[L4-TS] Introduction to Time Series Analysis

KNIME AG

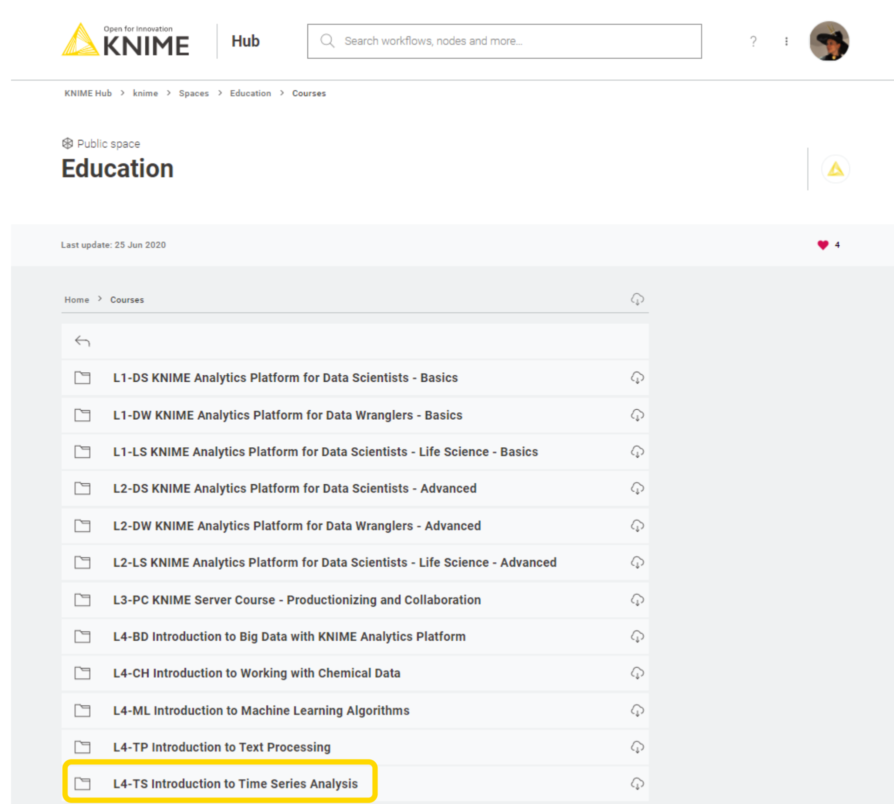
Open for Innovation
KNIME

Download your course materials from [the KNIME Hub]

Exercises – Getting Started

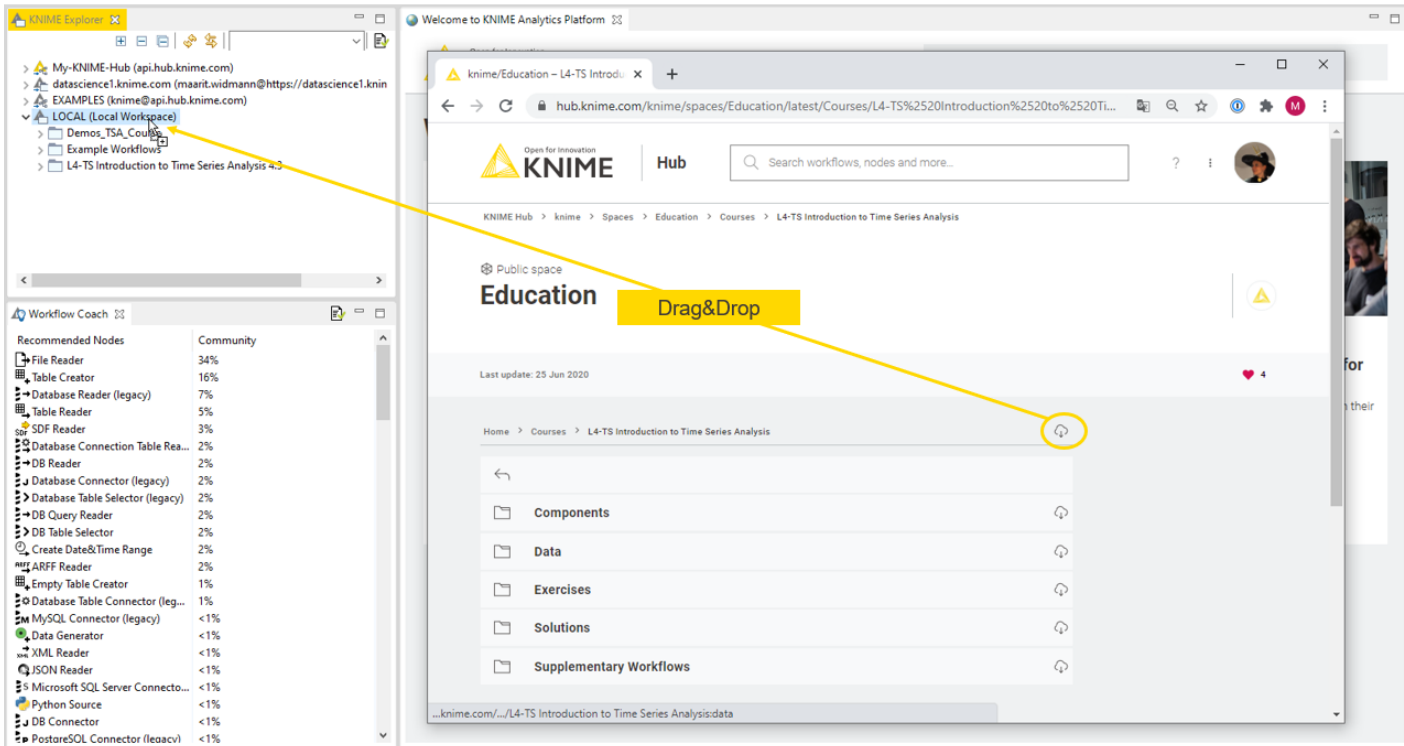
Click this KNIME Hub link to access the public Courses space on the KNIME Hub

Click L4-TS Introduction to Time Series Analysis



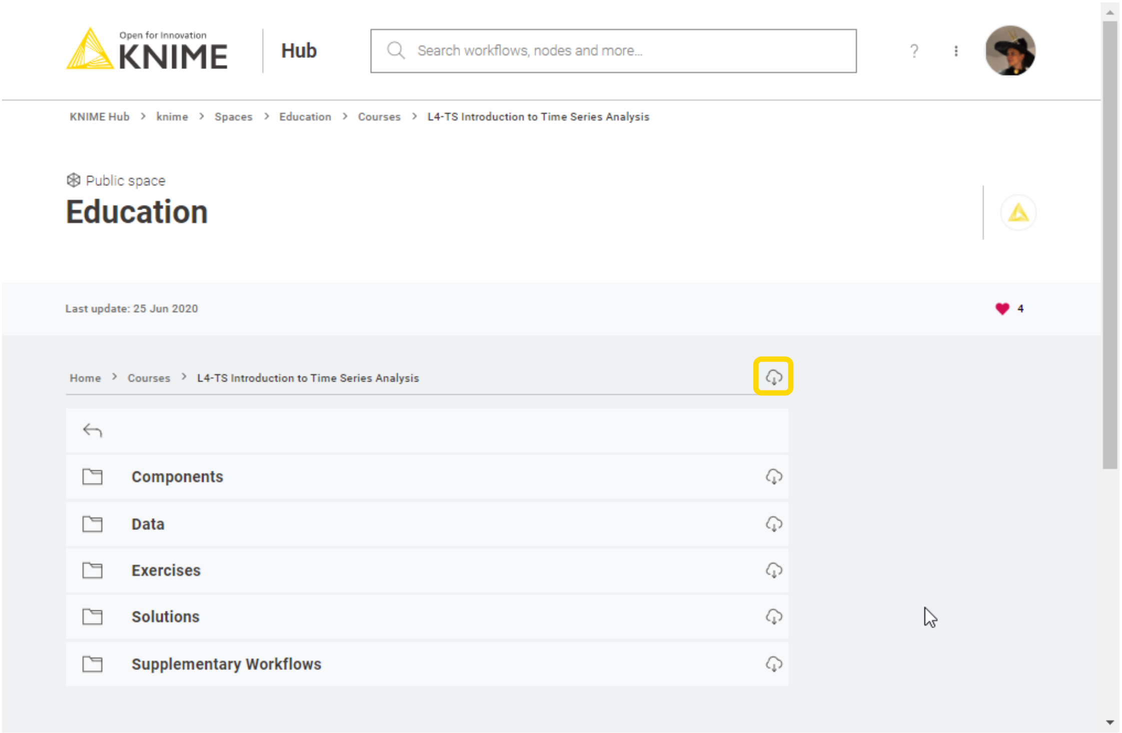
Exercises – Getting Started

Import the workflow group to your local workspace by drag and drop



Exercises – Getting Started

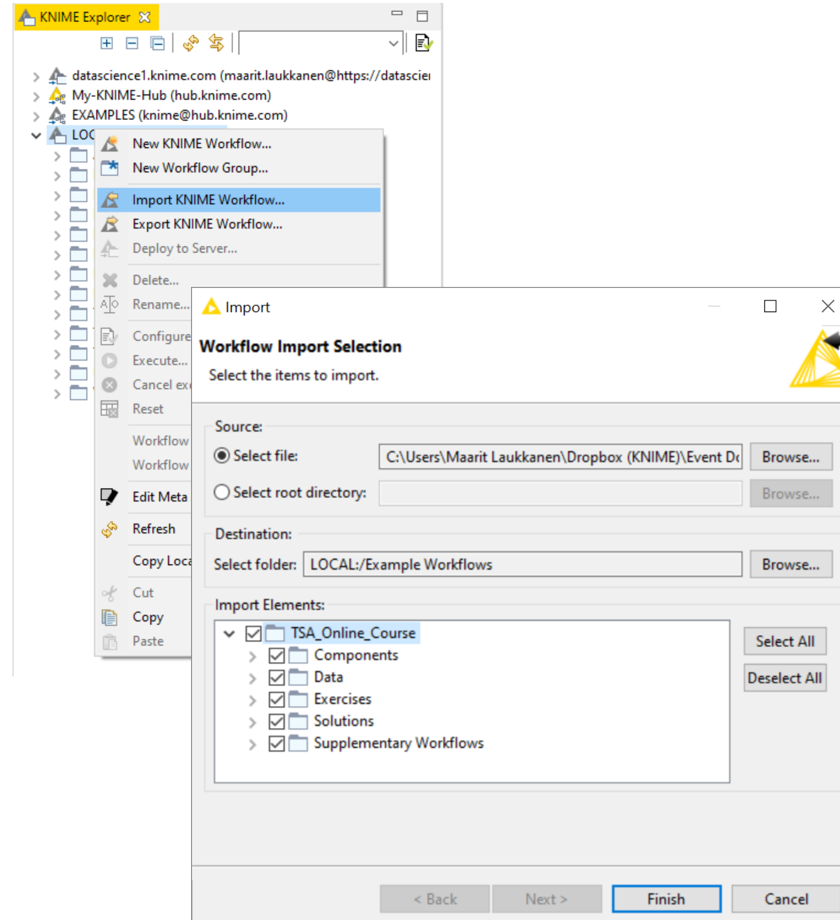
Alternatively, click the cloud icon to download the workflow group. Launch KNIME Analytics Platform.



Exercises – Getting Started

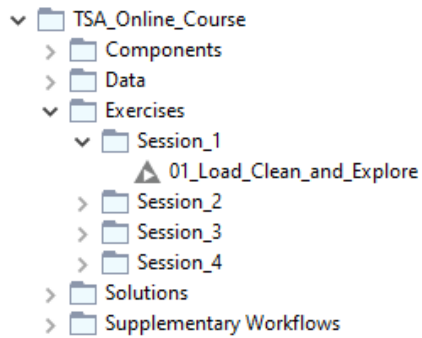
Right click LOCAL in the KNIME Explorer, and select Import KNIME Workflow…

Click Browse…, navigate to the “L4-TS Introduction to Time Series Analysis.knar” file and click Finish

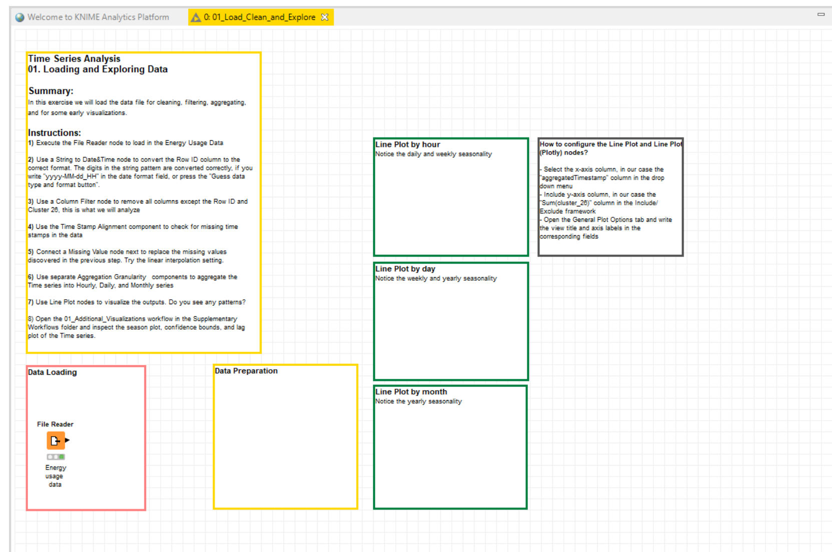


Exercises – Getting Started

Find the exercise materials in the KNIME Explorer



Double click an exercise workflow to open it. Follow the instructions.



KNIME Time Series Analysis Course - Session 1

KNIME AG

Prof. Daniele Tonini

- Daniele.Tonini@UniBocconi.it

Maarit Widmann

- Maarit.Widmann@knime.com

Corey Weisinger

- Corey.Weisinger@knime.com

Agenda

- Introduction: What is Time Series Analysis
- Today’s Task, Dataset & Components
- Descriptive Analytics: Load, Clean, Explore
- Descriptive Analytics: Non-stationarity, Seasonality, Trend
- Quantitative Forecasting: Classical techniques
- ARIMA Models: ARIMA(p,d,q)
- Machine Learning based Models
- Hyperparameter Optimization
- Quick Intro to LSTM Networks
- Example of Time Series Analysis on Spark
- Conclusions & Summary

Introduction
What is Time Series Analysis?

Introduction

Since social and economic conditions are **constantly changing over time**, data analysts must be able to **assess and predict the effects of these changes**, in order to suggest the most appropriate actions to take

- It's therefore required to use appropriate **forecasting techniques** to support business, operations, technology, research, etc.
- **More accurate** and **less biased** forecasts can be one of the most effective driver of performance in many fields

→ **Time Series Analysis**, using statistical methods, allows to enhance comprehension and predictions on any quantitative variable of interest (sales, resources, financial KPIs, logistics, sensors' measurements, etc.)

Applications

The fields of application of **Time series Analysis** are numerous: *Demand Planning* is one of the most common application, however, from industry to industry there are other possible uses. For instance:

Logistics & Transportation

- Forecasting of **shipped packages:** workforce planning

Retail grocery

- Forecasting of **sales during promotions:** optimizing warehouses

Insurance

- **Claims prediction:** determining insurance policies

Manufacturing

- **Predictive Maintenance:** improving operational efficiency

Energy & Utilities

- **Energy load forecasting:** better planning and trading strategies

TS data vs. Cross Sectional data

A Time series is made up by **dynamic data** collected over time! Consider the differences between:

1. Cross Sectional Data

- Multiple objects observed at a particular point of time
- *Examples*: customers' behavioral data at today's update, companies' account balances at the end of the last year, patients' medical records at the end of the current month, …

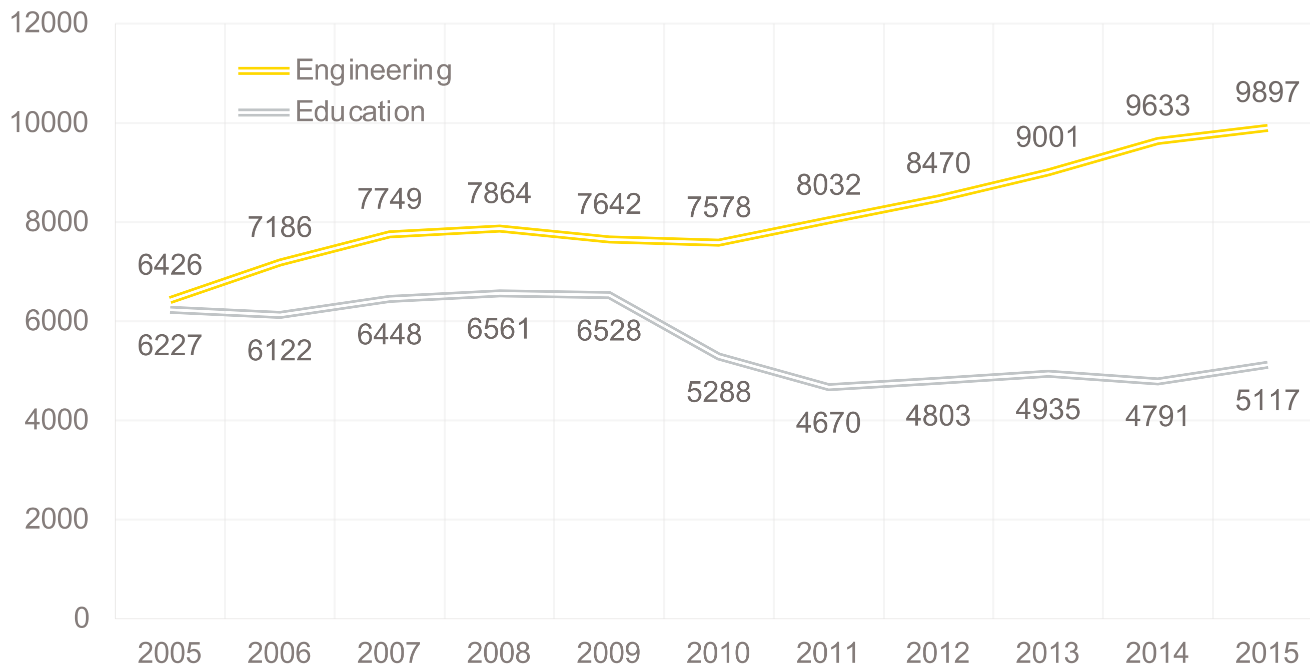
2. Time Series Data

- One single object (product, country, sensor, ..) observed over multiple equally-spaced time periods
- *Examples:* quarterly Italian GDP of the last 10 years, weekly supermarket sales of the previous year, yesterday's hourly temperature measurements, …

Examples

Time series example 1

Numbers of Doctorates Awarded in US, annual data – Engineering Vs. Education



At a glance

Annual data

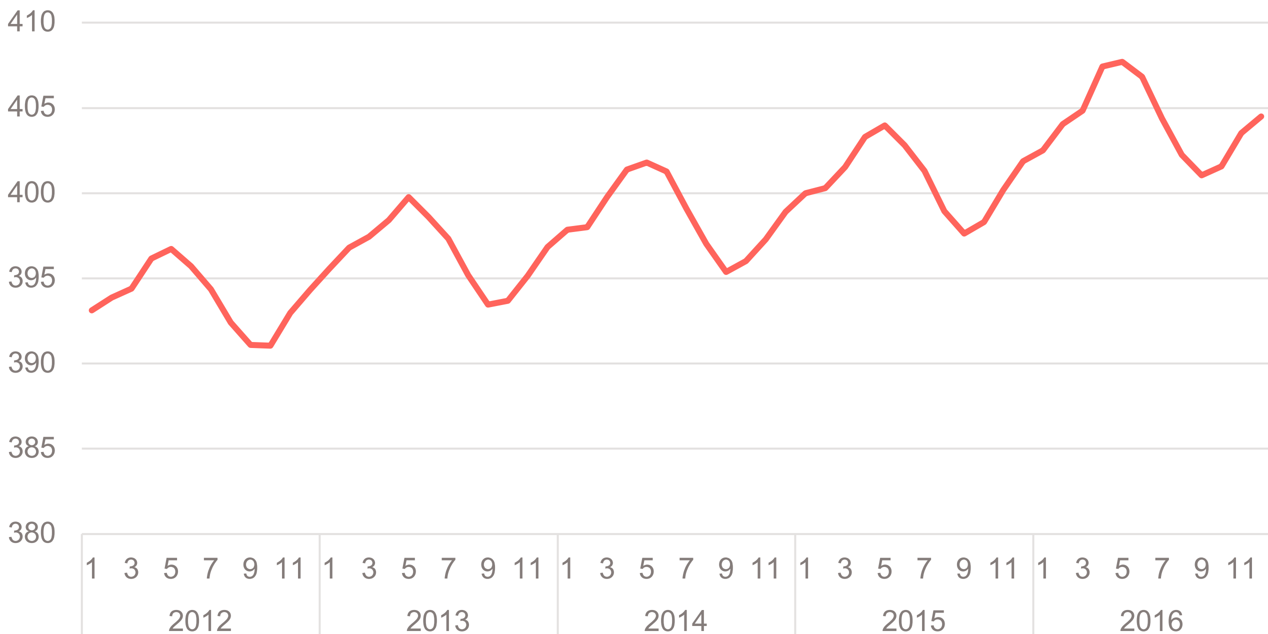
Different «directions»

No big fluctuations

Examples

Time series example 2

Monthly carbon dioxide concentration (globally averaged over marine surface sites)



At a glance

Monthly basis data

Regular pattern

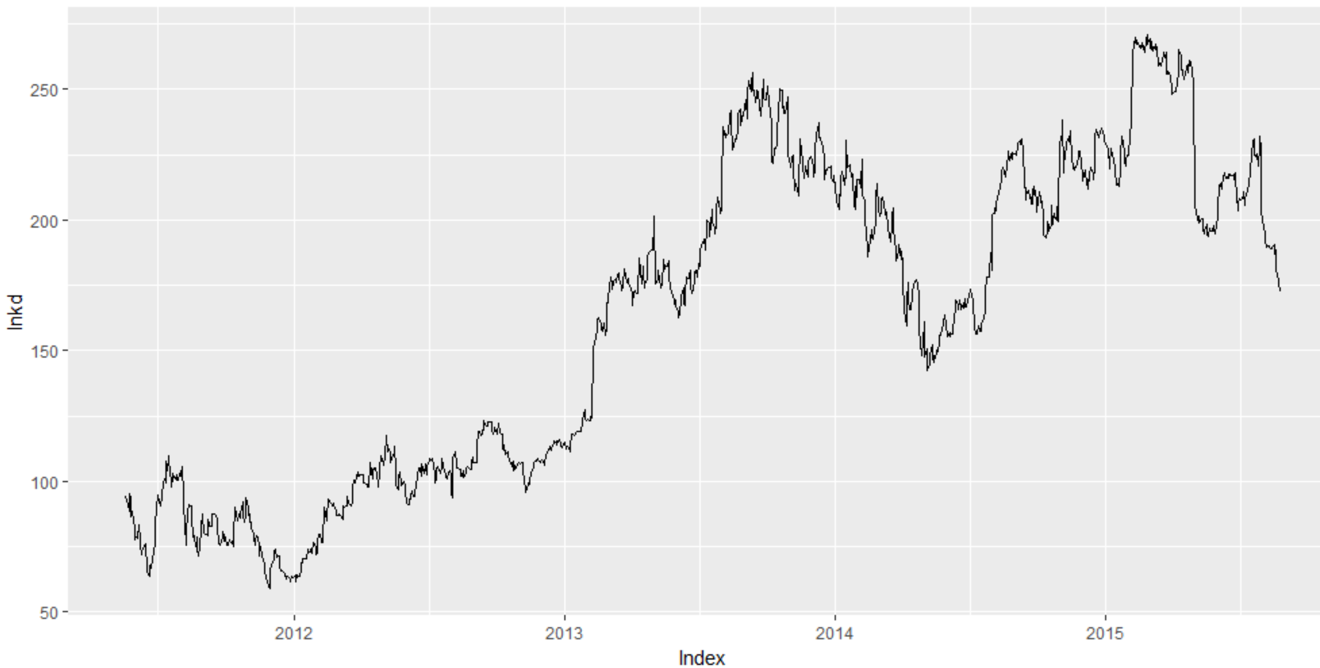
Constant fluctuations

Average value increases year by year

Examples

Time series example 3

LinkedIn daily stock market closing price



At a glance

Daily basis data

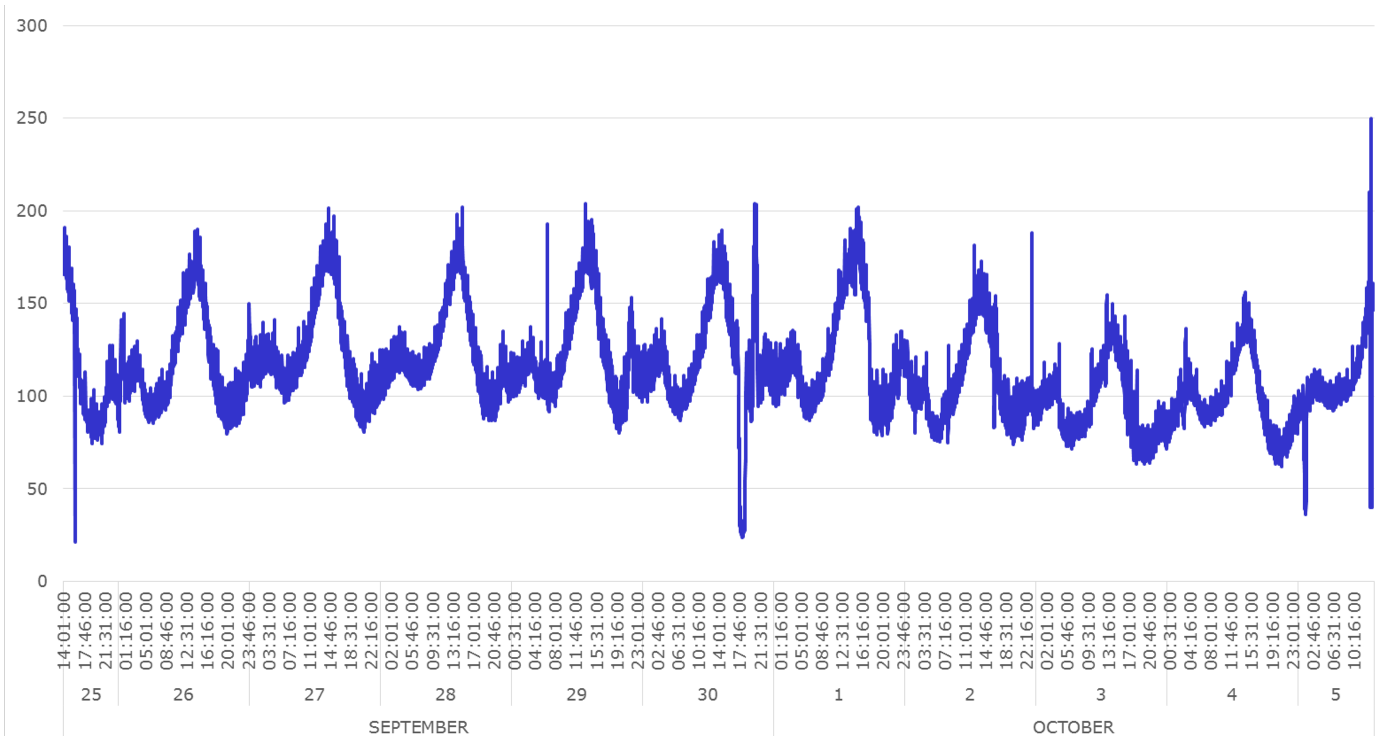
Very irregular dynamic

Many sudden changes

Examples

Time series example 4

Number of photos uploaded on the Instagram every minute (regional sub-sample)



At a glance

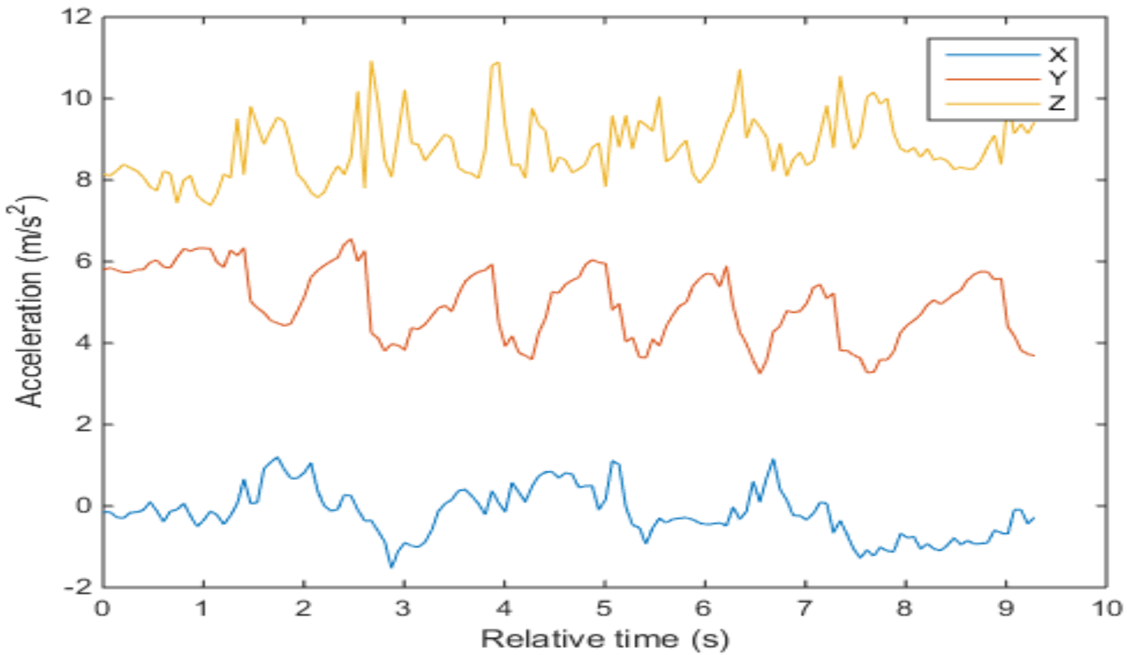
Minute basis data

Almost regular daily pattern but with some anomalies and spikes

Examples

Time series example 5

Acceleration detected by a smartphone sensors during a workout session (10 seconds)



At a glance

Milliseconds basis data

Each sensor has its own dynamics

Objectives

Main Objectives of Time Series Analysis

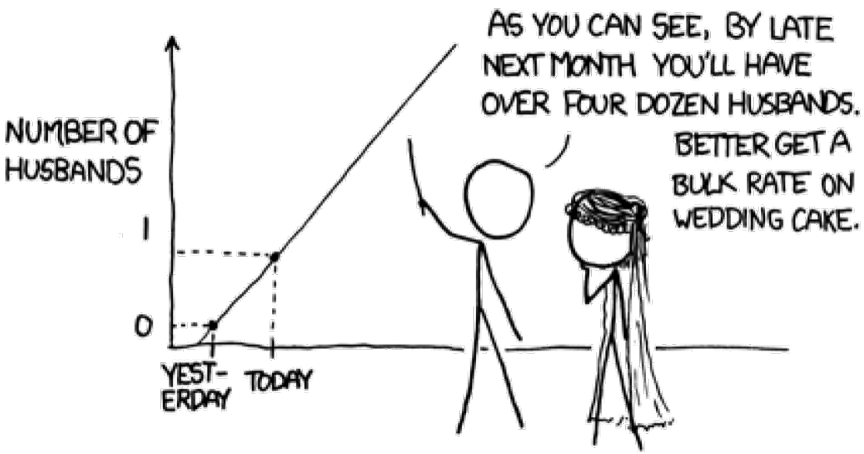
- **Summary description** (graphical and numerical) of data point vs. time
- **Interpretation** of specific series features (e.g. seasonality, trend, relationship with other series)
- **Forecasting** (e.g. predict the series values in $t+1, t+2, \dots, t+k$)
- **Hypothesis testing and Simulation** (comparing different scenarios)

Objectives

Once someone said: ***«Forecasting is the art of saying what will happen in the future and then explaining why it didn't»***

- Frequently true... history is full of examples of «bad forecasts», just like IBM Chairman's famous quote in 1943: "*there is a world market for maybe five computers in the future.*"

The reality is that forecasting is a really tough task, and you can do really bad, just like in this cartoon..



But we can do definitely better using **quantitative methods**.. and **common sense**!

