Ile mleka produkuje krowa?
R w środowisku produkcyjnym.

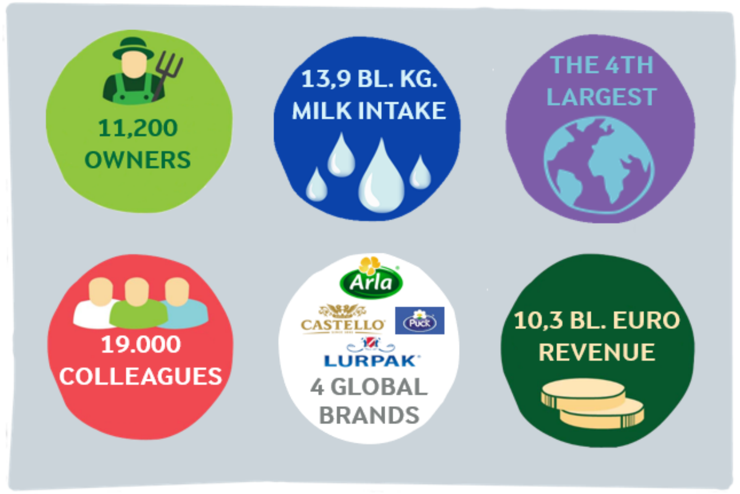
ML Gdańsk 2021

Mateusz Bogdański

Agenda

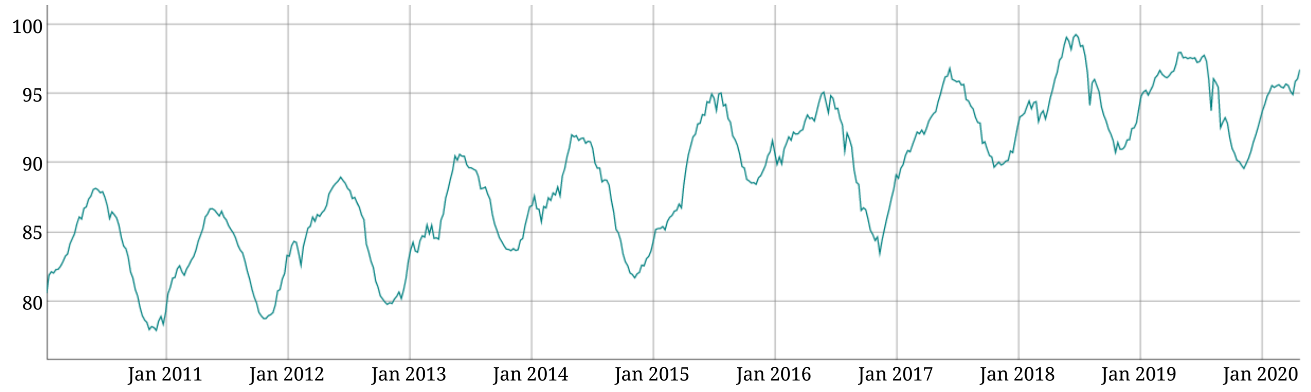
1. **Arla as a company.**
2. **The forecasting problem and PoC.**
3. **Putting the PoC into production.**
4. **Business requirements and what came out of it.**
5. **What exactly was built?**
6. **What we learned?**
7. **Questions.**

About Arla Foods



The forecasting problem

We were asked to forecast the amount of milk collected at Arla farmers in Denmark per week in millions of kgs:



Requirements: A forecast horizon of 12 months or more, forecasts split into various sub-aggregates, and performance on par with the current approach in use.

A brief look at the model

The project started as a proof-of-concept, where we were given a prepared data set to build the model.

