i_2\}>, \ldots, <\{i_{n-1}\}\ \{i_n\}>, <\{i_n\}\ \{i_n\}>,$
  $<\{i_2, i_1\}>, <\{i_3, i_1\}>, \ldots, <\{i_n, i_{n-1}\}>,$ $<\{i_2\}\ \{i_1\}>, \ldots, <\{i_n\}\ \{i_{n-1}\}>$
- Candidate 3-subsequences:
  $<\{i_1, i_2, i_3\}>, <\{i_1, i_2, i_4\}>, \ldots, <\{i_1, i_2\}\ \{i_1\}>, <\{i_1, i_2\}\ \{i_2\}>, \ldots,$
  $<\{i_1\}\ \{i_1, i_2\}>, <\{i_1\}\ \{i_1, i_3\}>, \ldots, <\{i_1\}\ \{i_1\}\ \{i_1\}>, <\{i_1\}\ \{i_1\}\ \{i_2\}>, \ldots$

Generalized Sequential Pattern Algorithm (GSP)

- Step 1:
  - Make the first pass over the sequence database D to yield all the 1-element frequent subsequences

- Step 2: Repeat until no new frequent subsequences are found
  1. Candidate Generation:
     - Merge pairs of frequent subsequences found in the (k-1)*th* pass to generate candidate sequences that contain k items
  2. Candidate Pruning:
     - Prune candidate k-sequences that contain infrequent (k-1)-subsequences (Apriori principle)
  3. Support Counting:
     - Make a new pass over the sequence database D to find the support for these candidate sequences
  4. Candidate Elimination:
     - Eliminate candidate k-sequences whose actual support is less than *minsup*

GSP Example

- Only one 4-sequence survives the candidate pruning step
- All other 4-sequences are removed because they contain subsequences that are not part of the set of frequent 3-sequences

Frequent 3-sequences

< {1} {2} {3} >
< {1} {2 5} >
< {1} {5} {3} >
< {2} {3} {4} >
< {2 5} {3} >
< {3} {4} {5} >
< {5} {3 4} >

Candidate Generation

< {1} {2} {3} {4} >
< {1} {2 5} {3} >
< {1} {5} {3 4} >
< {2} {3} {4} {5} >
< {2 5} {3 4} >

Candidate Pruning

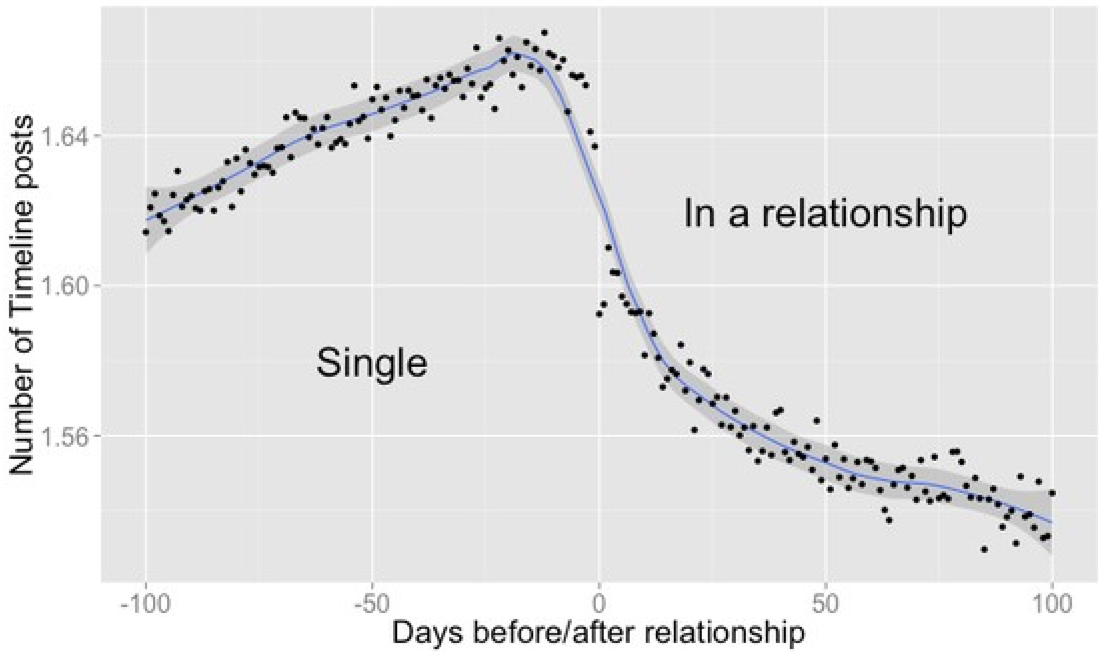
< {1} {2 5} {3} >

Trend Detection

- Task
  - given a time series
  - find out what the general trend is (e.g., rising or falling)
- Possible obstacles
  - random effects: ice cream sales have been low this week due to rain
    - but what does that tell about next week?
  - seasonal effects: sales have risen in December
    - but what does that tell about January?
  - cyclical effects: less people attend a lecture towards the end of the semester
    - but what does that tell about the next semester?

Trend Detection

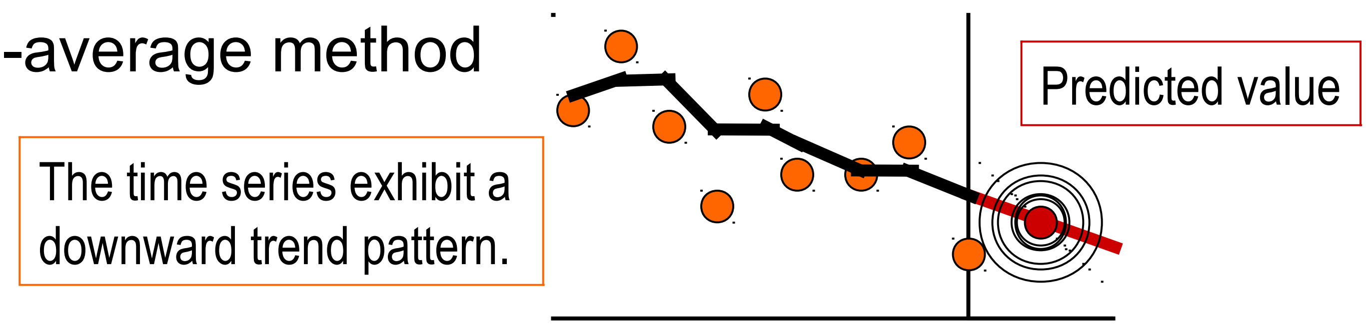
- Example: Data Analysis at Facebook



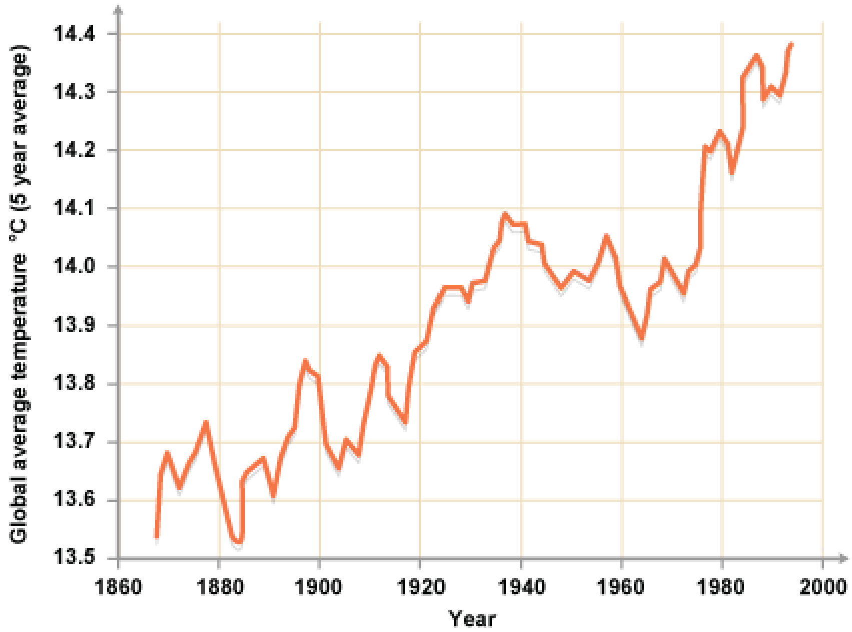
http://www.theatlantic.com/technology/archive/2014/02/when-you-fall-in-love-this-is-what-facebook-sees/283865/

Estimation of Trend Curves

- The freehand method
  - Fit the curve by looking at the graph
  - Costly and barely reliable for large-scale data mining
- The least-squares method
  - Find the curve minimizing the sum of the squares of the deviation of points on the curve from the corresponding data points
  - cf. linear regression
- The moving-average method

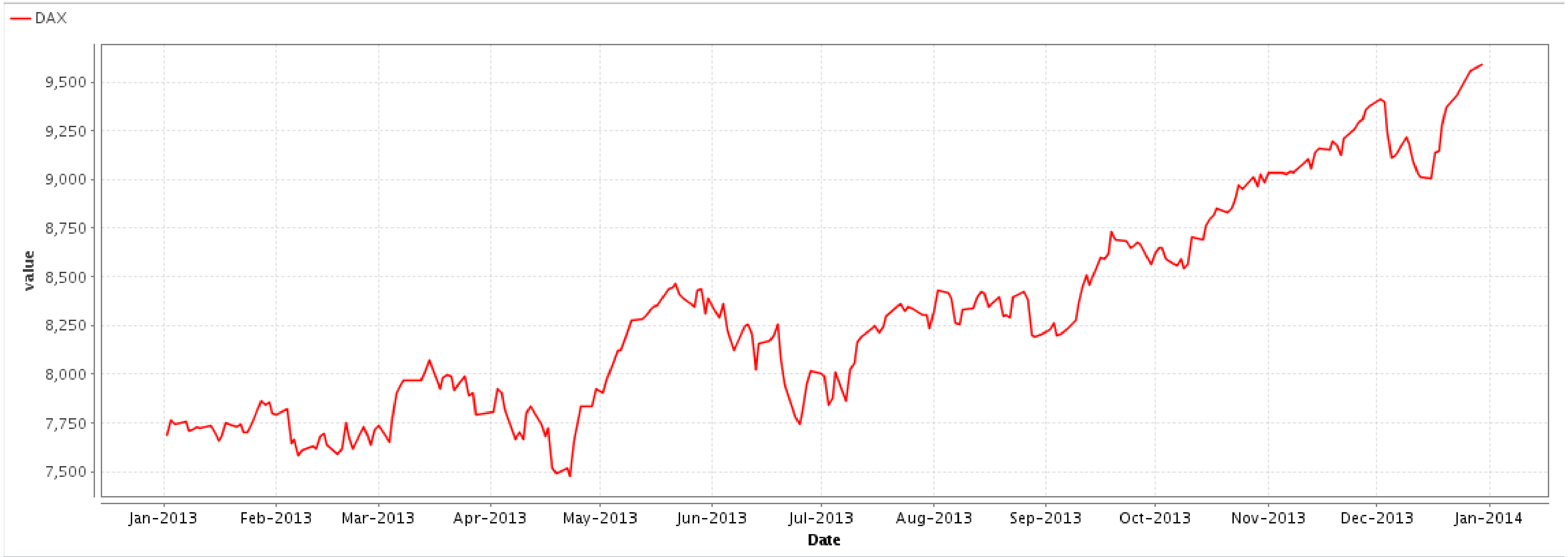


Example: Average Global Temperature



http://www.bbc.co.uk/schools/gcsebitesize/science/aqa_pre_2011/rocks/fuelsrev6.shtml