GOAL: Reduce uncertainty and improve the accuracy of our forecasts

Definition

General definition: "A time series is a collection of observations made sequentially through time, whose dynamics is often characterized by short/long period fluctuations (seasonality and cycles) and/or long period direction (trend)"

Such observations may be denoted by $Y_1, Y_2, Y_3, \ldots Y_t, \ldots, Y_T$ since data are usually collected at discrete points in time

Observation at time t

- The interval between observations can be any time interval (seconds, minute, hours, days, weeks, months, quarters, years, etc.) and we assume that these time periods are **equally spaced**
- One of the most distinctive characteristics of a time series is the mutual dependence between the observations, generally called **SERIAL CORRELATION** OR **AUTOCORRELATION**

Task & Dataset
Electricity Consumption by the Hour in Ireland

The Dataset: Electricity Consumption

Smart Meters to measure Electricity Usage

Task: Demand Prediction of kW used in the next hour

- Irish Smart Energy Trials
  - http://www.seai.ie/News_Events/Press_Releases/2012/Full_Data_from_National_Smart_Meter_Trial_Published.html
- 6000 households & businesses From Jul 2009 to Aug 2010

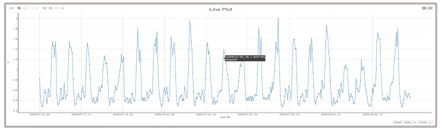
- The original Dataset
  - ID by household/store in Ireland
  - Date&Time (Jul 2009 - Aug 2010)
  - kW used in the past half an hour

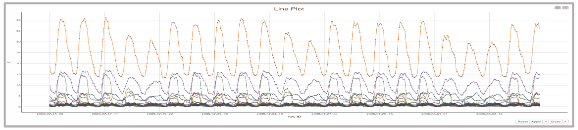
Task: Electricity Demand Prediction

A few, a bit fat, Time Series of kW used by **similar*** households in Ireland

One big fat Time Series of kW used every hour in the whole Ireland

Many fine Time Series of kW used every hour at each household in Ireland



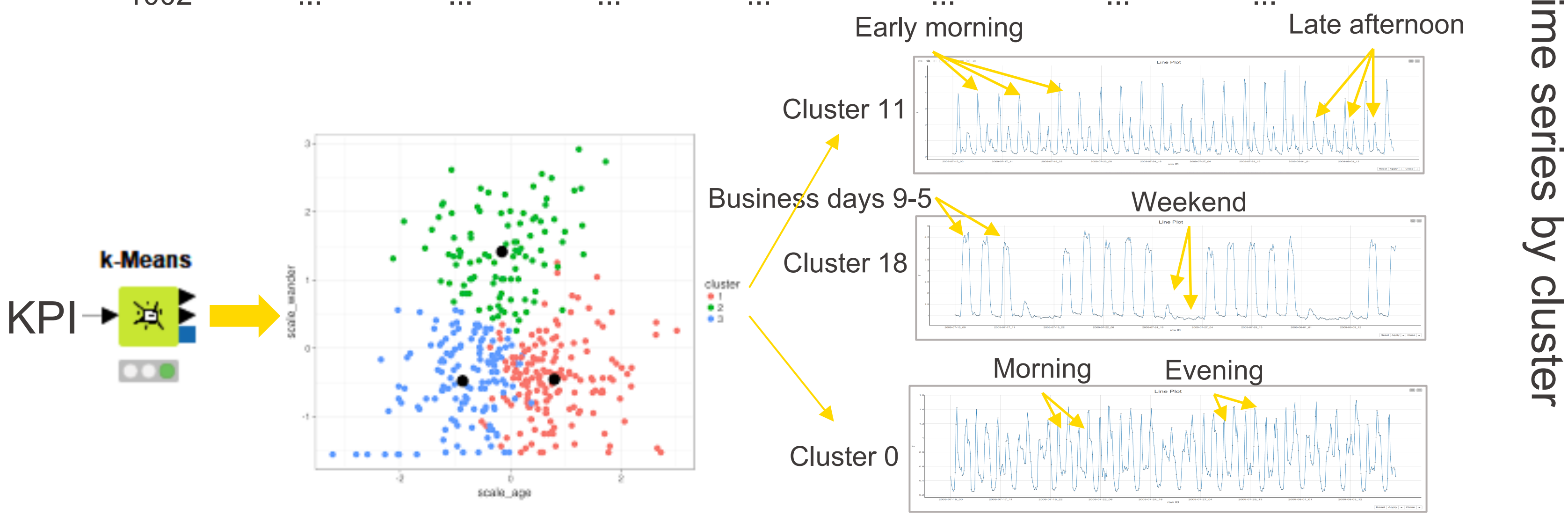


*Similar electrically speaking

Data Processing: daily & weekly KPI

Daily KPI = % of kW used in <time window> over total kW in day

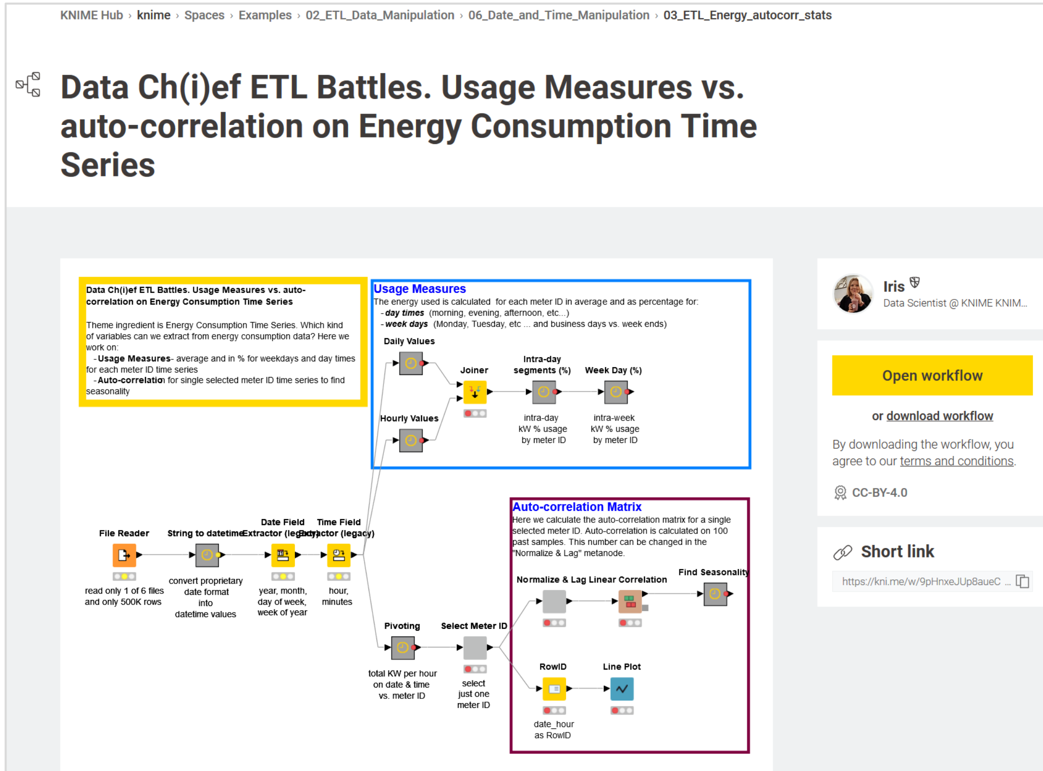
Meter ID	Early morning 06-09	Morning 09-12	Lunch 12-15	Early Afternoon 15-18	Late Afternoon 18-21	Evening 21-24	Night 24-06
1000	6	2	3	3	7	28	51
1001	5	22	16	24	23	5	5
1002	...	...	...	...	...	...	...



Clustering Energy Consumption Data

- Clustering in order to identify fewer groups with a particular behavior instead of inspecting and modeling many individual behaviors. We model the energy consumption of a prototype of one cluster.
- Clustering by k-Means algorithm based on
  - energy consumption on business days and over the weekend
  - total energy consumption
  - yearly, monthly, weekly, etc. energy consumption
- k-Means Algorithm
  - Based on Euclidean distance of numeric columns, and our data only contain numeric columns
  - Missing values need to be replaced, in our data by 0

Electricity KPIs on the KNIME Hub

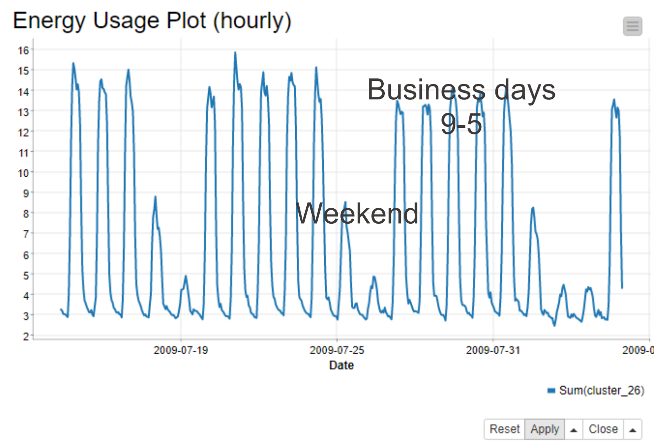


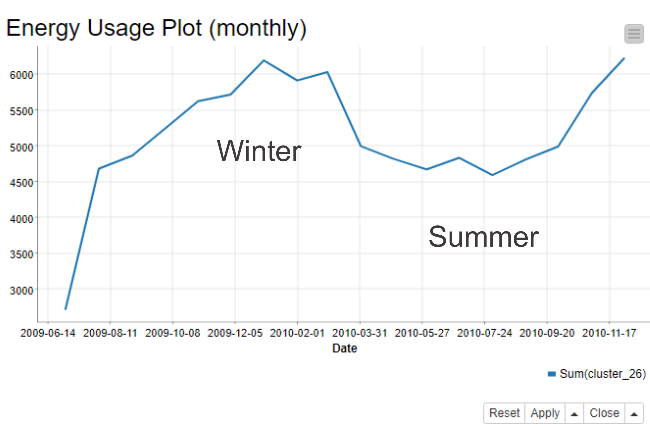
https://kni.me/w/9pHnxeJUp8aueCJT

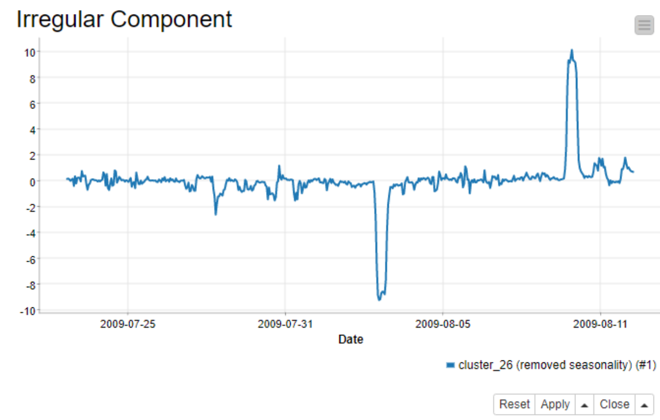
This Week‘s Challenge

- Isolate, preprocess, and visualize data in cluster 26
- Apply several techniques (e.g. Random Forest, ARIMA, LSTM) to generate in-sample and out-of-sample forecasts
- Evaluate the models and save them for forecasting comparison
- Compare the accuracy of the models in predicting Electricity Usage in **cluster 26** in kW in the next week (168 hours)

Today's Challenge – Cluster 26

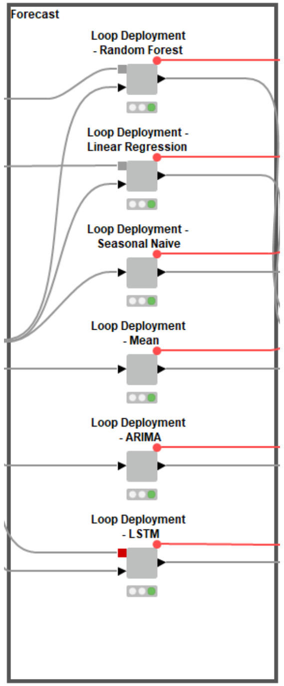


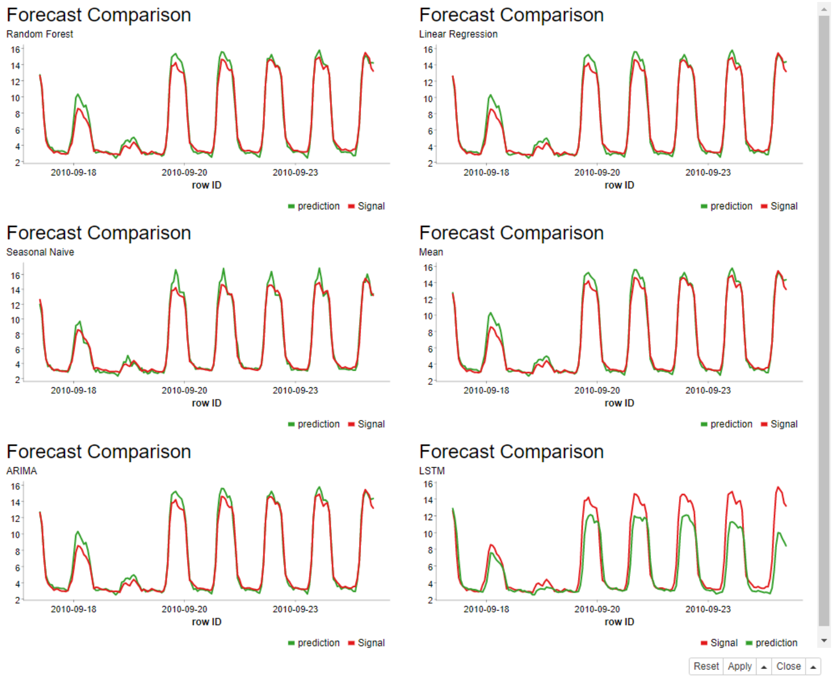




This Week's Challenge – Final Results

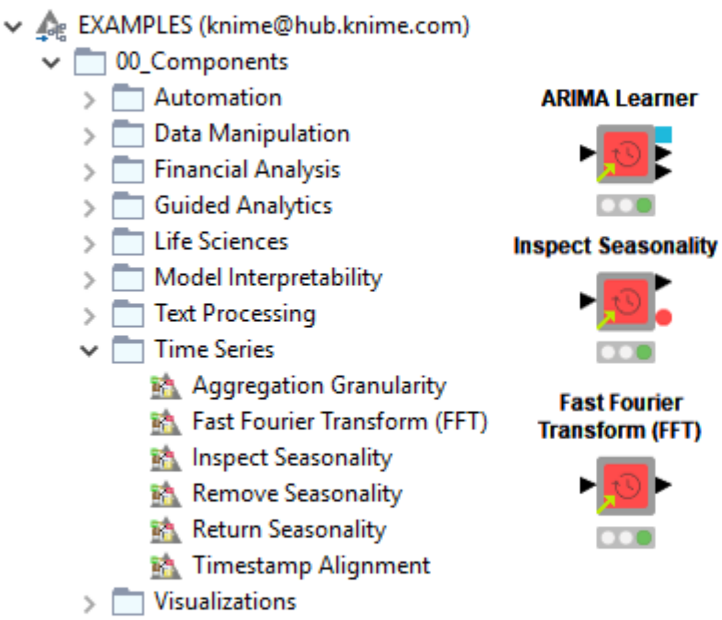
- Deploy the different techniques to generate out-of-sample forecasts for the next week



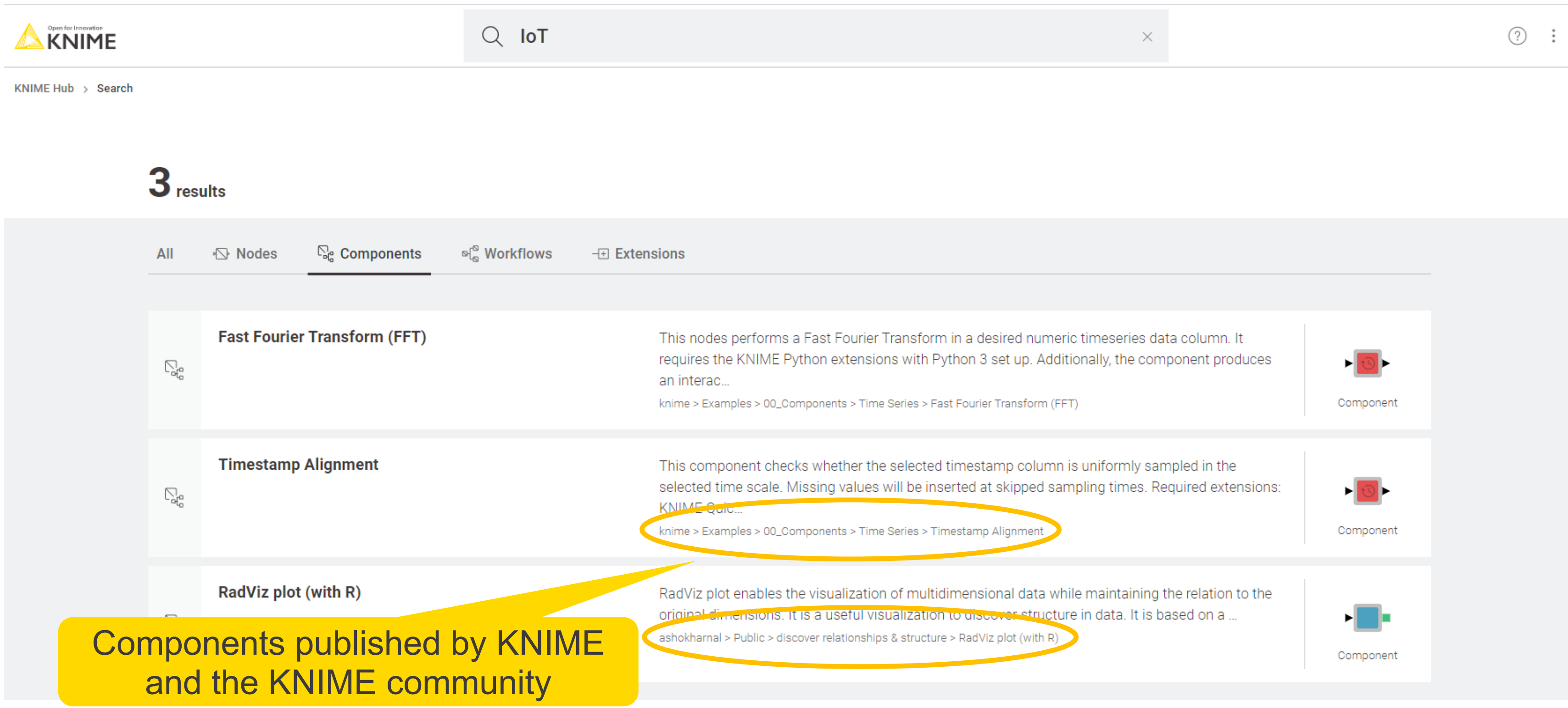


Components

- Encapsulates a functionality
  as a KNIME workflow
  - E. g. execute Python script by a component
    with a graphical UI
- Function like regular nodes:
  - Start using by drag and drop from the EXAMPLES Server/local directory
  - Configure in the component's configuration dialog
- Also available on the KNIME Hub

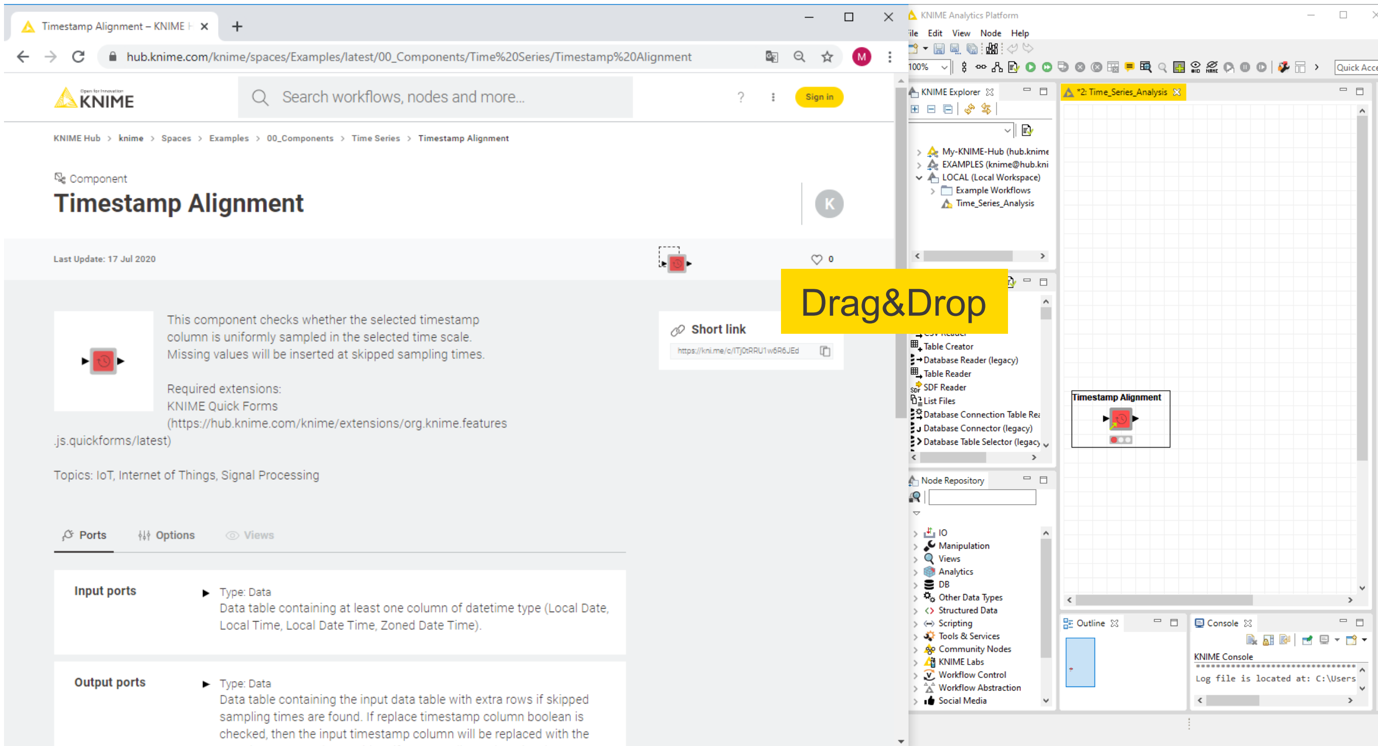


Components on the KNIME Hub



hub.knime.com

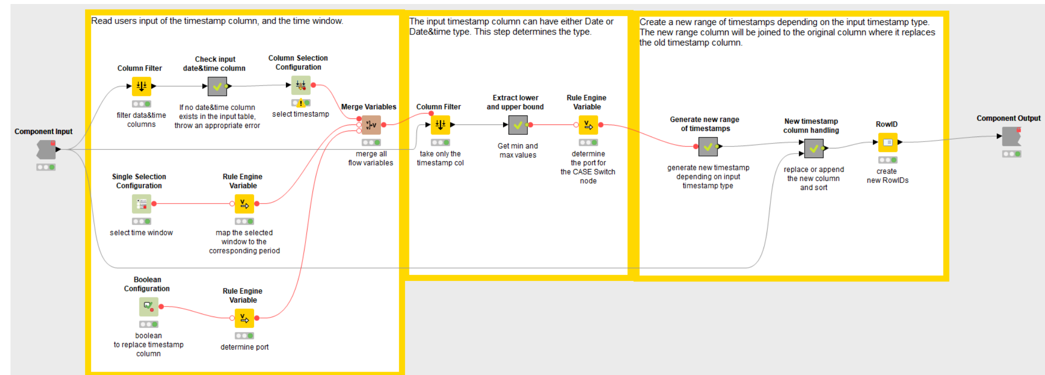
Components on the KNIME Hub

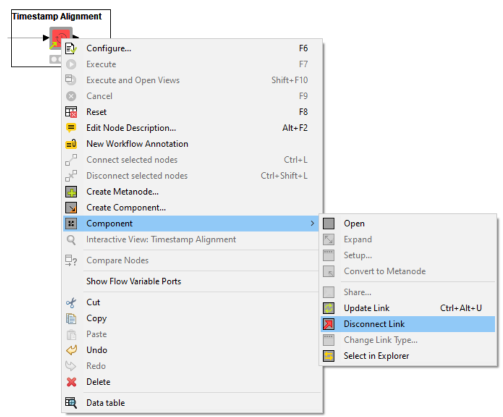


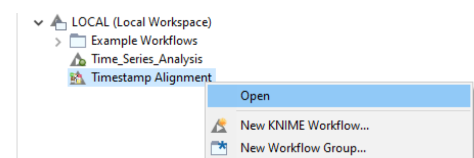
hub.knime.com

Components

- Instances of shared components are linked to the master and are therefore write-protected
- Editable after disconnecting the link or by a double click in the component editor

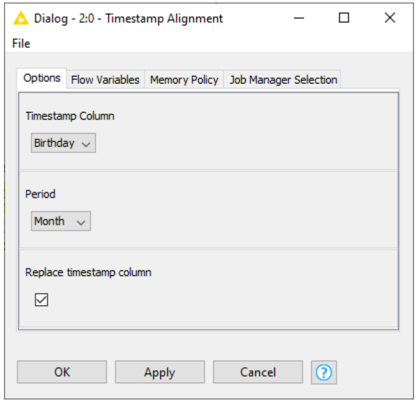




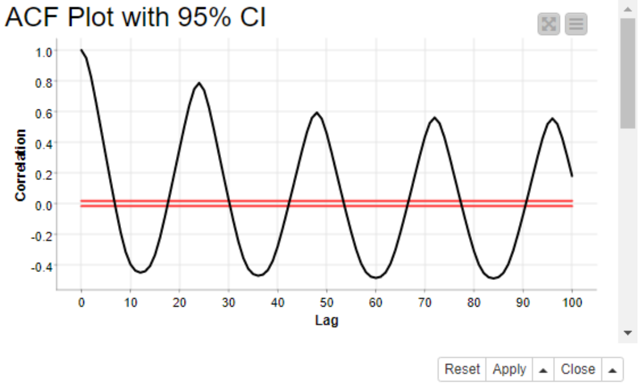


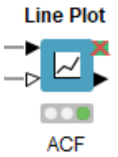
Components

- Components can have dialogs enabled by configuration nodes…



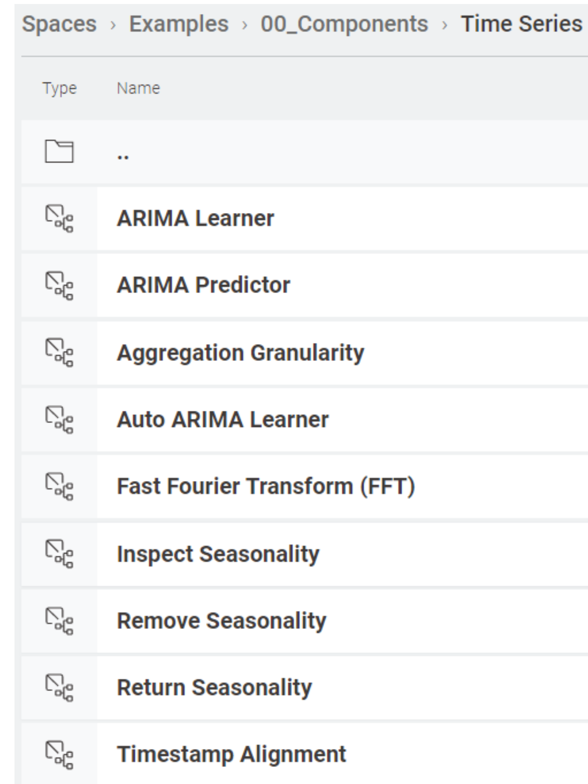
- … and interactive views enabled by widget nodes and JavaScript based nodes





Time Series Components

- Inspect, restore, and remove seasonality
- Train and apply ARIMA models
- Analyze residuals
- And many more!

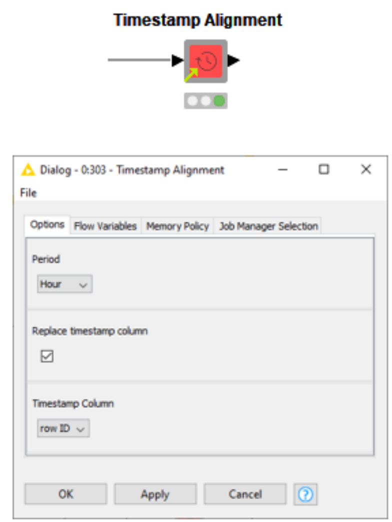


Component: Timestamp Alignment

- Acquire continuously spaced data
- In today's example we verify a record exists for every hour
- Otherwise create a missing value

cluster_26	Row ID
3.78	2010-03-24T22:00
3.85	2010-03-24T23:00
3.83	2010-03-25T01:00
3.95	2010-03-25T02:00
3.83	2010-03-25T03:00
3.75	2010-03-25T04:00

Input: Time series to check for uniform sampling



Output: Time series with skipped skipped sampling times

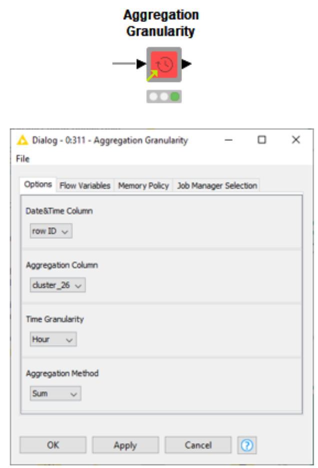
cluster_26	Row ID
3.78	2010-03-24T22:00
3.85	2010-03-24T23:00
?	2010-03-25T00:00
3.83	2010-03-25T01:00
3.95	2010-03-25T02:00
3.83	2010-03-25T03:00
3.75	2010-03-25T04:00

Component: Aggregation Granularity

- Extract granularities (year, month, hour, etc.) from a timestamp and aggregate (sum, average, mode, etc.) data at the selected granularity
- In today's example we calculate the total energy consumption by hour, day, and month

Input: Time series to aggregate

cluster_26	Row ID
3.25	2009-07-15T00:00
3.20	2009-07-15T01:00
3.04	2009-07-15T02:00
3.03	2009-07-15T03:00
3.01	2009-07-15T04:00
2.94	2009-07-15T05:00
…	…
3.18	2009-07-19T19:00
3.11	2009-07-19T20:00
3.01	2009-07-19T21:00
3.26	2009-07-19T22:00
3.20	2009-07-19T23:00



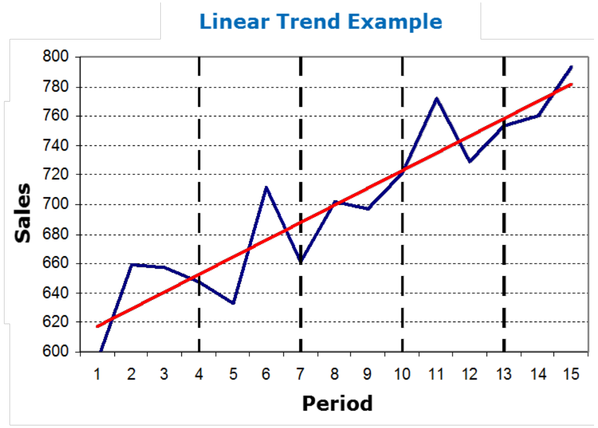
Output: Aggregated time series

Aggregated Timestamp	Sum (cluster_26)	Granularity
2009-07-15	185.82	Day
2009-07-16	182.49	Day
2009-07-17	177.87	Day
2009-07-18	116.48	Day
2009-07-19	83.95	Day

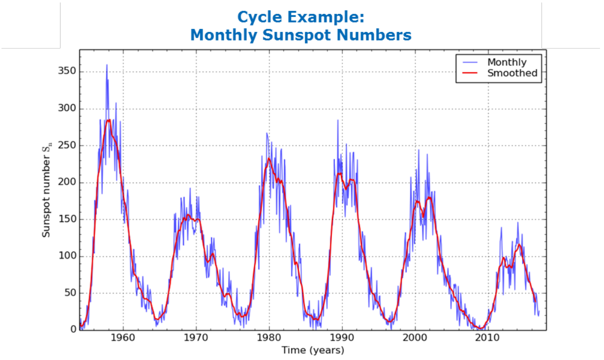
Descriptive Analytics
Load, Clean, and Explore

Time Series Properties: Main Elements

- TREND
  The general direction in which the series is running during a long period
  A **TREND** exists when there is a long-term increase or decrease in the data.
  It does not have to be necessarily linear (could be exponential or others functional form).