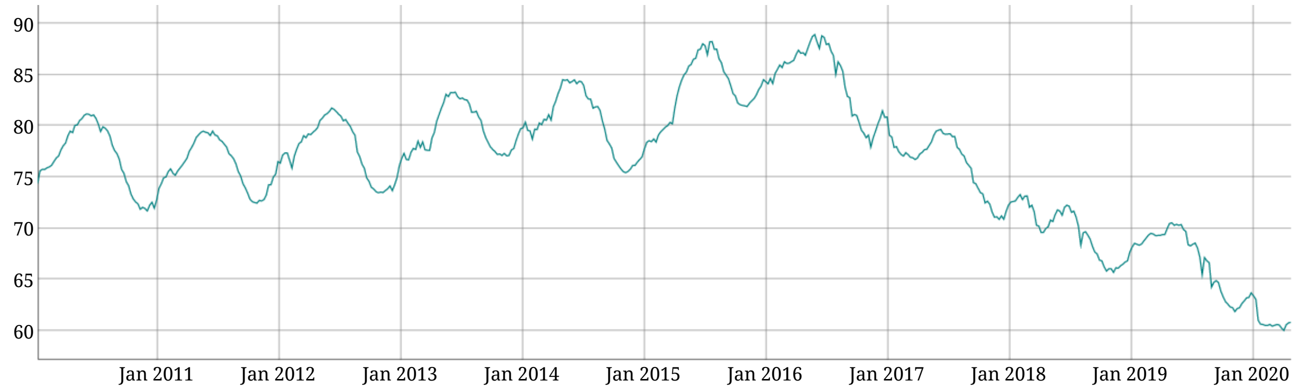
We chose to use an autoregressive model with seasonality, trend, relevant dummies and exogenous regressors:

$$y_t = \alpha + \sum_{i=1}^{p} \beta_i y_{t-i} + \theta' x_t + \sum_{k=1}^{q} [\gamma_k \sin(2\pi kt/52) + \delta_k \cos(2\pi kt/52)] + \varepsilon_t$$

The model was estimated by the LASSO using glmnet in order to do both model selection and determine the autoregressive order. The LASSO tuning parameter, λ, was selected by minimizing the BIC over the grid of λ-values selected by glmnet.

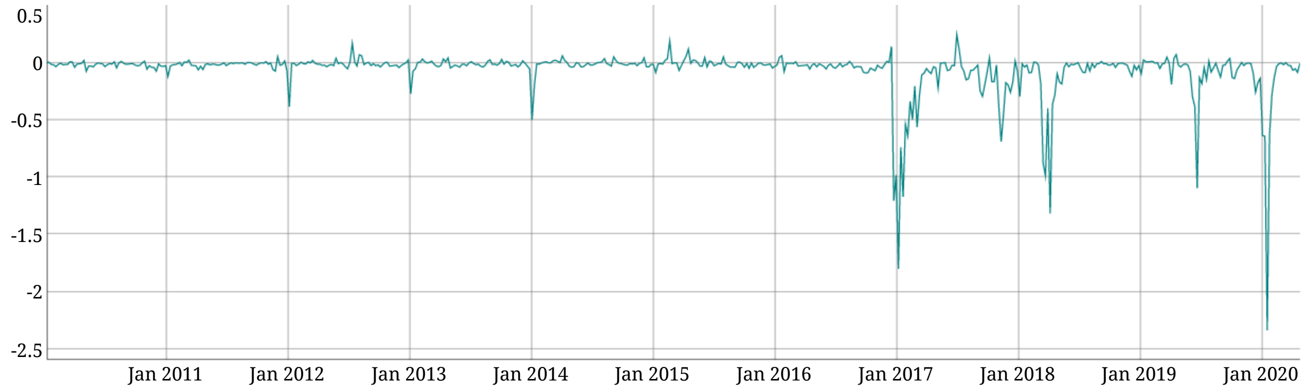
The data issues

The first plot was for the total intake, but the business also needs forecasts for different milk types, the following it the intake of conventional milk. What is going on?



Leavers/joiners/conversions

Well, we have farmers leaving, joining and converting to other milk types. The week-by-week net effect is plotted below. In this case it is mostly farmers converting to either non-GM or organic milk production.