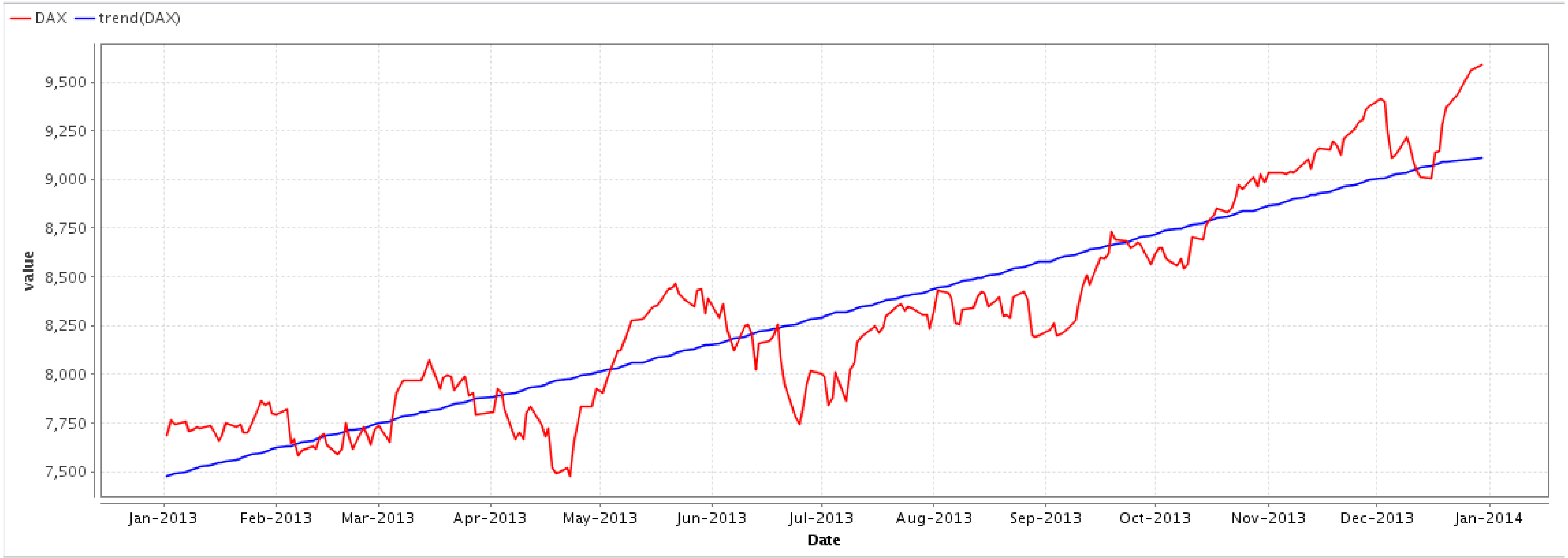
Example: German DAX 2013



Linear Trend

- Given a time series that has timestamps and values, i.e.,
  - (t_i,v_i), where t_i is a time stamp, and v_i is a value at that time stamp
- A linear trend is a linear function
  - $m*t_i + b$
- We can find via linear regression, e.g., using the least squares fit

Example: German DAX 2013



A Component Model of Time Series

A **time series** can consist of four components:

- Long - term trend (T_t)
- Cyclical effect (C_t)
- Seasonal effect (S_t)
- Random variation (R_t)

this is what we want to find

we need to eliminate those

Additive Model:

- Series = $T_t + C_t + S_t + R_t$

Multiplicative Model:

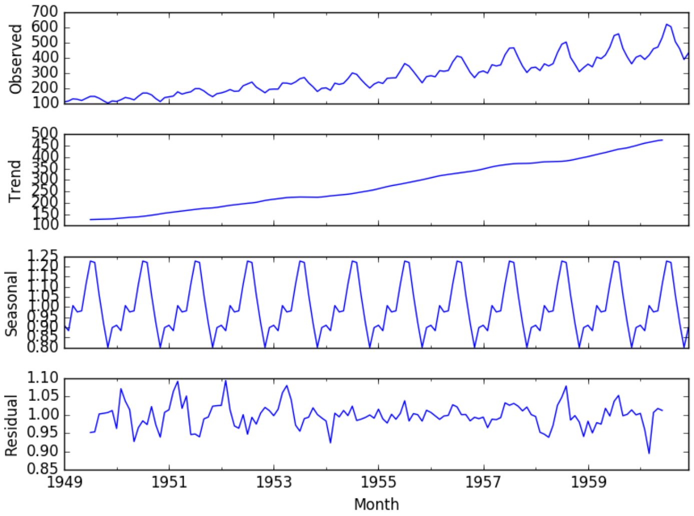
- Series = $T_t \times C_t \times S_t \times R_t$

Seasonal and Cyclical Effects

- Seasonal effects occur regularly each year
  - quarters
  - months
  - …
- Cyclical effects occur regularly over other intervals
  - every N years
  - in the beginning/end of the month
  - on certain weekdays or on weekends
  - at certain times of the day
  - ...

Identifying Seasonal and Cyclical Effects

- There are methods of identifying and isolating those effects
  - given that the periodicity is known



- Python: statsmodels package

```
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.seasonal
  import seasonal_decompose
series = Series.from_csv
  ('data.csv', header=0)
result = seasonal_decompose
  (series, model='multiplicative')
result.plot()
pyplot.show()
```

Identifying Seasonal and Cyclical Effects

- Variation may occur within a year or another period
- To measure the seasonal effects we compute *seasonal indexes*
- Seasonal index
  - degree of variation of seasons in relation to global average

http://davidsills.blogspot.de/2011/10/seasons.html

Identifying Seasonal and Cyclical Effects

- Algorithm
  - Compute the trend $\hat{y}_t$ (i.e., linear regression)
  - For each time period
    - compute the ratio $y_t/\hat{y}_t$
  - For each *season* (or other relevant period)
    - compute the average of $y_t/\hat{y}_t$
    - this gives us the average deviation for that season

here, we assume the multiplicative model

$$\frac{y_t}{\hat{y}_t} = \frac{T_t \times S_t \times R_t}{T_t} = S_t \times R_t$$

the computed ratios isolate the seasonal and random variation from the overall trend*

*) given that no additional cyclical variation exists

Example for Seasonal Effects

- Calculate the quarterly seasonal indexes for hotel occupancy rate in order to measure seasonal variation
- Data:

Year	Quarter	Rate	Year	Quarter	Rate	Year	Quarter	Rate
1996	1	0.561	1998	1	0.594	2000	1	0.665
	2	0.702		2	0.738		2	0.835
	3	0.8		3	0.729		3	0.873
	4	0.568		4	0.6		4	0.67
1997	1	0.575	1999	1	0.622			
	2	0.738		2	0.708			
	3	0.868		3	0.806			
	4	0.605		4	0.632			

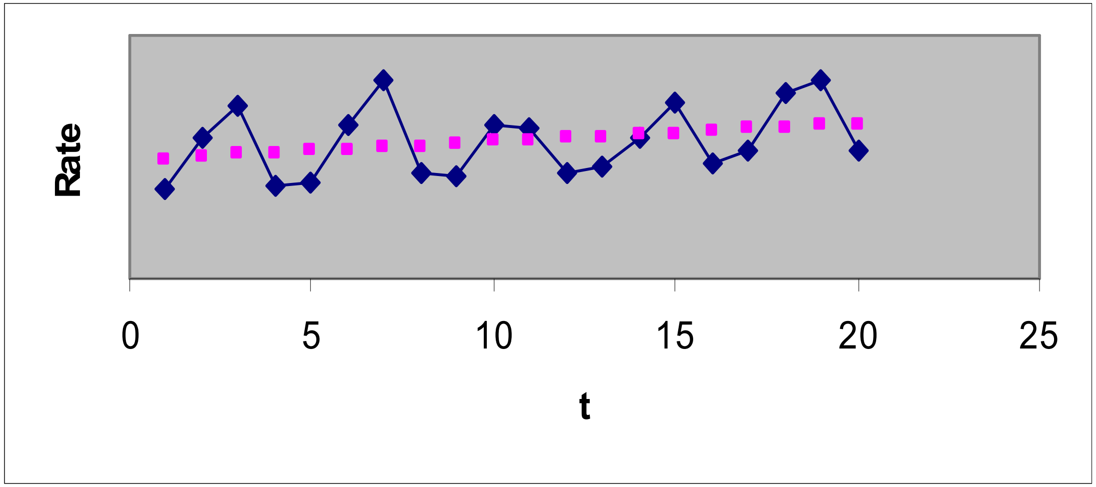
This example is taken from the course “Regression Analysis”
at University of Umeå, Department of Statistics

Example for Seasonal Effects

- First step: compute trend from the data
  - e.g., linear regression

Time (t)	Rate
1	0.561
2	0.702
3	0.800
4	0.568
5	0.575
6	0.738
7	0.868
8	0.605
.	.
.	.

$$\hat{y} = 0.639368 + 0.005246\,t$$

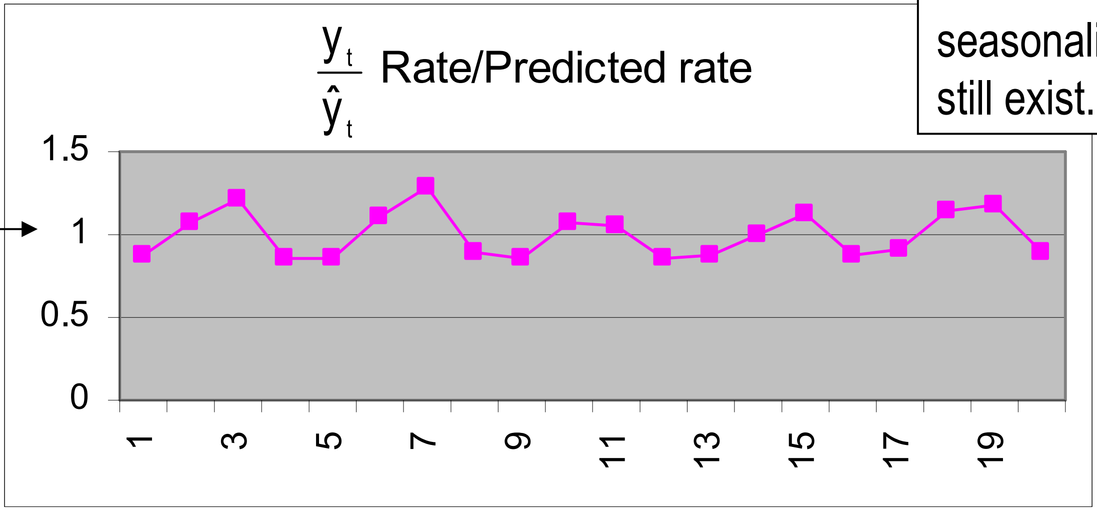


Example for Seasonal Effects

- Second step: compute ratios $y_t/\hat{y}_t$

t	y_t	$\hat{y}_t$	Ratio
1	.561	.645	.561/.645=.870
2	.702	.650	.702/.650=1.08
3	…………………………………………		