- CYCLE
  Long-term fluctuations that occur regularly in the series A CYCLE is an oscillatory component (i.e. Upward or Downward swings) which is repeated after a certain number of years, so:
  - May vary in length and usually lasts several years (from 2 up to 20/30)
  - Difficult to detect, because it is often confused with the trend component

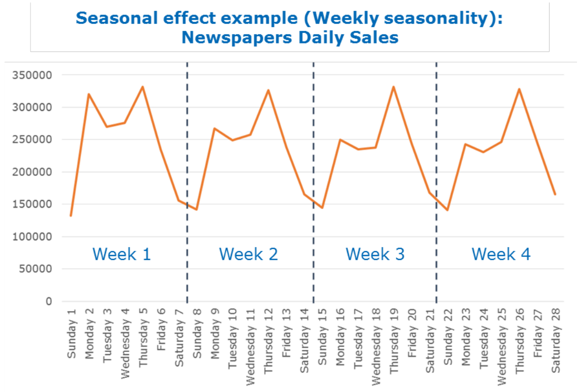


Time Series Properties: Main Elements

- **SEASONAL EFFECTS**

  Short-term fluctuations that occur regularly – often associated with months or quarters

  A **SEASONAL PATTERN** exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, day of the week). Seasonality is always of a fixed and known period.

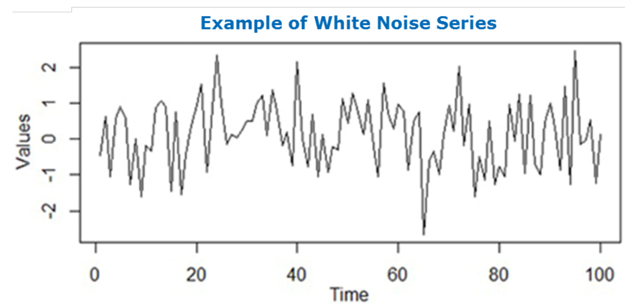


- **RESIDUAL**

  Whatever remains after the other components have been taken into account

  The residual/error component is everything that is not considered in previous components

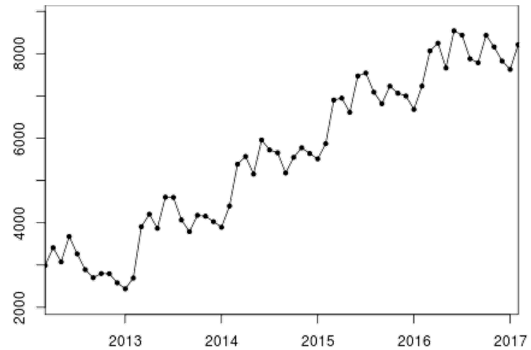
  Typically, it is assumed to be the sum of a set of random factors (e.g. a **white noise series**) not relevant for describing the dynamics of the series

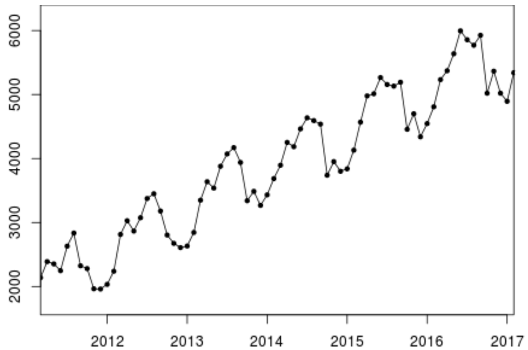


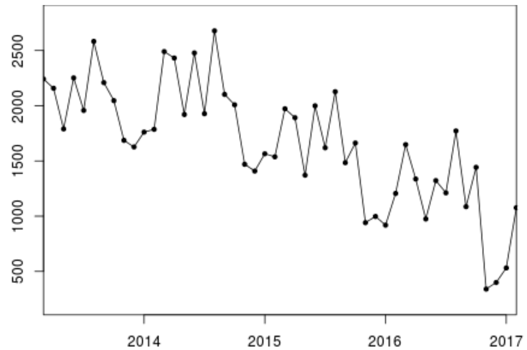
Seasonal effect: additive seasonality

- When the seasonality in Additive, the dynamics of the components are **independents from each other**; for instance, an increase in the trend-cycle will not cause an increase in the magnitude of seasonal dips
- The difference of the trend and the raw data is **roughly constant in similar periods of time** (months, quarters) irrespectively of the tendency of the trend

EXAMPLES OF ADDITIVE SEASONALITY



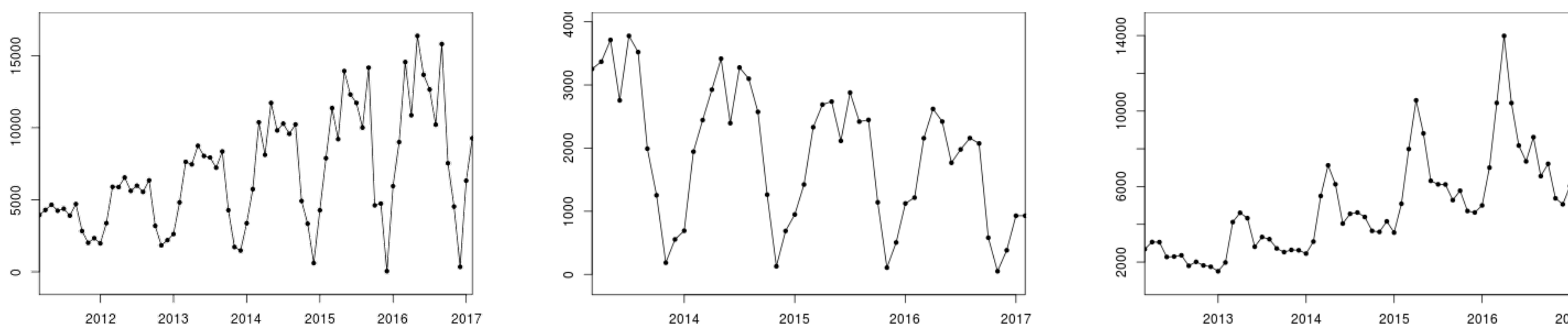




Seasonal effect: multiplicative seasonality

- In the multiplicative model the amplitude of the seasonality increase (decrease) with an increasing (decreasing) trend, therefore, on the contrary to the additive case, the **components are not independent from each other**
- When the variation in the seasonal pattern (or the variation around the trend-cycle) **appears to be proportional** to the level of the time series, then a multiplicative model is more appropriate.

EXAMPLES OF MULTIPLICATIVE SEASONALITY



Seasonal effect: frequency

According to the **data granularity** and to the **type of seasonality** you want to model, it is important to consider the right **seasonal frequency** (i.e. how many observations you have for every seasonal cycle)

- No problem if your data points are years, quarters, months or weeks (in this case you will face only annual seasonality), but if the frequency of observations is smaller than a week, things get more complicated
- For example, hourly data might have a daily seasonality (frequency=24), a weekly seasonality (frequency=24×7=168) and an annual seasonality (frequency=24×365.25=8766)

Frequency		Cycle type			
		Hour	Day	Week	Year
Data granularity	Annual				1
	Quarterly				4
	Monthly				12
	Weekly			1	52.18
	Daily		1	7	365.25
	Hourly	1	24	168	8766
	Minutes	60	1440	10080	525960

*Every year, on average, is made up of 365 days and 6 hours → so 365.25 days and 365.25/7=52.18 weeks

KNIME Open for Innovation

Numerical and graphical description of Time Series

- The first step in Time Series Analysis is to produce a **detailed exploratory analysis** of the data to get some insights about the distribution of the series over time
- This part must be performed using both **numerical descriptive analyses** and **graphical analyses**, such as:

Graphical descriptive analyses

- Time plot
- Seasonal plot
- Box plot analysis
- Scatterplots (Lag plots)
- Plotting auto-correlation and cross-correlation functions

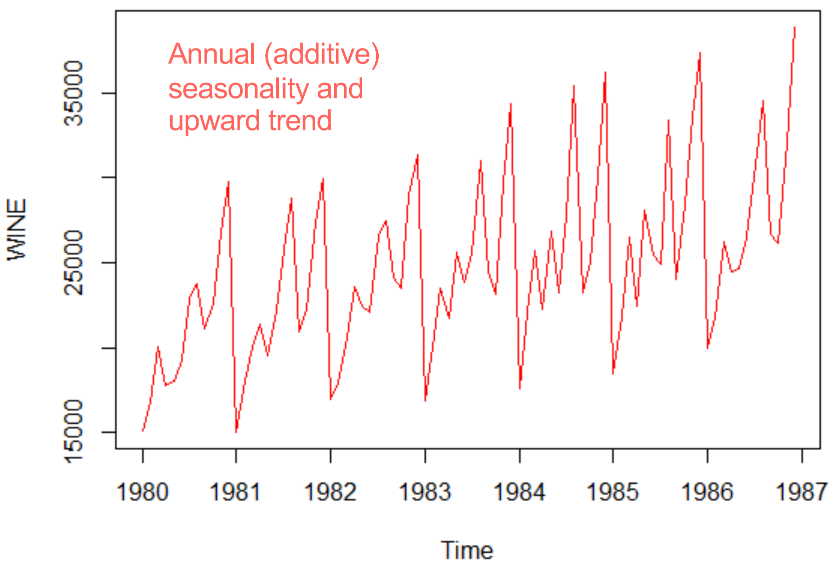
Numerical descriptive analyses

- Sampling period evaluation (start, end, data points features)
- Number of data available
- Missing value and outlier evaluation
- Frequency distribution analysis
- Summary descriptive statistics (overall and by season)

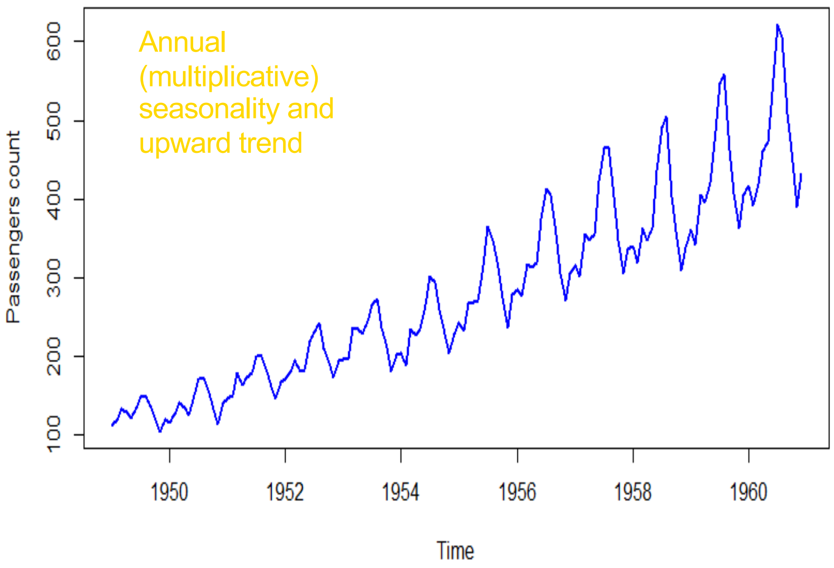
Graphical Analysis: Time Plot

- The first chart in time series analysis is the **TIME PLOT** → the observations are plotted against the time of observation, normally with consecutive observations joined by straight lines

Example of TS Plot of Australian monthly wine sales



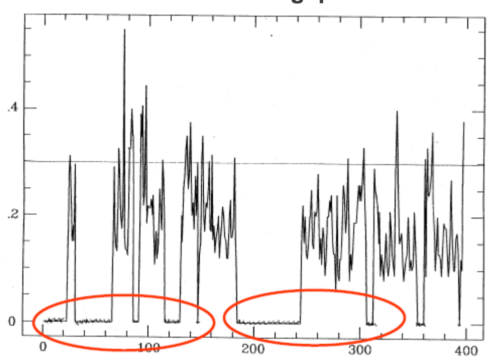
Example of TS Plot of Air Passengers (monthly) series



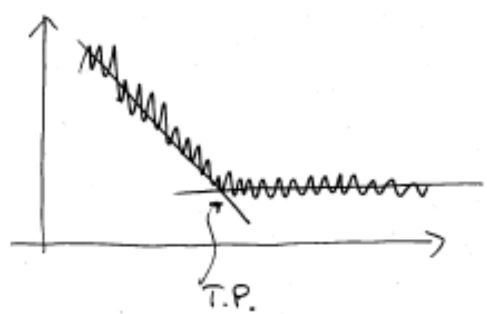
Graphical Analysis: Time Plot

- Insights you can get just from a simple **Time plot**
  - Is there a trend? Could it be linear or not?
  - Is there a seasonality effect?
  - Are there any long term cycles?
  - Are there any sharp changes in behaviour? Can such changes be explained?
  - Are there any missing values or “gap” in the series?
  - Are there any outliers, i.e. observations that differ greatly from the general pattern?
  - Is there any turning point/changing trend?

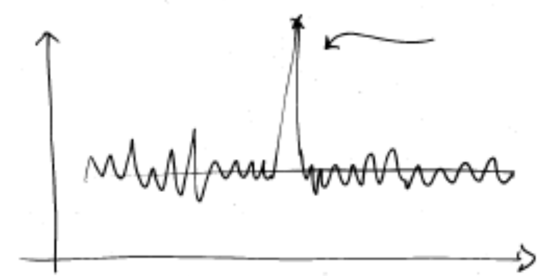
Series with gaps



Series with a turning point



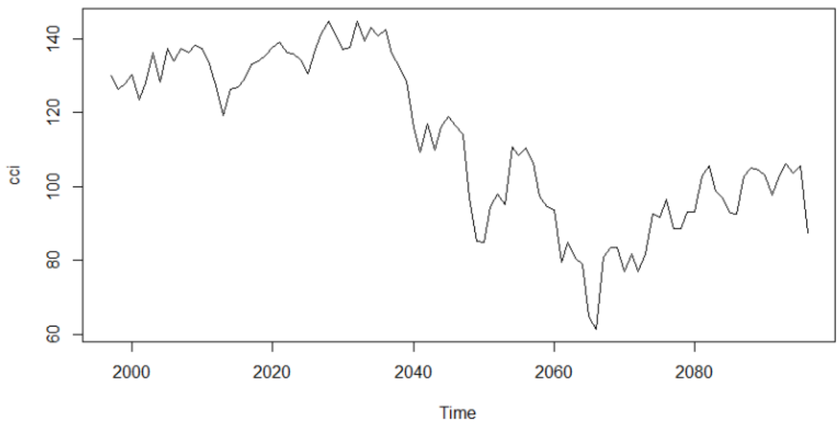
Series with an outlier



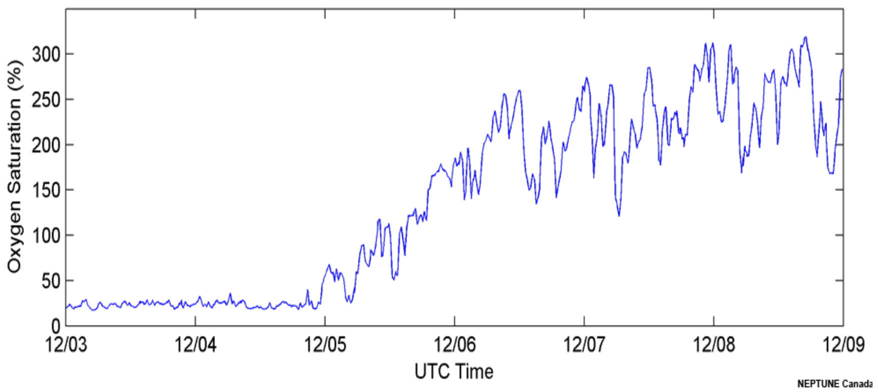
Graphical Analysis: Time Plot

- The **TIME PLOT** is very useful in cases where the series shows a very constant/simple dynamic (strong trend and strong seasonality), but in other cases could be difficult to draw clear conclusions

Consumer Cost Index



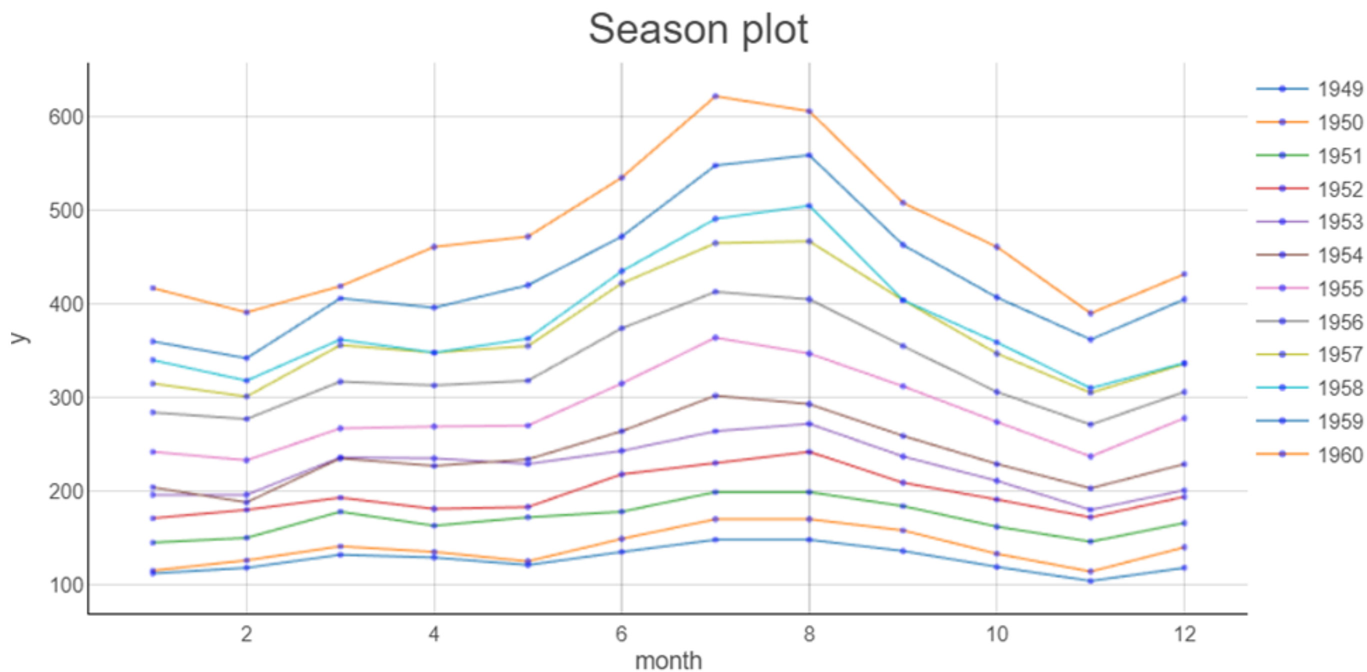
Oxygen saturation



  - Other graphical analyses and summary statistics could improve/extend the insights given by the simple time plot!

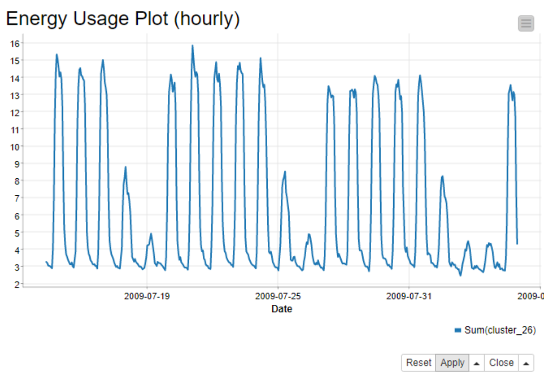
Graphical Analysis: Seasonal Plot

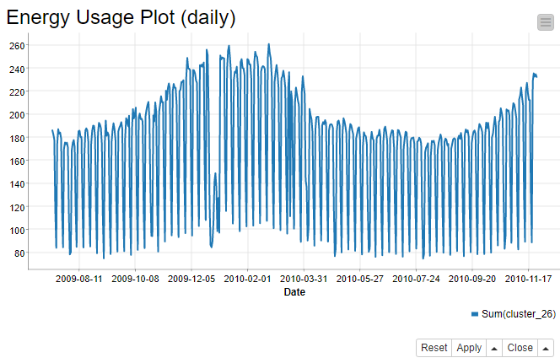
- Produce the **Seasonal plot** of the Time series in order to analyze more in detail the seasonal component (and possible changes in seasonality over time)

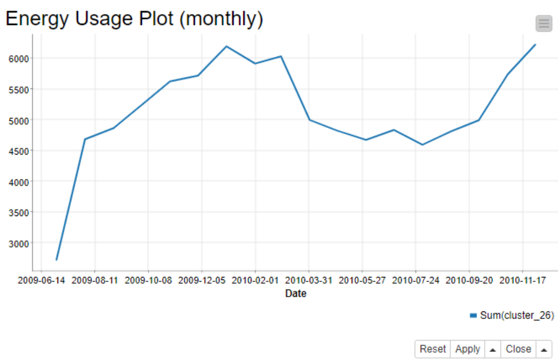


Granularity and Line Plots

- Show time series by hour, day, and month in line plots
- Identify daily, weekly, and yearly seasonality

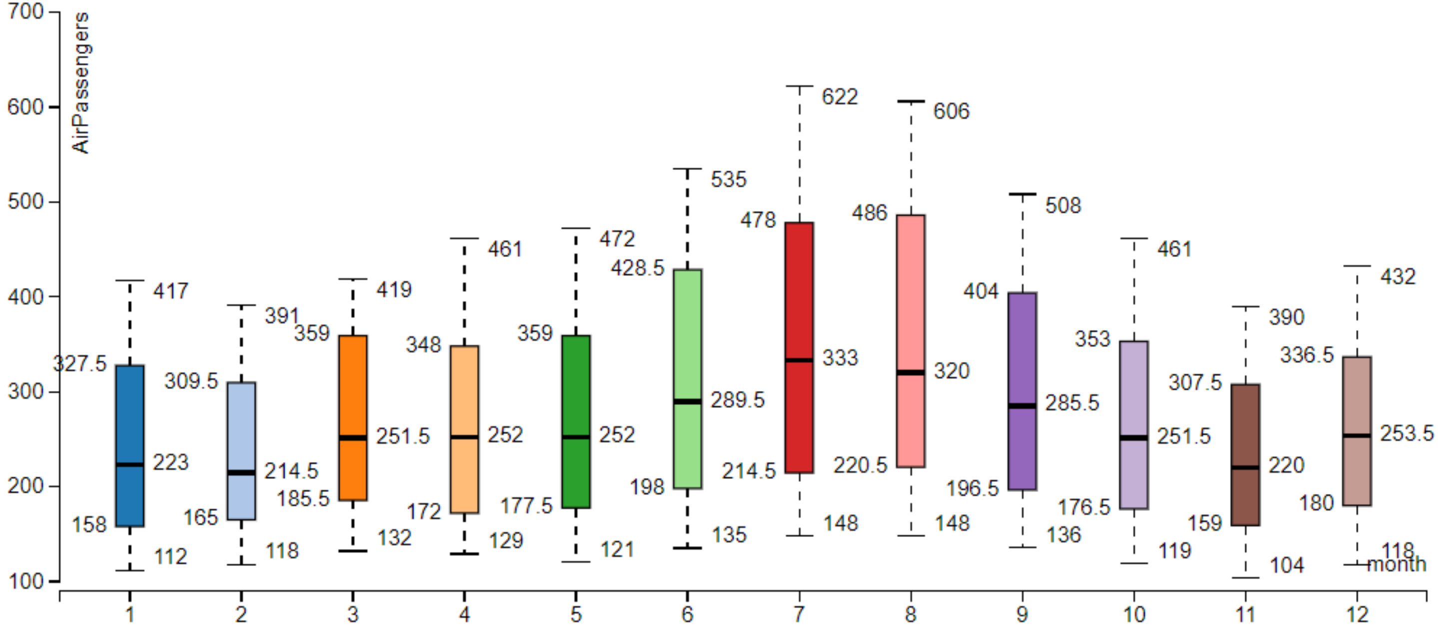






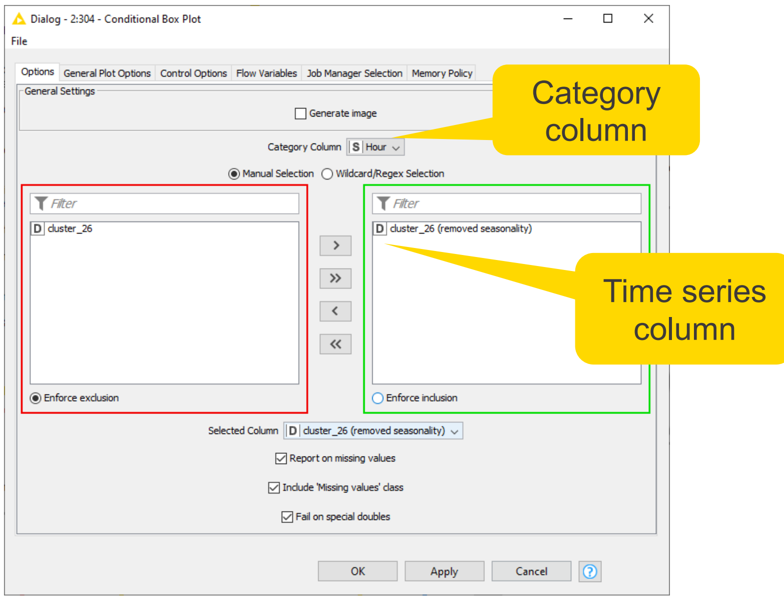
Graphical Analysis: Box Plot

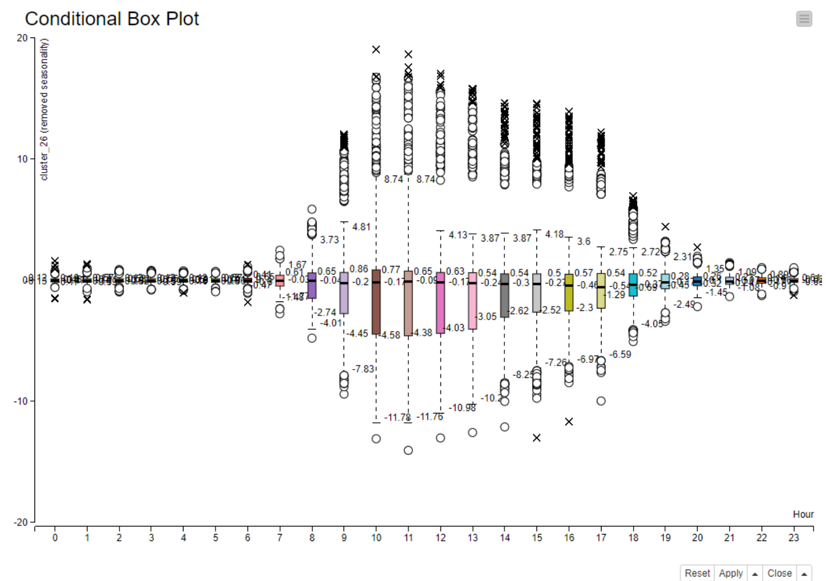
Create the **conditional Box plot** of the Time series in order to deeply understand the distribution of data in the same period of each seasons and focusing on specific aspects such as outliers, skewness, variability,…



Conditional Box Plot

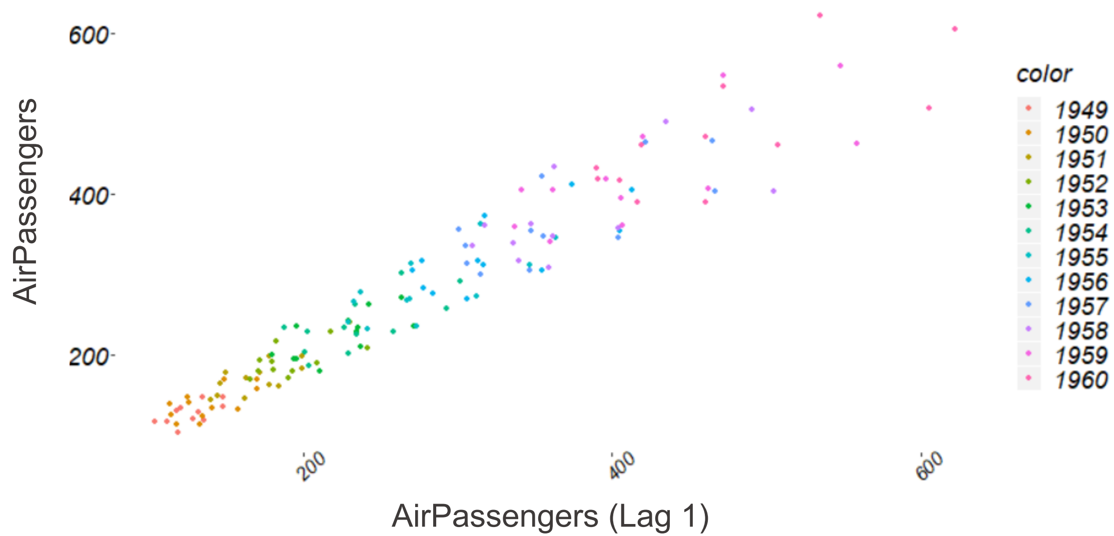
- Inspect the distribution of energy consumption hour by hour





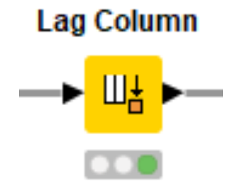
Graphical Analysis: Lag plot

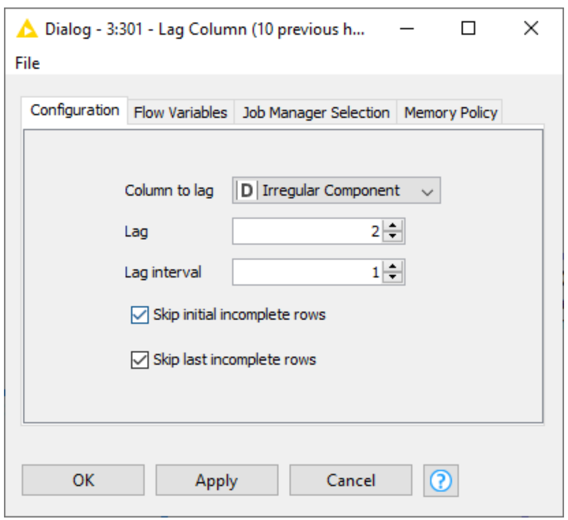
In time series analysis it's important to analyze the correlation between the *lagged values* of a time series (*autocorrelation*): the **lag plot** is a bivariate analysis, consisting in a simple scatter plot of the values of the target variable in ***t*** vs. the values of the same variable in ***t-k***; focusing on the correlation with the first lag (t-1) you can see from the plot below that there is a strong linear relation between the values in t and the values in t-1

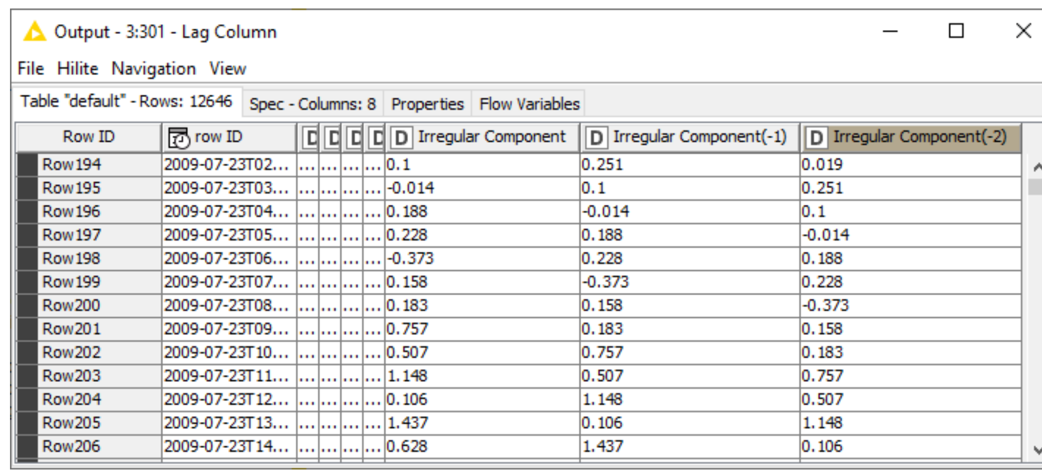


Lag Column Node

- Append past values as new columns
  - Shift cells *I* (lag interval) steps up
  - Duplicate the lag column *L* (lag value) times.
    In each column the rows are shifted *I, 2*I, .., L*I* steps up



