ARCADA

ARCADA UNIVERSITY OF APPLIED SCIENCES

Introduction to Analytics

Magnus.Westerlund @arcada.fi

Researcher & Programme Director
01.04.2017

Intro to Analytics - Course Schedule

- Week 1
  - 6.9: Intro to Analytics, Machine Learning, and AI
  - 7.9: Feature engineering, Pandas
- Week 2
  - 20.9: Time series processing, linear modeling and setting targets/labels
  - 21.9: Time series data visualization and regression
- Week 3
  - 4.10: External Presentation, understanding model output, and going from output to decision
  - 5.10: Open discussion, creating decisions, finalizing project

Todays Agenda

- Course project
- Time series components
- Rolling calculations in data frames
- Setup of Model

Course project

Project – Stock forecasting

- We will replicate parts of the following paper: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4873195/
- We will focus on features and determining a decision, by forecasting 1 step ahead
- Analyze data for a single stock, choose the MU symbol
- You get data from IEX in OHLC + Vol format
- Take five years of data and if you use a learning model, segment it as 80% training and 20% testing
- Implement some of the input features from the paper

Project – Stock forecasting (cont.)

- Grading (1=pass; 5=best): (grades 3-5 can be done in any order)
  - 1) Implement 2 features and visualize the price and the features in the same graph.
  - 2) Do a regression based on the features, forecast 1-day ahead.
  - 3) Plot (as a line) the regression and expected output, make the plot zoomable.
  - 4) Add 2 more features from the “Type 2” category of features presented in the paper.
  - 5) Design a decision for when to invest and when to sell based on your regression. The model can be naïve, meaning you can create a rule (if .. X .. then .. Y).
- Submission **deadline: 14.10!!**

Constructing Analytics Software

Time Series

Getting stock market data

```
import datetime as dt
import numpy as np
import pandas as pd
# https://stackoverflow.com/a/50970152
pd.core.common.is_list_like = pd.api.types.is_list_like
from pandas_datareader.data import DataReader
```

https://anaconda.org/anaconda/pandas-datareader

```
# Define timeframe of stocks we retreive
end = dt.datetime.now()
start = end - dt.timedelta(days=5*365)
```

```
# Use DataReader to get Apples stock data from IEX https://iextrading.com/developer/
# df = DataReader('AAPL','iex', start, end)
df = DataReader('AAPL','iex', start, end)
df.head(10)
```

```
5y
```

	open	high	low	close	volume
date					
2013-09-13	61.3916	61.7172	60.7847	60.8108	74578903
2013-09-16	60.3007	60.3805	58.4982	58.8776	136823442
2013-09-17	58.5950	60.1320	58.5349	59.5577	99756489
2013-09-18	60.5859	61.0005	60.2562	60.7821	113743049

Data format

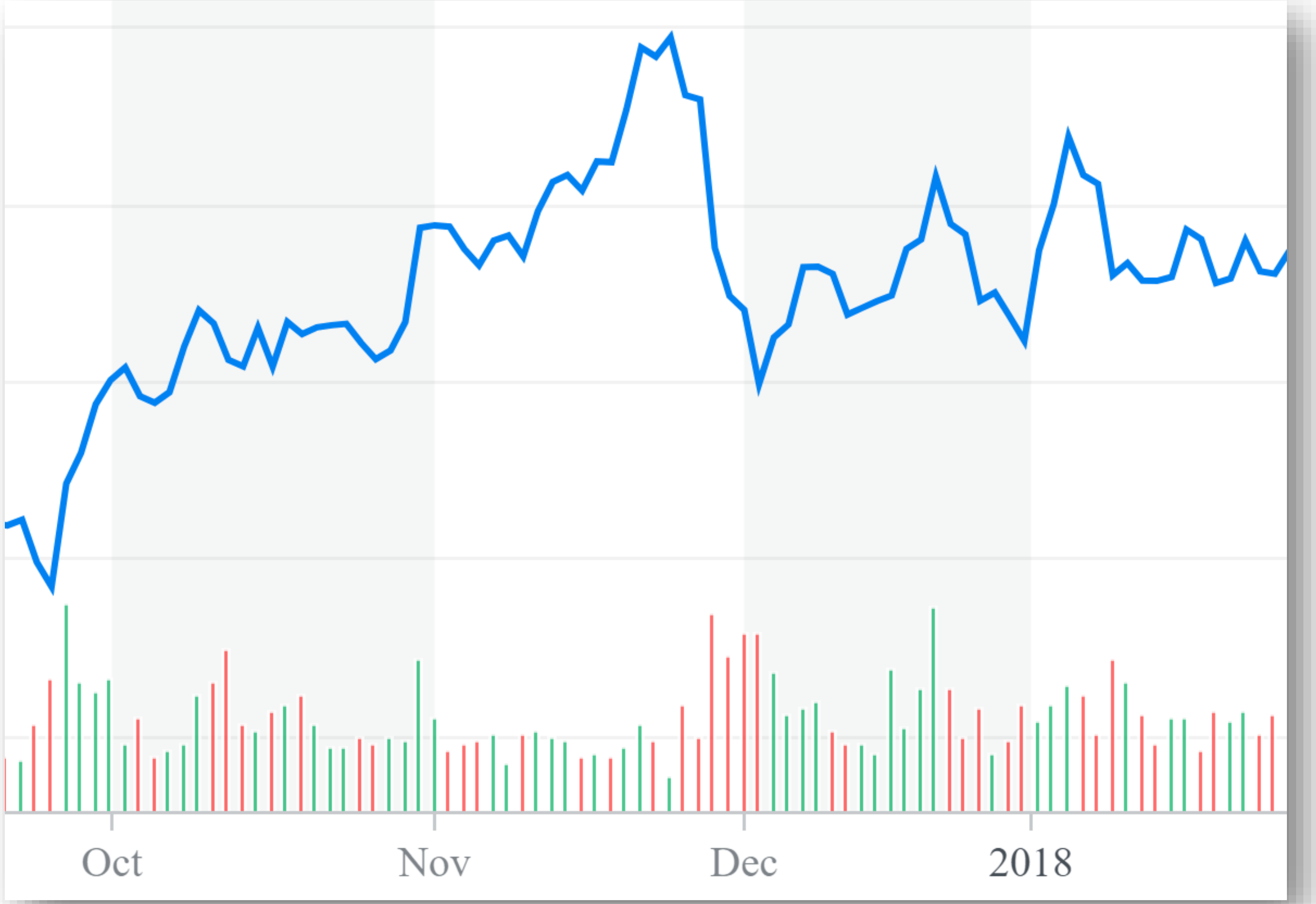
	open	high	low	close	volume
date					
2013-08-23	65.8298	65.8403	65.3170	65.5355	55587686
2013-08-26	65.5002	66.7363	65.4675	65.7906	82398085
2013-08-27	65.1405	65.7304	63.6101	63.9096	105930335
2013-08-28	63.5708	64.8527	63.5708	64.2112	76793066
2013-08-29	64.3099	64.9443	64.2418	64.3164	59807748