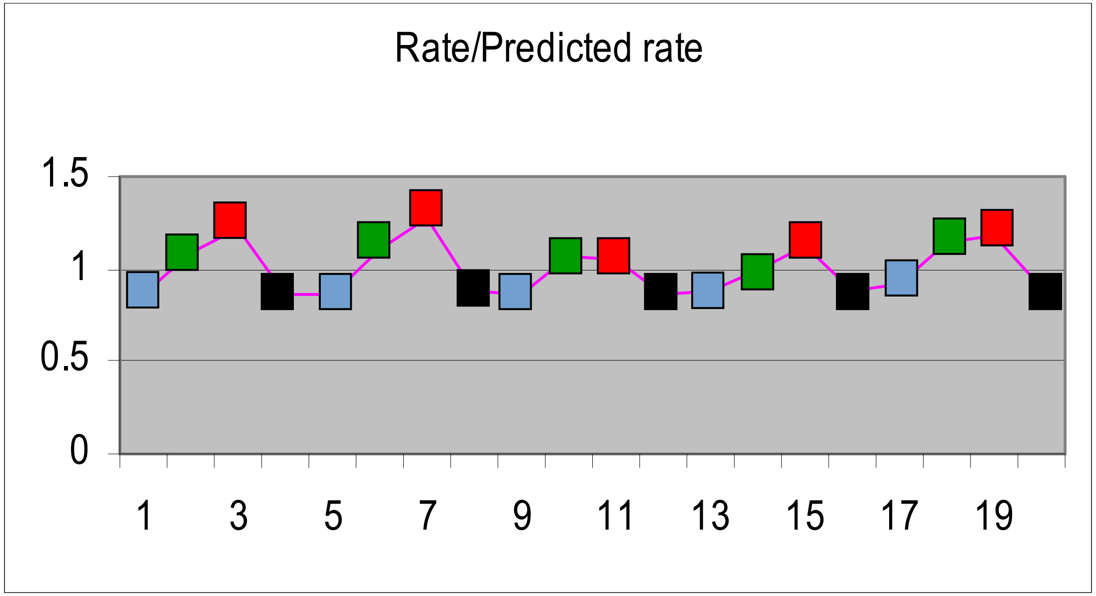
=0.639368+0.005245*t



No trend is observed, but seasonality and randomness still exist.

Example for Seasonal Effects

- Third step: compute average ratios by season



Rate/Predicted rate
0.870
1.080
1.221
0.860
0.864
1.100
1.284
0.888
0.865
1.067
1.046
0.854
0.879
0.993
1.122
0.874
0.913
1.138
1.181
0.900

Average ratio for quarter 1: (.870 + .864 + .865 + .879 + .913)/5 = .878

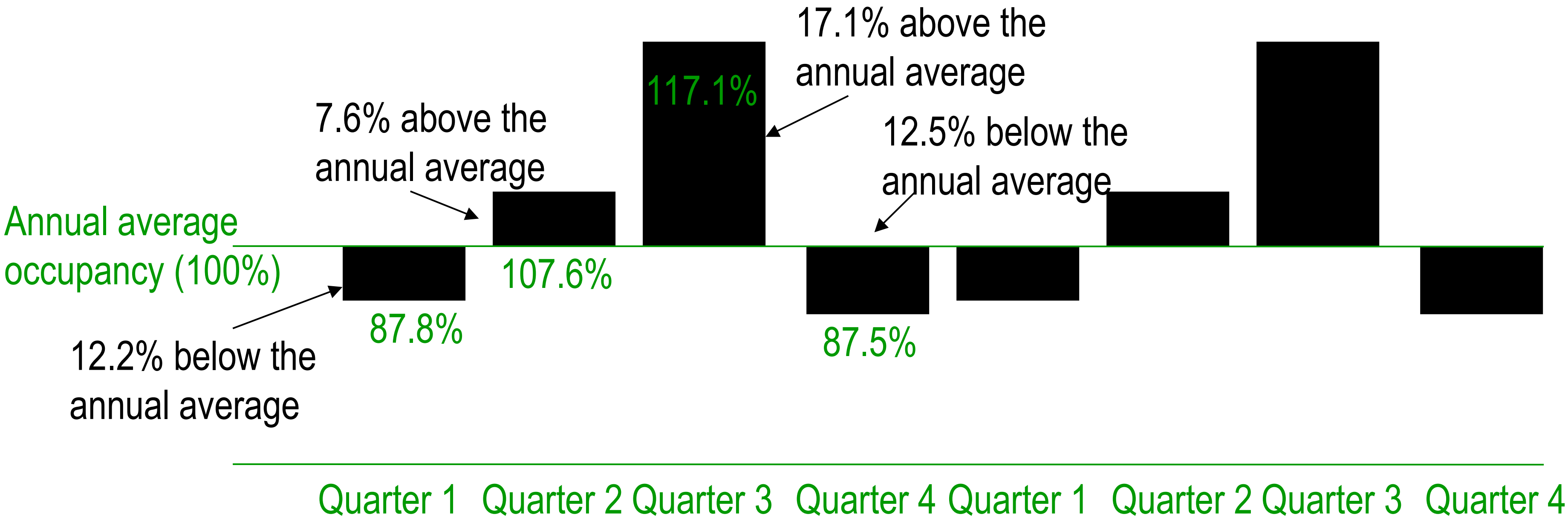
Average ratio for quarter 2: (1.080+1.100+1.067+.993+1.138)/5 = 1.076

Average ratio for quarter 3: (1.221+1.284+1.046+1.122+1.181)/5 = 1.171

Average ratio for quarter 4: (.860 +.888 + .854 + .874 + .900)/ 5 = .875

Example for Seasonal Effects

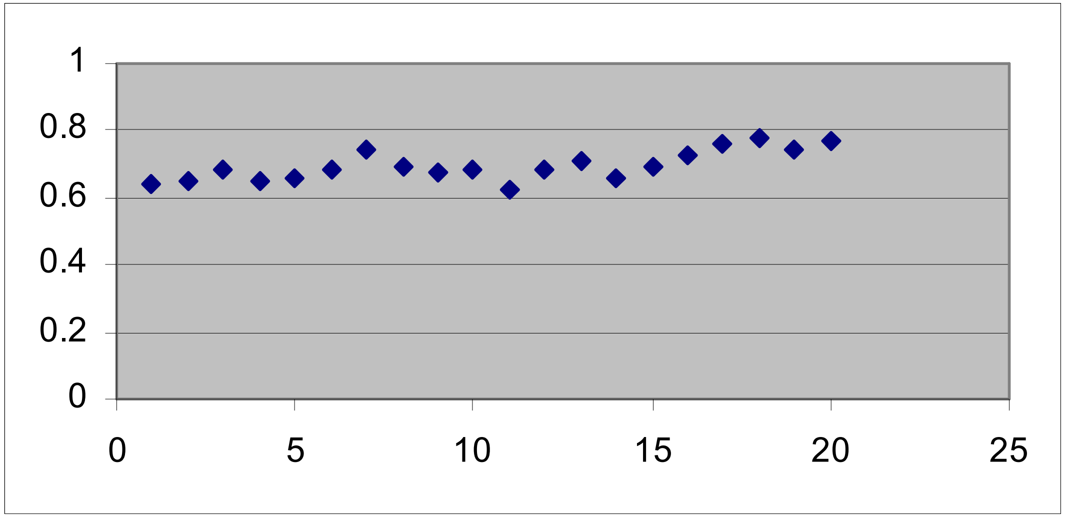
- Interpretation of seasonal indexes:
  - ratio between the time series' value at a certain season and the overall seasonal average
- In our problem:



Example for Seasonal Effects

- Deseasonalizing time series
  - when ignoring seasonal effects, is there still an increase?

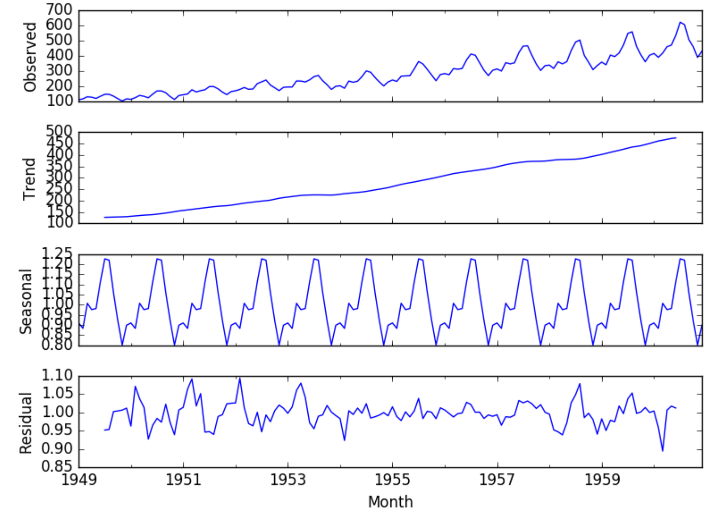
$$\text{Seasonally adjusted time series} = \frac{\text{Actual time series}}{\text{Seasonal index}}$$



Trend on deseasonalized time series: slightly positive

Determining the Periodicity

- There are methods of identifying and isolating those effects
  - given that the periodicity is known
- What if we don't know the periodicity?