Adjustment

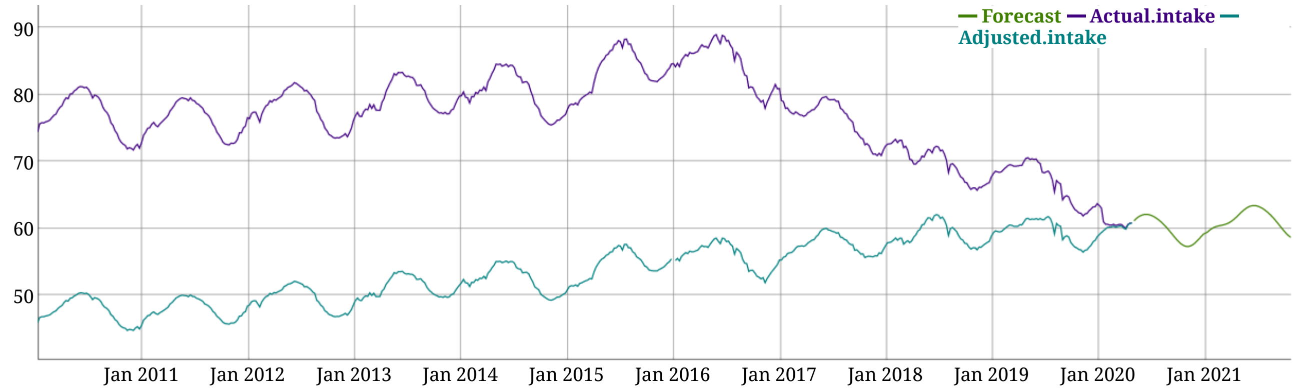
Let Δ_t be the net effect of farmers leaving/joining at time t and y_t be the actual intake (in millions of kgs), then we wish to define a new variable, $\tilde{y}_t$, which is adjusted so the effect of leavers/joiners is removed and further is log-transformed:

$$\begin{aligned}\tilde{y}_t &= \log\left(y_t \prod_{i=0}^{t-1}(1+\Delta_{t-i}/y_{t-i})\right)\\ &= \log(y_t) + \sum_{i=0}^{t-1}\log(1+\Delta_{t-i}/y_{t-i})\end{aligned}$$

The log-transformation ensures that the model forecasts a constant percentage growth in intake over time (as opposed to a constant growth in kgs).