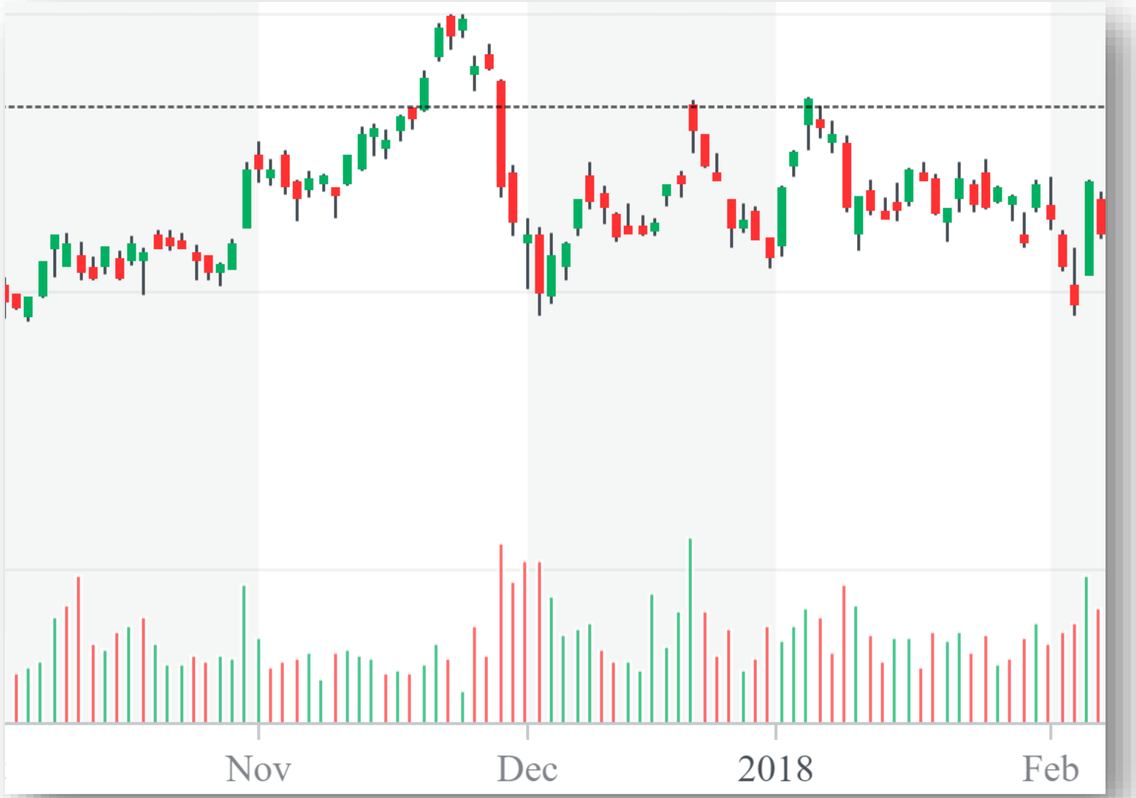
- This is End of Day data
- Each day has a record for:
  - Open price
  - Day high and low
  - Close price
  - Each day also have a volume
- Look at data statistics, run `.describe()` on your columns in your dataframe.
- Look at shape (row*col) and data,

```
print res.shape
display(res)
```

- Volume and prices are disparate and difference in size is too large.

Plotting a chart with lines or candles





https://finance.yahoo.com/quote/MU/chart?p=MU

Calculate change between days

- Simple return, up/down changes are different

$$(p_t - p_{t-1})/p_{t-1},$$

- Log return, up/down remains same

$$\log(p_t/p_{t-1})$$

- Note shift function, learn using shift!

1	
3	0.4771
5	0.2218
7	0.1461
5	-0.1461
3	-0.2218
1	-0.4771
1	0

```
import numpy as np
#df["DPC"] = np.log(df["Adj Close"].iloc[1:] / df["Adj Close"].iloc[:-1].values)
df['Log_Ret'] = np.log(df["Adj Close"] / df["Adj Close"].shift(1))
```

Exercise – Create features based on quantiles

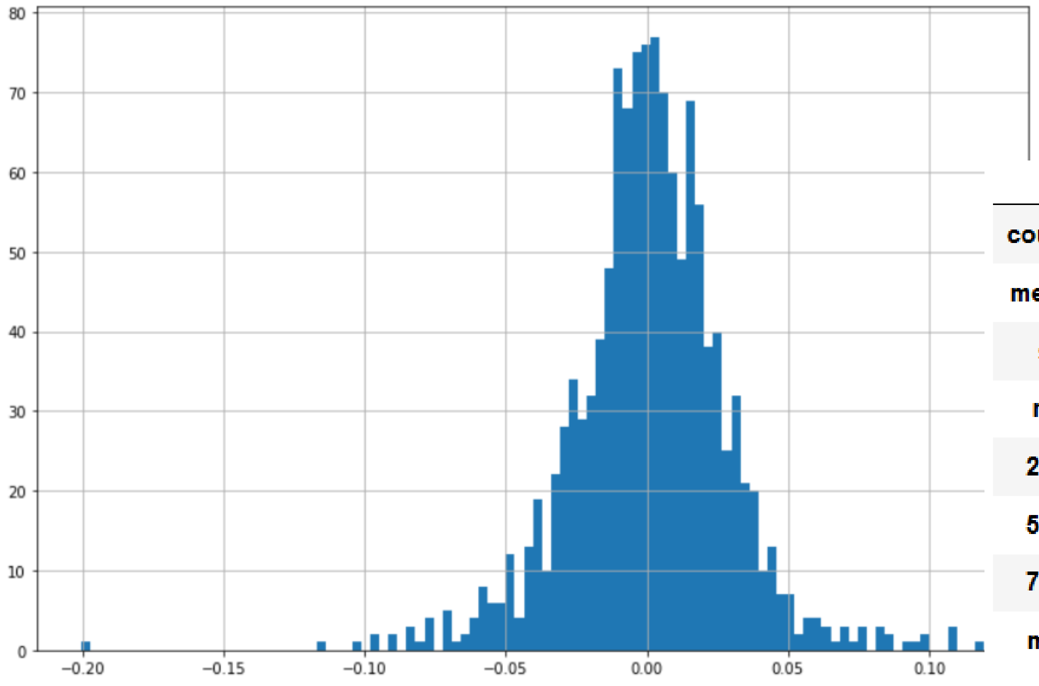
- Implement the log-return for the close prices
  - Calculate the quantiles for the log-return column
  - Filter each category of quantiles and set 1/0 in respective column

0	25	50	75	100
	Crash	Down	Up	Jump
22	1	0	0	0
70	0	0	1	0

Standardization vs. Normalization

```
print("Max value:", df["Log_Ret"].max())
print("Min value:", df["Log_Ret"].min())
df["Log_Ret"].hist(bins=100, figsize=(12,8))
plt.show()
```

```
Max value: 0.1194014208615368
Min value: -0.20030070193281696
```