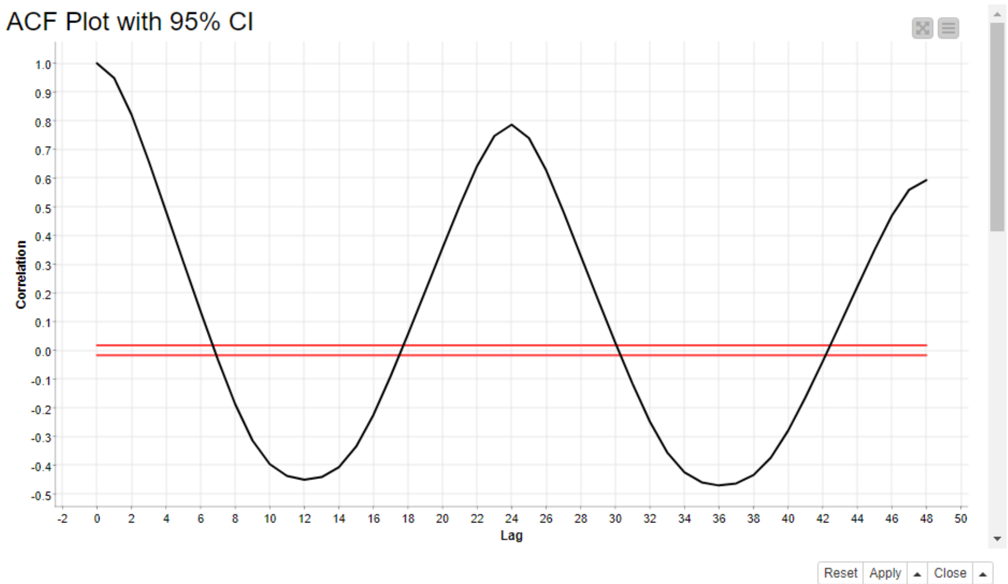




Row ID	row ID	D	D	D	D	Irregular Component	Irregular Component(-1)	Irregular Component(-2)
Row194	2009-07-23T02...	...	...	...	...	0.1	0.251	0.019
Row195	2009-07-23T03...	...	...	...	...	-0.014	0.1	0.251
Row196	2009-07-23T04...	...	...	...	...	0.188	-0.014	0.1
Row197	2009-07-23T05...	...	...	...	...	0.228	0.188	-0.014
Row198	2009-07-23T06...	...	...	...	...	-0.373	0.228	0.188
Row199	2009-07-23T07...	...	...	...	...	0.158	-0.373	0.228
Row200	2009-07-23T08...	...	...	...	...	0.183	0.158	-0.373
Row201	2009-07-23T09...	...	...	...	...	0.757	0.183	0.158
Row202	2009-07-23T10...	...	...	...	...	0.507	0.757	0.183
Row203	2009-07-23T11...	...	...	...	...	1.148	0.507	0.757
Row204	2009-07-23T12...	...	...	...	...	0.106	1.148	0.507
Row205	2009-07-23T13...	...	...	...	...	1.437	0.106	1.148
Row206	2009-07-23T14...	...	...	...	...	0.628	1.437	0.106

Numerical analysis: Auto Correlation Function (and ACF plot)

In order to go deeper inside the autocorrelation structure of the time series, you can create the Auto Correlation Function plot (**ACF plot**), also called *correlogram*: in this chart you can read the linear correlation index between the values in t and all the possible lags (t-1, t-2, …, t-k); the chart below shows all the correlations up to lag number 48

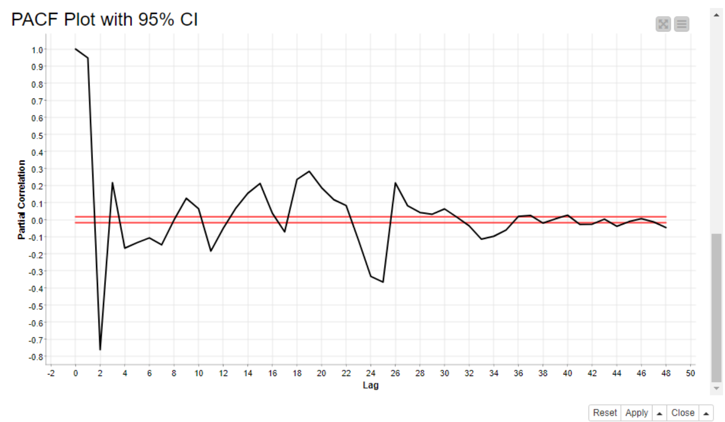


Numerical analysis: Auto Correlation Function (and ACF plot)

Together with the ACF, sometimes it is useful to analyze also the Partial Autocorrelation Function

The ACF plot shows the autocorrelations which measure the linear relationship between y_t and $y_{t-\mathrm{k}}$ for different values of *k* but consider that:

- if y_t and y_{t-1} are correlated, then y_{t-1} and y_{t-2} must also be correlated
- But then y_t and y_{t-2} might be correlated, simply because they are both connected to y_{t-1}
- → The **Partial Autocorrelation Function (PACF)** consider the linear relationship between y_t and $y_{t-\mathrm{k}}$ after *removing* the effects of other time lags $1, 2, 3, \ldots, k-1$



Numerical analysis: Descriptive statistics

From a numerical point of view, it's important to **produce statistics (total sample and split by seasonal periods) of the time series**, in order to have a more precise idea of: number of valid data points vs. missing data, central tendency measures, dispersions measures, percentiles, confidence intervals of the means, etc.

Time Series	Month	N obs	Missing	Mean	Std. Dev	Min	Max	95% LCL	95% UCL
AirPassengers	1	12	0	241.8	101.0	112	417	177.6	305.9
AirPassengers	2	12	0	235.0	89.6	118	391	178.1	291.9
AirPassengers	3	12	0	270.2	100.6	132	419	206.3	334.1
AirPassengers	4	12	0	267.1	107.4	129	461	198.9	335.3
AirPassengers	5	12	0	271.8	114.7	121	472	198.9	344.7
AirPassengers	6	12	0	311.7	134.2	135	535	226.4	396.9
AirPassengers	7	12	0	351.3	156.8	148	622	251.7	451.0
AirPassengers	8	12	0	351.1	155.8	148	606	252.1	450.1
AirPassengers	9	12	0	302.4	124.0	136	508	223.7	381.2
AirPassengers	10	12	0	266.6	110.7	119	461	196.2	336.9
AirPassengers	11	12	0	232.8	95.2	104	390	172.4	293.3
AirPassengers	12	12	0	261.8	103.1	118	432	196.3	327.3
AirPassengers	**Total**	**144**	**0**	**280.3**	**120.0**	**104**	**622**	**260.5**	**300.1**

Exercise 1: Loading and Exploring Data

- Load in the Energy Usage Data
- Perform preprocessing:
  - Convert string to Date&Time
  - Filter out unnecessary columns
  - Fill skipped sampling times with missing values
  - Handle missing values
  - Calculate hourly, daily, and monthly total energy consumption
- Plot the hourly, daily, and monthly totals in line plots

Time Series Analysis
01. Loading and Exploring Data

Summary:
In this exercise we will load the data file for cleaning, filtering, aggregating, and for some early visualizations.

Instructions:
1) Execute the File Reader node to load in the Energy Usage Data

2) Use a String to Date&Time node to convert the Row ID column to the correct format. The digits in the string pattern are converted correctly, if you write "yyyy-MM-dd_HH" in the date format field, or press the "Guess data type and format button".

3) Use a Column Filter node to remove all columns except the Row ID and Cluster 26, this is what we will analyze

4) Use the Time Stamp Alignment component to check for missing time stamps in the data

5) Connect a Missing Value node next to replace the missing values discovered in the previous step. Try the linear interpolation setting.

6) Use separate Aggregation Granularity components to aggregate the Time series into Hourly, Daily, and Monthly series

7) Use Line Plot nodes to visualize the outputs. Do you see any patterns?

8) Open the 01_Additional_Visualizations workflow in the Supplementary Workflows folder and inspect the season plot, confidence bounds, and lag plot of the Time series.

Review of Installation Requirements

- KNIME v4.02
- Python Environment
  - StatsModels
  - Keras=2.2.4 & TensorFlow=1.8.0 hp5=
- KNIME Python Integration
- KNIME Deep Learning Keras Integration

Open for Innovation
KNIME

KNIME Time Series Analysis Course - Session 2

KNIME AG

Agenda

1. Introduction: What is Time Series Analysis
2. Today's Task, Dataset & Components
3. Descriptive Analytics: Load, Clean, Explore
4. Descriptive Analytics: Non-stationarity, Seasonality, Trend
5. Quantitative Forecasting: Classical techniques
6. ARIMA Models: ARIMA(p,d,q)
7. Machine Learning based Models
8. Hyperparameter Optimization
9. Quick Intro to LSTM Networks
10. Example of Time Series Analysis on Spark
11. Conclusions & Summary

Exercise 1: Loading and Exploring Data

- Load in the Energy Usage Data
- Perform preprocessing:
  - Convert string to Date&Time
  - Filter out unnecessary columns
  - Fill skipped sampling times with missing values
  - Handle missing values
  - Calculate hourly, daily, and monthly total energy consumption
- Plot the hourly, daily, and monthly totals in line plots

Time Series Analysis
01. Loading and Exploring Data

Summary:
In this exercise we will load the data file for cleaning, filtering, aggregating, and for some early visualizations.

Instructions:
1) Execute the File Reader node to load in the Energy Usage Data

2) Use a String to Date&Time node to convert the Row ID column to the correct format. The digits in the string pattern are converted correctly, if you write "yyyy-MM-dd_HH" in the date format field, or press the "Guess data type and format button".

3) Use a Column Filter node to remove all columns except the Row ID and Cluster 26, this is what we will analyze

4) Use the Time Stamp Alignment component to check for missing time stamps in the data

5) Connect a Missing Value node next to replace the missing values discovered in the previous step. Try the linear interpolation setting.

6) Use separate Aggregation Granularity components to aggregate the Time series into Hourly, Daily, and Monthly series

7) Use Line Plot nodes to visualize the outputs. Do you see any patterns?

8) Open the 01_Additional_Visualizations workflow in the Supplementary Workflows folder and inspect the season plot, confidence bounds, and lag plot of the Time series.

Descriptive Analytics
Stationarity, Seasonality, Trend

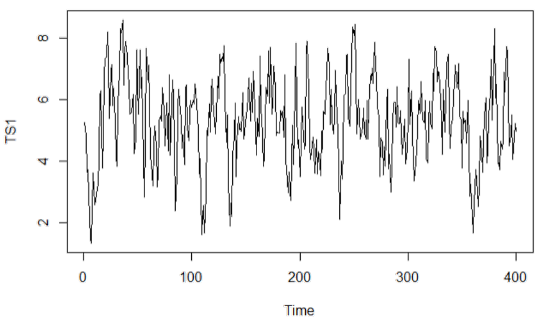
Stationarity

A time series can be defined as "**stationary**" when *its properties does not depend on the time at which the series is observed*, so that:

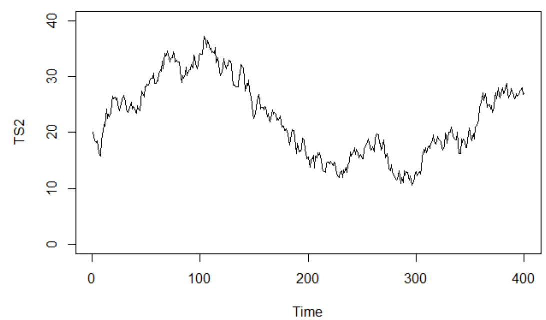
- the values oscillate frequently around the mean, independently from time
- the variance of the fluctuations remains constant across time
- the autocorrelation structure is constant over time and no periodic fluctuations exist

So, a time series that shows trend or seasonality is not stationary

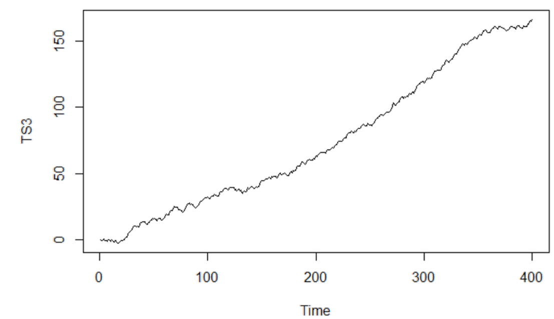
Stationary Time Series example



Non-Stationary Time Series example 1



Non-Stationary Time Series example 2



Stationarity

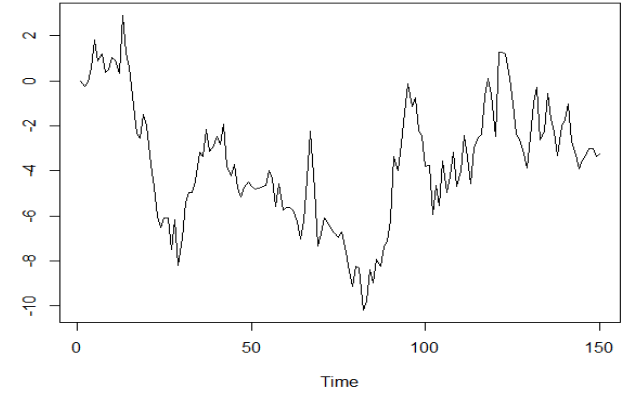
Typical examples of non-stationary series are all series that exhibit a deterministic trend (i.e. $y_t = \alpha + \beta \cdot t + \varepsilon_t$) or the so-called **"Random Walk"**

Random Walk (without drift) → $y_t = y_{t-1} + \varepsilon_t$ (where ε_t is white noise)

A random walk model is very widely used for non-stationary data, particularly financial and economic data.

- Random walks typically have:
  - long periods of apparent trends up or down
  - sudden and unpredictable changes in direction
  - variance and autocorrelation that depends on time!

Random Walk Example

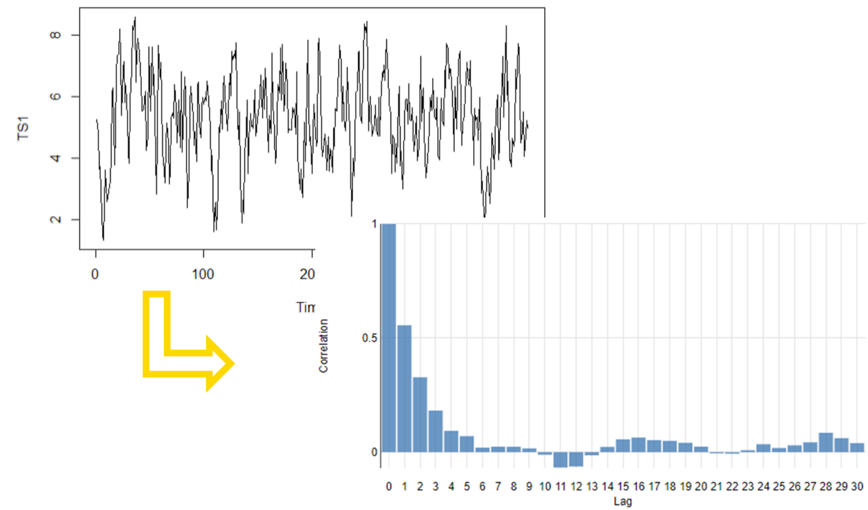


Stationarity

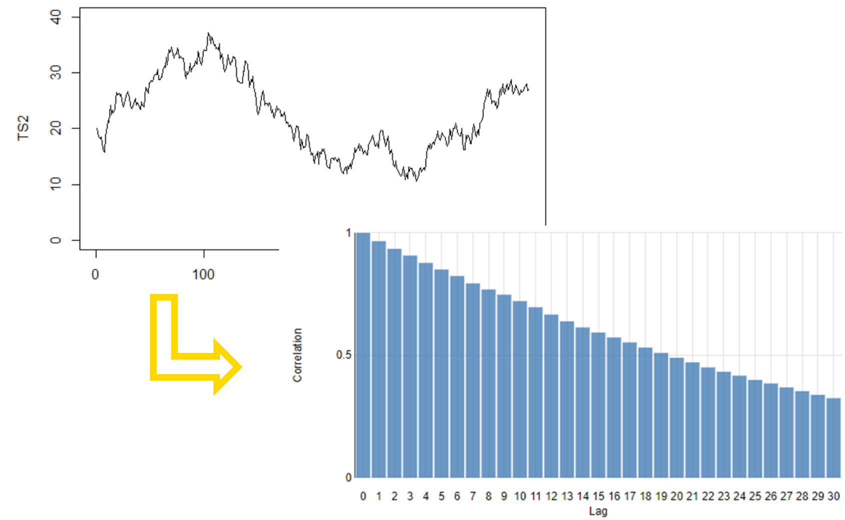
Besides looking at the time plot of the data, the ACF plot is also useful for identifying non-stationary TS:

→ for a stationary time series, the ACF will **drop to zero (i.e. within confidence bounds) relatively quickly**, while the ACF of non-stationary data **decreases slowly**

Stationary Time Series example



Non-Stationary Time Series example 1 (random walk!)



Differencing

- One way to make a time series stationary is to compute the differences between consecutive observations → This is known as DIFFERENCING
  - Differencing can help **stabilize the mean** of a time series by removing changes in the level of a time series, and so eliminating trend (and also seasonality, using a specific differencing order)
  - The **Order of Integration** for a Time Series, denoted ***I*(*d*)**, reports the minimum number of differences (d) required to obtain a stationary series (*note:* ***I*(*0*)** → it means the series is stationary!)
  - Transformations such as logarithms can help to stabilize the variance of a time series

Differenced Time Series (first order)

y_t $\quad$ $y'_t = y_t - y_{t-1}$

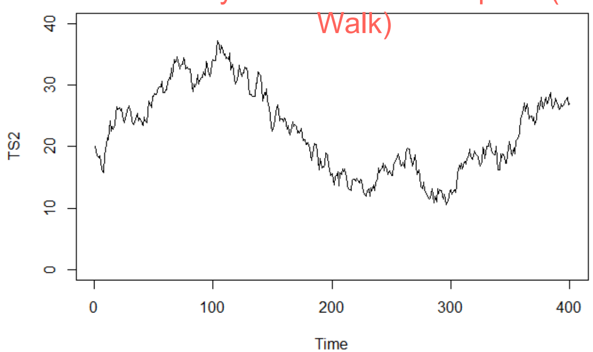
CYCLE_	WEEK_	DATE_	COLLI_ARR	DIFF_1
1	1	1 1	983	.
1	2	1 2	1478	495
1	3	1 3	1822	345
1	4	1 4	1883	61
1	5	1 5	1913	30
1	6	1 6	2001	88
1	7	1 7	2077	76

1478 – 983 = **495**

Differencing

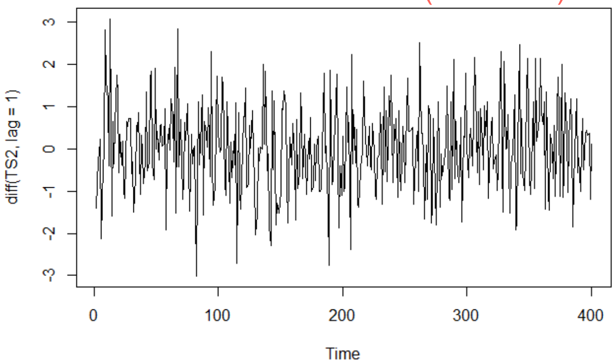
Example: use differencing to make stationary a non-stationary series

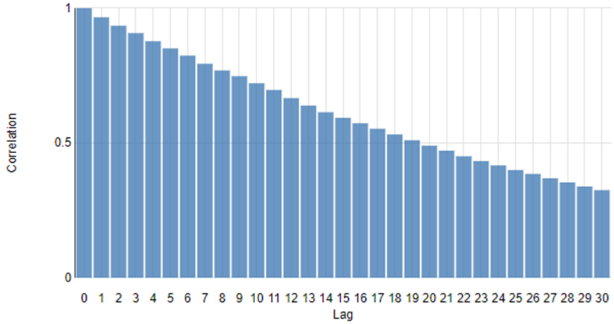
Non-Stationary Time Series example 1 (Random Walk)

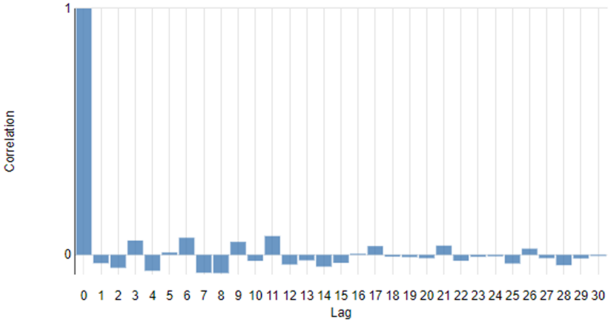


$TS2_t - TS2_{t-1}$

Differenced Time Series (first order)





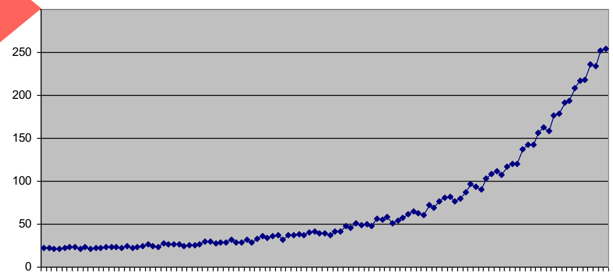


No significative autocorrelation exists → applying first differences to a random walk generates a white noise

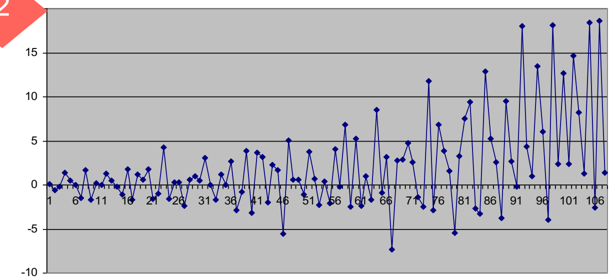
Differencing

Occasionally the differenced data will not appear stationary and it may be necessary to difference the data a second time to obtain a stationary series ($y''_t = y'_t - y'_{t-1} = [y_t - y_{t-1}] - [y_{t-1} - y_{t-2}]$)*

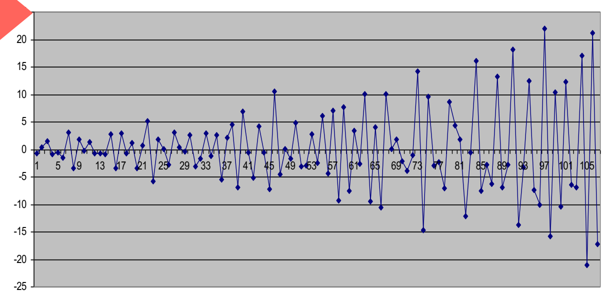
1 Original Time Series: non-stationary (mean and variance)



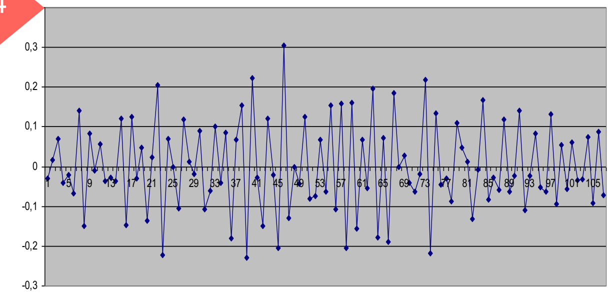
2 First Order Differencing: non-stationary (mean and variance)



3 Second Order Diff.: stationary in mean, but not in variance



4 Double Differencing applied to Log(Series): stationary series



* it's almost never necessary to go beyond second-order differences

Differencing

A **seasonal difference** is the difference between an observation and the corresponding observation from the previous (seasonal) cycle

$$y'_t = y_t - y_{t-F}$$

Where F is the (seasonal) cycle frequency

→ *The seasonal differencing removes strong and stable seasonality pattern* (and transform into a white noise the so called “seasonal random walk”, i.e. $y_t = y_{t-F} + \varepsilon_t$)

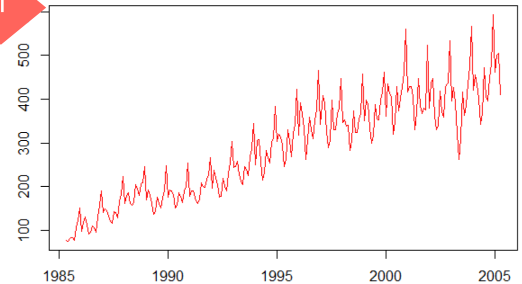
Consider that:

- Sometimes it’s needed to apply both “simple” first differencing and seasonal differencing in order to obtain a stationary series
- It makes no difference which is done first—the result will be the same
- However, if the data have a strong seasonal pattern, it’s recommended that seasonal differencing be done first because sometimes the resulting series will be stationary and there will be no need for a further non-seasonal differencing

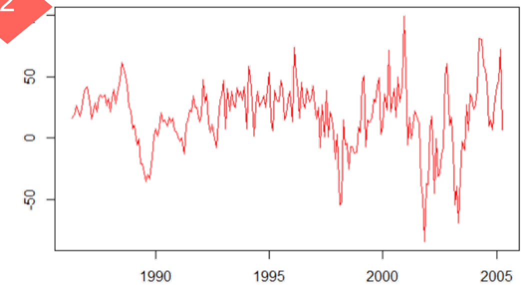
Differencing

Consider the following example where a set of differencing has been applied to "Monthly Australian overseas visitors" TS

1 Original Time Series (y_t)

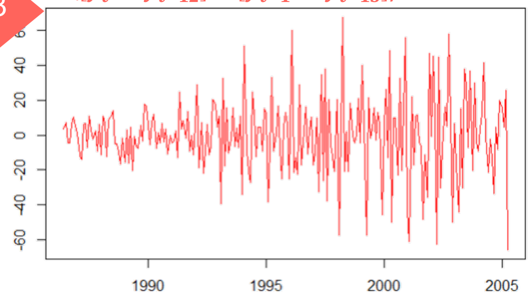


2 Seasonal Differencing ($y_t - y_{t-12}$)



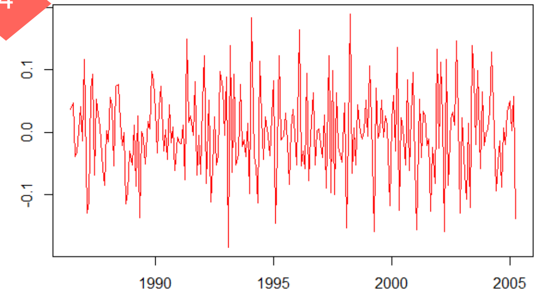
3 Applying first differencing to seasonal differenced series

$([y_t - y_{t-12}] - [y_{t-1} - y_{t-13}])$



4 Use log trasformation in order to stabilize the variance

$([log(y_t) - log(y_{t-1})] - [log(y_{t-12}) - log(y_{t-13})])$

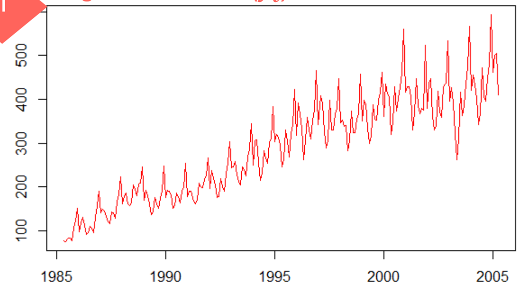


The series now appears to be stationary

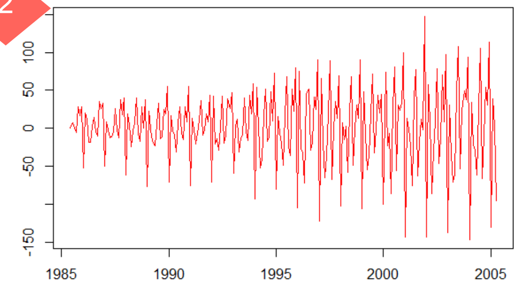
Differencing

Same example of the previous slide, but changing the differencing process order → the final result is…

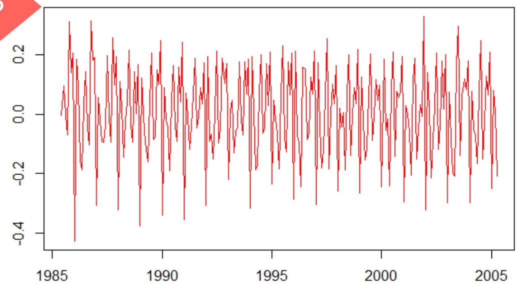
1 Original Time Series (y_t)



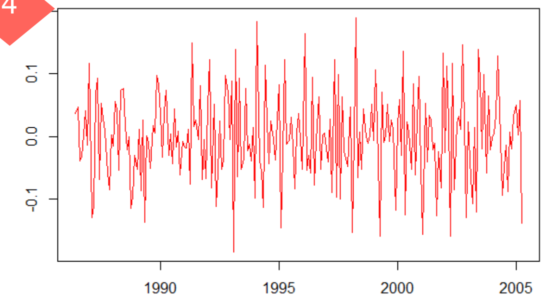
2 First Order Differencing ($y_t - y_{t-1}$)



3 First Order Diff. after log transformation ($log(y_t) - log(y_{t-1})$)



4 Applying seasonal differencing to first order differenced of log series ($[log(y_t) - log(y_{t-1})] - [log(y_{t-12}) - log(y_{t-13})]$)

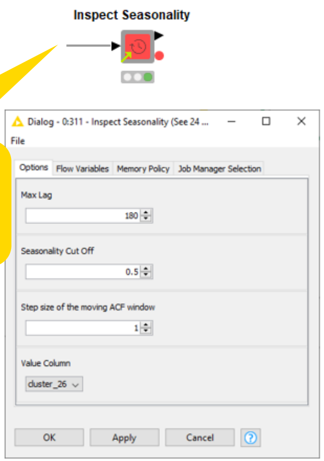


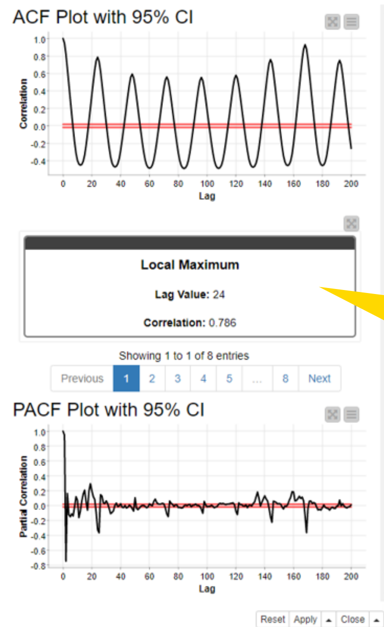
The series is now stationary

Component: Inspect Seasonality

- Calculates (partial) autocorrelation with lagged values
- In today's example we inspect daily seasonality in the energy consumption data

Input: Time series to inspect seasonality





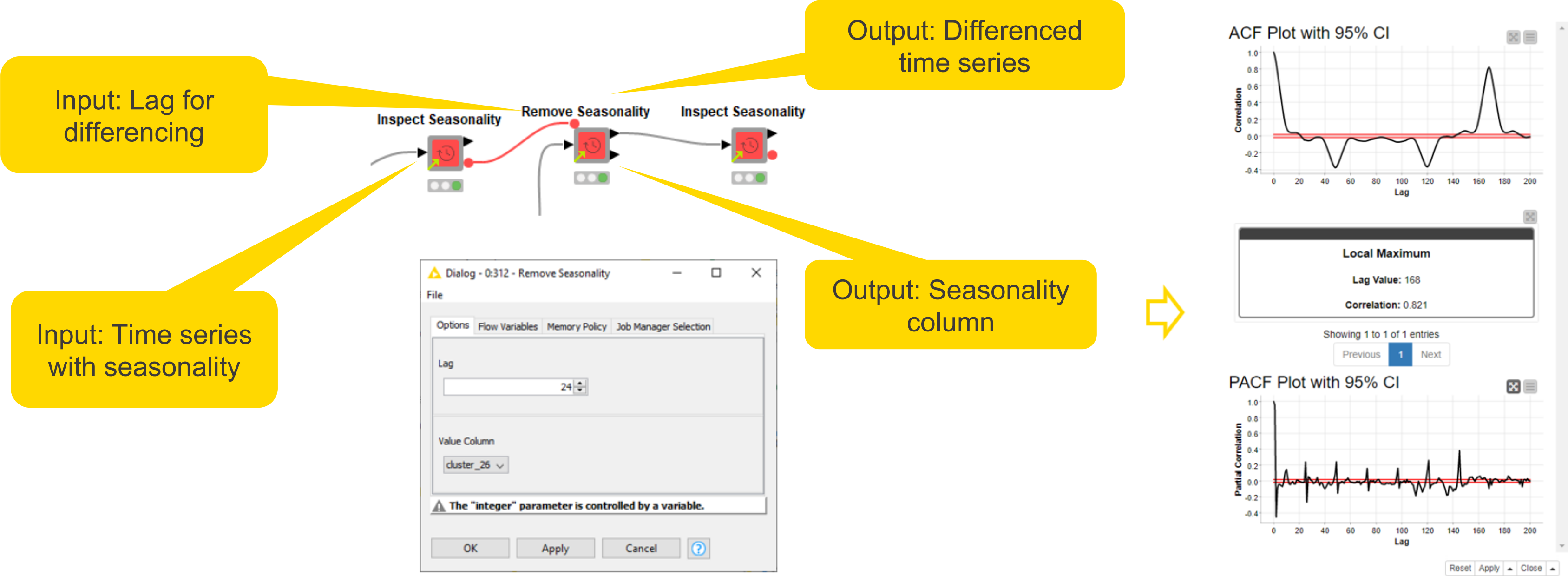
Output: Autocorrelation plot

Output: Lag with maximum correlation available in the flow variable output. Local maximums are listed in the data output.