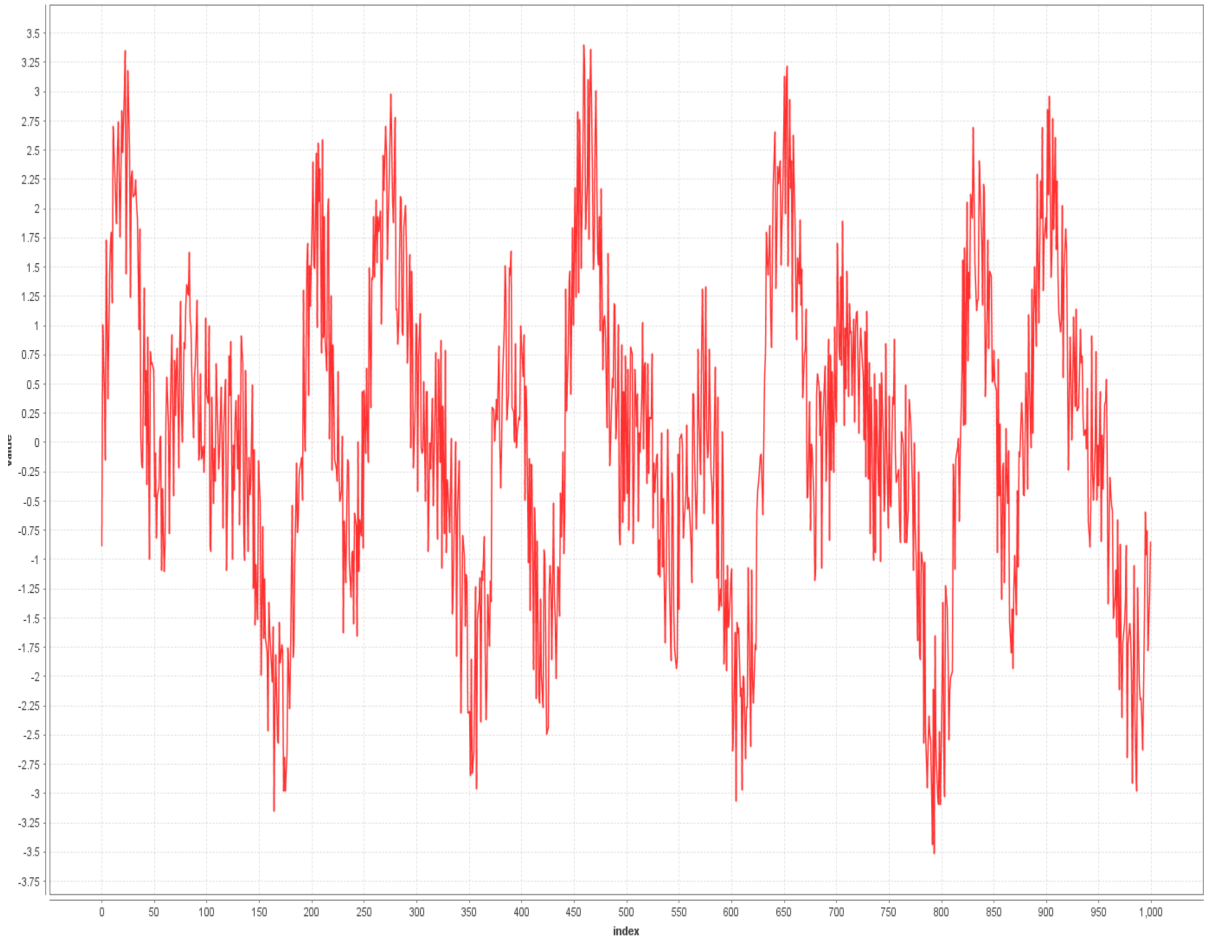
Determining the Periodicity

- Assumption: time series is a sum of sine waves
  - With different periodicity
  - Different representation of the time series

- The frequencies of those sine waves is called *spectrum*
  - *Fourier transformation* transforms between spectrum and series
  - Spectrum gives hints at the frequency of periodic effects
  - Details: see textbooks

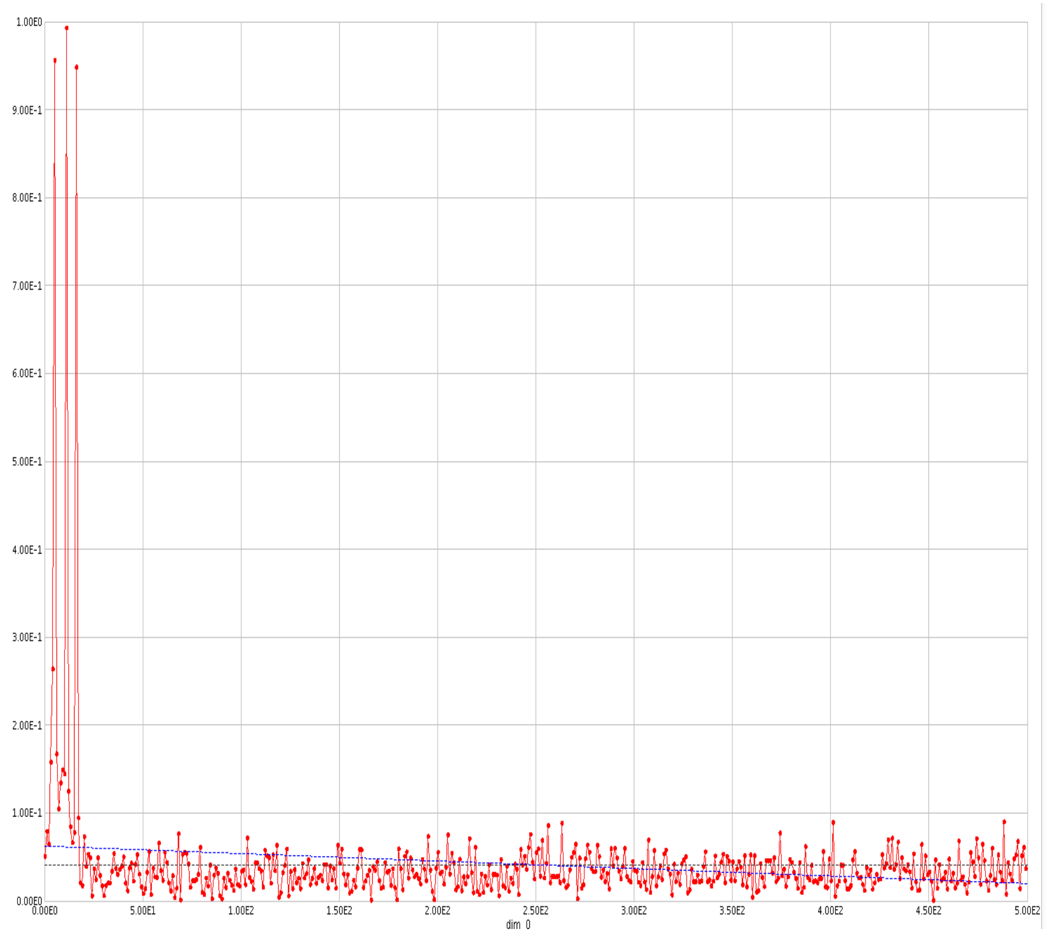
Determining the Periodicity

- Example: three interfering sine waves with noise added



Determining the Periodicity

- The corresponding spectrum



Dealing with Random Variations

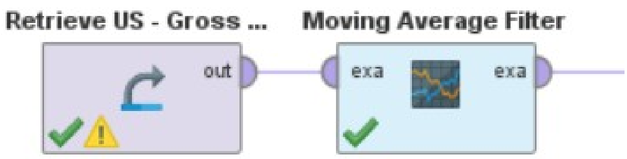
- *Moving average* of order n

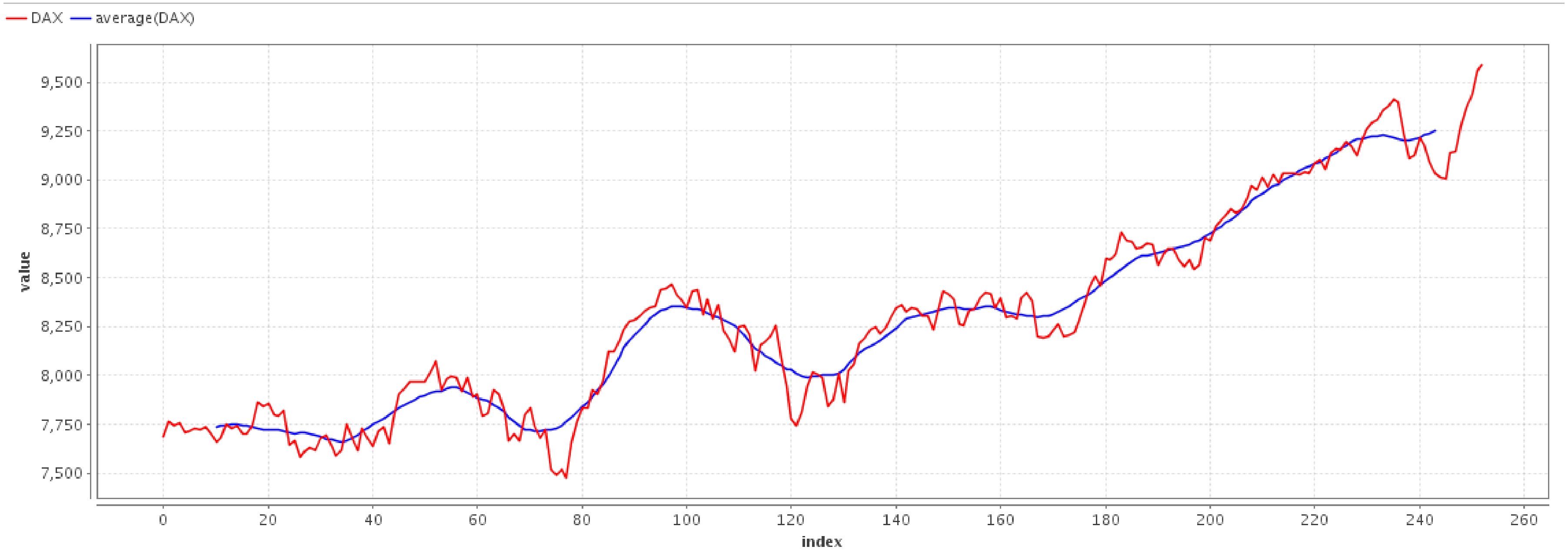
$$\frac{y_1 + y_2 + \cdots + y_n}{n}, \frac{y_2 + y_3 + \cdots + y_{n+1}}{n}, \frac{y_3 + y_4 + \cdots + y_{n+2}}{n}, \cdots$$

- Key idea:
  - upcoming value is the average of the last n
  - cf.: nearest neighbors
- Properties:
  - Smoothes the data
  - Eliminates *random* movements
  - Loses the data at the beginning or end of a series
  - Sensitive to outliers (can be reduced by weighted moving average)