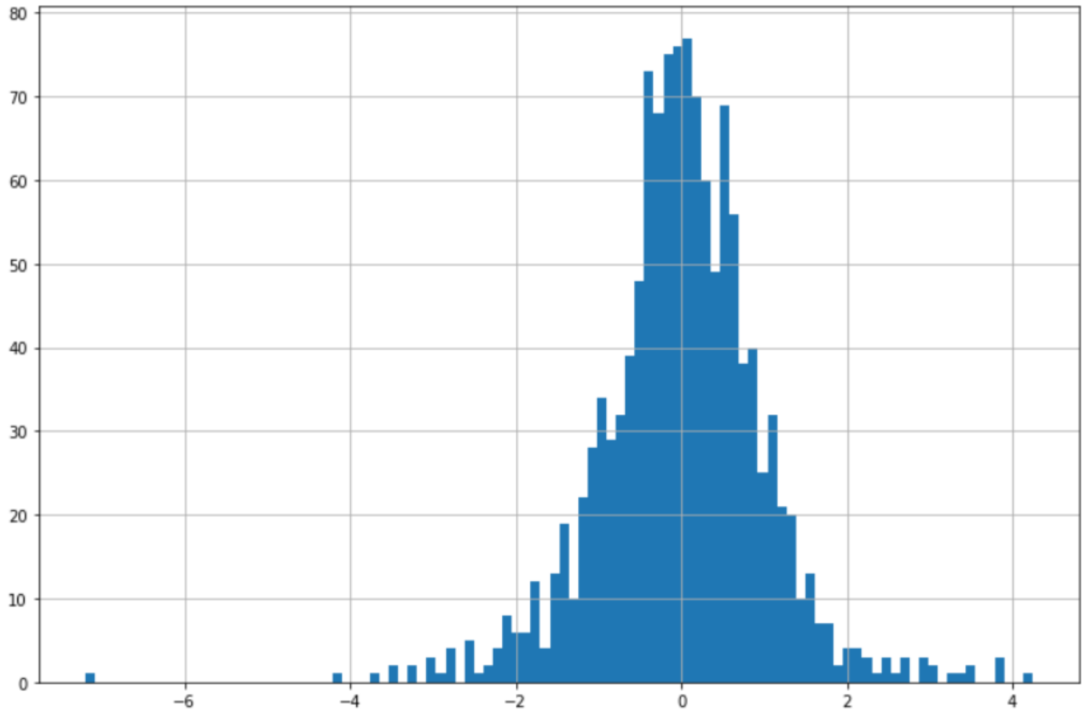



```
df["stdNorm"] = (df["Log_Ret"] - df["Log_Ret"].mean()) / (df["Log_Ret"].max() - df["Log_Ret"].min())
df["stdL_R"] = (df["Log_Ret"] - df["Log_Ret"].mean()) / (df["Log_Ret"].std())
df.describe()
```

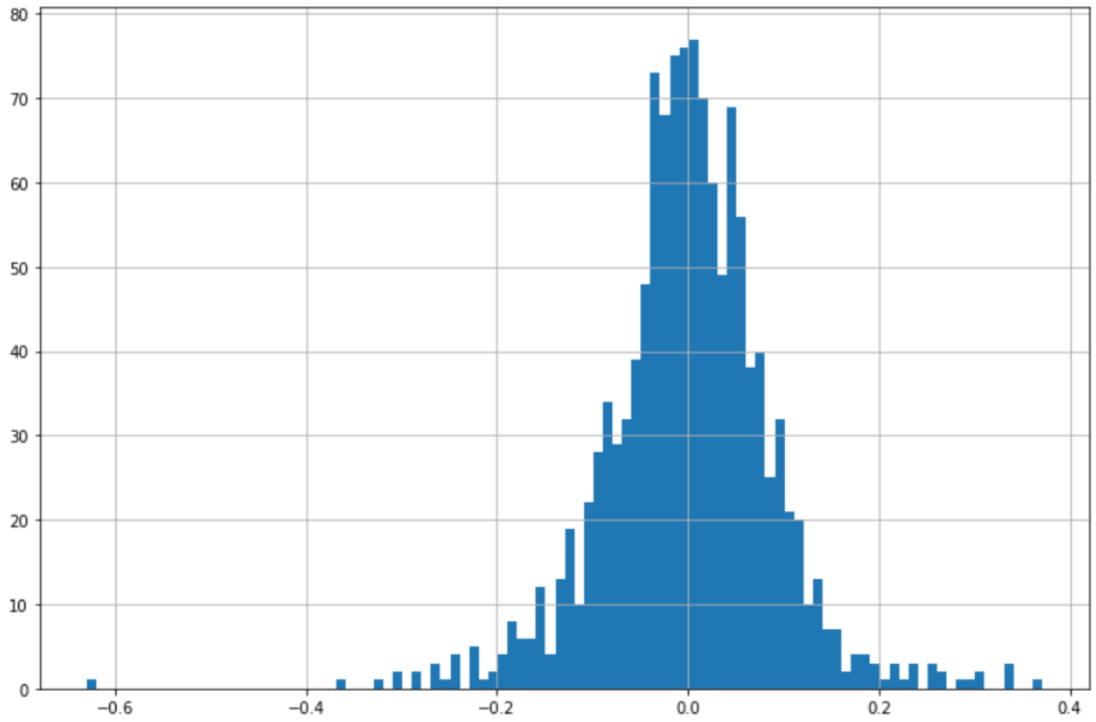
	open	high	low	close	volume	Log_Ret	stdNorm	stdL_R
count	1257.000000	1257.000000	1257.000000	1257.000000	1.257000e+03	1257.000000	1.257000e+03	1.257000e+03
mean	28.021510	28.454775	27.532495	27.996195	3.153365e+07	0.000773	-7.783485e-19	3.047151e-18
std	12.627152	12.806026	12.388175	12.608508	1.640837e+07	0.027960	8.745688e-02	1.000000e+00
min	9.610000	9.690000	9.310000	9.560000	4.672964e+06	-0.200301	-6.289412e-01	-7.191443e+00
25%	17.630000	17.900000	17.270000	17.570000	2.119826e+07	-0.013357	-4.419693e-02	-5.053568e-01
50%	26.910000	27.240000	26.470000	26.860000	2.722599e+07	0.001010	7.411741e-04	8.474737e-03
75%	33.180000	33.480000	32.770000	33.190000	3.718453e+07	0.016519	4.925101e-02	5.631462e-01
max	63.700000	64.660000	61.350000	62.620000	1.539061e+08	0.119401	3.710588e-01	4.242762e+00

Distribution

```
df["stdL_R"].hist(bins=100, figsize=(12,8))
plt.show()
```




```
df["stdNorm"].hist(bins=100, figsize=(12,8))
plt.show()
```

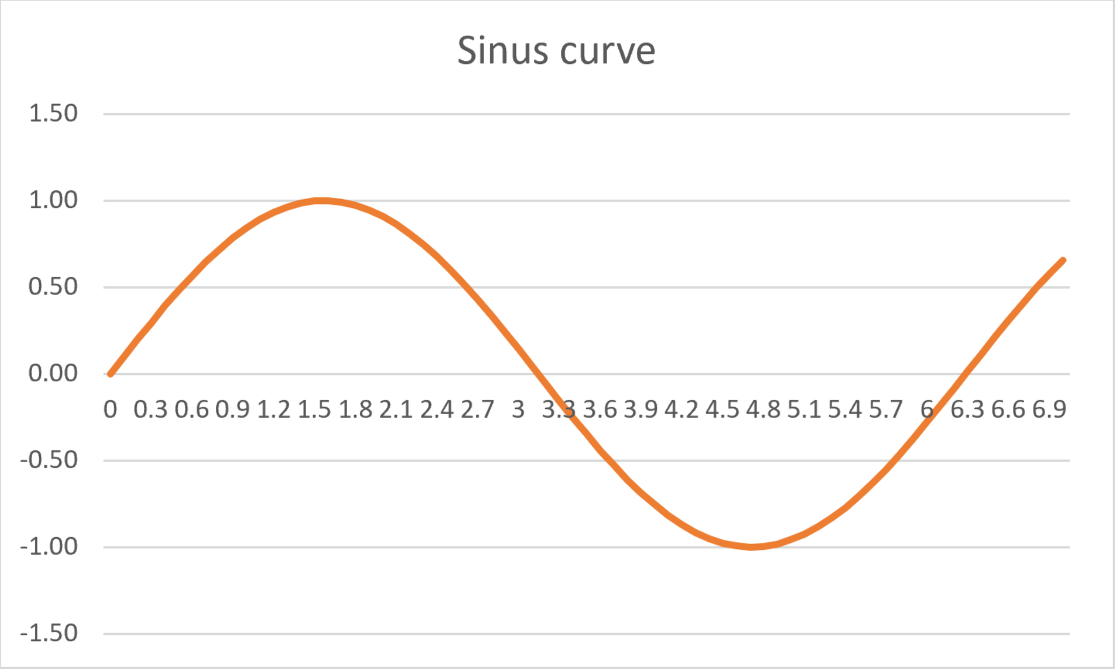


Evaluating and maintaining results; Do we trust the output?