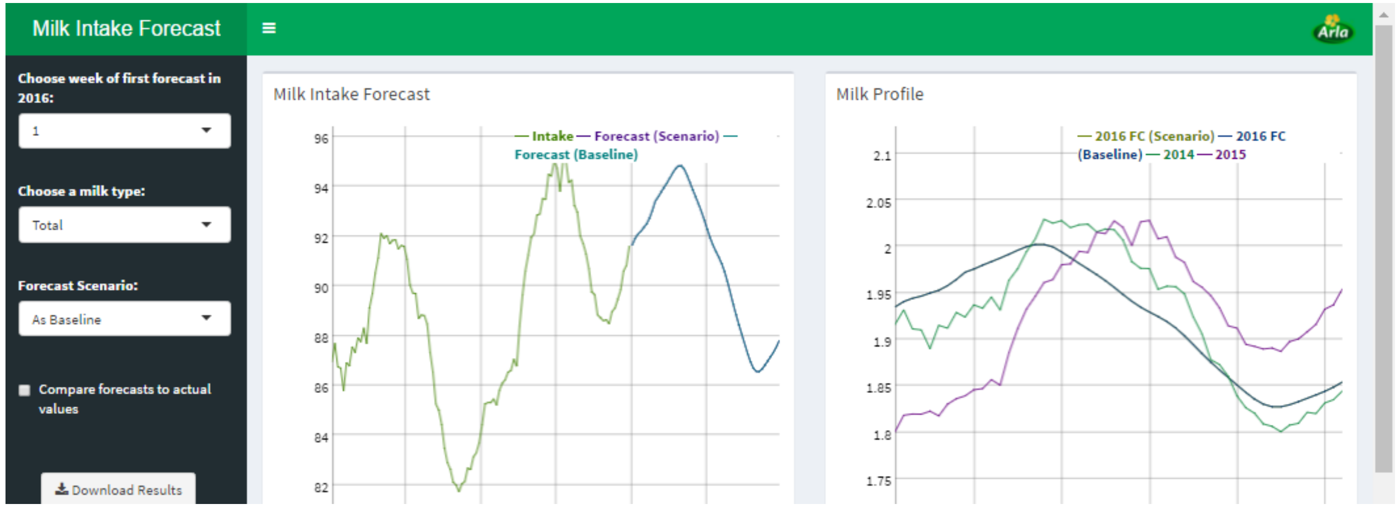
The forecast

By training our model on the adjusted intake data we are able to get a more plausible forecast.



Presenting the results

We used Shiny to visualize the outcome of the PoC in an interactive way with the possibility of exploring the forecasting performance at various points in time.


Milk Intake Forecast
Arla
Choose week of first forecast in 2016:
1
Choose a milk type:
Total
Forecast Scenario:
As Baseline
Compare forecasts to actual values
Download Results
Milk Intake Forecast
— Intake — Forecast (Scenario) — Forecast (Baseline)
96
94
92
90
88
86
84
82
Milk Profile
— 2016 FC (Scenario) — 2016 FC (Baseline) — 2014 — 2015
2.1
2.05
2
1.95
1.9
1.85
1.8
1.75

Putting the model into production

Our stakeholders were happy with the outcome of the PoC and we were asked to put the model into production, but how do we do that?

First problem: The code from the PoC didn't work anymore. We thought we had done enough by keeping my code under version control, but we had not tracked the package versions used in the code.

First change: Use `checkpoint`. Define a global configuration, e.g. `cpconf.R`:

```r
.cpconf <- list(snapshotDate="2019-08-06", R.version = "3.6.1")
```

And then in the beginning of the scripts we need to run, e.g. the Shiny dashboard:

```r
source("cpconf.R")
do.call(checkpoint::checkpoint, c(.cpconf, scanForPackages = FALSE))
```

Caveat: `snapshotDate` and `R.version` must "match". Alternatives: `packrat`, `renv`.

Putting the model into production

1. What are the actual business requirements?

2. Where did our data set come from? We need to track down the source systems and ensure that we can get reliable frequent updates.

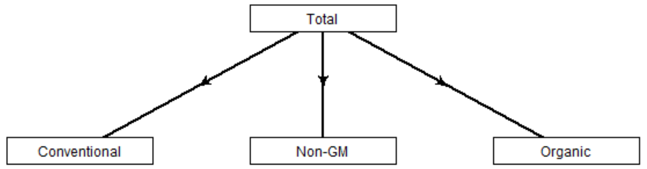
3. Where in our IT landscape can we actually run our model? And what about Shiny? Examples of the questions we met:

    - *Can you even run R code in production?*
    - *Why don't you just rewrite it in C#?*
    - *Couldn't you at least use Python instead?*

Hierarchy of models

One requirement we hadn't met in the PoC was summability of forecasts. Basically, the problem is that we need to produce forecasts both for individual milk types and for the total intake, and the sum of the forecasts of the milk types should equal the forecast for the total intake.

For DK the problem is quite simple. We train four models, one for each of the three milk types and one for the total intake, hence we have a hierarchy:



Summability of forecasts

The problem is that the sum of the milk type forecasts will not equal the forecast of total intake. We would like to impose this summability condition. We do so by using Hyndman et al. (2011). Let

$$\begin{aligned} Y_t &= [y_{t,total}, y_{t,conv}, y_{t,nongm}, y_{t,org}]' \\ Y_t^* &= [y_{t,conv}, y_{t,nongm}, y_{t,org}] \end{aligned}$$

then the following relationship holds:

$$Y_t = SY_t^*$$

where

$$S = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Summability of forecasts

According to Hyndman et al. (2011) the optimal weighting that imposes summability is given by:

$$\tilde{Y}_t = S(S'S)^{-1}S'\hat{Y}_t$$

where $\hat{Y}_t$ is the forecast of Y_t, i.e. the four individual series we started out with.