Moving Average in RapidMiner and Python

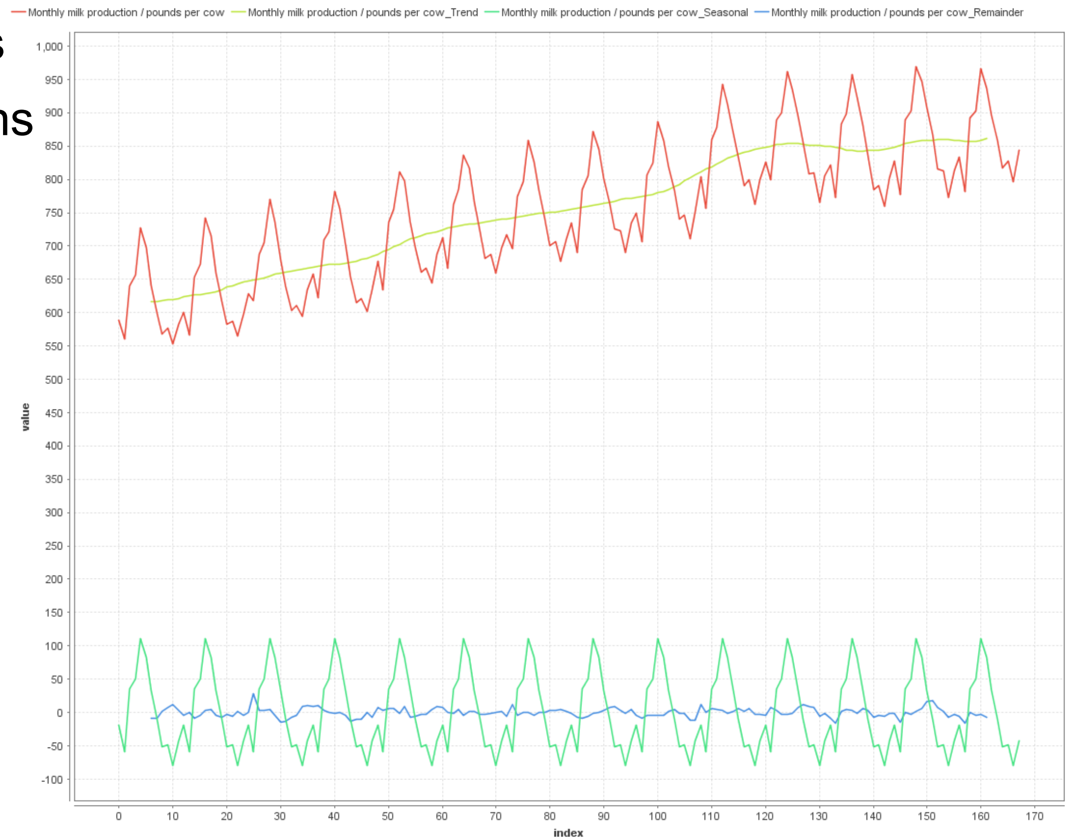
- Python:
  - e.g., rolling_mean in pandas
- Alternatives for average:
  - median, mode, …





Moving Average and Decomposition

- Often, moving averages are used for the trend
  - instead of a linear trend
  - less susceptible to outliers
  - the remaining computations stay the same



Dealing with Random Variations

- Exponential Smoothing
  - $S_t = \alpha y_t + (1-\alpha)S_{t-1}$
  - α is a smoothing factor
  - recursive definition
    - in practice, start with $S_0 = y_0$

`Python: statsmodels package`

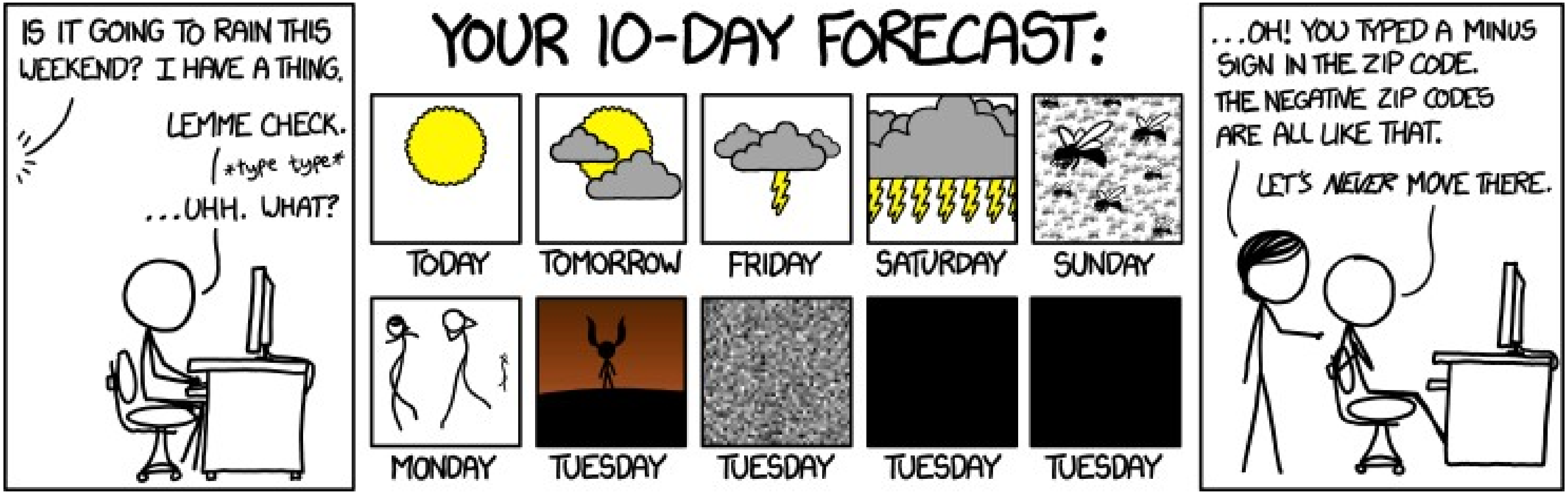
- Properties:
  - Smoothes the data
  - Eliminates random movements
    - and even seasonal effects for smaller values of α
  - Smoothing values for whole series
  - More recent values have higher influence

Dealing with Random Variations

Recap: Trend Analysis

- Allows to identify general trends (upward, downward)
- Overall approach:
  - eliminate all other components so that only the trend remains
- Method for factoring out seasonal variations
  - and compute deseasonalized time series
- Methods for eliminating with random variations (smoothing)
  - moving average
  - exponential smoothing

Time Series Prediction: Definition



http://xkcd.com/1245/

From Moving Averages to Autoregressive Models

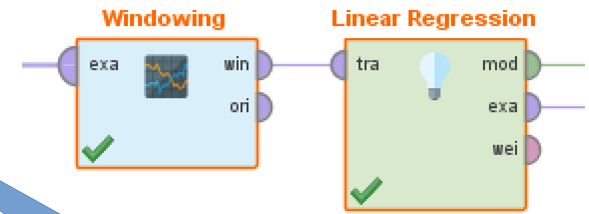
- Recap moving average for smoothing
  - each value is replaced by the average of its surrounding ones
- Moving average for prediction
  - predict the average of the last n values
  - $y_t = 1/n * (y_{t-1} + \ldots yt-n)$

- Here: weights are uniform
  - advanced: weights are learned from the data
  - $y_t = \delta_1 y_{t-1} + \delta_2 y_{t-2} + \ldots \delta_n y_{t-n} + \beta + \varepsilon_t$
  - just like linear regression learning
  - this is called an *autoregressive* model
    - i.e., regression trained on the time series itself

Autoregressive Models in RapidMiner / Python

- RapidMiner: only with a twist
  - generate windowed representation for learning first
  - learn linear model on top

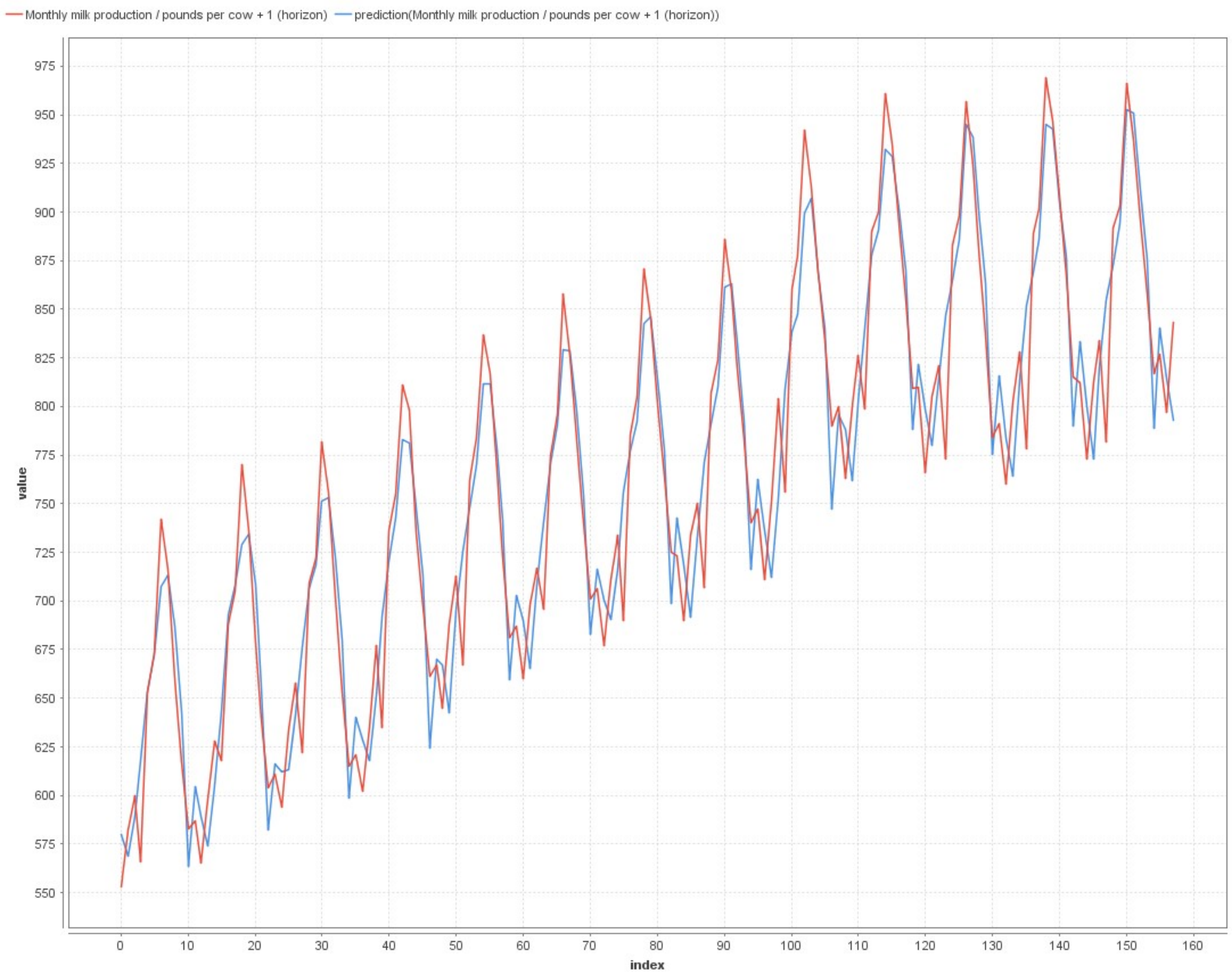
Row No. ↑	Window id	Copper price + 1 (horizon)	Copper price - 9	Copper price - 8	Copper price - 7	Copper price - 6	Copper price - 5	Copper price - 4	Copper price - 3
1	0	2.268	0.246	0.627	0.529	0.528	1.086	1.001	1.491
2	1	0.450	0.627	0.529	0.528	1.086	1.001	1.491	0.293
3	2	0.746	0.529	0.528	1.086	1.001	1.491	0.293	[illegible]
4	3	0.059	0.528	1.086	1.001	1.491	0.293	0.189	0.536
5	4	1.111	1.086	1.001	1.491	0.293	0.189	0.536	2.268
6	5	1.981	1.001	1.491	0.293	0.189	0.536	2.268	0.450
7	6	3.232	1.491	0.293	0.189	0.536	2.268	0.450	0.746
8	7	2.565	0.293	0.189	0.536	2.268	0.450	0.746	0.059
9	8	2.336	0.189	0.536	2.268	0.450	0.746	0.059	1.111
10	9	1.978	0.536	2.268	0.450	0.746	0.059	1.111	1.981
11	10	1.391	2.268	0.450	0.746	0.059	1.111	1.981	3.232
12	11	1.744	0.450	0.746	0.059	1.111	1.981	3.232	2.565
13	12	1.538	0.746	0.059	1.111	1.981	3.232	2.565	2.336
14	13	1.114	0.059	1.111	1.981	3.232	2.565	2.336	1.978
15	14	0.084	1.111	1.981	3.232	2.565	2.336	1.978	1.391
16	15	0.050	1.981	3.232	2.565	2.336	1.978	1.391	1.744
17	16	0.923	3.232	2.565	2.336	1.978	1.391	1.744	1.538
18	17	1.072	2.565	2.336	1.978	1.391	1.744	1.538	1.114
19	18	1.149	2.336	1.978	1.391	1.744	1.538	1.114	0.084
20	19	1.520	1.978	1.391	1.744	1.538	1.114	0.084	0.050
21	20	1.415	1.391	1.744	1.538	1.114	0.084	0.050	0.923
22	21	0.862	1.744	1.538	1.114	0.084	0.050	0.923	1.072
23	22	0.428	1.538	1.114	0.084	0.050	0.923	1.072	1.149
24	23	0.467	1.114	0.084	0.050	0.923	1.072	1.149	1.520