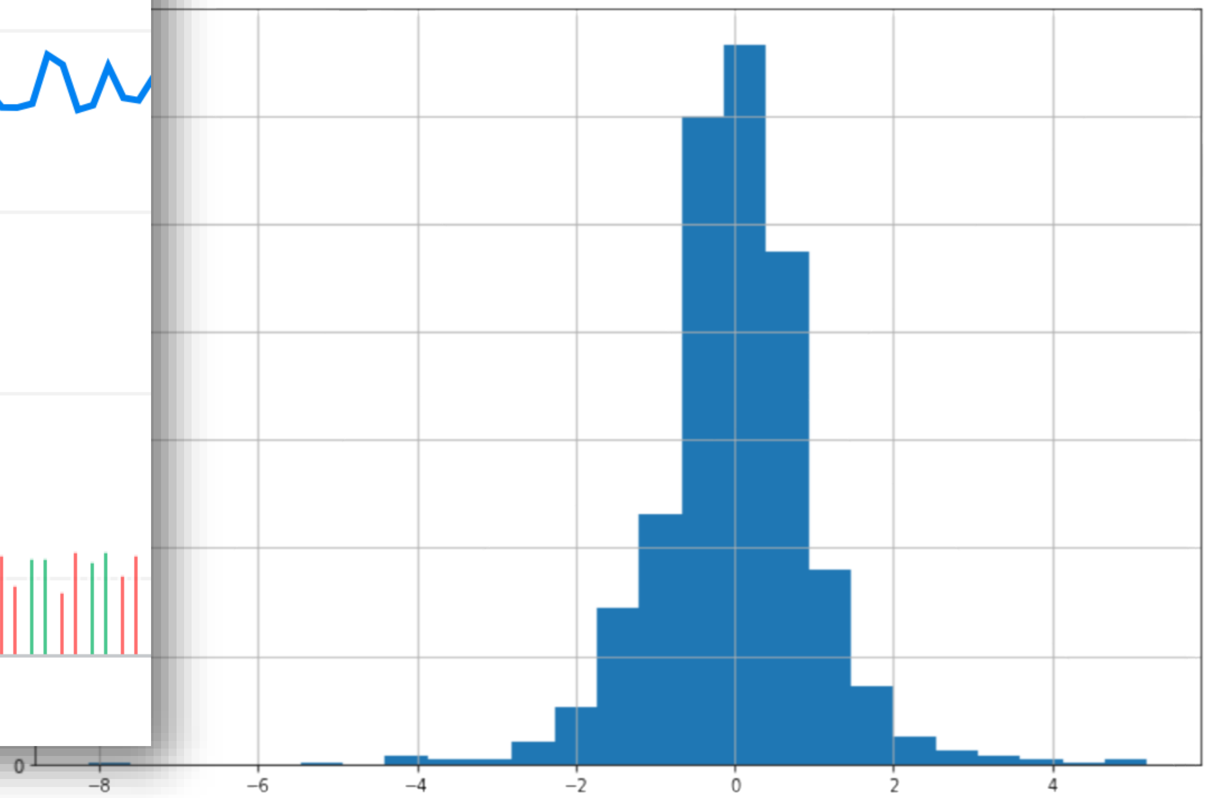
Dealing with rolling calculations in data frames

A basic time series, the sinus curve

	A	B
1	Step	=SIN(X)
2	0	0.00
3	0.1	0.10
4	0.2	0.20
5	0.3	0.30
6	0.4	0.39
7	0.5	0.48
8	0.6	0.56
9	0.7	0.64
10	0.8	0.72
11	0.9	0.78
12	1	0.84
13	1.1	0.89
14	1.2	0.93
15	1.3	0.96
16	1.4	0.99
17	1.5	1.00
18	1.6	1.00
19	1.7	0.99
20	1.8	0.97
21	1.9	0.95
22	2	0.91
23	2.1	0.86
24	2.2	0.81
25	2.3	0.75



Continuous function (price) vs Discrete function (ratio)



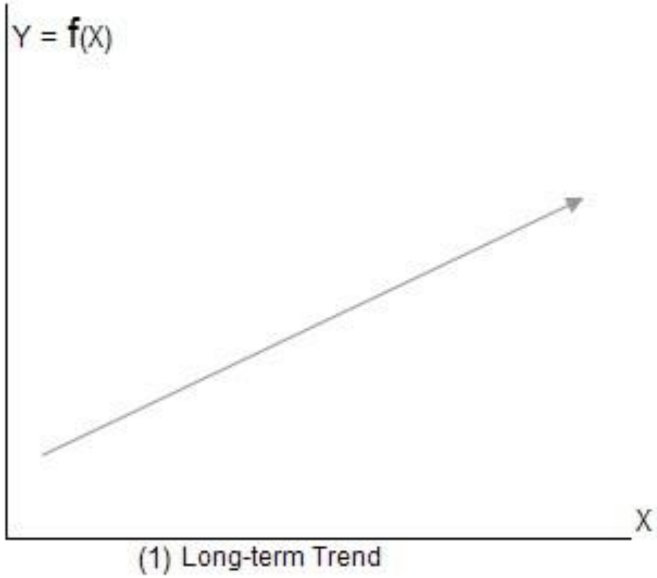
Time series components

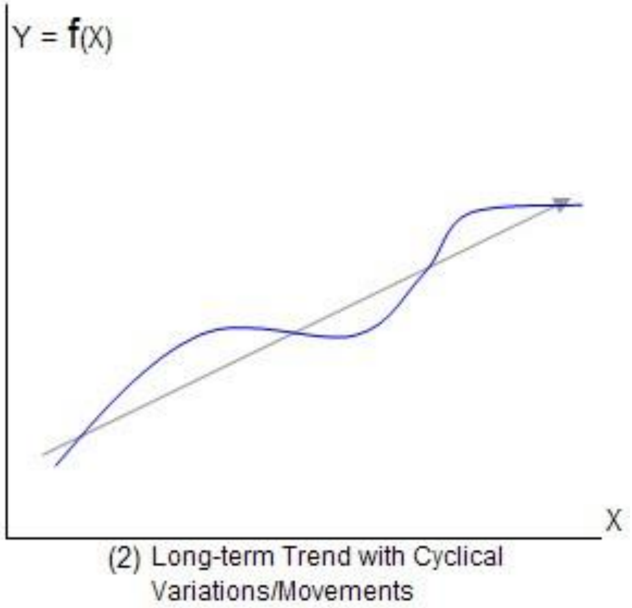
- A time series consists of base, trend, season, and residual components.
  - The base is the long-term mean of the time series while the trend represents the long-term change of the mean.
  - A season is behavior that is cyclically repeated.
  - A time series can have several seasonal components with different season lengths.
    - We refer to the base, trend, and season components as deterministic components.
  - Residuals form the stochastic component of a time series. They are unstructured information that is usually assumed to be random.
- Together, these components describe the time series model which is adopted in this work.