Potential issue:

- If one of the forecasts is bad, this will now affect all forecasts! It is crucial that your stakeholders understand the trade-off between a high quality aggregated forecast and many, potentially low quality, sub-aggregates.

Business requirements

The business requirements appeared quite simple - take the model from the PoC, put it into production so we can get new forecasts every week. But...

1. How good should the model be? Was the forecasting performance from the PoC good enough? And what is "good enough"? Can performance be used as an acceptance criteria?
2. How should the forecasts be "delivered" to the business?
3. How do we ensure that the business trusts the model and actually uses the forecasts?
4. When should the forecasts be available? And what if we miss that deadline?
5. Could you expand the capabilities of the model? Scenario analysis? 5 year forecasts?
6. And, finally, the model of course needed to be expanded to cover all our markets, but that's just a matter of swapping the data set, right?

Technical requirements

Our first challenge was to turn the business requirements into technical requirements.

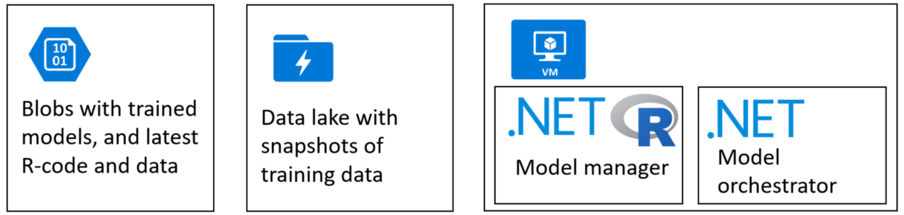
1. Performance: With the heavy focus on forecast performance, how could we ensure that we could test and document the performance of our models?
    - We did not trust the source systems, so we needed a solution to keep track of upstream data changes.
    - Any forecast produced needed to be linked to the model that produced it.
2. Delivery: We used a Shiny dashboard in the PoC - so let's do the same in the full solution.
3. Trust: Let's give the business a solution where they can approve newly trained models and revert back to older versions if needed.
4. Scenario analysis: We need a solution where the business can interact with the model and explore scenarios.

What we built

We had no experience with putting models into production, but we had a team of software developers that could help us build a solution.

- We are using Microsoft Azure and initially looked at the services available there, but nothing matched our requirements.
- The advice from the developers was clear: You should not build an API on top of your R code, let us wrap it in C#/.Net instead. That way you only have to worry about your R code and we will orchestrate the entire thing and save what needs to be saved in a database.
- That created a separation of responsibilities between data science and data engineering. As a data scientists we "just" needed to create code that read their CSV-files and saved the results in a pre-specified way.
- We could still use Shiny for dash-boarding, we just needed to connect to their APIs instead of calling the model directly in the code.

Model Manager



Blobs with trained models, and latest R-code and data
Data lake with snapshots of training data
VM
.NET
R
Model manager
.NET
Model orchestrator

- All R code is uploaded to a blob. The code needs to include two scripts `train.R` and `predict.R`.
- The Model Manager exposes two endpoints:
  - train: Copies the input data to the `input` folder. Runs the `train.R` script, trains the model, assigns an id and saves it in the blob together with its context. R versions are handled by the Model Manager, package versions by `checkpoint`.
  - predict: The id of the model is supplied. The Model manager retrieves the model (including context) from the blob and runs the `predict.R` script passing any parameters passed to the API. The result is returned to the caller.