lagged values/
lag variables

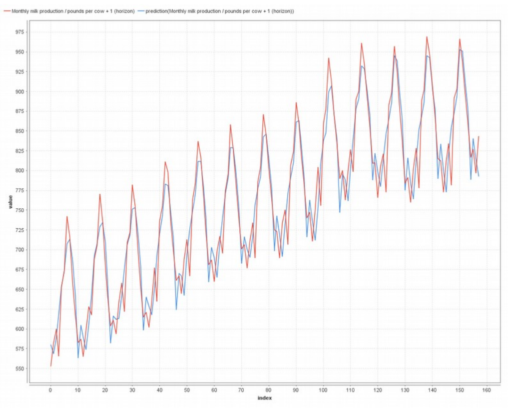
```
from statsmodels.tsa.ar_model import AR
```

Autoregressive Models



Autoregressive Models

- First observation:
  - we have learned a linear model using the lag values
  - but the prediction itself is not linear!



- Second observation:
  - periodicities are learned well
- Why?
  - e.g., given that we have a strong weekly trend
  - we will learn a high weight for δ_{t-7}
  - multiple periodicities can also be learned
    - e.g., time series with weekly and monthly component

Extension of AR models

- ARMA
  - Fits an AR model
  - Fits a second model to estimate the errors made by the AR model
  - $y_t = \delta_1 y_{t-1} + \delta_2 y_{t-2} + \ldots \delta_p y_{t-p} + \beta + \gamma_1 \varepsilon_{t-1} + \ldots + \gamma_q \varepsilon_{q-1}$
- ARIMA
  - Tries to predict a differenced model
    - i.e., the relative change of a time series instead of the absolute value
  - ARIMA models come with three parameters:
    - p: number of terms in the AR part
    - q: number of terms in the MA part
    - d: number of times the time series is differenced

Lag Variables for Nominal Prediction

Date	Weather
1.1.	Sunny
2.1.	Cloudy
3.1.	
4.1.	
5.1.	
6.1.	
7.1.	
8.1.	
9.1.	

Date	Weather-3	Weather-2	Weather-1	Weather
1.1.	?	?	?	Sunny
2.1.	?	?	Sunny	Cloudy
3.1.	?	Sunny	Cloudy	Cloudy
4.1.	Sunny	Cloudy	Cloudy	Rainy
5.1.	Cloudy	Cloudy	Rainy	Cloudy
6.1.	Cloudy	Rainy	Cloudy	Sunny
7.1.	Rainy	Cloudy	Sunny	Sunny
8.1.	Cloudy	Sunny	Sunny	Sunny
9.1.	Sunny	Sunny	Sunny	Rainy

Lag Variables in Multivariate Series

- Also possible for multi-variate data

Result Overview | ExampleSet (Windowing)

Data View | Meta Data View | Plot View | Advanced Charts | Annotations

ExampleSet (250 examples, 2 special attributes, 6 regular attributes)

Row No.	Date	Weather-2	Weather-1	Weather-0	Temperature-2	Temperature-1	Temperature-0	label
1	04.01.2013	sunny	cloudy	cloudy	23	24	28	cloudy
2	07.01.2013	cloudy	cloudy	cloudy	24	28	32	rainy
3	08.01.2013	cloudy	cloudy	rainy	28	32	19	sunny
4	09.01.2013	cloudy	rainy	sunny	32	19	24	rainy
5	10.01.2013	rainy	sunny	rainy	19	24	25	cloudy
6	11.01.2013	sunny	rainy	cloudy	24	25	17	sunny
7	14.01.2013	rainy	cloudy	sunny	25	17	14	sunny
8	15.01.2013	cloudy	sunny	sunny	17	14	12	rainy
9	16.01.2013	sunny	sunny	rainy	14	12	26	sunny
10	17.01.2013	sunny	rainy	sunny	12	26	23	cloudy
11	18.01.2013	rainy	sunny	cloudy	26	23	24	cloudy
12	21.01.2013	sunny	cloudy	cloudy	23	24	28	cloudy
13	22.01.2013	cloudy	cloudy	cloudy	24	28	32	rainy
14	23.01.2013	cloudy	cloudy	rainy	28	32	19	sunny
15	24.01.2013	cloudy	rainy	sunny	32	19	24	rainy
16	25.01.2013	rainy	sunny	rainy	19	24	25	cloudy
17	28.01.2013	sunny	rainy	cloudy	24	25	17	sunny

Predicting with Exponential Smoothing

- Recap exponential smoothing
  - $S_t = \alpha y_t + (1-\alpha)S_{t-1}$
    - We can also understand S_t as a prediction of y_{t+1}
    - i.e., we predict the average of the last value and the last prediction

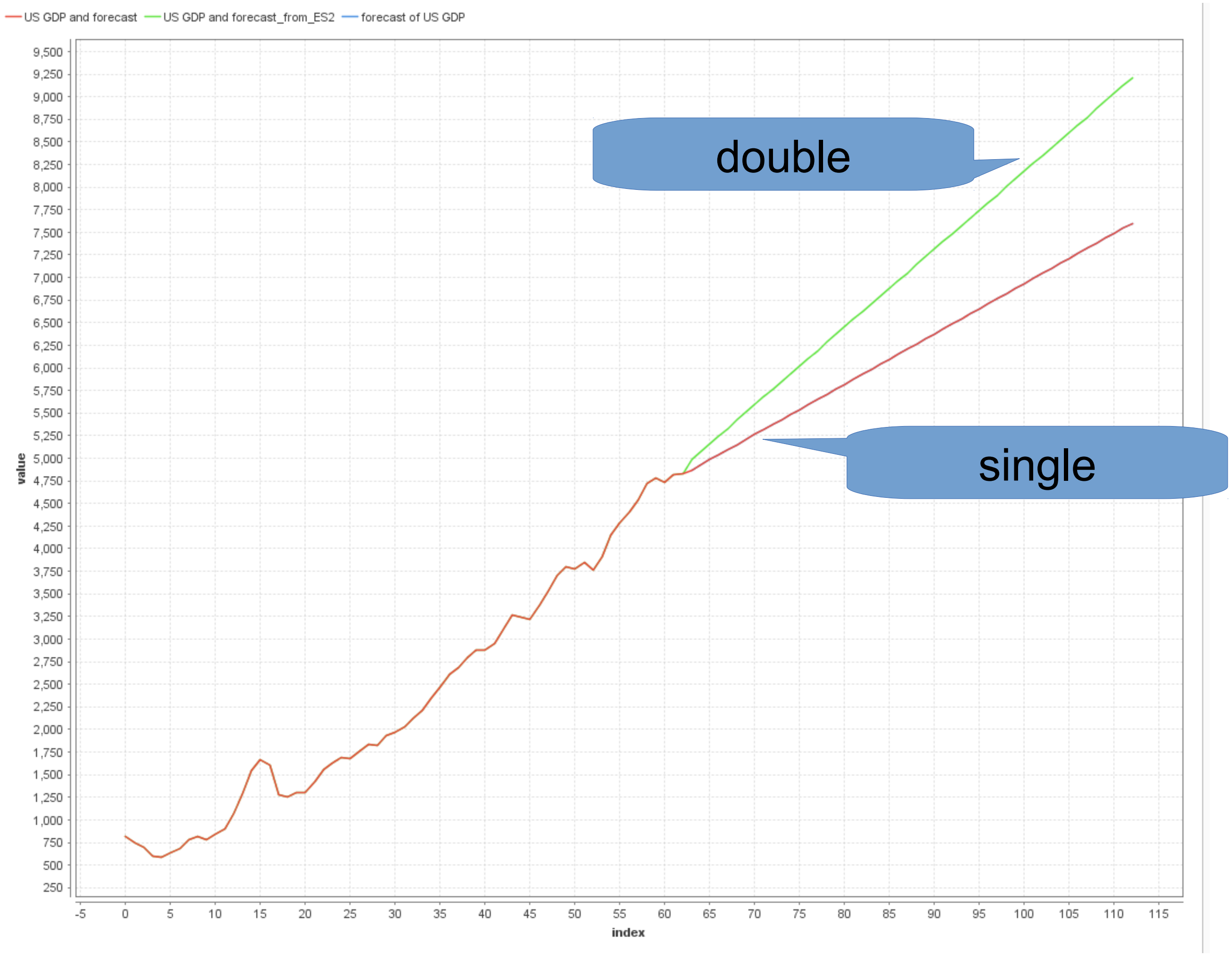
- By recursion, we can use exponential smoothing for prediction
  - i.e., predict one step into the future
    - then use this prediction as input to the next step
  - works OK for short forecasting windows
  - at some point, the predictions usually diverge

Predicting with Exponential Smoothing

Double Exponential Smoothing

- Larger values for α:
  - more cancellation of random noise, but
  - exponential smoothing takes longer to adapt to trend
- With a trend, the smoothed time series will rise/fall over time
  - $S_t = \alpha y_t + (1-\alpha)(S_{t-1}+b_{t-1})$ Estimated trend
  - $b_t = \beta(S_t-S_{t-1})+(1-\beta)b_{t-1}$
- Explanation:
  - $S_t - S_{t-1}$ describes the change of the estimate
  - b is the exponentially smoothed time series of those changes
- S is called *level smoothing*, b is called *trend smoothing*