Feature-based Comparison and Generation of Time Series. Lars Kegel, Martin Hahmann, and Wolfgang Lehner SSDBM ’18, July 9–11, 2018, Bozen-Bolzano, Italy

Trend component

- Long term direction,
- A smoothed average that reverse direction





Seasonal component

- Systematic or
- calendar related movements

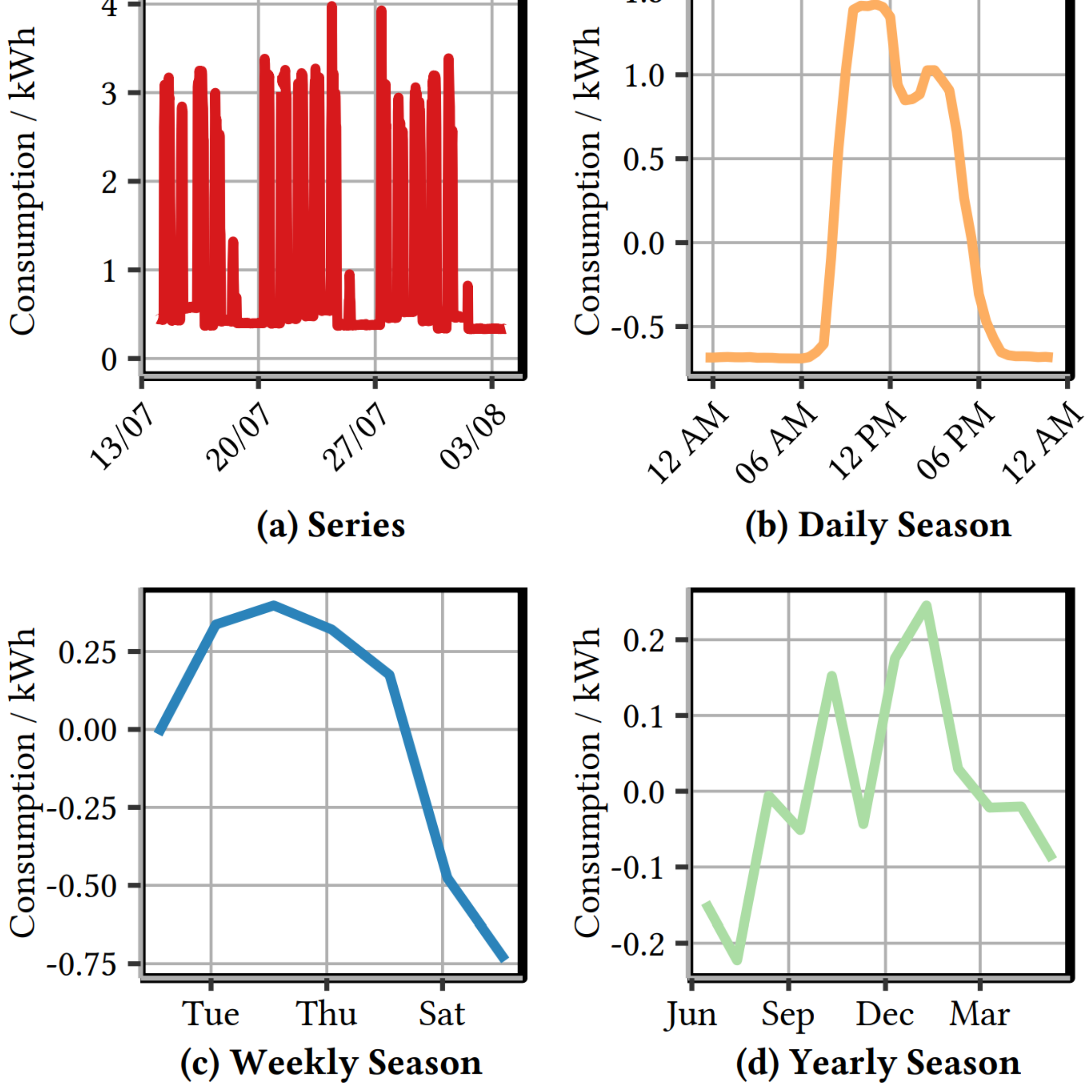
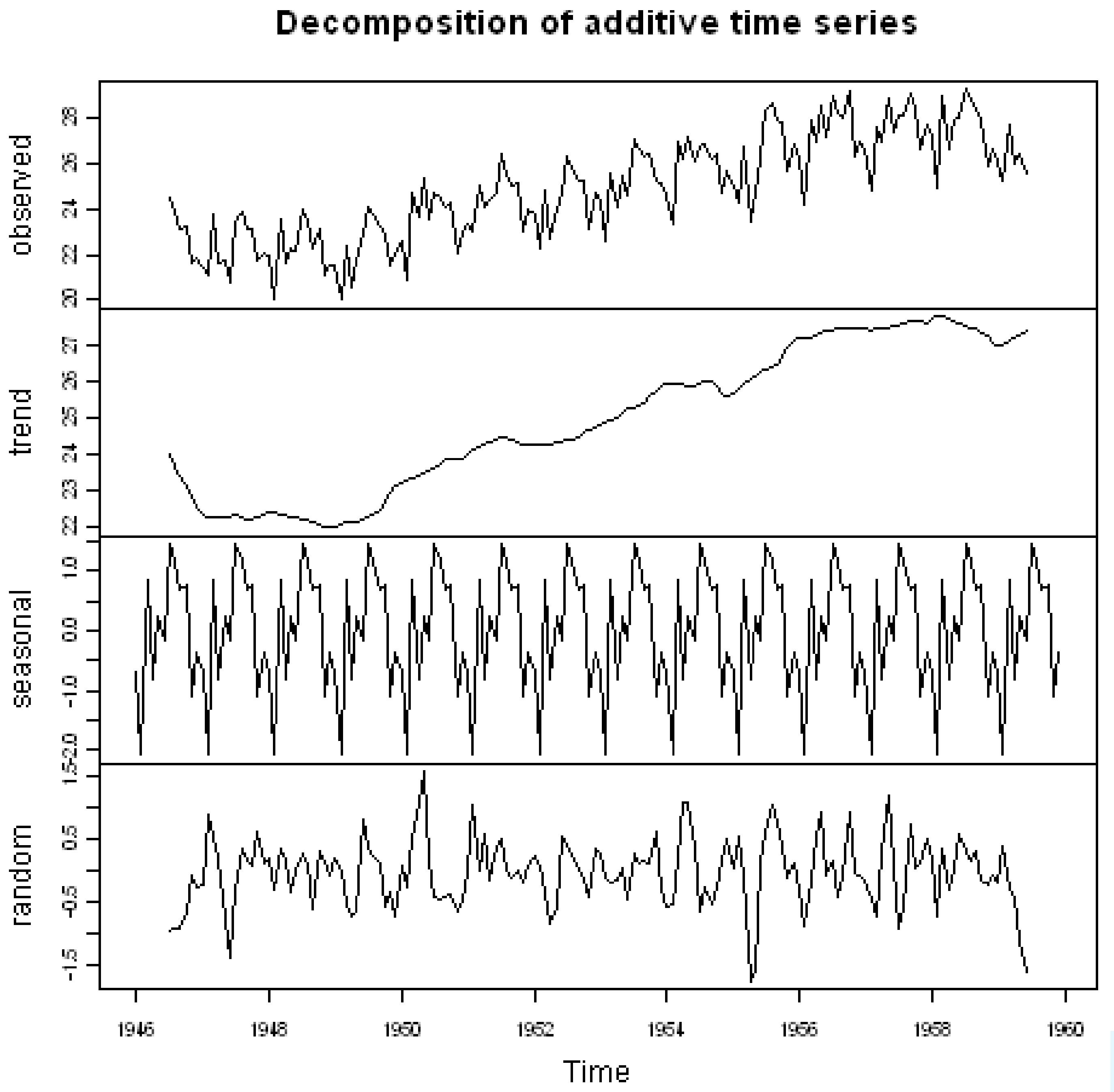