Deployment

- All back-end components follow software development best practices. E.g.
  - Using git for source control.
  - Having separate development and production environments.
  - Using build and release pipelines in Azure.
- The R code for the forecasting model and Shiny dashboard is also under source control in git.
- There is no automatic/managed deployment of the model code, it is manually copied to a blob in either our development or production environment.
  - The biggest issue with this is that there is no (easy) way of knowing which model is running in production. It would be better if deployment was linked to a commit in the git repository.
- The Shiny dashboard packaged in a Docker image and is built/deployed using Azure Pipelines.
  - The Docker container is running on manually set-up VMs.
  - Authentication is handled by Azure Application Gateway.

Data sources

We of course had all the usual problems with data and data quality, but a few issues stand out:

- The business provided the data for the PoC, but between the reports where they extract data from and the source systems things happen - data is altered and tracing out that logic is very difficult.
- Some of the data from the PoC did not exist in any systems, but were maintained in local Excel files or sent by email from external parties.
- Uncertainty about whether we were allowed to use the data under GDPR.

Having spent a lot of time building what is essentially now a small data warehouse with consolidated and cleaned data for our model we now face a new issue:

- People are now coming to us for the data instead of going to the source systems.
- On top of that our forecasts are now considered source data for other systems - how do we best supply them?

We did not realize that the data collected and cleaned "just" for the model would end up being this valuable.

What we learned: Did we over-engineer it?

In the end our Model Manager has enabled us to put our model into production, and it has many great features, but an obvious questions is whether we over-engineered it?

- We assumed that the business wanted control over putting newly trained models into production/reverting back to old versions. In the end they didn't, they just wanted us to take care of it.
- We assumed they wanted a Shiny dashboard as in the PoC, but in the end it turned out they preferred having all data and forecasts available in PowerBI and then do the dash-boarding themselves.
- In the PoC we also experimented with scenario analysis and the need for this ended up in the requirements for the Model Manager. It, however, turned our that the model was not suited for scenario analysis so it was never implemented.

What we learned: User adoption

Once the model/solution has been delivered, how do we then ensure that the business actually uses it?

- In our case the business (and not just the manager) is fortunately very interested in adopting the model, but that does not mean they fully trust it yet. We are attempting to build this trust by having monthly review meetings where we compare the model's forecasts with the business'.
- In order to ensure full user adoption we have implemented a feature where they can (arbitrarily) adjust the forecasts of the model. We accept this solution because we know the alternative would be that the users still made adjustments but in Excel sheets out of our control. By doing like this we can track the changes the make and compare performance.
- While it could be argued that our Model Manager was over-engineered, the comprehensive solution we now had made it very easy to implement the manual adjustments feature - a feature that would not have been possible in a pure PowerBI solution but was easily implemented in Shiny.

What we learned: Separation

Throughout the design of the solution there has been a focus on separation:

1. Separation between data engineering and data science: The data scientist does not need to know about the details of the data model / the design is better left to the data engineer.

2. Separation between data science and operations: The data scientist does not need to know about APIs, accessing the data lake, blobs, etc. All that is needed is to deliver the R files.

While this in principle makes life easier for the data scientist, it creates a large dependency on your data engineer and software developer. For example, even changes to the model that appear small may now require the software developer to make changes to the Model Manager.

In our case it means that we now need to have a .Net developer in our team in order to maintain the solution.

What we learned: Operations

Of course, putting a model into production isn't just about that one step, it is just as much about maintenance and daily operations. Being part of a large IT department, this is of course nothing new when it comes to IT solutions, but we had never tried it before for a forecasting model.

Basically, our Operations department could not (should not?) take over the model, which means:

- Anything we build we have to run and maintain.
    - Shiny is a great tool for dash-boarding and prototyping but you easily end up in spending a lot of time supporting it.
- We need to support the solution. What is the support process? Help desk? SLA? Support outside business hours?
- How do we monitor our models: Are they still working and what are their performance?
- You need a team of data scientists to handle support and maintenance, i.e. more than one person!

Questions?