Double Exponential Smoothing: Example



Triple Exponential Smoothing

- Double exponential smoothing
  - Uses level and trend, but no seasonality

- Triple exponential smoothing (also known as Holt Winters Method)
  - Introduces seasonal component
  - $S_t = \alpha(y_t - c_{t-L}) + (1-\alpha)(S_{t-1}+b_{t-1})$
  - $b_t = \beta(S_t - S_{t-1})+(1-\beta)b_{t-1}$
  - $c_t = \gamma(yt - S_t) + (1-\gamma)c_{t-L}$

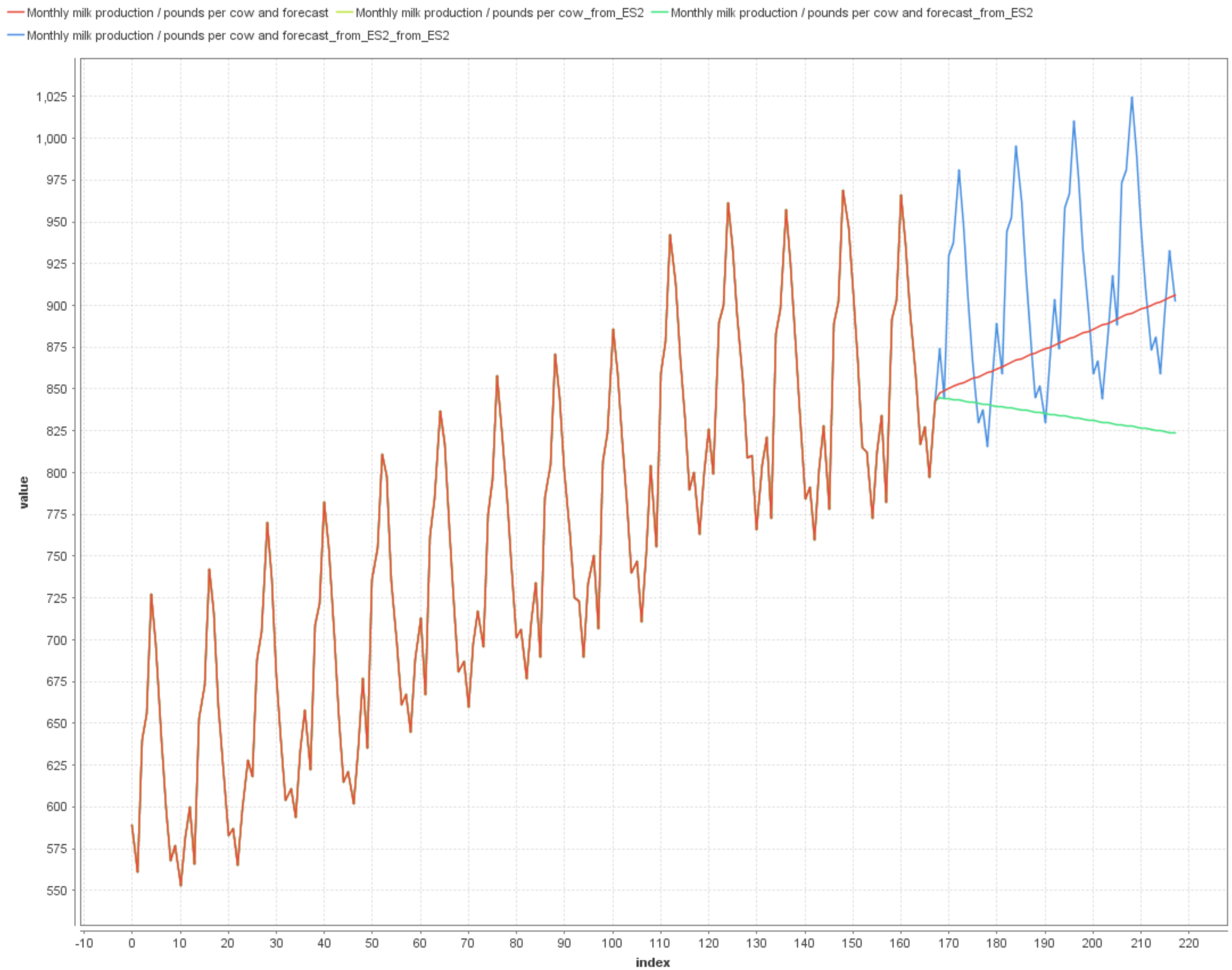
Most recent value and its trend component

L is the cycle length of the seasonality

Triple Exponential Smoothing

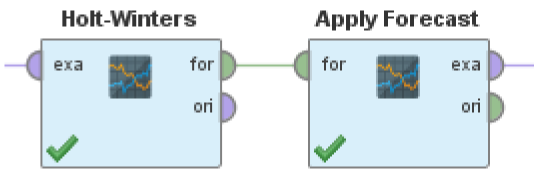
- Cycle length L
  - counted in number of observations
- Examples:
  - weekly cycles, one observation = one day: 7
  - yearly cycles, one observation = one month: 12
  - hourly cycles, one observation = one second: 3600

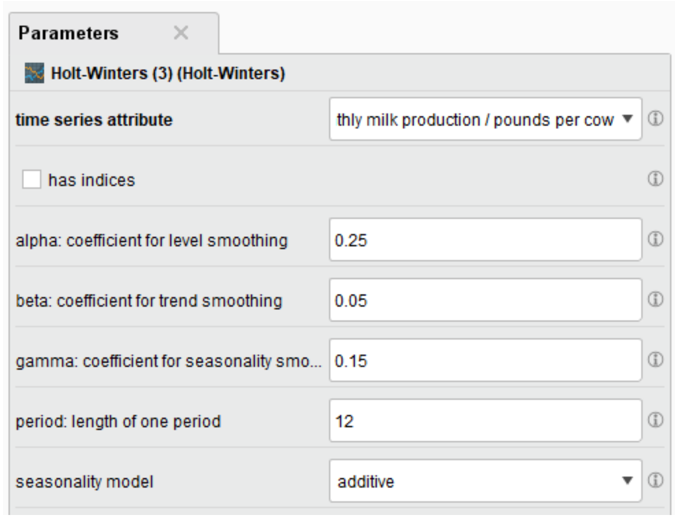
Triple Exponential Smoothing



Holt Winters in RapidMiner and Python

- Parameters:
  - α, β, γ
  - period length
- Python implemention:
  - can also estimate parameters
  - as to fit the given data best
- Both implementations:
  - have additive and multiplicative variant
  - multiplicative often works better

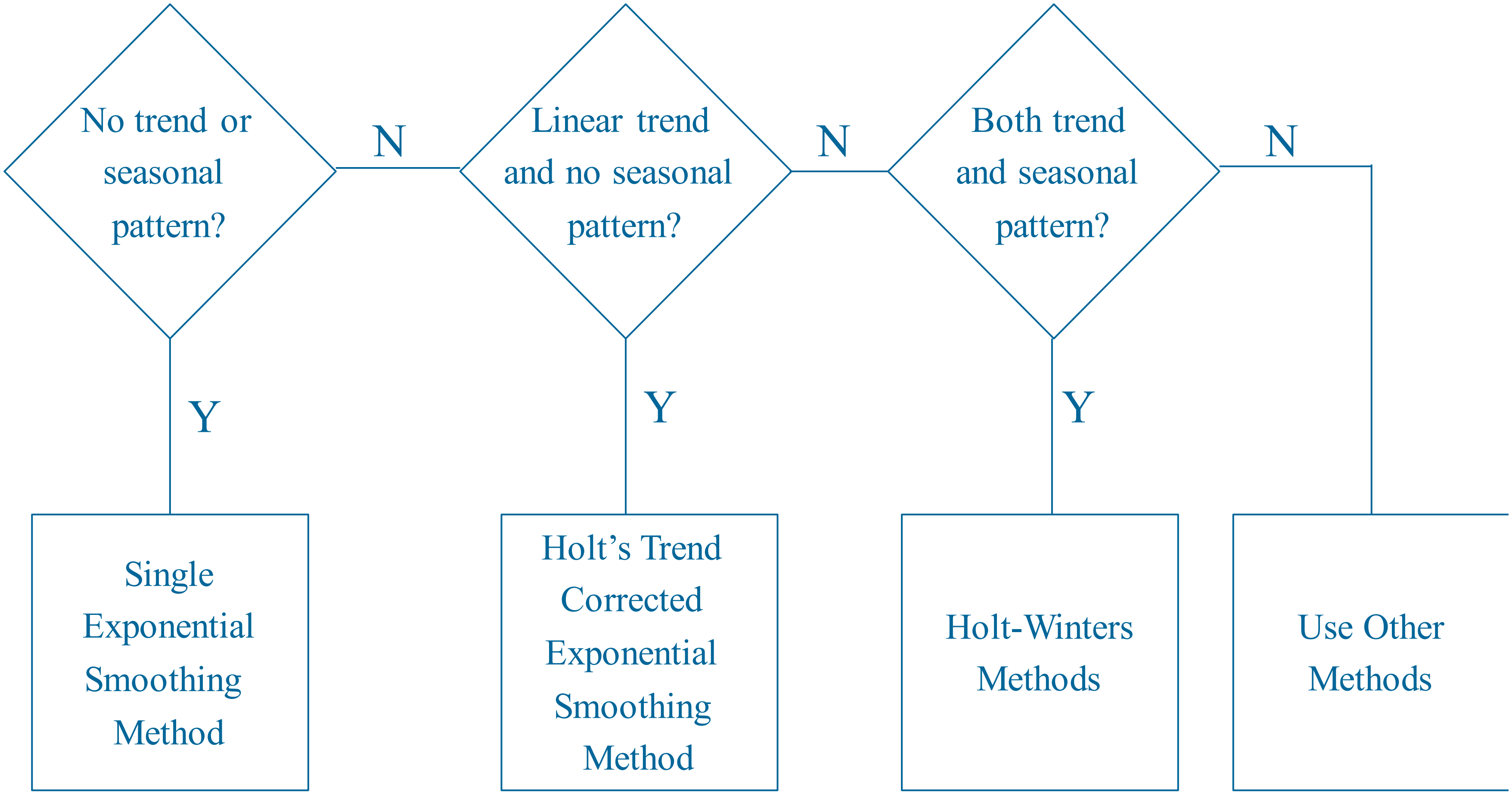





```
from statsmodels.tsa.holtwinters
import ExponentialSmoothing
```

Selecting an Exponential Smoothing Model

- Taken from Alan Wan, Forecasting Methods for Business



Missing Values in Series Data

- Remedies in non-series data:
  - replace with average, median, most frequent
  - Imputation (e.g., k-NN)
  - replace with most frequent
  - …