Output: Partial autocorrelation plot

Component: Remove Seasonality

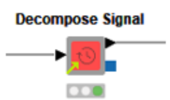
- Removes seasonality by differencing at the selected lag
- In today's example we remove daily seasonality from the energy consumption data



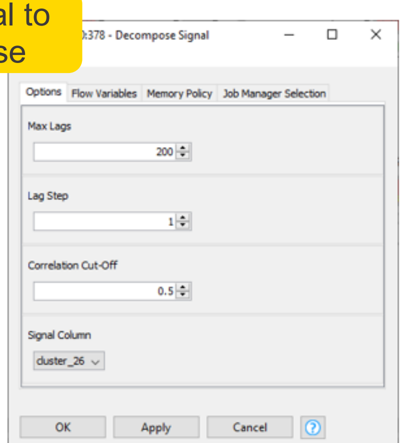
Component: Decompose Signal

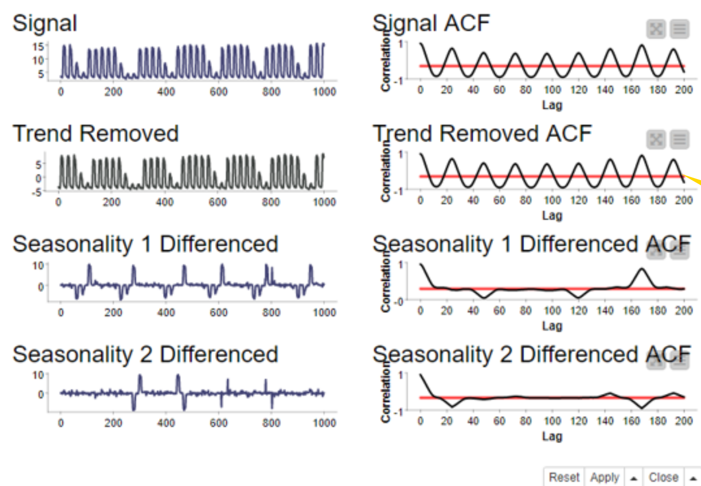
- Extract trend, first and second seasonality, and residual from time series and show the progress of time series in line plots and ACF plots

Output: Signal and columns for trend, seasonality, and residual



Input: Signal to decompose





Output: Line plots and ACF plots at the different stages of decomposing

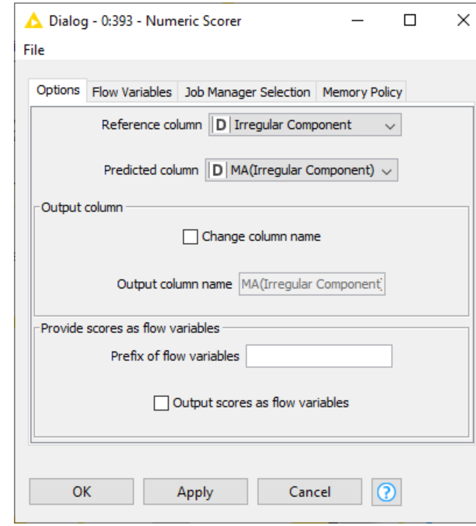
Numeric Errors: Formulas

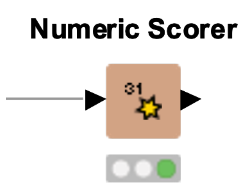
Error Metric	Formula	Notes
R-squared	$1 - \frac{\sum_{i=1}^{n}(f(x_i) - y_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$	Universal range: the closer to 1 the better
Mean absolute error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert f(x_i) - y_i\rvert$	Equal weights to all distances Same unit as the target column
Mean squared error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$	Common loss function
Root mean squared error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2}$	Weights big differences more Same unit as the target column
Mean signed difference	$\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)$	Only informative about the direction of the error
Mean absolute percentage error (MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert f(x_i) - y_i\rvert}{\lvert y_i\rvert}$	Requires non-zero target column values

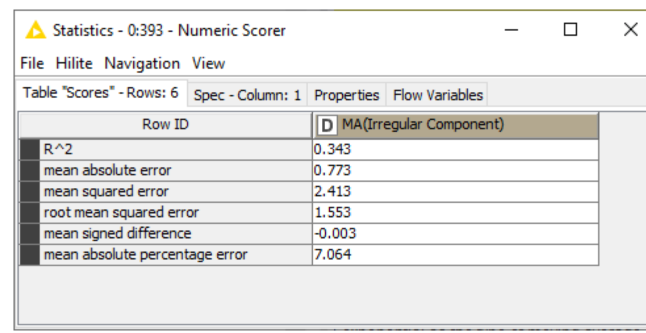
Numeric Scorer Node

Evaluate numeric predictions

- Compare actual target column values to predicted values to evaluate goodness of fit.
- Report R^2, RMSE, MAPE, etc.

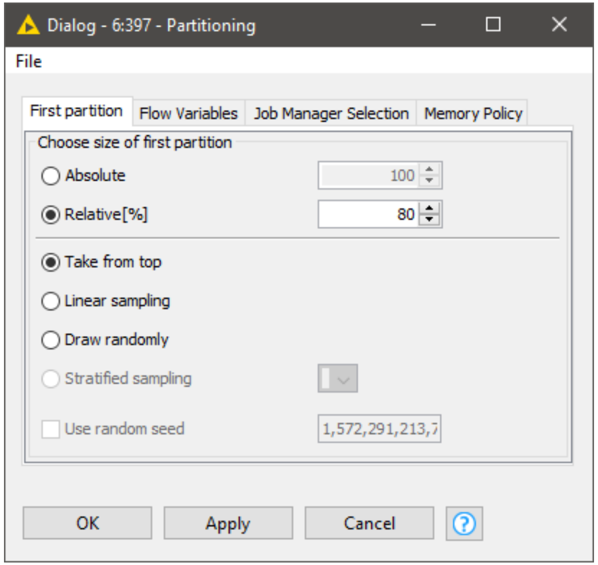






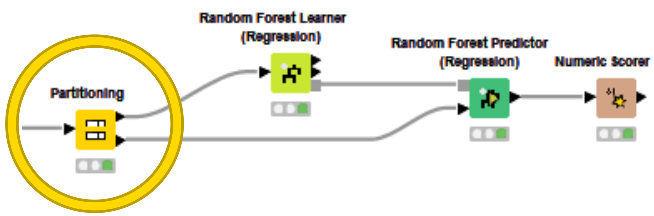
Partitioning for Time Series

- When Partitioning data for training a Time Series model it is important your training data comes before your test data chronologically.
- This will mirror how the model is used in deployment, always forecasting the future.
- To do this make sure your data is properly sorted and partition with the “Take from top” option. In the KNIME node.

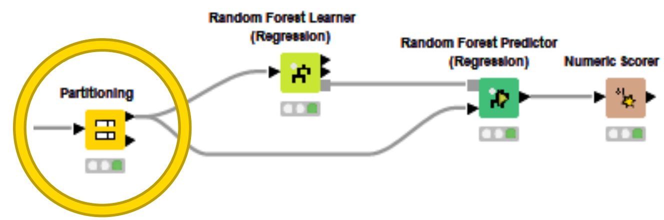


In-Sample vs. Out-sample

Out-Sample Static



In-Sample Static

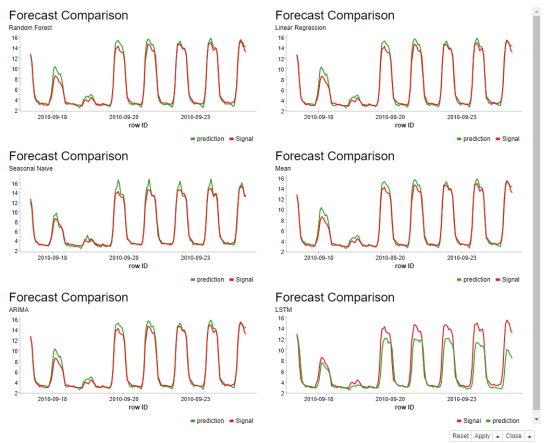


- Data used to train is the sample data
- Forecasts on the sample data are called **In-Sample** Forecasts
- Forecasts on other data are called **Out-Sample** Forecasts
- Either Forecast is called **Dynamic** if it uses prior Forecasts as its inputs, if real values are used it is called **Static**

Model Evaluation

- Assess the expected forecast accuracy of your model by comparing actual and predicted time series
  - Training data vs. in-sample predictions
  - Test data vs. out-of-sample predictions

Visual comparison in a line plot:



Numeric comparison by error metrics:

Forecast Accuracy

Comparison of Different Methods

Name	Time	R^2	mean absolute error	mean squared error	root mean squared error	mean signed difference	mean absolute percentage error
LSTM	0 Minutes 39 Seconds	0.8	1.241	3.858	1.964	-0.982	0.15
Random Forest	5 Minutes 19 Seconds	0.98	0.458	0.385	0.62	0.16	0.079
Linear Regression	0 Minutes 7 Seconds	0.98	0.451	0.394	0.628	0.159	0.074
ARIMA	27 Minutes 24 Seconds	0.98	0.441	0.377	0.614	0.203	0.072
Seasonal Naive	0 Minutes 3 Seconds	0.979	0.436	0.397	0.63	0.111	0.069
Mean	0 Minutes 3 Seconds	0.98	0.444	0.381	0.617	0.21	0.072

Showing 1 to 6 of 6 entries

Reset Apply Close

Exercise 2: Inspecting and Removing Seasonality

- Use ACF plots to inspect seasonality from energy consumption data
- Remove seasonality and check again the ACF plot
- Compare hourly energy consumption values before and after removing seasonality
- Optional: split energy consumption data into a trend, seasonality, and residual

Time Series Analysis
02. Inspecting & Removing Seasonality

Summary:
In this exercise we'll explore seasonality in the time series using conditional box plots and the (P)ACF plots.

Instructions:
1) Run the workflow up through the Missing Value node, this is where we left off in the previous exercise

2) Use the Inspect Seasonality Component to kook at the ACF and PACF plots of the Time Series. Do we have any Seasonality?

3) Use the Remove Seasonality Component to remove the seasonality we discovered

4) Apply another copy of the Inspect Seasonality component after the removal. Does the ACF plot look better?

5) Use the Extract Date&Time Fields node to extract the Hour from the timestamp (Row ID column) after the Missing Value node

6) Use the Number to String node to convert the Hour values into string

7) Use the Conditional Box Plot node to visualize the Energy Usage by hour, do we see a pattern?

8) Repeat steps 5-7 after the Remove Seasonality component, does it look better?

Optional) Use the Decompose Signal component after the Missing Value node and look at the view

KNIME Time Series Analysis Course - Session 3

KNIME AG

Agenda

1. Introduction: What is Time Series Analysis
2. Today's Task, Dataset & Components
3. Descriptive Analytics: Load, Clean, Explore
4. Descriptive Analytics: Non-stationarity, Seasonality, Trend
5. Quantitative Forecasting: Classical techniques
6. ARIMA Models: ARIMA(p,d,q)
7. Machine Learning based Models
8. Hyperparameter Optimization
9. Quick Intro to LSTM Networks
10. Example of Time Series Analysis on Spark
11. Conclusions & Summary

Exercise 2: Inspecting and Removing Seasonality

- Use ACF plots to inspect seasonality from energy consumption data
- Remove seasonality and check again the ACF plot
- Compare hourly energy consumption values before and after removing seasonality
- Optional: split energy consumption data into a trend, seasonality, and residual

Time Series Analysis
02. Inspecting & Removing Seasonality

Summary:
In this exercise we'll explore seasonality in the time series using conditional box plots and the (P)ACF plots.

Instructions:
1) Run the workflow up through the Missing Value node, this is where we left off in the previous exercise

2) Use the Inspect Seasonality Component to kook at the ACF and PACF plots of the Time Series. Do we have any Seasonality?

3) Use the Remove Seasonality Component to remove the seasonality we discovered

4) Apply another copy of the Inspect Seasonality component after the removal. Does the ACF plot look better?

5) Use the Extract Date&Time Fields node to extract the Hour from the timestamp (Row ID column) after the Missing Value node

6) Use the Number to String node to convert the Hour values into string

7) Use the Conditional Box Plot node to visualize the Energy Usage by hour, do we see a pattern?

8) Repeat steps 5-7 after the Remove Seasonality component, does it look better?

Optional) Use the Decompose Signal component after the Missing Value node and look at the view

Quantitative Forecasting

Classical Techniques

Qualitative vs. Quantitative

The approaches to forecasting are essentially two: *qualitative approach and quantitative approach*

- **Qualitative forecasting** methods are adopted when historical data are not available (e.g. estimate the revenues of a new company that clearly doesn't have any data available). They are highly subjective methods.

- **Quantitative forecasting** techniques are based on *historical quantitative data*;
  →the analyst, starting from those data, tries to understand the underlying structure of the phenomenon of interest and then to use the same historical data for forecasting purposes

Our focus

Quantitative forecasting

The basis for quantitative analysis of time series is the assumption that there are factors that influenced the dynamics of the series in the past and these factors continue to **bring similar effects in also in the future**

Main methods used in Quantitative Forecasting:

1. **Classical Time Series Analysis:** analysis and forecasts are based on identification of structural components, like trend and seasonality, and on the study of the serial correlation → *univariate time series analysis*
2. **Explanatory models:** analysis and forecasts are based both on past observations of the series itself and also on the relation with other possible predictors → *multivariate time series analysis*
3. **Machine learning models:** Different Artificial Neural Networks algorithms used to forecast time series (both in univariate or multivariate fashion)

Classical Time Series Analysis

The main tools used in the Classical Time Series Analysis are:

- **Classical Decomposition:** considers the time series as the overlap of several elementary components (i.e. trend, cycle, seasonality, error)
- **Exponential Smoothing:** method based on the weighting of past observations, taking into account the overlap of some key time series components (trend and seasonality)
- **ARIMA** (*AutoRegressive Integrated Moving Average*): class of statistical models that aim to treat the correlation between values of the series at different points in time using a regression-like approach and controlling for seasonality

Which model?

The choice of **the most appropriate method of forecasting** is influenced by a number of factors, that are:

- **Forecast horizon**, in relation to TSA objectives
- Type/amount of **available data**
- Expected **forecastability**
- Required **readability** of the results
- **Number of series** to forecast
- **Deployment** frequency of the models
- Development **complexity**
- Development **costs**

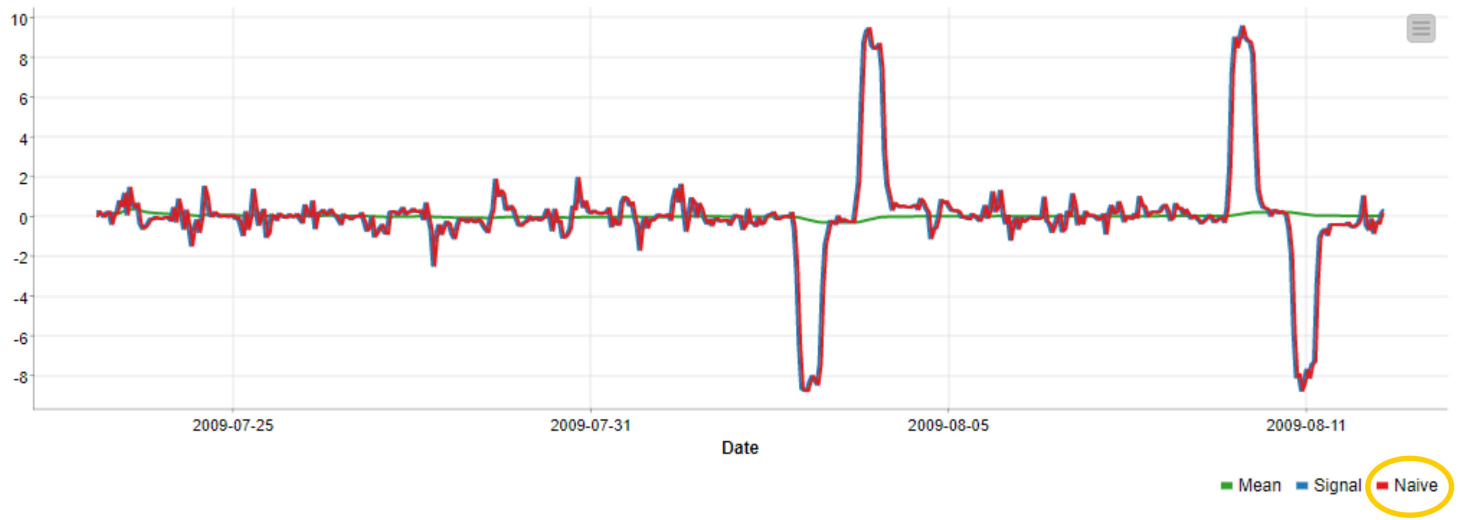
Naïve Prediction

- Predict values by the most recent known value

$$\hat{y}_{T+h|T} = y_T,$$

where y_T is the most recent known value and $h=1,2,3$

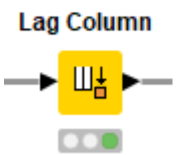
- Best predictor for true random walk data



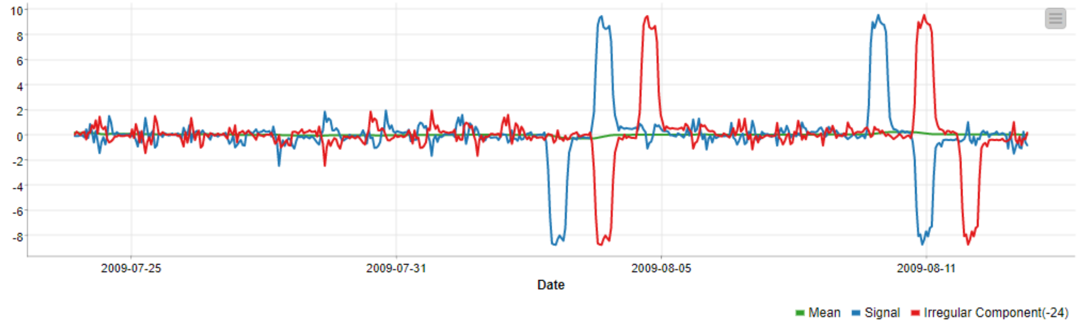
Naïve seasonal Prediction

- Predict values by the most recent known value

$$\hat{y}_{T+h|T} = y_{T+h-m(k+1)},$$



- where m is the seasonal period, and k is the integer part of (h−1)/m (i.e., the number of complete years in the forecast period prior to time T+h).
- For example, with hourly data, the forecast for all future 6pm values is equal to the last observed 6pm value.
- Best predictor for seasonal random walk data



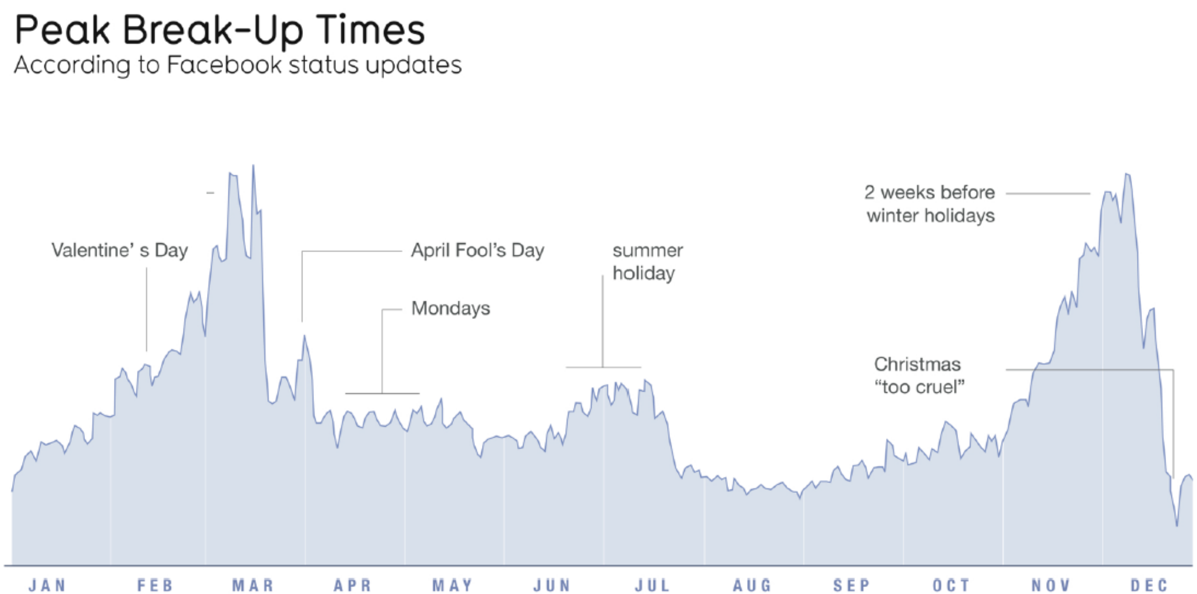
Interpretation issues

IMPORTANT: Remember that quantitative data ARE NOT JUST NUMBERS..

.. they have **a story to tell**, especially if your data are time series!

So.. always try to understand what's going on from a logical/business point of view: try to give an interpretation to the observed dynamics!

Example 1:
can you draw something useful looking at this series?



Open for Innovation KNIME

ARIMA Models

ARIMA(p,d,q)

Goal of this Section

1. Introduction to ARIMA
2. ARIMA Models
3. ARIMA Model selection
4. ARIMAX

Exponential Smoothing vs. ARIMA

While exponential smoothing models are based on a description of level, trend and seasonality in the data, ARIMA models aim to describe the **autocorrelations in the data**

REMINDER: Just as correlation measures the amount of a linear relationship between two variables, AUTOCORRELATION measures the linear relationship between *lagged values* of a time series

- There are several autocorrelation coefficients, depending on the lag length
- r_1 measures the relationship between y_t and y_{t-1}, r_2 measures the relationship between y_t and y_{t-2}, and so on

Before starting with ARIMA models is useful to give a look to a preliminary concept: what is a **linear regression model**?

ARIMA Models: General framework

An ARIMA model is a numerical expression indicating how the observations of a target **variable are statistically correlated with past observations of the same variable**

- ARIMA models are, in theory, the most general class of models for forecasting a time series which can be "**stationarized**" by transformations such as differencing and lagging
- The easiest way to think of ARIMA models is as fine-tuned versions of random-walk models: the fine-tuning consists of adding lags of the differenced series and/or lags of the forecast errors to the prediction equation, as needed to remove any remains of autocorrelation from the forecast errors

In an ARIMA model, in its most complete formulation, are considered:

- An **Autoregressive (AR)** component, seasonal and not
- A **Moving Average (MA)** component, seasonal and not
- The order of **Integration (I)** of the series

That's why we call it ARIMA (Autoregressive Integrated Moving Average)

ARIMA Models: General framework

The most common notation used for ARIMA models is:

$$ARIMA(p, d, q)\ (P, D, Q)s$$

where:

- **p** is the number of autoregressive terms
- **d** is the number of non-seasonal differences
- **q** is the number of lagged forecast errors in the equation
- **P** is the number of seasonal autoregressive terms
- **D** is the number of seasonal differences
- **Q** is the number of seasonal lagged forecast errors in the equation
- **s** is the seasonal period (cycle frequency using R terminology)

→ **In the next slides we will explain each single component of ARIMA models!**

ARIMA Models: Autoregressive part (AR)

In a **multiple regression model**, we predict the target variable Y using a linear combination of independent variables (predictors)→ In an **autoregression model**, we forecast the variable of interest using a linear combination of past values of the variable itself

The term autoregression indicates that it is a regression of the variable against itself

- An **Autoregressive model of order $\boldsymbol{p}$**, denoted $AR(p)$ model, can be written as

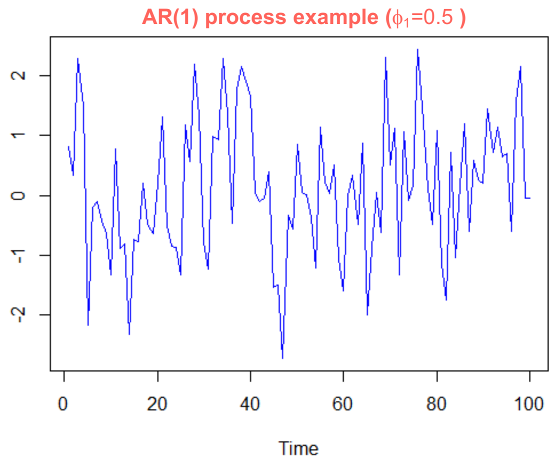
$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

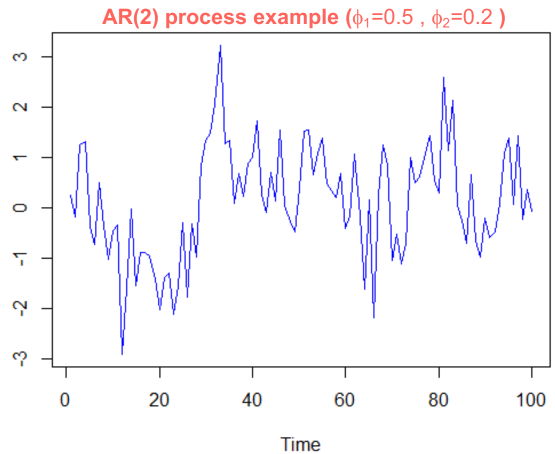
Where:

- y_t = dependent variable
- $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$ = independent variables (i.e. lagged values of y_t as predictors)
- $\phi_1, \phi_2, \ldots, \phi_p$ = regression coefficients
- ε_t = error term (must be white noise)

ARIMA Models: Autoregressive part (AR)

Autoregressive simulated process examples:





Consider that, in case of **AR(1)** model:

- When $\phi_1 = 0$, y_t is a white noise
- When $\phi_1 = 1$ and $c = 0$, y_t is a random walk
- In order to have a stationary series the following condition must be true: $-1 < \phi_1 < 1$

ARIMA Models: Moving Average part (MA)

Rather than use past values of the forecast variable in a regression, a Moving Average model uses **past forecast errors** in a regression-like model

In general, a moving average process of order q, MA (q), is defined as:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$$

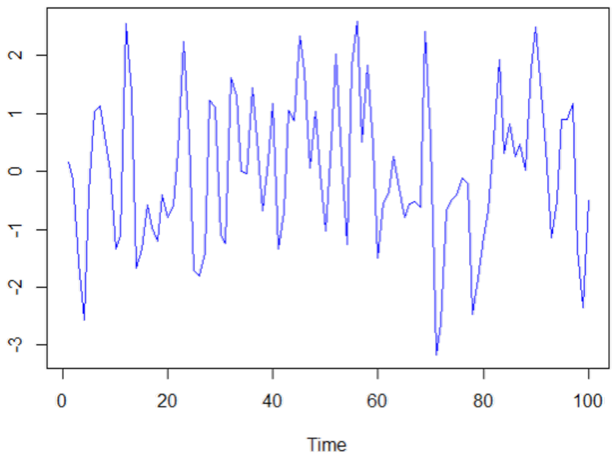
The lagged values of ε_t are not actually observed, so it is not a standard regression

Moving average models should not be confused with **moving average smoothing** (the process used in classical decomposition in order to obtain the trend component)→ A **moving average model** is used for forecasting future values while moving average smoothing is used for estimating the trend-cycle of past values

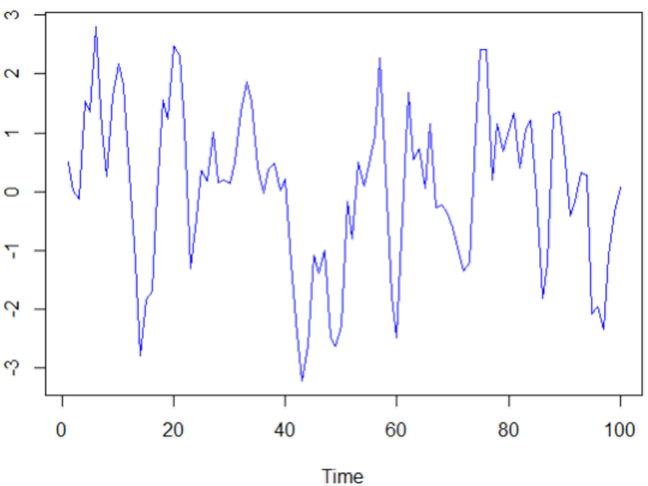
ARIMA Models: Moving Average part (MA)

Moving Average simulated process examples:

MA(1) process example (θ_1=0.7)



MA(2) process example (θ_1=0.8 , θ_2=0.5)



- Looking just the time plot it’s hard to distinguish between an AR process and a MA process!

ARIMA Models: ARMA and ARIMA

If we combine autoregression and a moving average model,
we obtain an **ARMA(p,q)** model:

$$y_t = c + \underbrace{\phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p}}_{\text{Autoregressive component of order p}} + \underbrace{\theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}}_{\text{Moving Average component of order q}} + \varepsilon_t$$

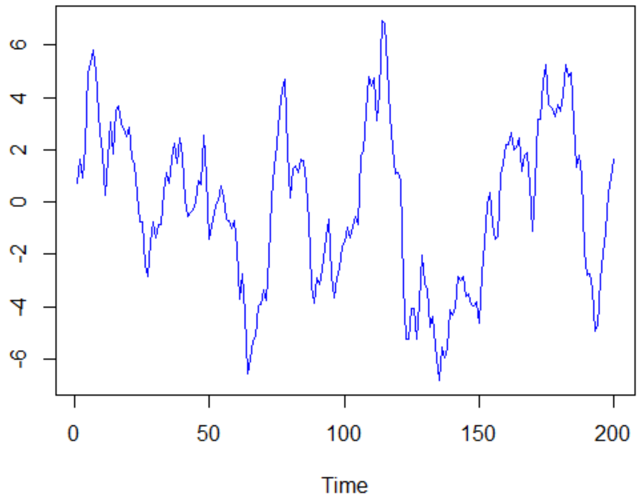
To use an ARMA model, the series must be **STATIONARY**!