Figure 1: Multi-seasonal Decomposition

Residual component

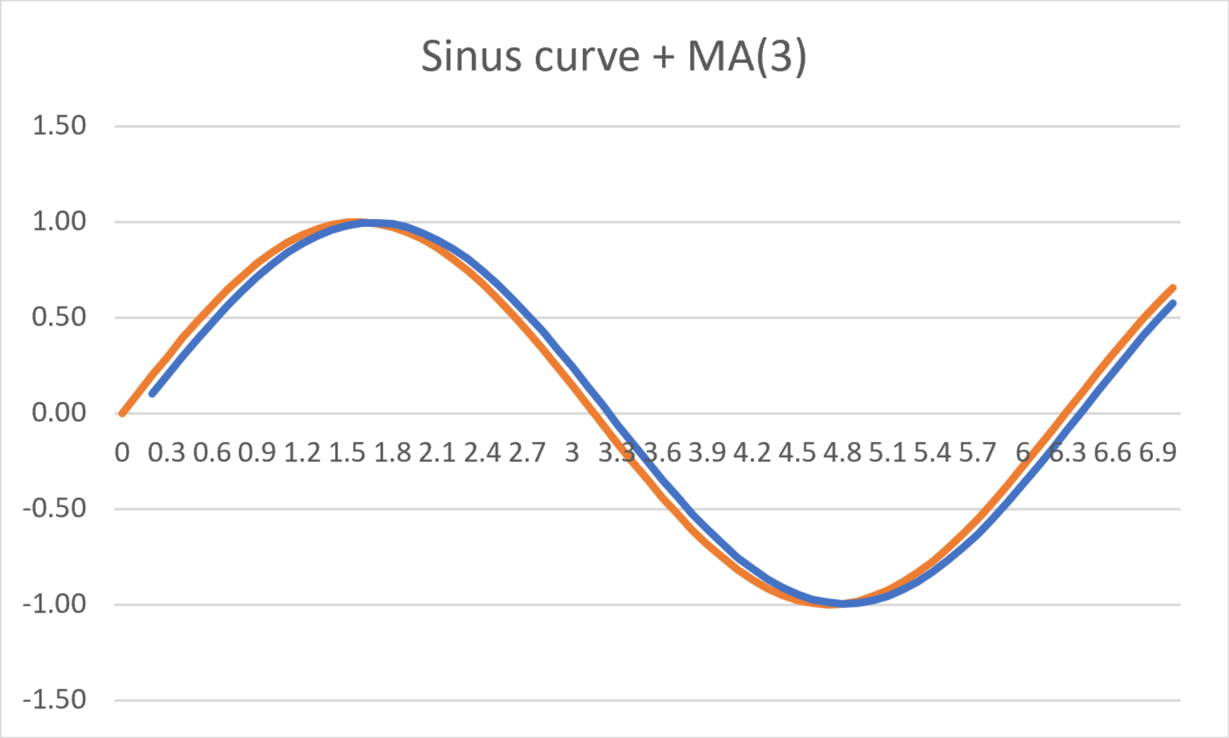
- Randomness
- Noise

- Depending on magnitude (effect size) the components are more/less observable visually.



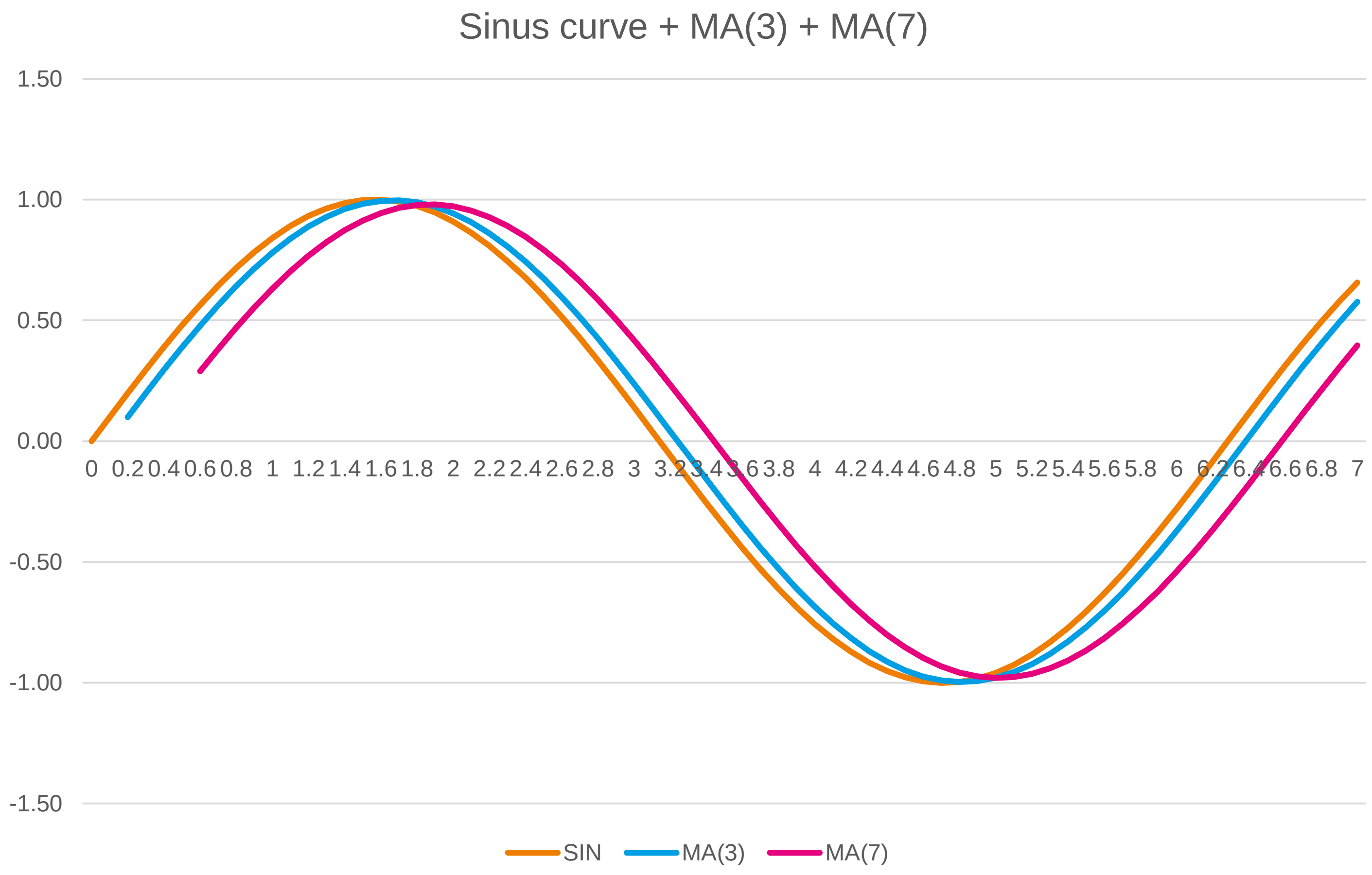
A moving average is calculated over a rolling period, note the NaN in beginning of data

	A	B	C
1	Step	=SIN(X)	MA(3)= SUM(B2:B4)/3
2	0	0.00	
3	0.1	0.10	
4	0.2	0.20	0.10
5	0.3	0.30	0.20
6	0.4	0.39	0.29
7	0.5	0.48	0.39
8	0.6	0.56	0.48
9	0.7	0.64	0.56
10	0.8	0.72	0.64
11	0.9	0.78	0.71
12	1	0.84	0.78
13	1.1	0.89	0.84
14	1.2	0.93	0.89
15	1.3	0.96	0.93
16	1.4	0.99	0.96
17	1.5	1.00	0.98
18	1.6	1.00	0.99
19	1.7	0.99	1.00