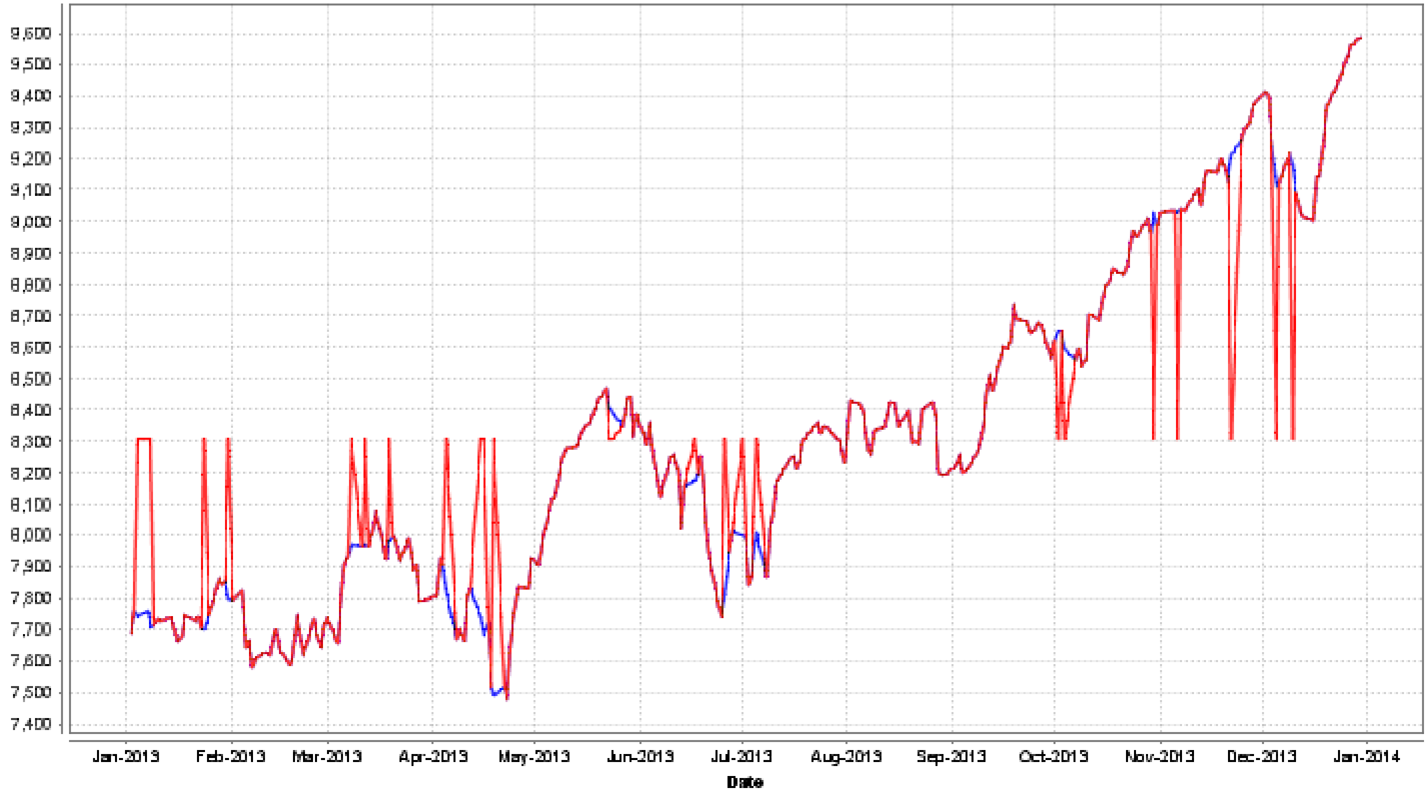
- What happens if we apply those to time series?

Missing Values in Series Data

- Original time series
  - with missing values inserted

Missing Values in Series Data

- Replace with average



Missing Values in Series Data

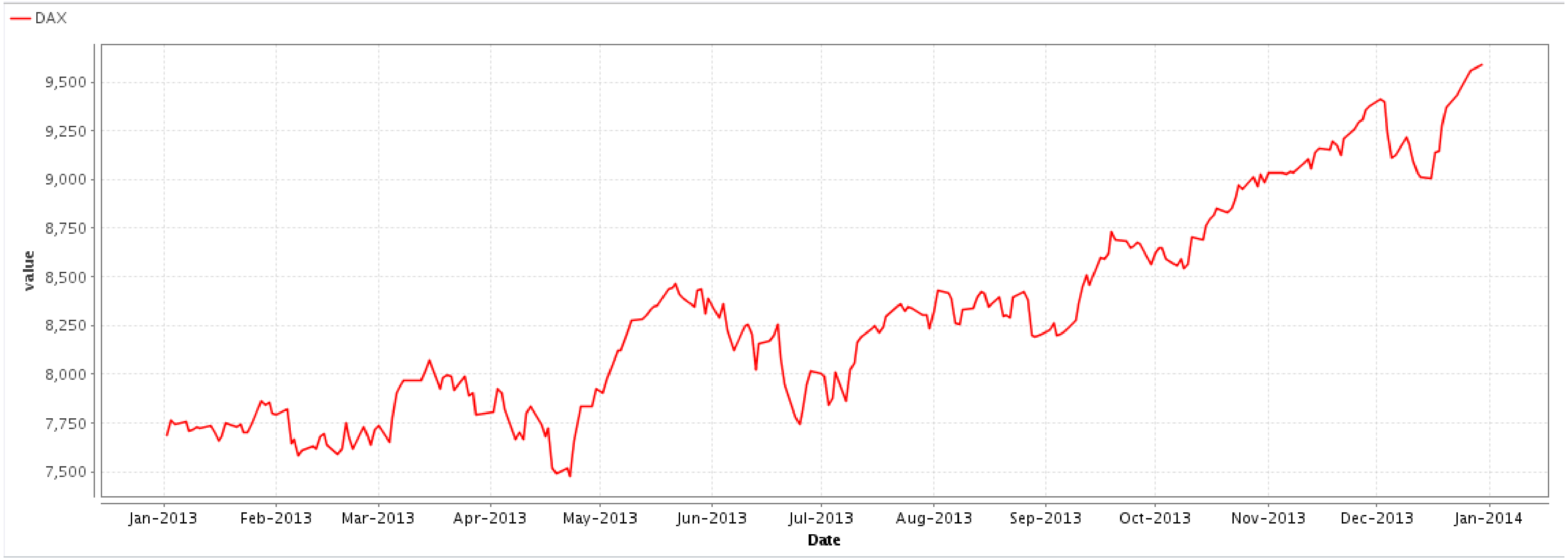
- Alternatives
  - Linear interpolation
  - Replace with previous
  - Replace with next
  - K-NN imputation
    - Essentially: this is the average of previous and next

Missing Values in Series Data

- Linear interpolation plotted

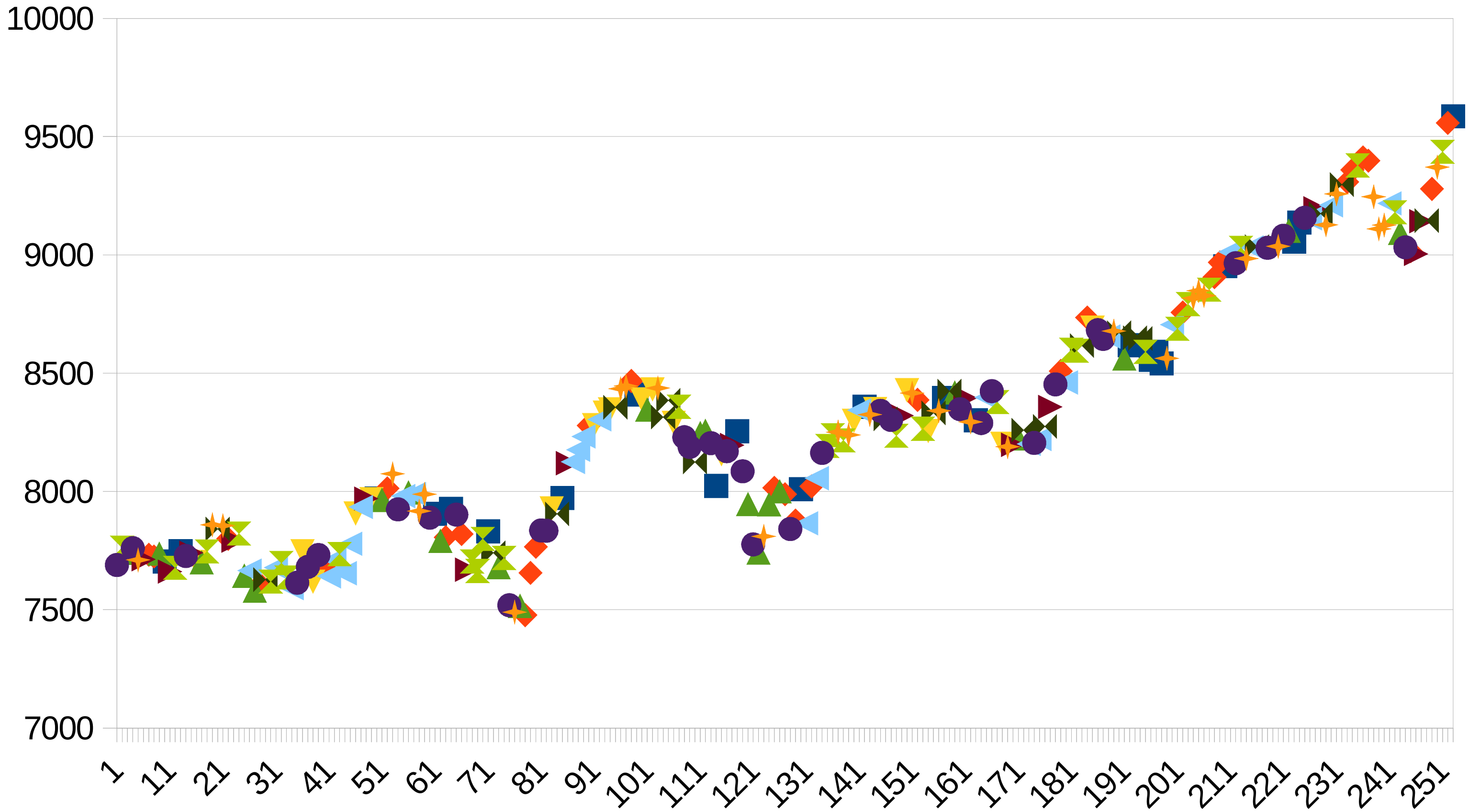
Evaluating Time Series Prediction

- So far, our gold standard has been 10-fold cross validation
  - Divide data into 10 equal shares
  - Random sampling:
    - Each data point is randomly assigned to a fold



Evaluating Time Series Prediction

- Using Cross Validation?



Evaluating Time Series Prediction

- Variant 1
  - Use hold out set *at the end* of the training data
  - E.g., train on 2000-2015, evaluate on 2016

- Variant 2:
  - Sliding window evaluation
  - E.g., train on one year, evaluate on consecutive year

Wrap-up

- Time series data is data sequentially collected at different times

- Analysis methods discussed in this lecture
  - frequent pattern mining
  - trend analysis
  - different prediction methods

Questions?