- If the series is NOT stationary, before estimating and ARMA model, we need to apply one or more differences in order to make the series stationary: this is the integration process, called ***I(d)***, where d= number of differences needed to get stationarity
- If we model *the integrated* series using an ARMA model, we get an **ARIMA (p,d,q)** model where p=order of the autoregressive part; d=order of integration; q= order of the moving average part

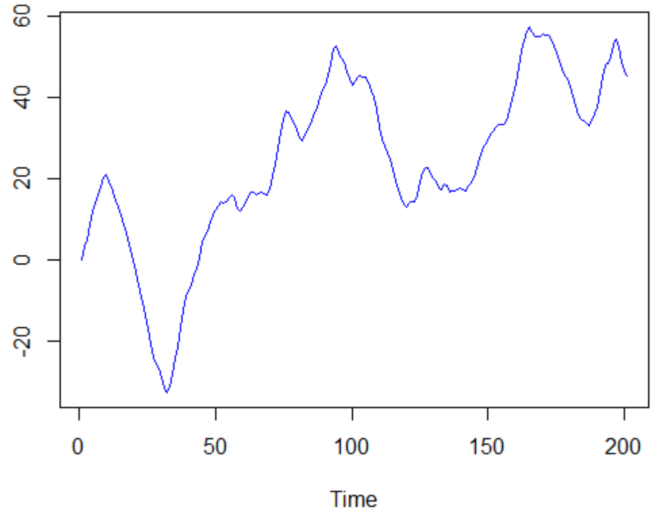
ARIMA Models: ARMA and ARIMA

ARIMA simulated process examples

ARMA(2,1) process example, equal to ARIMA(2,0,1)
(ϕ_1=0.5, ϕ_2=0.4, θ_1=0.8)



ARIMA(2,1,1) process example (ϕ_1=0.5, ϕ_2=0.4, θ_1=0.8)



ARIMA Models: Model identification

General rules for model indentification based on ACF and PACF plots:

The data may follow an $\boldsymbol{ARIMA(p,d,0)}$ model if the ACF and PACF plots of the differenced data show the following patterns:

- the ACF is exponentially decaying or sinusoidal
- there is a significant spike at lags p in PACF, but none beyond lag p

The data may follow an $\boldsymbol{ARIMA(0,d,q)}$ model if the ACF and PACF plots of the differenced data show the following patterns:

- the PACF is exponentially decaying or sinusoidal
- there is a significant spike at lags q in ACF, but none beyond lag q

→ For a general $\boldsymbol{ARIMA(p,d,q)}$ model (with both **p** and **q > 1**) both ACF and PACF plots show exponential or sinusoidal decay and it's more difficult to understand the structure of the model

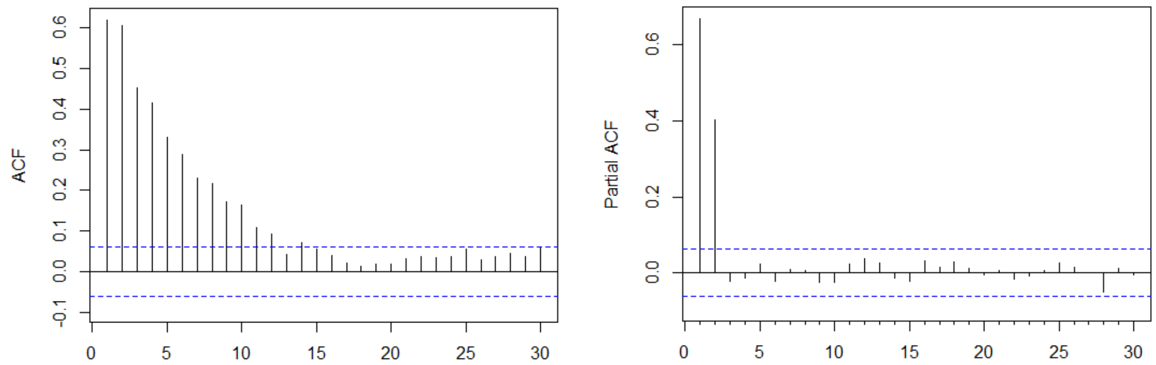
ARIMA Models: Model identification

Specifically:

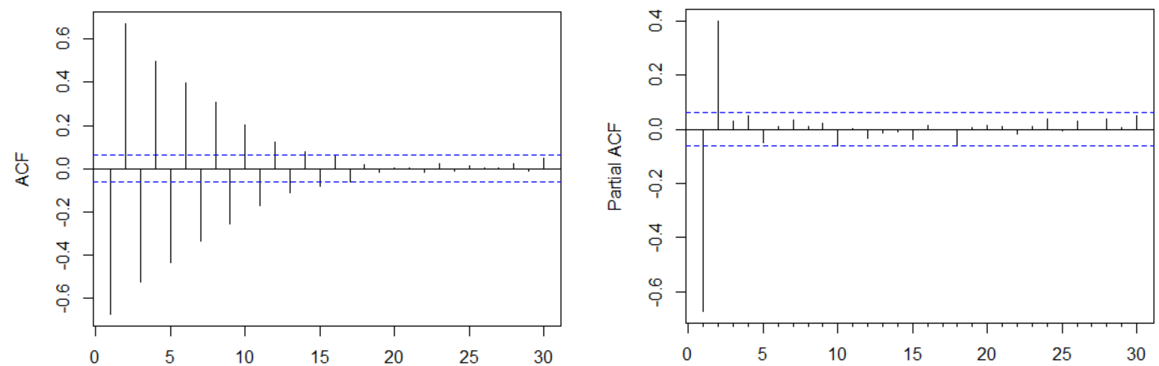
TIME SERIES	ACF	PACF
AR(1)	Exponential decay: From positive side or alternating (depending on the sign of the AR coefficient)	Peak at lag 1, then decays to zero: positive peak if the AR coefficient is positive, negative otherwise
AR(p)	Exponential decay or alternate sinusoidal decay	Peaks at lags 1 up to p
MA(1)	Peak at lag 1, then decays to zero: positive peak if the MA coefficient is positive, negative otherwise	Exponential decay: From positive side or alternating (depending on the sign of the MA coefficient)
MA(q)	Peaks at lags 1 up to q	Exponential decay or alternate sinusoidal decay

ARIMA Models: Model identification

$\boldsymbol{AR(2)}$: Φ1>0, Φ2>0

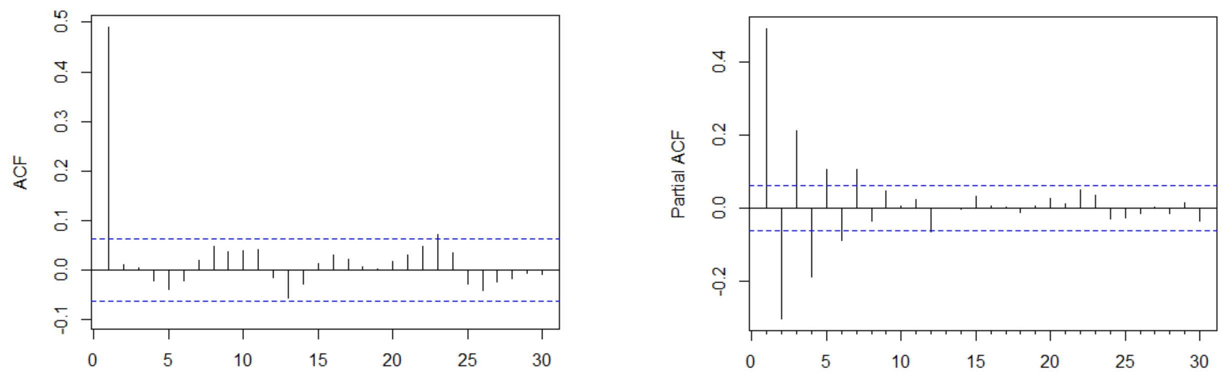


$\boldsymbol{AR(2)}$: Φ1<0, Φ2>0

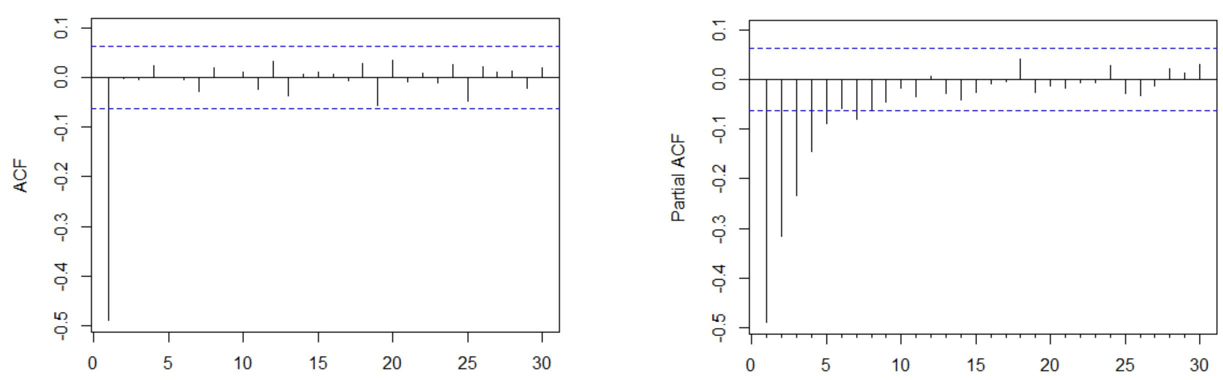


ARIMA Models: Model identification

$MA(1)$: θ1>0



$MA(1)$: θ1<0



ARIMAX Models: Adding explicative variables

A **special case** of ARIMA models allows you to generate forecasts that depend on both the historical data of the target time series (Y) and on other exogenous variables (X_k)→ we call them **ARIMAX models**

- This is not possible with other classical time series analysis techniques (e.g. ETS), where the prediction depends only on past observations of the series itself
- The advantage of ARIMAX models, therefore consists in the possibility to **include additional explanatory variables** in addition to the target dependent variable lags

$$Y_t = c + \boxed{\emptyset_1 Y_{t-1} + \ldots + \emptyset_p Y_{t-p}} + \boxed{\theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}} + \boxed{\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k} + \boxed{\varepsilon_t}$$

AUTOREGRESSIVE
the forecast depends on past observations (weighted with the regression coefficients)

MOVING AVERAGE
the forecast depends on the past errors (the difference between the observed value and estimated value)

EXPLICATIVE VARIABLES
Independent variables that provide additional information, useful to improve prediction: you can add also LAGGED effect of explicative variables!!

ERROR TERM
White noise (i.i.d, 0 mean and constant variance)

ARIMA Models: Seasonal ARIMA

A seasonal ARIMA model is formed by including **additional seasonal terms in the ARIMA models** we have seen so far

$$ARIMA\underbrace{(p,d,q)}_{\substack{\uparrow \\ \left(\text{Non-seasonal part of the model}\right)}}\ \underbrace{(P,D,Q)}_{\substack{\uparrow \\ \left(\text{Seasonal part of the model}\right)}}s$$

where s = number of periods per season (i.e. the frequency of seasonal cycle)

We use uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model

→ As usual, d / D are the number of differences/seasonal differences necessary to make the series stationary

ARIMA Models: Seasonal ARIMA identification

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF

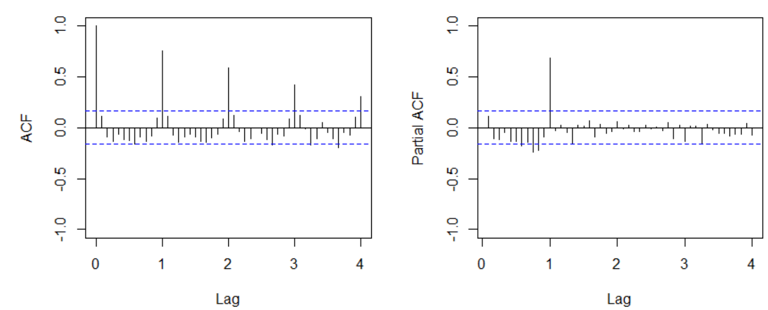
For example, an $ARIMA(0,0,0)(0,0,1)_{12}$ model will show:

- A spike at lag 12 in the ACF but no other significant spikes
- The PACF will show exponential decay in the seasonal lags; that is, at lags 12, 24, 36, …

Similarly, an $ARIMA(0,0,0)(1,0,0)_{12}$ model will show:

- Exponential decay in the seasonal lags of the ACF
- A single significant spike at lag 12 in the PACF

Example of $ARIMA(0,0,0)(1,0,0)_{12}$ process



ARIMA Models: estimation and AIC

Parameters estimation

In order to estimate an ARIMA model, normally it's used the **Maximum Likelihood Estimation (MLE)**

This technique **finds the values of the parameters which maximize the probability of obtaining the data that we have observed** → For *given values* of $(\boldsymbol{p}, \boldsymbol{d}, \boldsymbol{q})$ $(\boldsymbol{P}, \boldsymbol{D}, \boldsymbol{Q})$ (i.e. model order) the algorithm will try to **maximize the log likelihood** when finding parameter estimates

ARIMA model order

A commonly used criteria to compare different ARIMA models (i.e. with different values for $(\boldsymbol{p},\boldsymbol{q})$ $(\boldsymbol{P},\boldsymbol{Q})$ but fixed $\boldsymbol{d}$, $\boldsymbol{D}$) and to determine the optimal ARIMA order, is the **Akaike Information Criterion (AIC)**

$$\mathbf{AIC} = -2\log\left(Likelihood\right) + 2(p)$$

- where *p* is the number of estimated parameters in the model
- AIC is a goodness of fit measure
- **The best ARIMA model is that with the lower AIC** → most of automatic model selection method (e.g *auto.arima* in R) uses the AIC for determining the optimal ARIMA model order

ARIMA Model selection criteria: Manual procedure (outline)

- After preliminary analysis (and time series transformations, if needed), follow these steps:

(1) Obtain stationary series using differencing

(2) Figure out possible order(s) for the model looking at ACF (and PACF) plot

(3) Compare models from different point of view (goodness of fit, accuracy, bias, …)

(4) Examine the residuals of the best model

ARIMA Model selection criteria: Manual procedure (details)

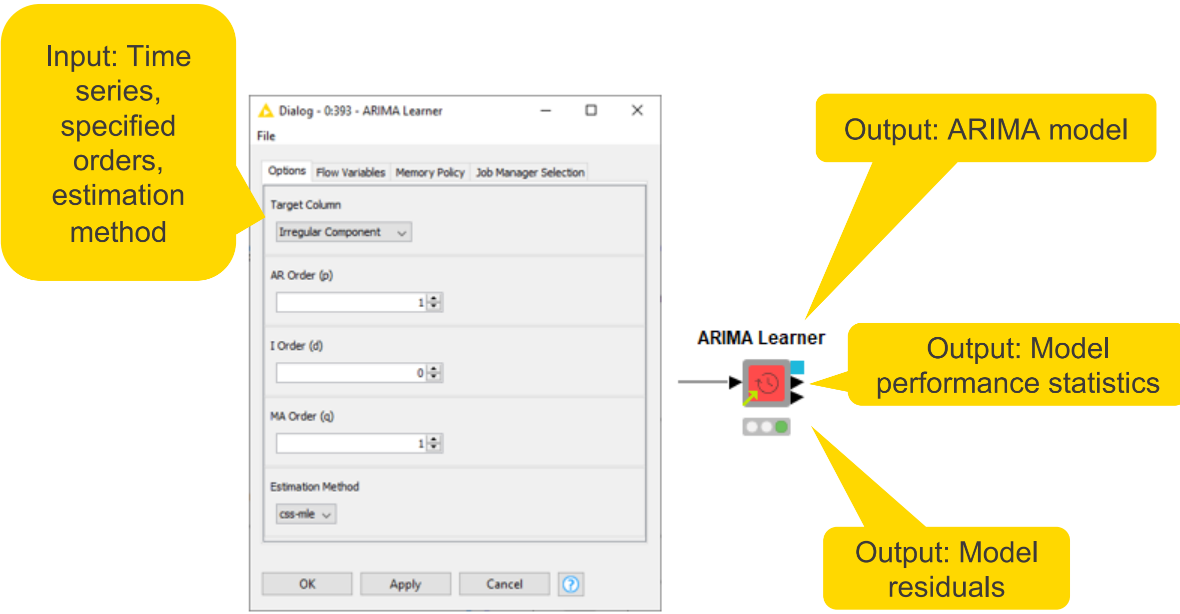
After preliminary analysis (and time series transformations, if needed), follow these steps:

1. If the series is not stationary, **use differencing (simple and/or seasonal) in order to obtain a stationary series** → together with graphical analysis, there are specific statistical tests (e.g. ADF) useful to understand if the series is stationary
2. Examine the **ACF/PACF of the stationary series and try to obtain an idea about residual structure of correlation** → Is an AR(p) / MA(q) model appropriate or you need more complex model? Do you need to model the seasonality using seasonal autoregressive lags? **It is frequent that you need to consider more candidate models to test**
3. Try your chosen model(s)*, and **use different metrics to compare the performance**:
   - Compare <u>goodness of fit </u>using AIC
   - Compare <u>accuracy</u> using measures like MAPE (in-sample and out-of-sample!)
   - Model <u>complexity</u> (simple is better!)
4. Finally, **check the residuals** from your chosen model by plotting the ACF of the residuals and doing some test on the residuals (e.g. Ljung-Box test of autocorrelation) → **they must be white noise when the model is ok!**

* Always consider slight variations of models selected in point 2: e.g. **vary one or both p and q from current model by 1**

Component: ARIMA Learner

- Learns ARIMA model of specified orders on selected target column.



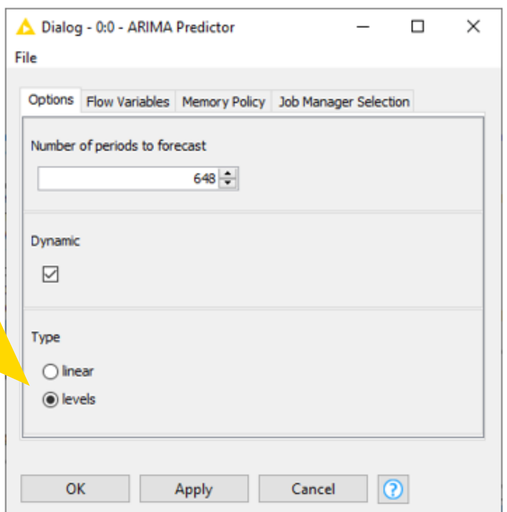
Row ID	value
RMSE	0.85
MAE	0.48
MAPE	5.47
R2	0.81
Log Likelihood	-12699.09
AIC	25406
BIC	25435
AR.L1.D.Irregular Component	0.90
AR.L1.D.Irregular Component Std Error	0.005
MA.L1.D.Irregular Component	0.008
MA.L1.D.Irregular Component Std Error	0.01

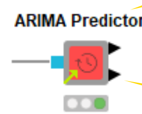
Component: ARIMA Predictor

- Generates number of forecasts set in configuration and in-sample predictions based on range used in training
- Checking the dynamic box will use predicted values for in-sample prediction

Input: ARIMA Model

Predict differenced (linear) or original (level) time series if I > 0





Output: Forecasted values and their standard errors

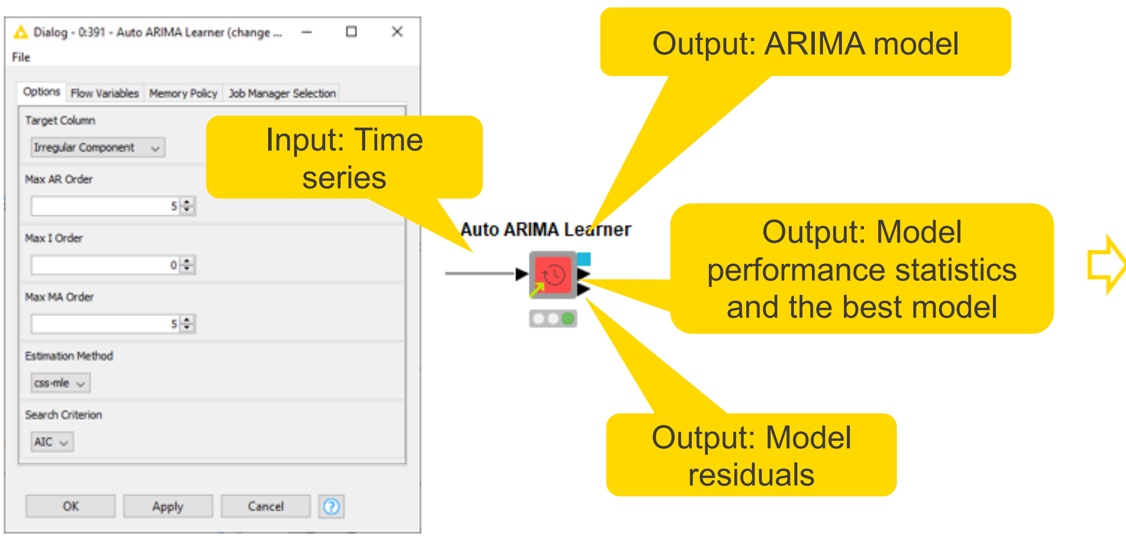
Output: In-sample predictions

out-of-sample forecast	standard error
0.415	0.849
0.373	1.145
0.334	1.337
0.300	1.473
0.269	1.574
0.242	1.651
...	...

in-sample forecast
0.017
0.015
0.014
0.013
0.011
...

Component: Auto ARIMA Learner

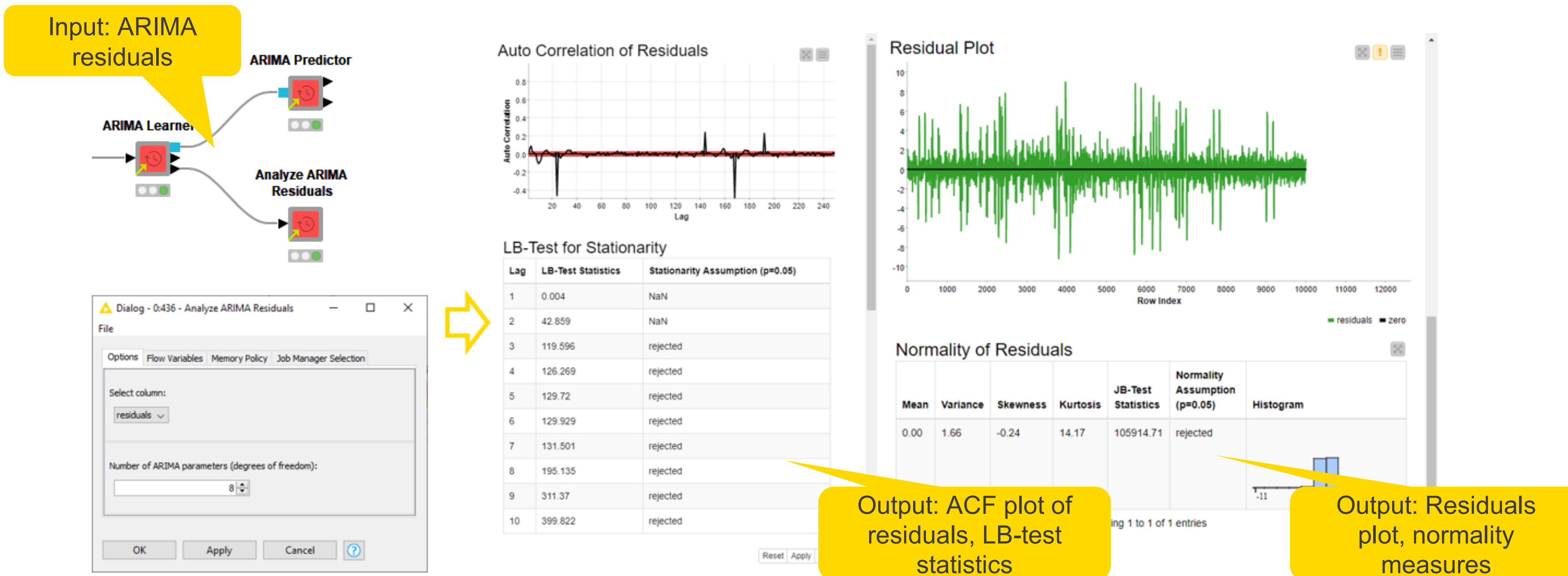
- Creates all combinations of ARIMA Models up to specified Orders.
- Select best model based on either AIC or BIC.
- ! Can take a long time to execute due to brute force approach.



Row ID	value
RMSE	0.85
MAE	0.48
MAPE	5.47
R2	0.81
ARMA(1, 0)	NaN
Log Likelihood	-12699.41
AIC	25405
BIC	25426
AR.L1.D.Irregular Component	0.90
AR.L1.D.Irregular Component Std Error	0.004

Component: Analyze ARIMA Residuals

- Inspect ACF plot, residuals plot, Ljung-Box test statistics, and normality measures → are residuals stationary and normally distributed?



ARIMA Performance Comparison

- (2,1,1) vs (1,0,0) vs (0,1,0)

ARIMA(p,d,q)	R^2	AIC	MAPE	RMSE
ARIMA(2,1,1)	0.798	25,899	6.073	0.870
ARIMA(1,0,0)	0.808	25,405	5.466	0.871
ARIMA(0,1,0)	0.798	25,924	6.048	0.871

Exercise 3: ARIMA Models

- Train a model with both the ARIMA Learner and Auto ARIMA Learner.
- Generate a Forecast for each model using the ARIMA Predictor.
- Score your forecasts.
- Analyze ARIMA residuals.

Time Series Analysis
03. ARIMA Models

Summary:
In this exercise we'll train and score two ARIMA models.

Instructions:
1) Run the workflow up through the Decompose Signal component, we'll start this exercise from here

2) Partition the data using the Partioning node. Let's use an 80/20 split. Make sure you check the box to take data from the top. This is important with time series data.

3) Apply both the ARIMA Learner and Auto ARIMA Learner components to the residual column in the output from the Decompose Signal component. Note that the Auto ARIMA can take quite a while to run, so be careful to keep the settings low for now.

4) Use an ARIMA Predictor component after the learners, you can configure the number of values you want to forecast here.

5) Attach the Forecast output from the ARIMA Predictor to the top port of the scoring metanode and the other half of our Partitioning node to the bottom. Run the scoring metanode and look at the results. Try this with different numbers of forecasted values. Do the scores change?

6) Analyze the residuals of the ARIMA model with the Analyze ARIMA Residuals component. What can you say about the residuals?

KNIME Time Series Analysis Course - Session 4

KNIME AG

Agenda

1. Introduction: What is Time Series Analysis
2. Today's Task, Dataset & Components
3. Descriptive Analytics: Load, Clean, Explore
4. Descriptive Analytics: Non-stationarity, Seasonality, Trend
5. Quantitative Forecasting: Classical techniques
6. ARIMA Models: ARIMA(p,d,q)
7. Machine Learning based Models
8. Hyperparameter Optimization
9. Quick Look at LSTM Networks
10. Example of Time Series Analysis on Spark
11. Conclusions & Summary

Exercise 3: ARIMA Models

- Train a model with both the ARIMA Learner and Auto ARIMA Learner.
- Generate a Forecast for each model using the ARIMA Predictor.
- Score your forecasts.
- Analyze ARIMA residuals.

Time Series Analysis
03. ARIMA Models

Summary:
In this exercise we'll train and score two ARIMA models.

Instructions:
1) Run the workflow up through the Decompose Signal component, we'll start this exercise from here

2) Partition the data using the Partioning node. Let's use an 80/20 split. Make sure you check the box to take data from the top. This is important with time series data.

3) Apply both the ARIMA Learner and Auto ARIMA Learner components to the residual column in the output from the Decompose Signal component. Note that the Auto ARIMA can take quite a while to run, so be careful to keep the settings low for now.

4) Use an ARIMA Predictor component after the learners, you can configure the number of values you want to forecast here.

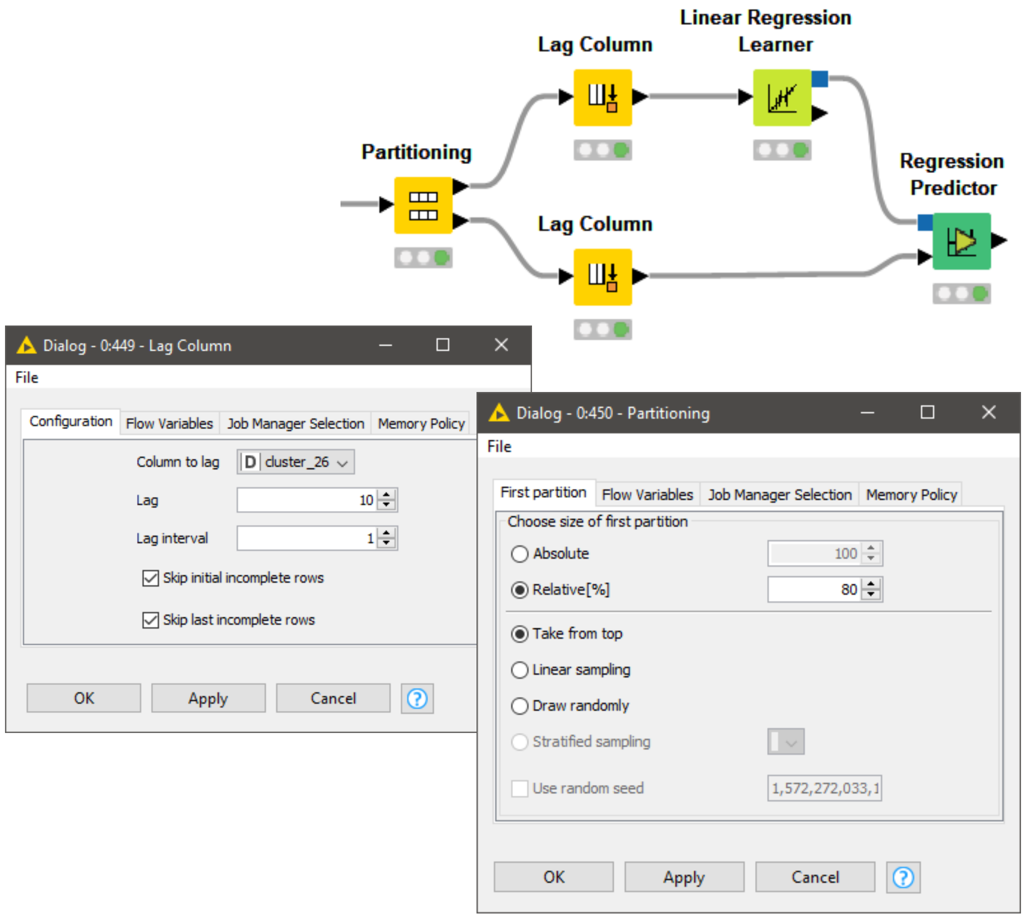
5) Attach the Forecast output from the ARIMA Predictor to the top port of the scoring metanode and the other half of our Partitioning node to the bottom. Run the scoring metanode and look at the results. Try this with different numbers of forecasted values. Do the scores change?

6) Analyze the residuals of the ARIMA model with the Analyze ARIMA Residuals component. What can you say about the residuals?