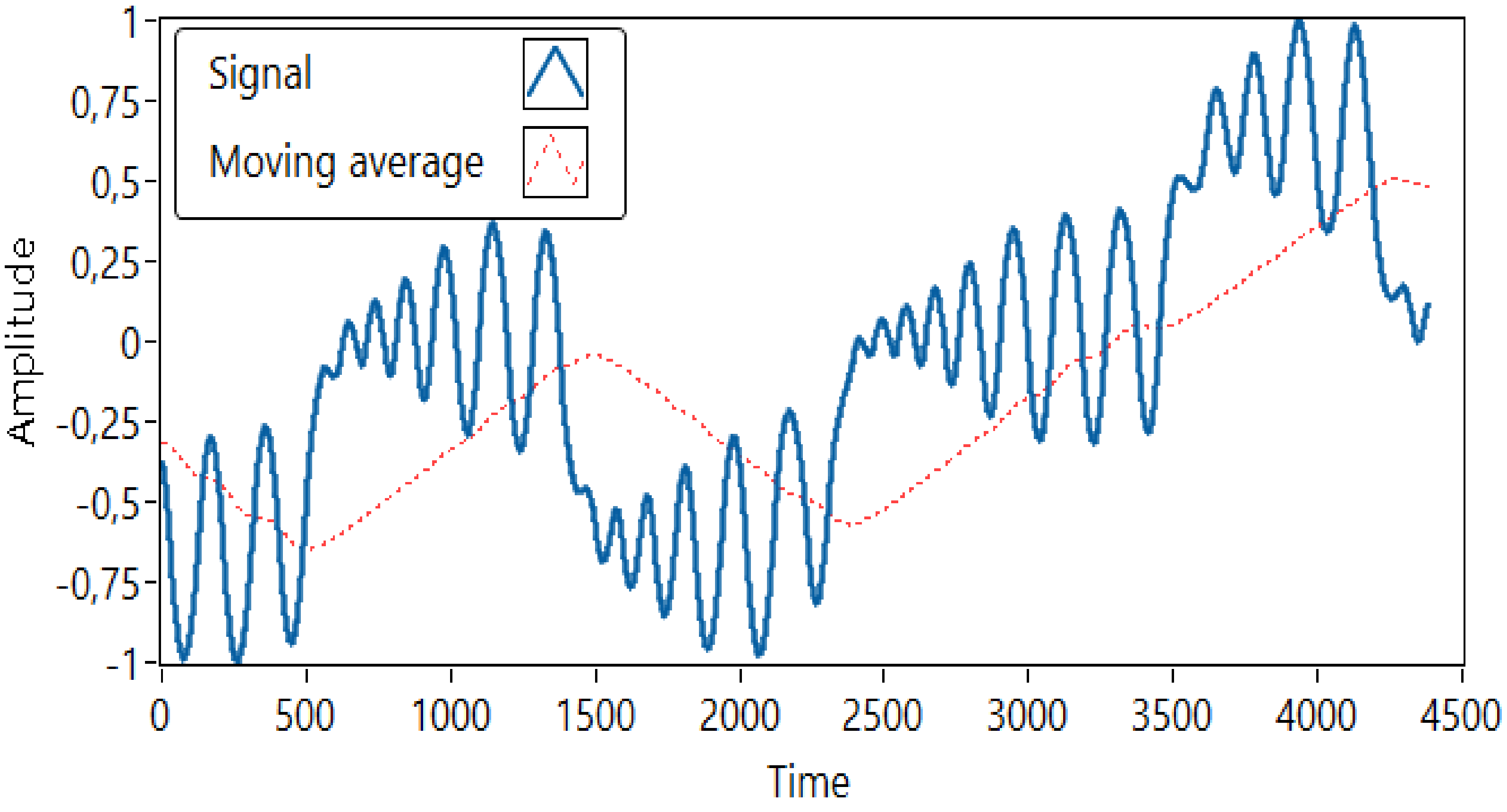
Properties of a moving average

- MA will smooth a curve the longer it is, cf. MA2 vs. MA7.
- MAs can be a naïve form of forecasting.
- MA work best when data follows trends.
- MA essentially shifts data forward.



Smoothing effect of MA

- The longer the MA, the more it smooths short term variations such as residuals
- Cf. MA(3) / MA(14)



Pandas existing rolling calculations

Use df.rolling(n).mean() or df [” x ”].rolling(n).mean()

Method	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
std()	Bessel-corrected sample standard deviation
var()	Unbiased variance
skew()	Sample skewness (3rd moment)
kurt()	Sample kurtosis (4th moment)
quantile()	Sample quantile (value at %)
apply()	Generic apply
cov()	Unbiased covariance (binary)
corr()	Correlation (binary)

Exercise – Compare 3 different MA lengths

- Plot three different MAs (3,14, 21) with close price
  - You calculate MA on the close price
- Try out e.g. Plot.ly library or make your library zoomable
- What do you find?

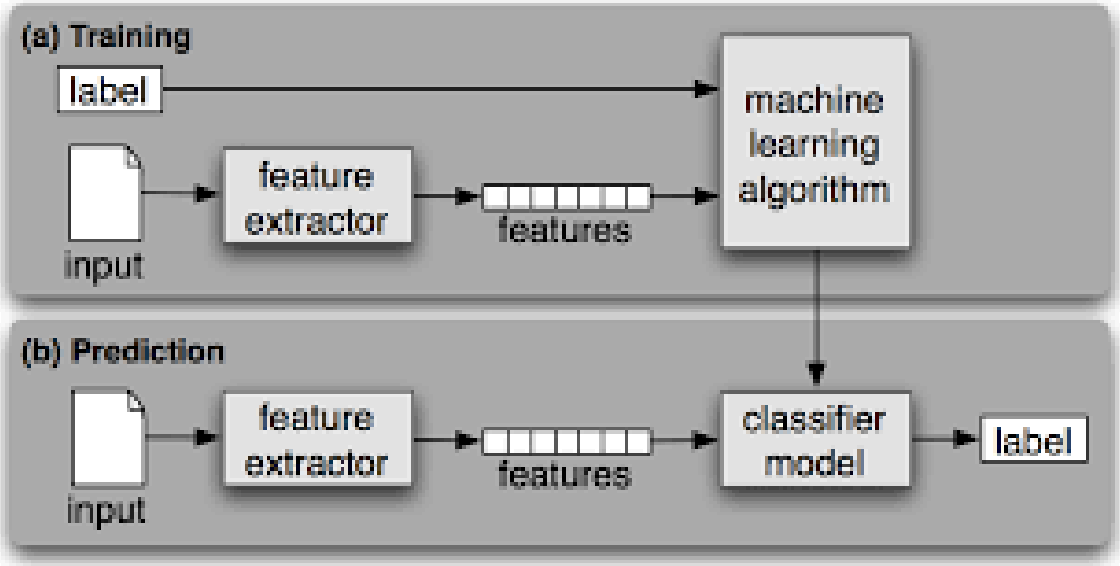
- Extra: Create new plot for MA(3, 7) on the Log Return with price

Constructing a forecast of a time series

Modeling - Regression

General idea of modeling

- Assume a distribution p(X,Y).
  - X : input
  - Y : output



- Given multiple features
  - X_i : one input feature
  - $X_{i,t}$: one input feature, at a time or index
- Given multiple outputs
  - Y_i : one output type
  - $Y_{i,t}$: one output, at a time or index

Creating labels for the expected output

- To create a regression label for the model to learn is quite easy, as you just need to shift your input data and create a label column.
- For a forecast 3-steps ahead we need to shift data backwards (depends how your data is organized) to create the correct label.
  - `df['Label'] = df["Close"].shift(-3)`
- Be careful so that this goes correct!