Machine Learning based Models
Lag Column + Regressions

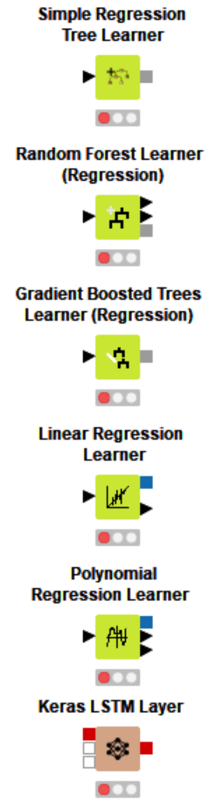
Using Machine Learning Techniques

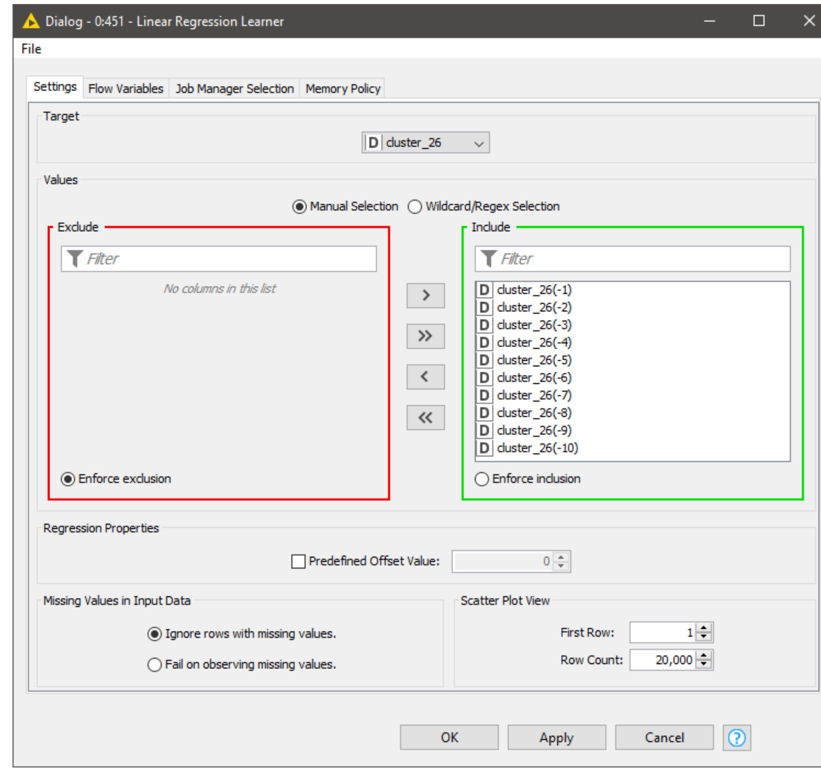
- Use Lag Column Node(s) to create features
- Lagged Columns for input
- Original Column for target
- When Partitioning make sure data is sorted and take from top



Useful Models on lagged inputs

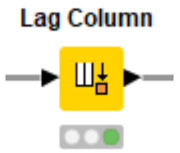
- Regression Trees and Forests
- Linear and Polynomial Regression
- Deep Learning
- Options with Spark, H2O, XGBoost, Keras, and TensorFlow

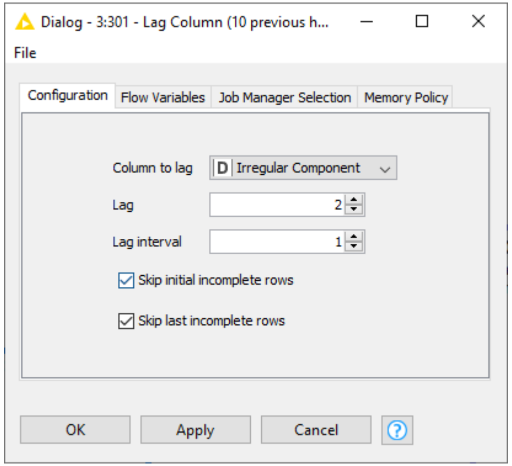


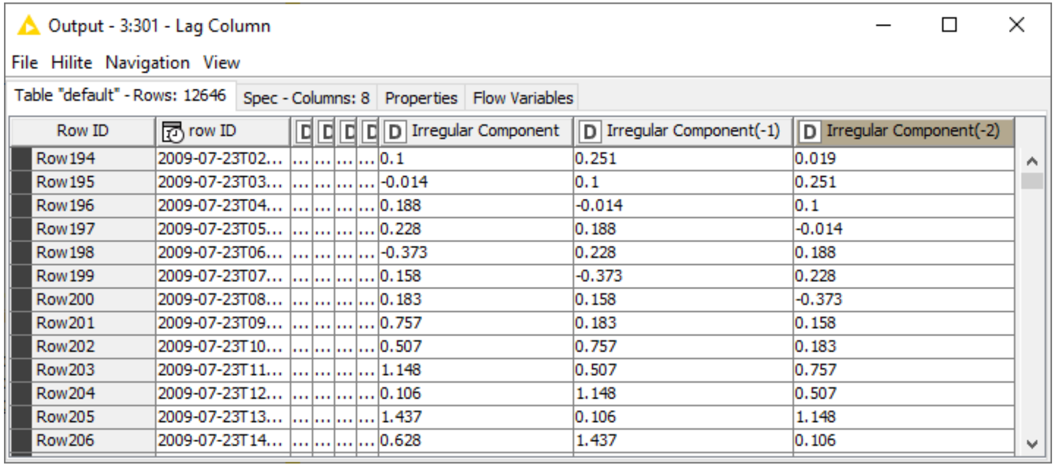


Recap: Lag Column Node

- Append past values as new columns
  - Shift cells *l* (lag interval) steps up
  - Duplicate the lag column *L* (lag value) times.
    In each column the rows are shifted *l, 2*l, .., L*l* steps up







Row ID	row ID	D	D	D	D	Irregular Component	Irregular Component(-1)	Irregular Component(-2)
Row194	2009-07-23T02...	...	...	...	...	0.1	0.251	0.019
Row195	2009-07-23T03...	...	...	...	...	-0.014	0.1	0.251
Row196	2009-07-23T04...	...	...	...	...	0.188	-0.014	0.1
Row197	2009-07-23T05...	...	...	...	...	0.228	0.188	-0.014
Row198	2009-07-23T06...	...	...	...	...	-0.373	0.228	0.188
Row199	2009-07-23T07...	...	...	...	...	0.158	-0.373	0.228
Row200	2009-07-23T08...	...	...	...	...	0.183	0.158	-0.373
Row201	2009-07-23T09...	...	...	...	...	0.757	0.183	0.158
Row202	2009-07-23T10...	...	...	...	...	0.507	0.757	0.183
Row203	2009-07-23T11...	...	...	...	...	1.148	0.507	0.757
Row204	2009-07-23T12...	...	...	...	...	0.106	1.148	0.507
Row205	2009-07-23T13...	...	...	...	...	1.437	0.106	1.148
Row206	2009-07-23T14...	...	...	...	...	0.628	1.437	0.106

Hyperparameter Optimization

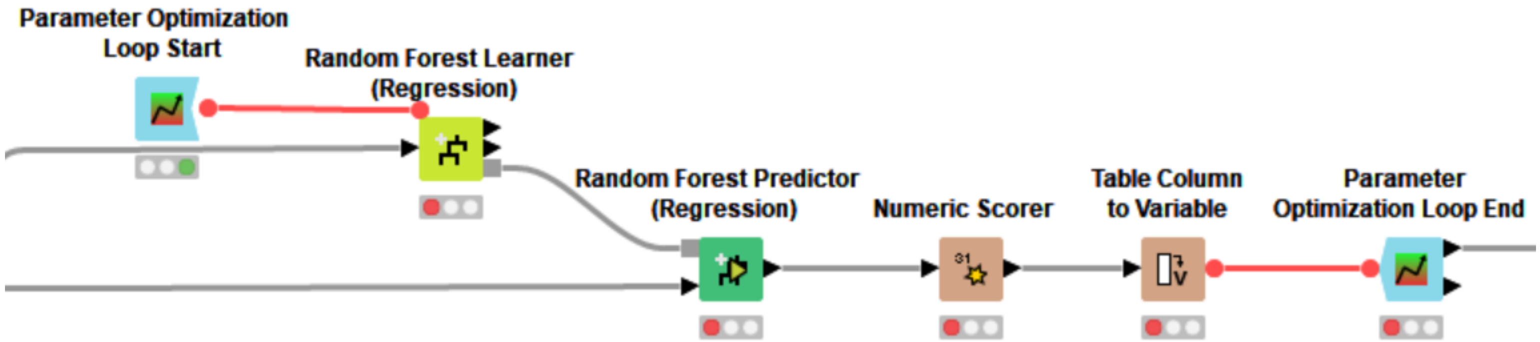
Performance on Tree Depth

- 1 v 10 v 100 performance
- Can we automate the selection?

Max Tree Depth	R^2	MAPE	RMSE
1	0.495	8.65	2.45
10	0.954	2.918	.736
100	0.957	3.16	0.713

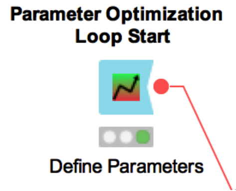
Hyperparameter Optimization

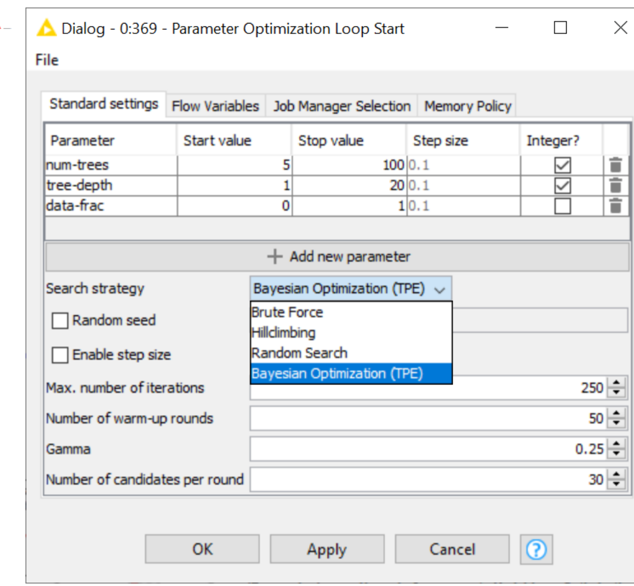
- Some modeling approaches are very sensitive to their configuration.
- Calculating optimum settings is not always possible.
- Hyperparameter Optimization loops may help find a good configuration



New Node: Parameter Optimization Loop Start

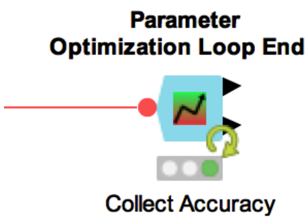
- Define some parameters to optimize
- Set upper/lower bounds and step sizes (and flag integers)
- Choose an optimization method
  - Brute force for maximum accuracy but slower computation
  - Hillclimbing for better faster runtimes but may get stuck in local optimum settings
  - Random search to randomly search for parameter values within a given range
  - Bayesian Optimization (TPE)

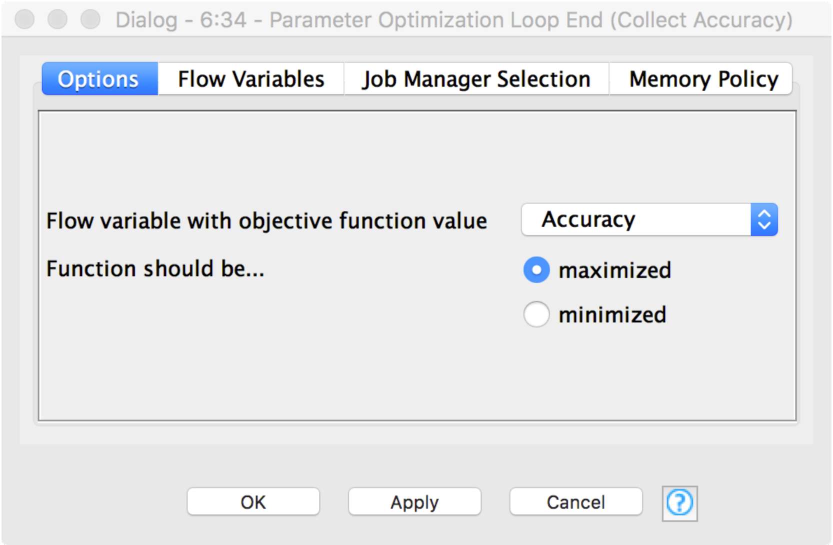




New Node: Parameter Optimization Loop End

- Collects a value to optimize as flow variable.
- Value may be maximized (accuracy) or minimized (error)





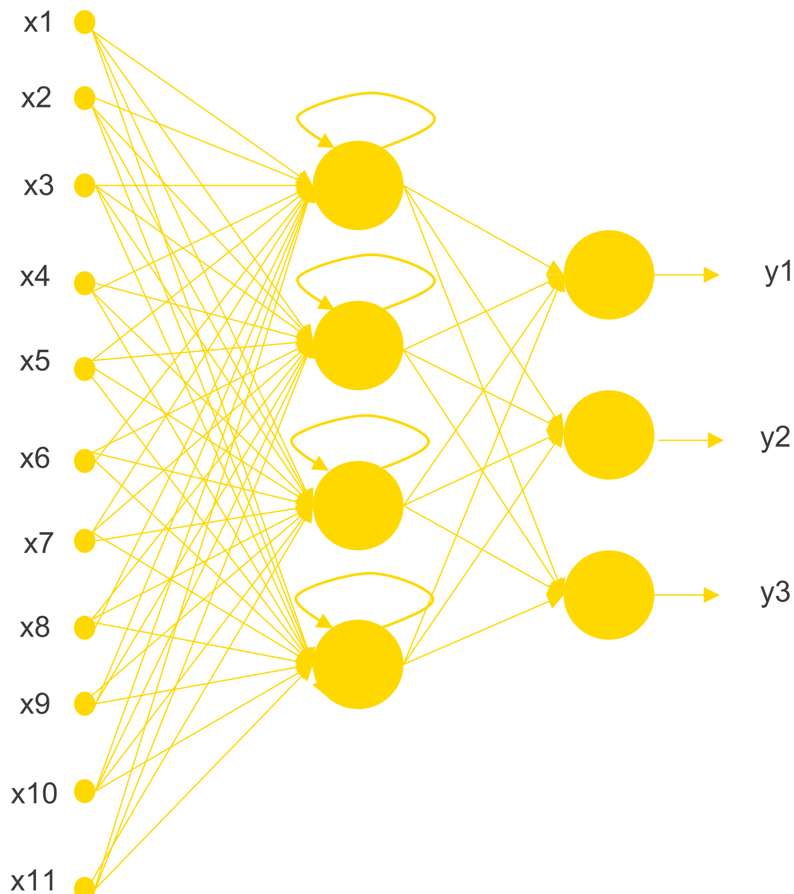
Time Series Analysis with LSTM Units in Deep Learning Networks

What is Deep Learning?

- Deep Learning extends the family of Artificial Neural Networks with:
  - Deeper architectures
  - A few additional paradigms, e.g. Recurrent Neural Networks (RNNs)

- The algorithms to train such networks are not new, but they have been enabled by recent advances in hardware performance and parallel execution.

Feed-Forward vs. Recurrent Neural Networks



Unrolling RNNs through time

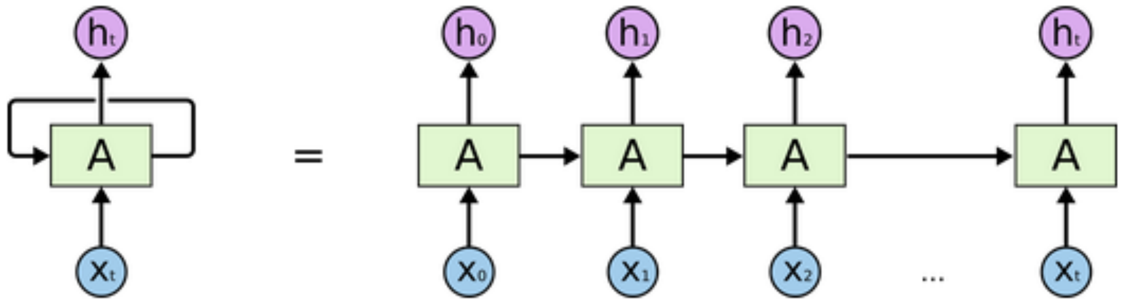


Image Source: Christopher Olah, "Understanding LSTM Networks"
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Capture the dynamics of a sequence through a loop connection
- RNNs are not constrained to a fixed sequence size
- Trained via BPTT (Back-Propagation Through Time)

LSTM = Long Short Term Memory Unit

- Special type of units with three gates
  - Input gate
  - Forget gate
  - Output gate

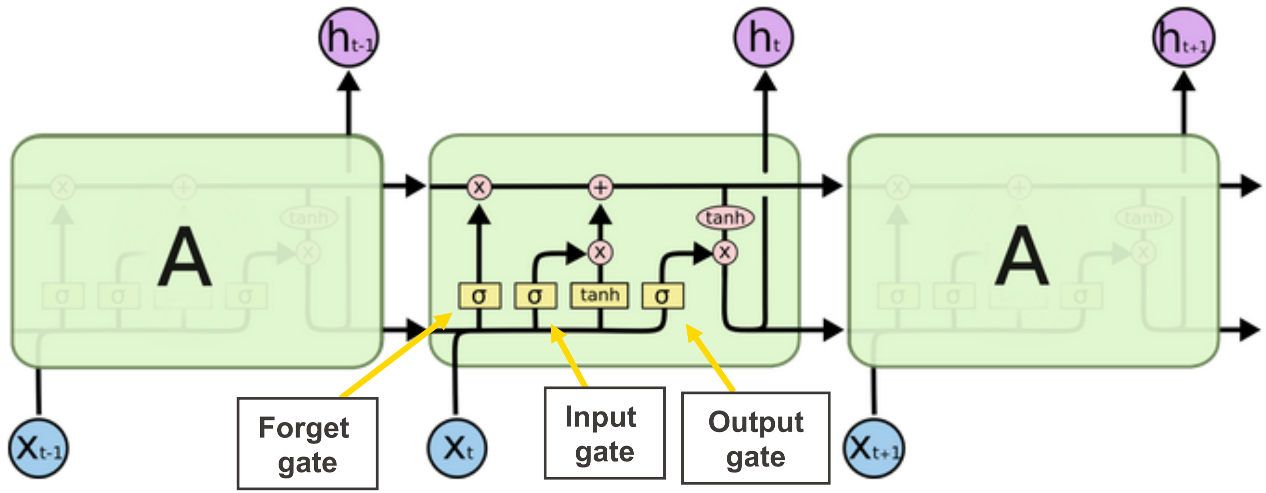
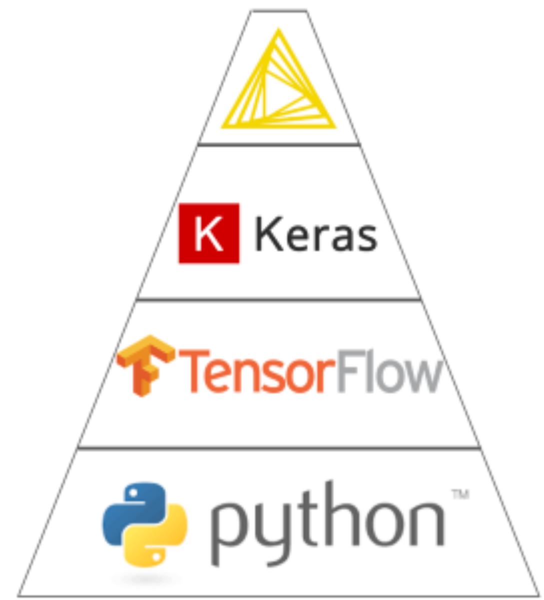


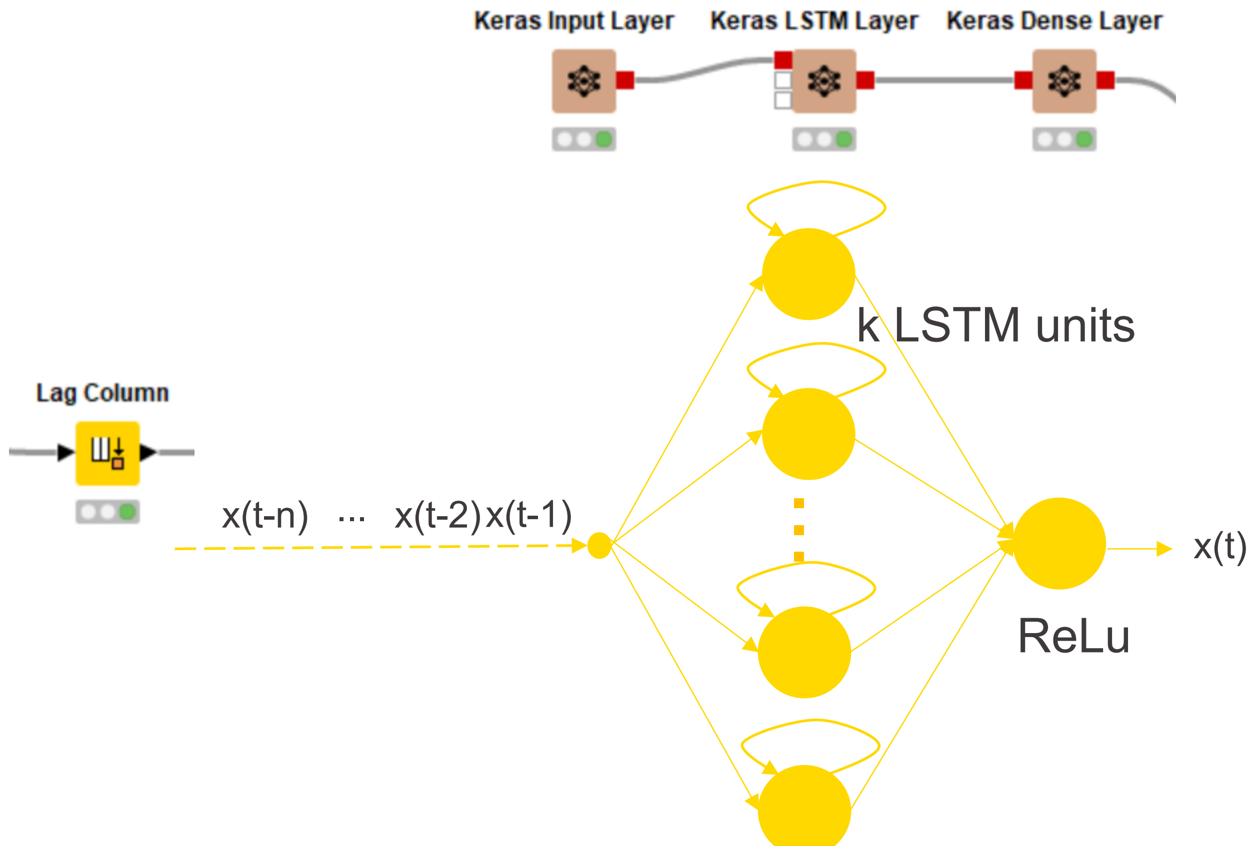
Image Source: Christopher Olah, “Understanding LSTM Networks”
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The KNIME Keras Integration for Deep Learning

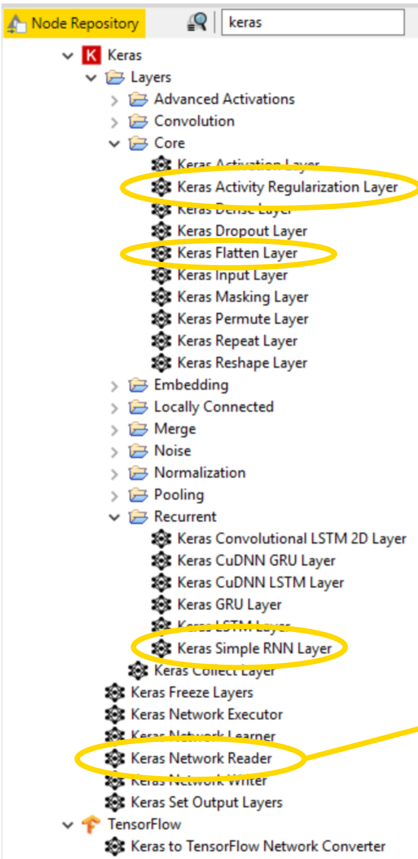
- The KNIME Deep Learning intergation that we will use is based on Keras
- We need to install:
  - KNIME Deep Learning Keras Integration
  - Keras
  - Python
- The Keras integration includes:
  - Activation Functions
  - Neural Layers (many!!!)
  - Learners / Executors
  - Layer Freezer
  - Network Reader/Writer
  - Network Converter to TensorFlow



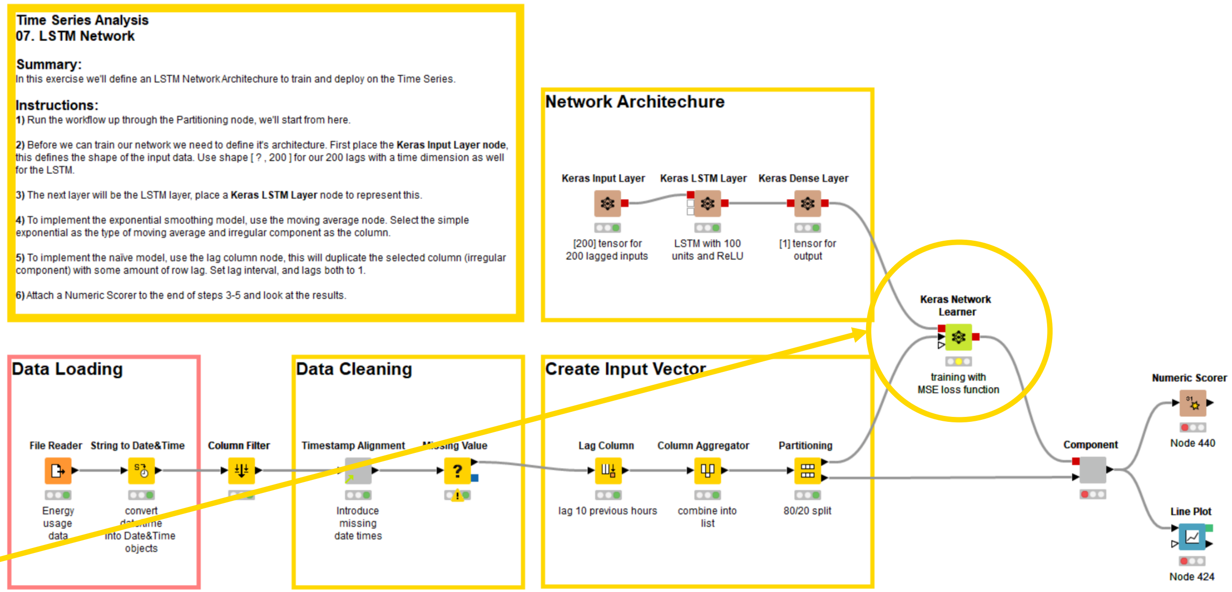
LSTM based Architecture



LSTM based Networks: Deep Learning nodes



In: Supplementary Workflows/02_LSTM_Networks



Results from LSTM based Network

- Out-sample Static testing

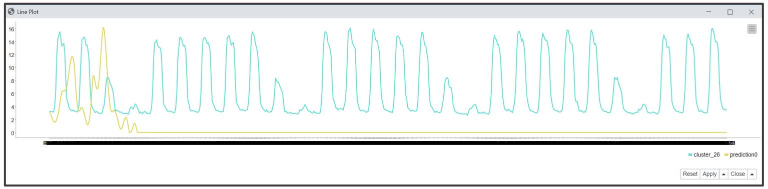
Past	RMSE	MAE	MAPE	R^2
10h	0.646	0.424	0.064	0.985
100h	0.474	0.312	0.047	0.992
200h	0.522	0.347	0.051	0.992

- Out-sample Dynamic testing

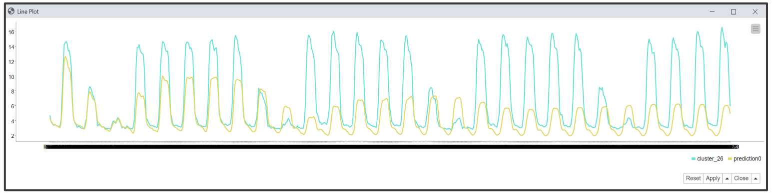
Past	RMSE	MAE	MAPE	R^2
10h	7.911	6.521	0.988	-2.115
100h	4.107	2.637	0.286	0.167
200h	2.624	1.742	0.225	0.662

Past Out-sample Dynamic testing

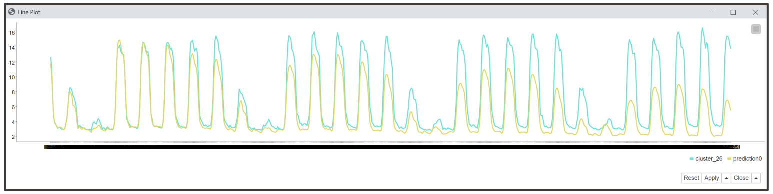
10 hours



100 hours

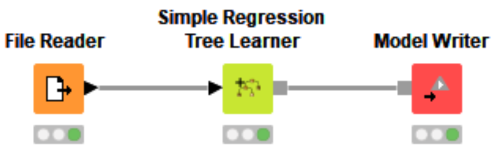


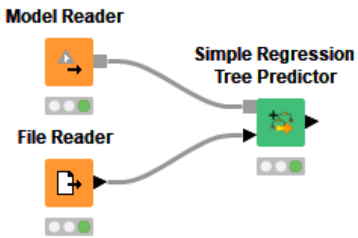
200 hours



Deployment

Saving and Reading Models

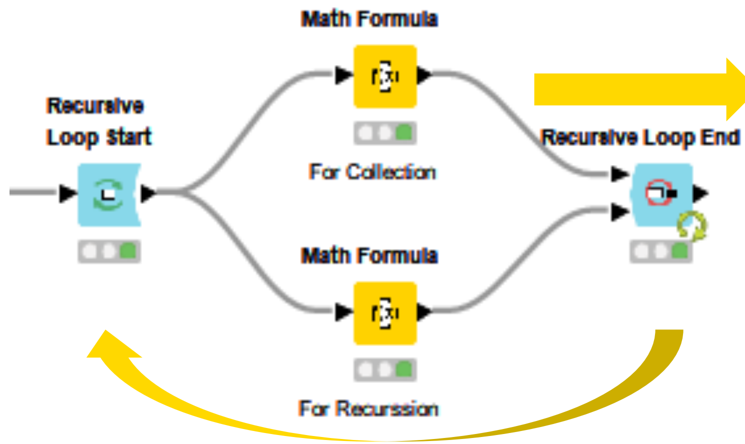




- Model Writer
  - After training your model attach the output of your learner node to the Model Writer to save your trained model.

- Model Reader
  - Use the Model Reader node to load a saved model and attach this to your Predictor for use in deployment.

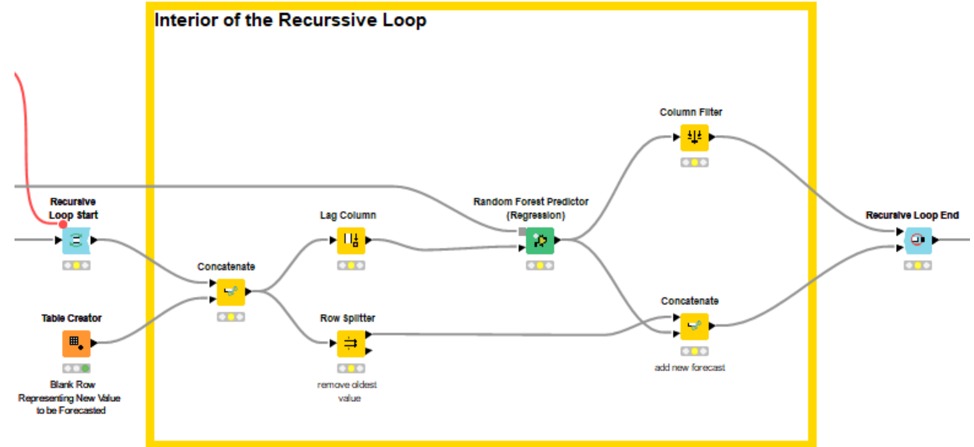
Recursive Loop Nodes



- The Recursive Loop Start and End nodes pass data back to the start of the loop with every iteration.
- This enables us to generate predictions based on predictions.

Model Deployment Workflow

- Generate dynamic predictions for a selected forecast horizon
- How well does the Forecast hold up on dynamic predictions?

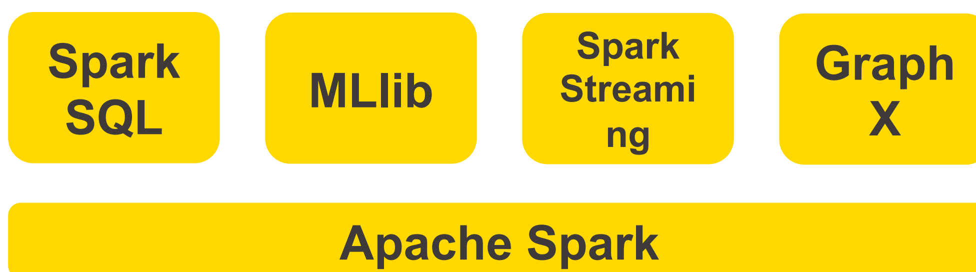


In: Supplementary Workflows/03_Deployment_and_Signal_Reconstruction

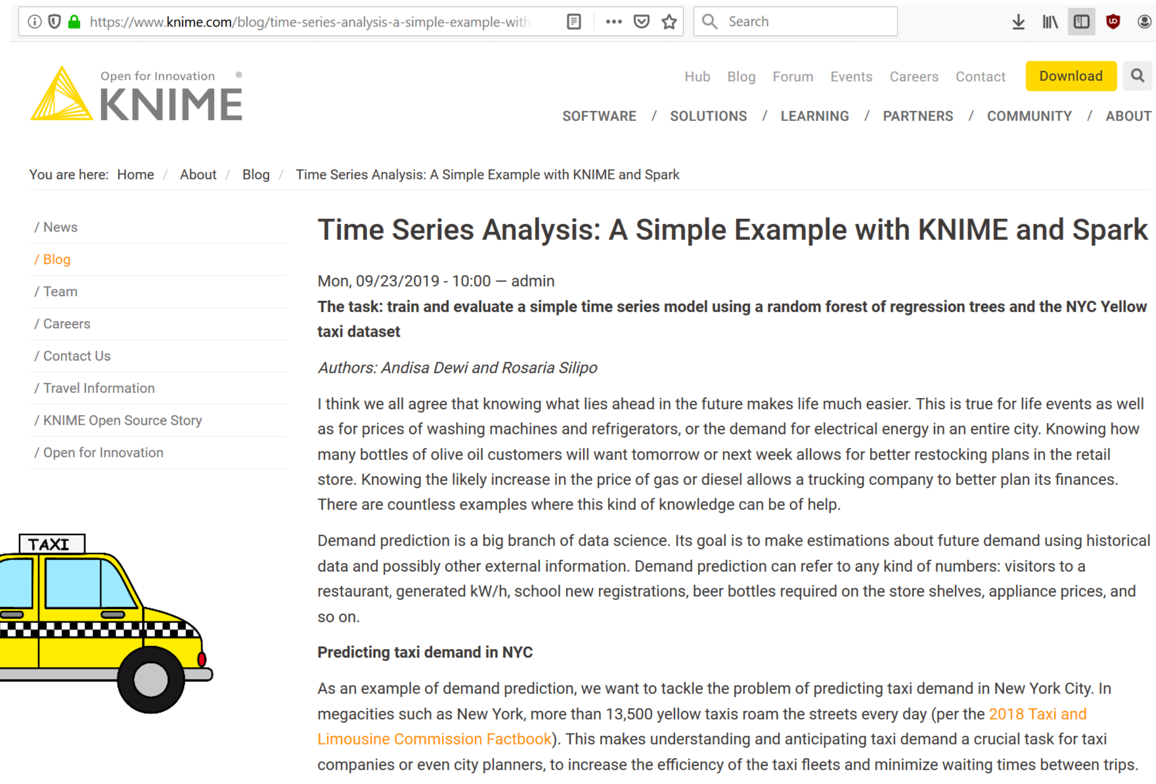
Time Series Analysis on Spark

What is Spark? And why should we use it?

- Spark is a general-purpose **distributed** data processing engine.
- Application developers incorporate Spark into their applications to rapidly query, analyze, and transform data **at scale**.
- Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, and machine learning tasks.



Taxi Demand Prediction on Spark – KNIME Blog



https://www.knime.com/blog/time-series-analysis-a-simple-example-with

Open for Innovation KNIME

Hub Blog Forum Events Careers Contact Download

SOFTWARE / SOLUTIONS / LEARNING / PARTNERS / COMMUNITY / ABOUT

You are here: Home / About / Blog / Time Series Analysis: A Simple Example with KNIME and Spark

/ News
/ Blog
/ Team
/ Careers
/ Contact Us
/ Travel Information
/ KNIME Open Source Story
/ Open for Innovation

Time Series Analysis: A Simple Example with KNIME and Spark

Mon, 09/23/2019 - 10:00 — admin

The task: train and evaluate a simple time series model using a random forest of regression trees and the NYC Yellow taxi dataset

Authors: Andisa Dewi and Rosaria Silipo

I think we all agree that knowing what lies ahead in the future makes life much easier. This is true for life events as well as for prices of washing machines and refrigerators, or the demand for electrical energy in an entire city. Knowing how many bottles of olive oil customers will want tomorrow or next week allows for better restocking plans in the retail store. Knowing the likely increase in the price of gas or diesel allows a trucking company to better plan its finances. There are countless examples where this kind of knowledge can be of help.

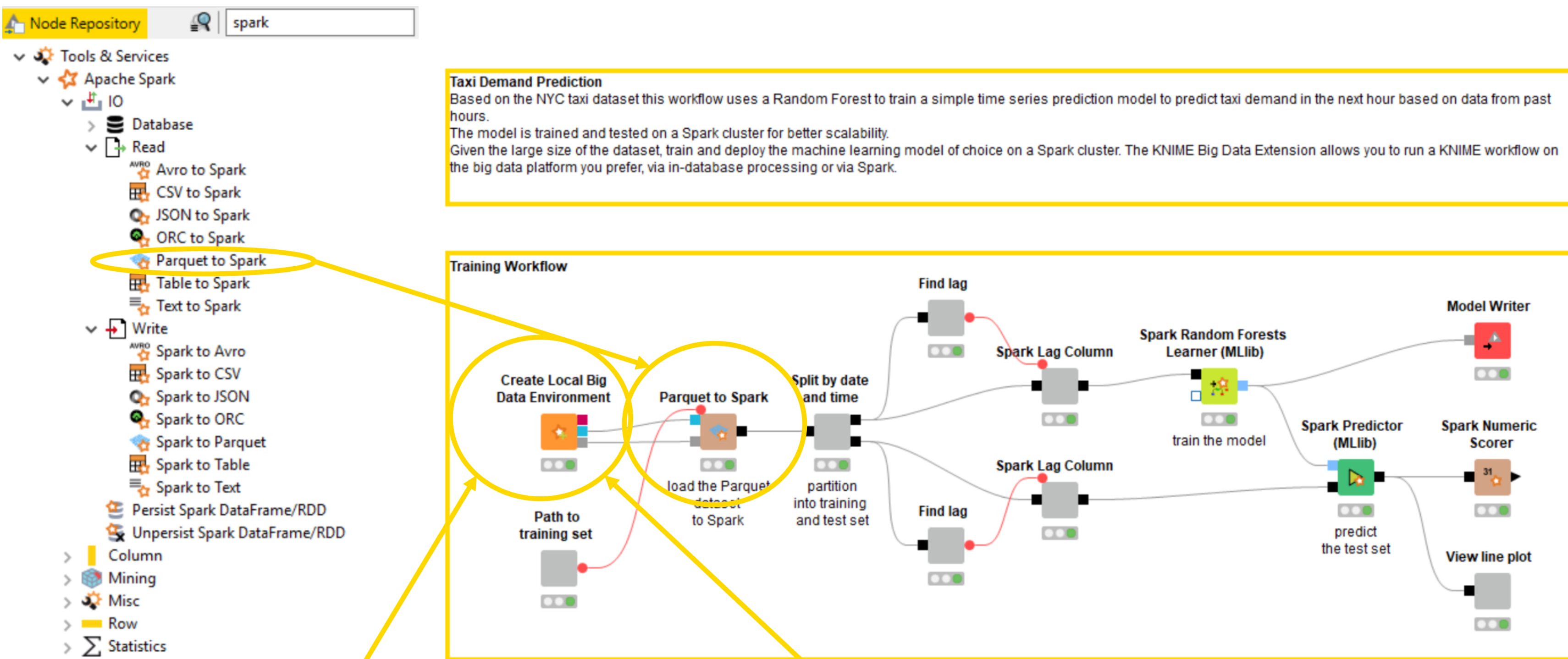
Demand prediction is a big branch of data science. Its goal is to make estimations about future demand using historical data and possibly other external information. Demand prediction can refer to any kind of numbers: visitors to a restaurant, generated kW/h, school new registrations, beer bottles required on the store shelves, appliance prices, and so on.

Predicting taxi demand in NYC

As an example of demand prediction, we want to tackle the problem of predicting taxi demand in New York City. In megacities such as New York, more than 13,500 yellow taxis roam the streets every day (per the 2018 Taxi and Limousine Commission Factbook). This makes understanding and anticipating taxi demand a crucial task for taxi companies or even city planners, to increase the efficiency of the taxi fleets and minimize waiting times between trips.

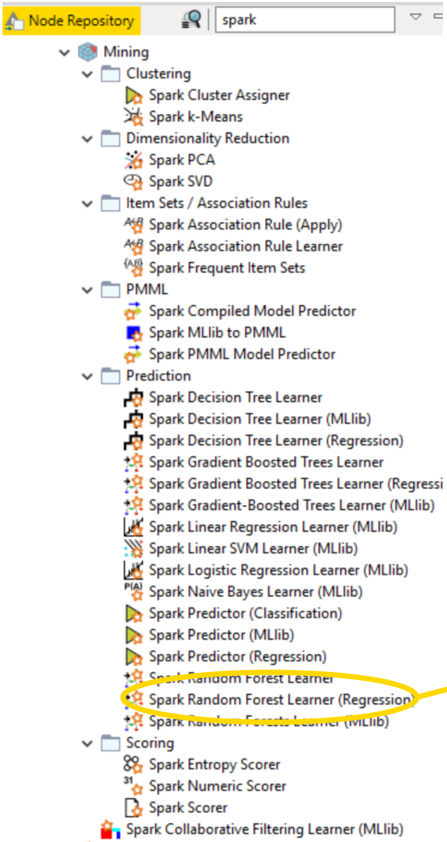
https://www.knime.com/blog/time-series-analysis-a-simple-example-with-knime-and-spark

Taxi Demand Prediction: Connect to Spark

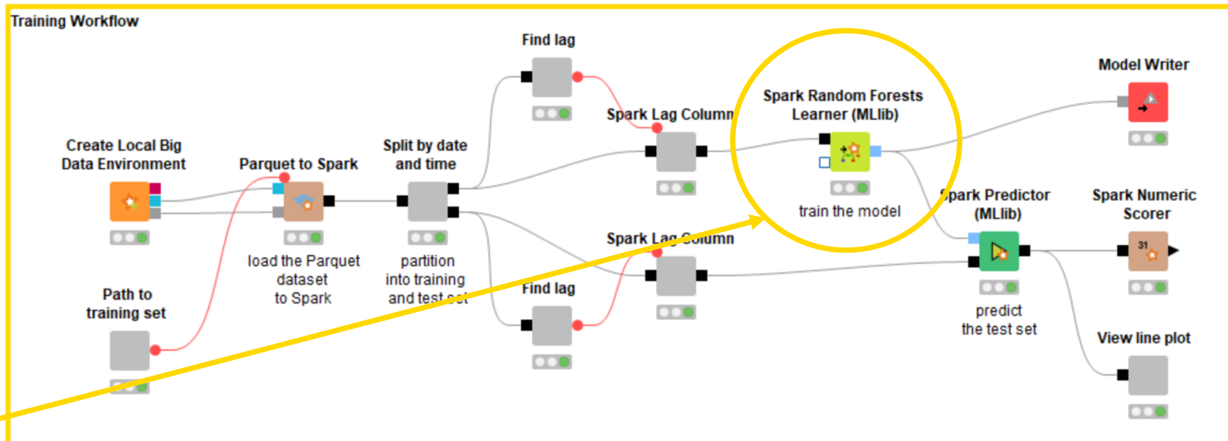


Creates a local Spark cluster.
To connect to an external Spark cluster use „Create Spark Context“

Taxi Demand Prediction: Training

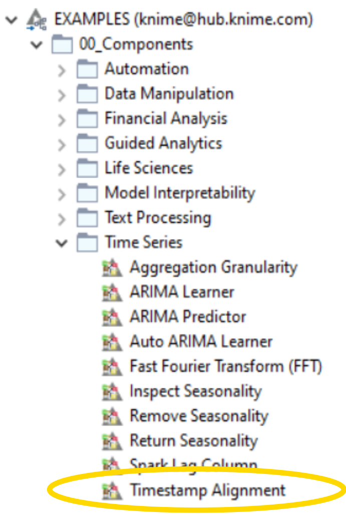


Taxi Demand Prediction
Based on the NYC taxi dataset this workflow uses a Random Forest to train a simple time series prediction model to predict taxi demand in the next hour based on data from past hours.
The model is trained and tested on a Spark cluster for better scalability.
Given the large size of the dataset, train and deploy the machine learning model of choice on a Spark cluster. The KNIME Big Data Extension allows you to run a KNIME workflow on the big data platform you prefer, via in-database processing or via Spark.

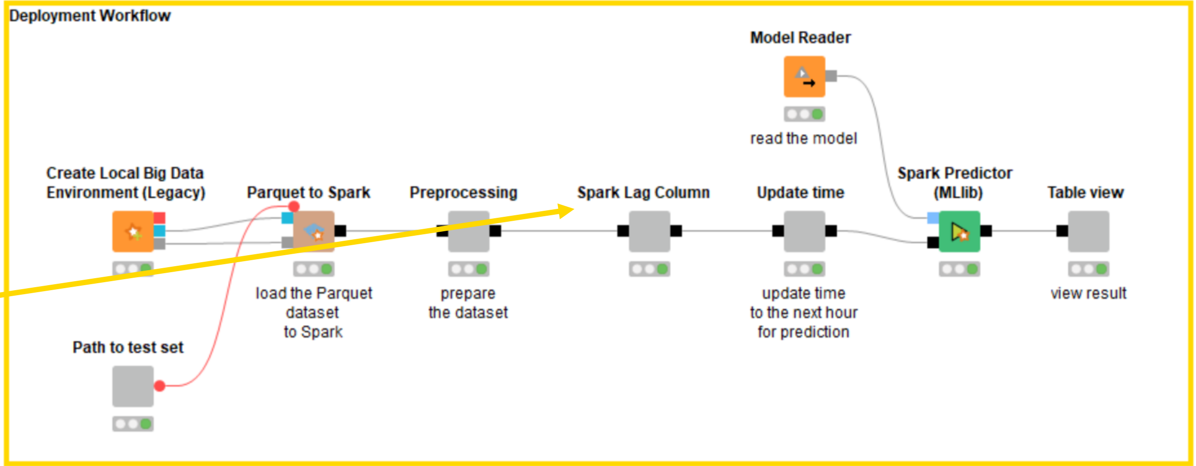


Open for Innovation KNIME

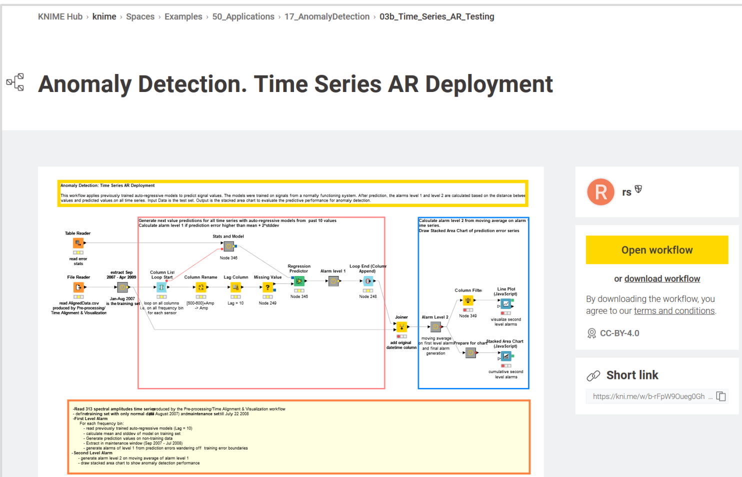
Taxi Demand Prediction: Deployment



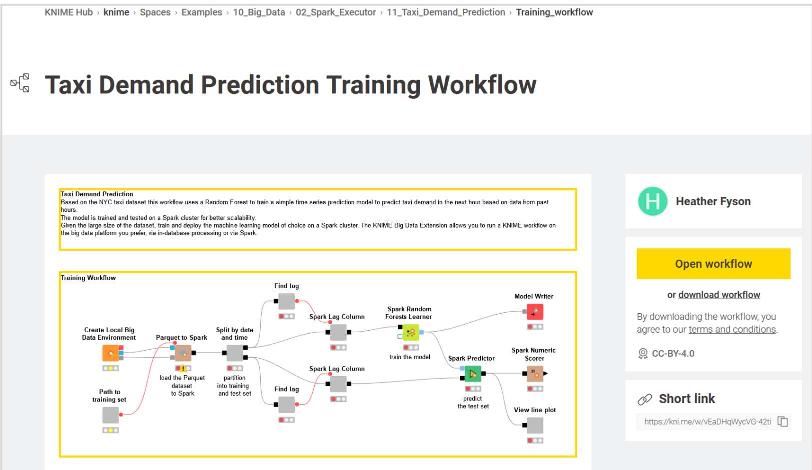
Taxi Demand Prediction
Based on the NYC taxi dataset this workflow uses a Random Forest to train a simple time series prediction model to predict taxi demand in the next hour based on data from past hours.
The model is trained and tested on a Spark cluster for better scalability.
Given the large size of the dataset, we train and deploy the machine learning model of choice on a Spark cluster. The KNIME Big Data Extension allows you to run a KNIME workflow on the big data platform you prefer, via in-database processing or via Spark.



IoT References on the KNIME Hub



https://kni.me/w/b-rFpW9Oueg0GhuN



https://kni.me/w/vEaDHqWycVG-42ti

Exercise 4: Machine Learning

- Predict the residual of the energy consumption with a Random Forest model and Linear Regression model
  - Use ten past values for prediction
- Evaluate the prediction results

Time Series Analysis
04. Machine Learning

Summary:
In this exercise we'll train and score a Random Forest and Linear Regression

Instructions:
1) Run the workflow up through the Decompose Signal component, we'll start this exercise from here

2) Use the Lag Column node with Lag Interval = 1 and Lags = 10. We'll use these 10 past values as the inputs for our models.

3) Partition the data using the Partioning node. Let's use an 80/20 split. Make sure you check the box to take data from the top. This is important with time series data.

4) Apply both the Linear Regression Learner and Random Forest Learner (Regression) to the top port of the Partioning node. Make sure your target is Residual and your inputs are the lagged values: Residual(-n)

5) Use the Random Forest Predictor (Regression) and the Regression Predictor nodes after their respective learners. Use the data from the bottom port of our Partitioning node for the input.

6) Apply the Numeric Scorer node to the output of both predictors and see how they did

Exercise 5: Hyper Parameter Optimization

- Find for the best number of trees and tree depth that give the highest accuracy of the Random Forest model. Test the following values:
  - Number of trees: min=5, max=100
  - Tree depth: min = 1, max = 20
- Optional: Train a Random Forest model using the best performing parameters

Time Series Analysis
05. Hyper Parameter Optimization

Summary:
In this exercise we'll optimize some of the hyper parameters in our Random Forest model.

Instructions:
1) Run the workflow up through the Random Forest Predictor, we'll start from here

2) Attach a Numeric Scorer to the output of the Predictor, verify the reference and prediction column are correct in the configuration

3) After the Scorer attach a Table Column to Variable node, we'll need these scores as flow variables later to select the best parameters
*Note that we use the Table Column to Variable instead of Table Row to Variable because our metrics are all in one column.

4) Next we'll add the Parameter Optimization Loop Start node to our workflow. It's output is a flow variable port. Attach this to the Random Forest Learner.

5) To configure the Parameter Optimization Loop Start we'll add new variables to the table in its configuration. These will represent the range of values we want to try when training.
Create one with the name: NumTrees, with min value 5 and max value 100
Create another with the name: TreeDepth with min value 1 and max value 20
Check the box to indicate both are integers
**Execute this node so you see your Flow Variables in the next step.

6) Next configure the Random Forest Learner to use these flow variables. Open the configuration window for the Learner and go to the Flow Variables tab.
In the drop down box next to maxLevels select your TreeDepth flow variable, and in the box next to nrModels select NumTrees. This will instruct KNIME to control those model parameters with your flow variables.

7) Finally add the Parameter Optimization Loop End to the end of your workflow. Attach the output of your Table Column to Variable node to it.
In the configuration window for the Loop End node you can select which metric to optimize for. We'll use Mean Absolute Percentage Error.

Optional) Train a model with the optimized parameters from the loop

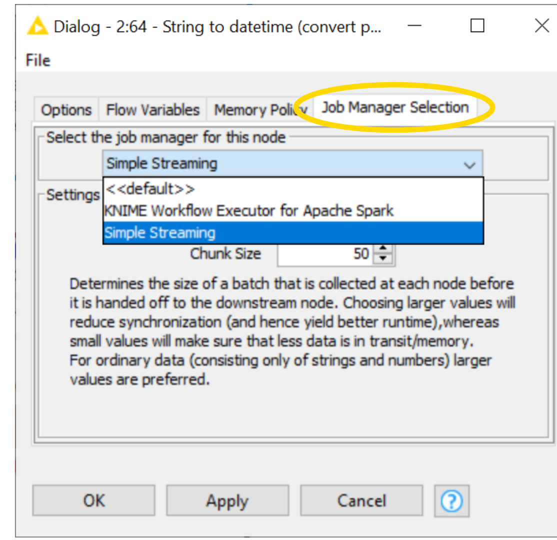
Real Time Streaming

What is Real-Time Streaming?

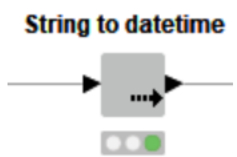
- In *real-time* data *streaming*, big volumes of data are received and processed quickly as soon as they are available.
- “Quickly” and “as soon as available” are two important factors, since they allow a reaction to changing conditions *in real time*.

Simple Streaming Execution – In batches

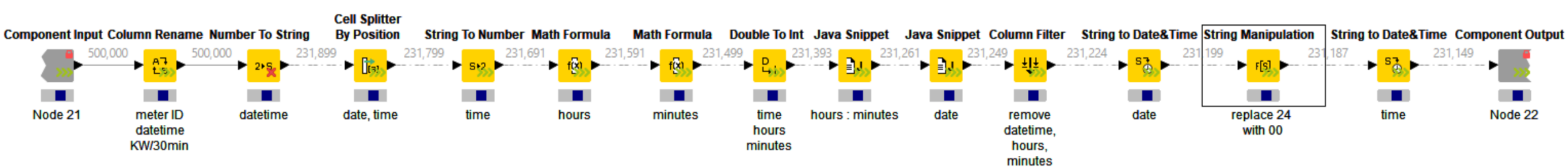
- When the first node has processed the first batch, it passes it to the next node which can then already begin with its processing.



Only for Components

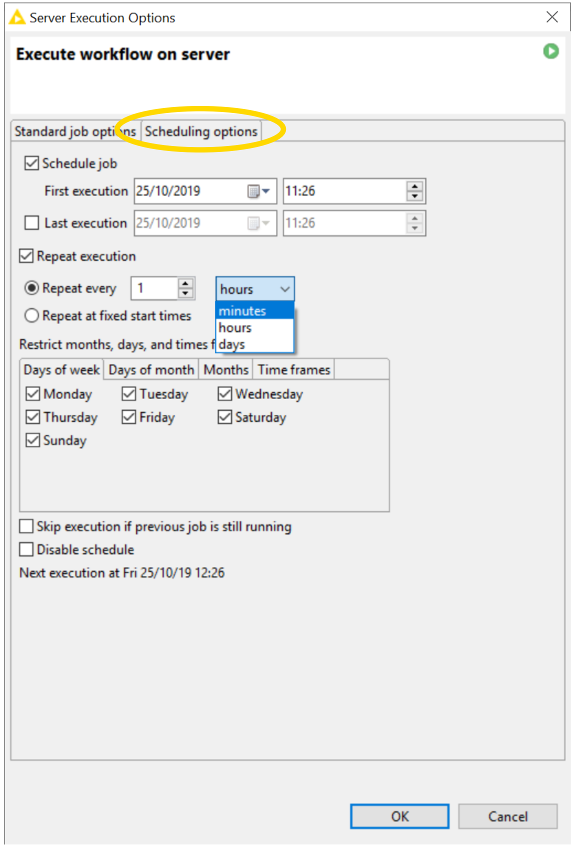


Only for most Nodes



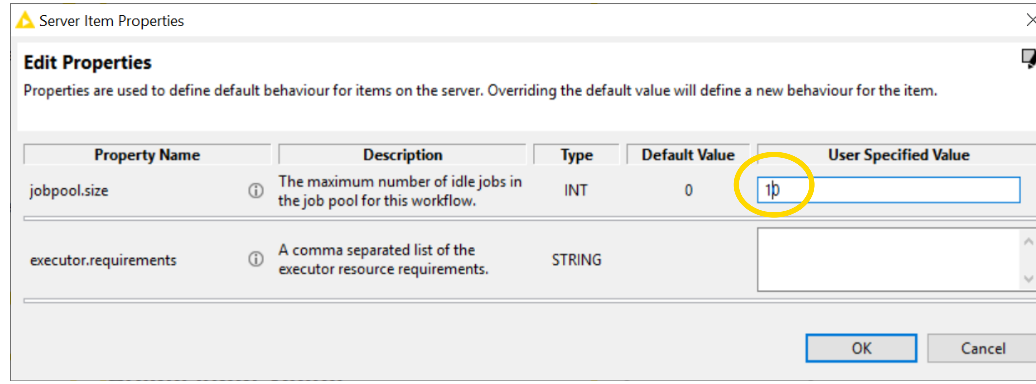
Streaming Solution: via Scheduler in KNIME Server

- Smallest time resolution: by the Minute
- Sometimes this is enough.

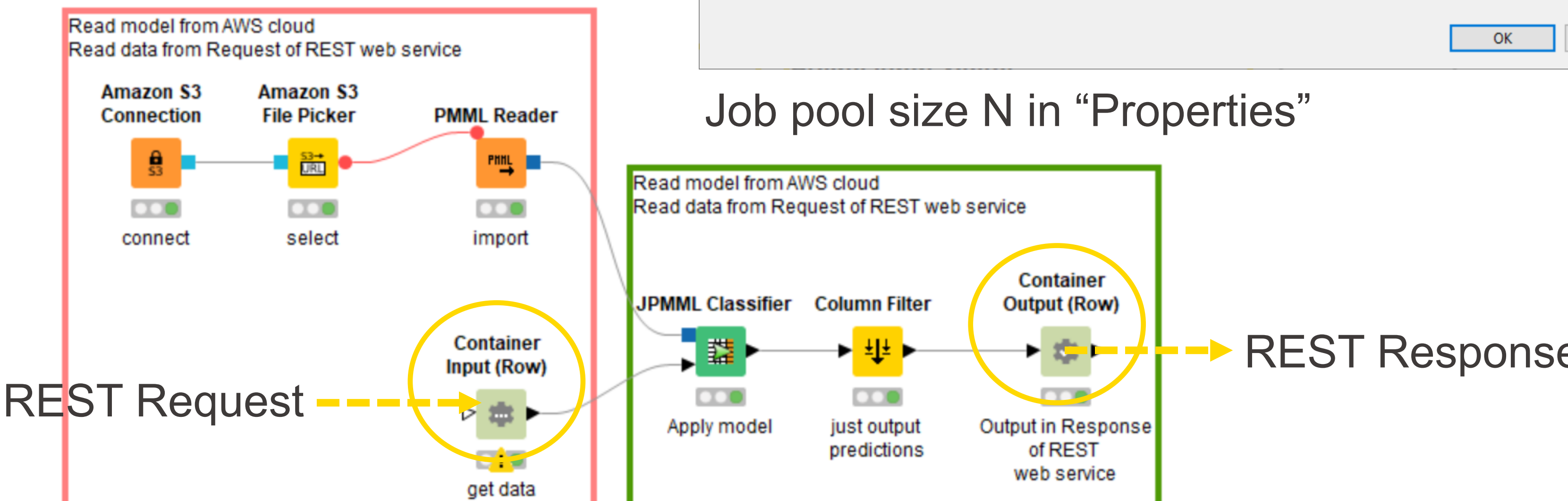


Streaming Solution: on demand via REST service

- The Job Pool keeps max. N active REST jobs at the same time. This speeds up the execution ofup to N concurrent REST calls.

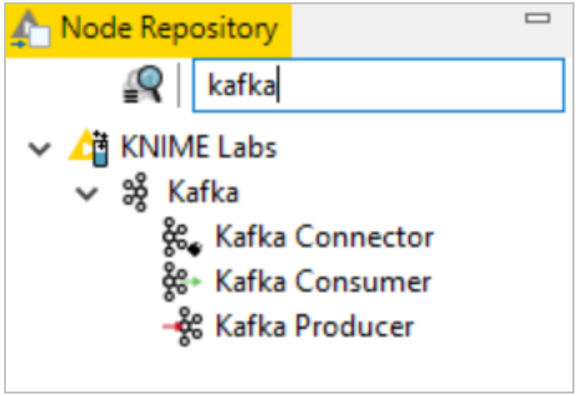


Job pool size N in “Properties”

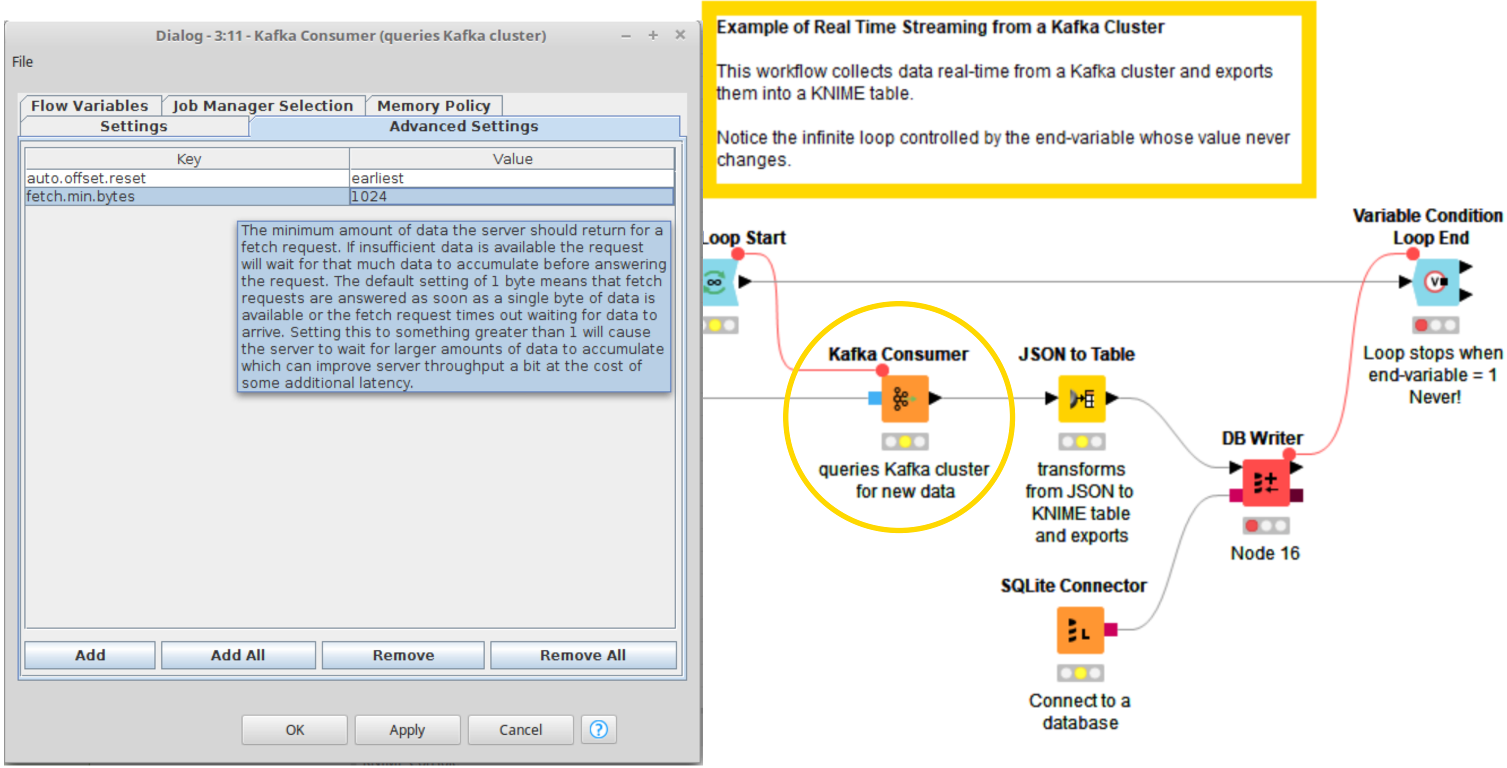


What is Kafka?

- Apache Kafka is a community **distributed** streaming platform capable of handling trillions of events a day.
- Initially conceived as a messaging queue, Kafka is based on an abstraction of a distributed commit log.
- Since being created and open sourced by LinkedIn in 2011, Kafka has quickly evolved from messaging queue to a full-fledged **event streaming platform**.
- KNIME Kafka Integration:



Streaming Solution: with Kafka



In: Kafka_write_and_read workflow on the KNIME Hub

References

- Hyndman, Rob J., and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.
- Gilliland, Michael, Len Tashman, and Udo Sglavo. *Business forecasting: Practical problems and solutions*. John Wiley & Sons, 2016.
- Franses, Philip Hans, and Philip Hans BF Franses. *Time series models for business and economic forecasting*. Cambridge university press, 1998.
- Chatfield, Chris, and Haipeng Xing. *The analysis of time series: an introduction with R*. CRC press, 2019.

Thank You!

education@knime.com