	Adj Close
Date	
2015-12-24	105.222536
2015-12-28	104.043982
2015-12-29	105.914084
2015-12-30	104.530989
2015-12-31	102.524526
2016-01-04	102.612183
2016-01-05	100.040792
2016-01-06	98.083025
2016-01-07	93.943473
2016-01-08	94.440222

Label
104.530989
102.524526
102.612183
100.040792
98.083025
93.943473
94.440222
NaN
NaN
NaN

A simplistic training setup

- Training a model means that you provide it with input data (train_X) and a target to reach (train_y).

- You would have many more feature columns in your input.

Input

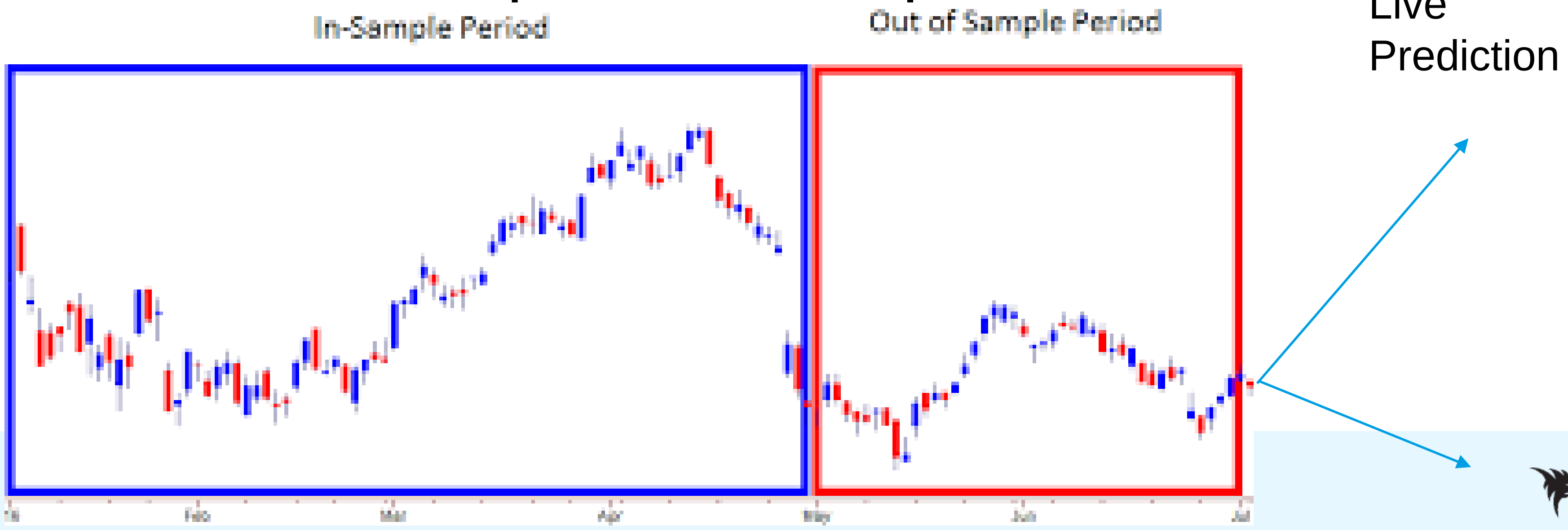
	Adj Close
Date	
2015-12-24	105.222536
2015-12-28	104.043982
2015-12-29	105.914084
2015-12-30	104.530989
2015-12-31	102.524526
2016-01-04	102.612183
2016-01-05	100.040792

Target

Label
104.530989
102.524526
102.612183
100.040792
98.083025
93.943473
94.440222

Segmenting data for Time Series

- You need to split your dataset into two parts to understand how well your model perform.
- **Train** on the In-Sample.
- **Test** model based on out-of-sample.
- The lines afterwards represents live prediction.



ARCADA

A simplistic forecast setup

- Training a model means that you provide it with input data (train_X) and a target to reach (train_y)
- To understand how well a model perform:
  - run it on **test_X**
  - measure the output (forecast) you receive from the model
  - compare forecast to the label **test_y**

	Input	Target	Output
	Adj Close	**Label**	Forecast
Date			
2015-12-24	105.222536	104.530989	?
2015-12-28	104.043982	102.524526	?
2015-12-29	105.914084	102.612183	?
2015-12-30	104.530989	100.040792	?
2015-12-31	102.524526	98.083025	?
2016-01-04	102.612183	93.943473	?
2016-01-05	100.040792	94.440222	?

Turn the data into arrays and scale data

- Two arrays hold our data
  - X = Input
  - y = Expected output

```
import numpy as np
```

```
In [10]: # X is the featureset, dont include Labels
X = np.array(df.drop(['Label'], 1))
```

```
In [11]: # y is the labels
y = np.array(df['Label'])
```

- You may want to scale data between -1 and 1
  - This approch may use future information!
  - Why?

```
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression, ElasticNetCV, Ridge
from sklearn.neural_network import MLPRegressor
```

```
# Scale values down, fit Stanard scaler to y so both X and y are using same scale
y = y.reshape(-1,1)

scaler = preprocessing.StandardScaler().fit(y)

X = scaler.transform(X)
y = scaler.transform(y)
```

Split data set into train and test

- X_train and y_train follow the same ordering index
- Same for X_test and y_test

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

```
# Docs: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

tscv = TimeSeriesSplit(n_splits=5)

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

Should give 80/20 split

Model setup

```
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression, ElasticNetCV, Ridge
from sklearn.neural_network import MLPRegressor
```

- Use a linear model, see scikit-learn for suitable regression models
- List of models in sk-learn:

  http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

- Test e.g.:
  - LinearRegression
  - Ridge
  - LogisticRegression

```
# Assign sklearn's model to a variable
# For other linear models: http://scikit-learn.org/stable/modules/linear_model.html
linear = ElasticNetCV()
```

```
# Fit or "train" the model, (reshape just to avoid error warning, works either way)
linear.fit(X_train, y_train.reshape(len(y_train),))
```

```
ElasticNetCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_intercept=True,
       l1_ratio=0.5, max_iter=1000, n_alphas=100, n_jobs=1,
       normalize=False, positive=False, precompute='auto',
       random_state=None, selection='cyclic', tol=0.0001, verbose=0)
```

http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNetCV.html

Test the model

- A simple score measure
- Run your regression forecast
- Store your forecast values in an array

```
# Score returns the coefficient of determination R^2 of the prediction
linear.score(X_test, y_test)
```

```
0.93437958984705 49
```

```
# First 5 featuresets of the testing data
X_test[:5]
```

```
array([[ 0.81049652,  0.99273542,  0.99801783],
       [ 0.99273542,  0.99801783,  0.87652508],
       [ 0.99801783,  0.87652508,  0.91922363],
       [ 0.87652508,  0.91922363,  0.9007357 ],
       [ 0.91922363,  0.9007357 ,  0.76955875]])
```

```
# .predict() uses the model to predict the values for the input
forecast_set = linear.predict(X_test)
```

```
# The first 5 predictions, compare to the featuresets above
forecast_set[:5]
```

```
array([ 0.99464965,  0.88582074,  0.91567377,  0.90093618,  0.78023303])
```

```
# Here we can see what the actual labels were for the featuresets
y_test[:5]
```

```
array([[ 0.87652508],
       [ 0.91922363],
       [ 0.9007357 ],
       [ 0.76955875],
       [ 0.73302285]])
```

Assignment 2 – Towards the project

- Set up the most basic analytics pipeline for the forecast project.
  - Use close price as input
  - Create a label 1-day ahead
  - Train your linear model
  - Create a prediction
  - Visualize prediction and price in same chart
- Grading is based on 0-5 scale, 10% of final grade

Deadline: 4.10.18